Data Management

GIT: A version control system

Malka Guillot

HEC Liège | ECON2306

Table of contents

The importance of version control

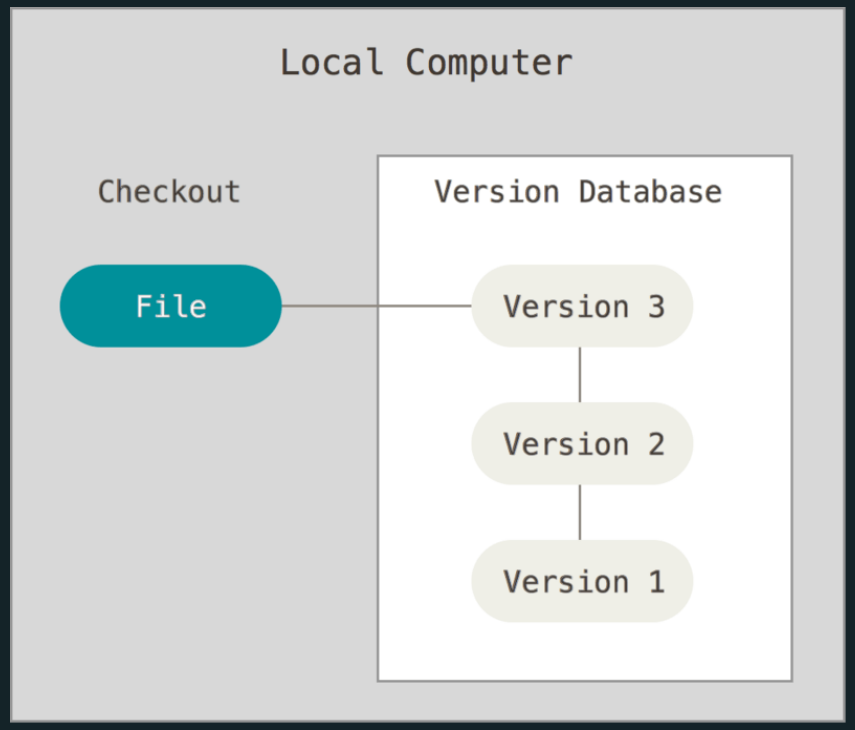
What is version control?

Version control is a way to keep track of changes to code, text, and documents. And data and outputs.

- It gives you an organized revision history
- It lets you experiment *without fear*
- It lets you go back and forth between many different versions of the same file, and see a list of the differences
- It makes (the technical aspects of) collaboration a breeze
- It lets you and your collaborators work on different versions and then merge them

From local to distributed version control system

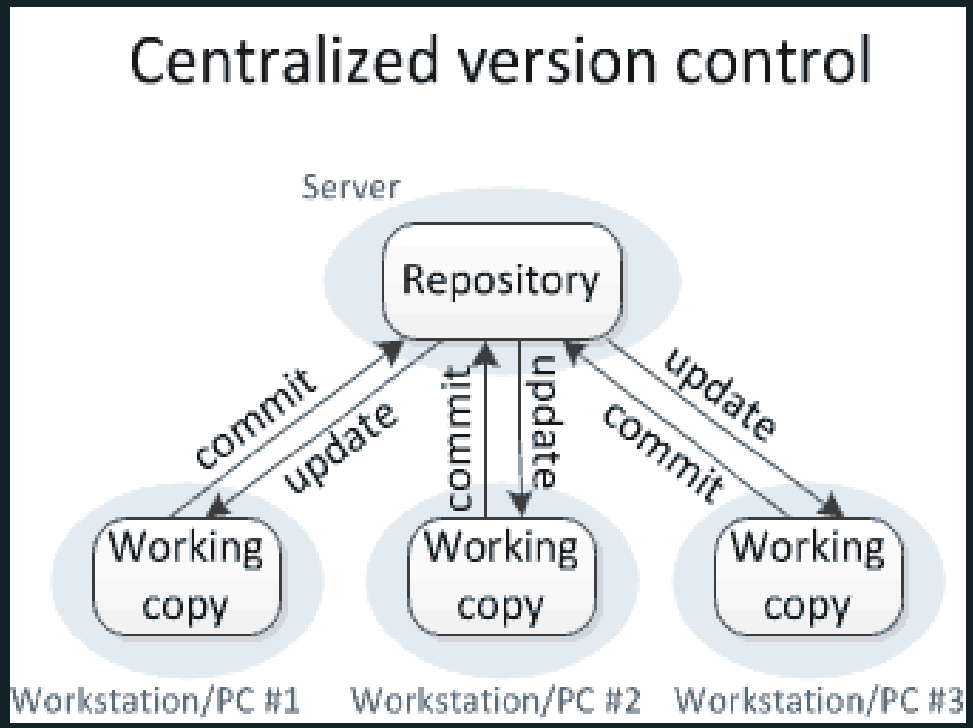
- **Local**: everything is on your computer



  - No collaboration
  - Not possible to retrieve files if the local machine crashes

From local to distributed version control system

- **Centralized**:
  - all files on 1 server
  - many collaborators checkout files

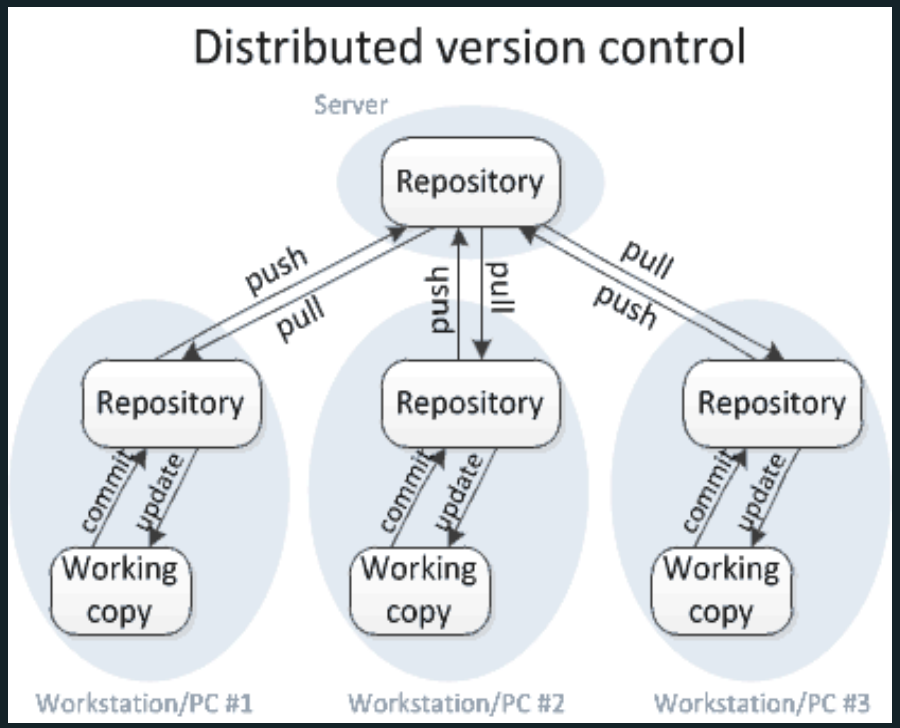


+ Collaboration

– Not possible to retrieve files if the central server crashes

From local to distributed version control system

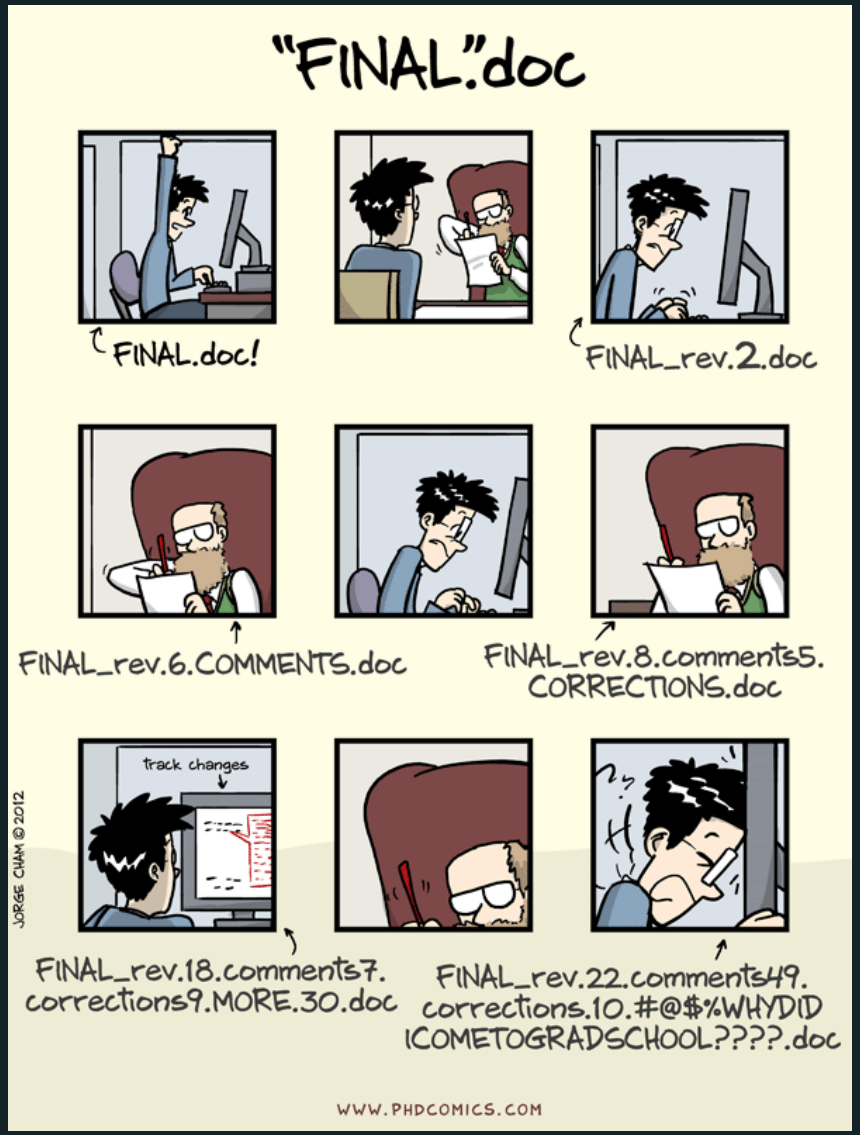
- **Distributed**:
  - one or more servers
  - many collaborators



+ Collaboration

+ Each user has their own repository and a working copy

Why bother?



Also git vs. Dropbox from a researcher's perspective

[CCL] Version control system

Enables **coordinatation** → no code change is lost or accidentally overwriten.

Provides an organized **sharing** platform → *open source* & documentation

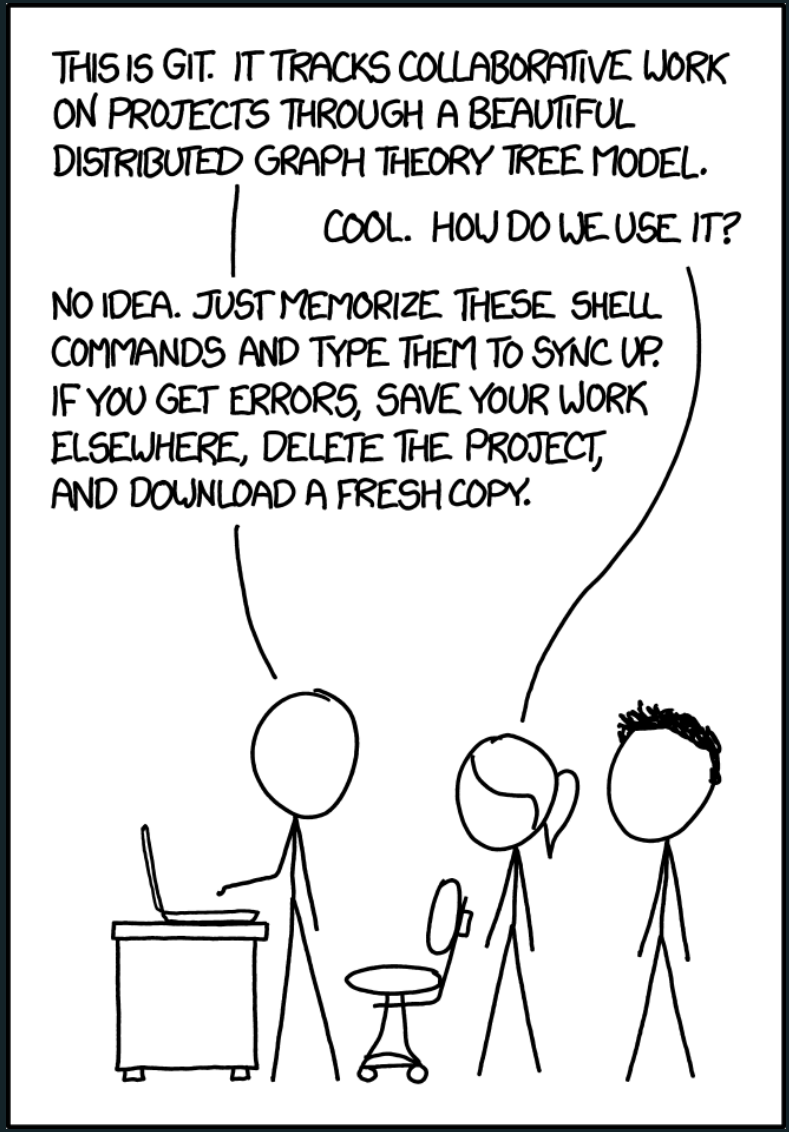
⇒ key tool from our **project management** perspective

⇒ widely used in a companies / not enough in research:

- Software development
- Scientific researcher
- Anything involving coding (even latex)

Git(Hub)

This is Git



Source: https://xkcd.com/1597/

Git(Hub): a solution

- Git:
  - Git is a **distributed version control system**. *(Wait, what?)*
  - *Okay, try this:* Imagine if Dropbox and the "Track changes" feature in MS Word had a baby. Git would be that baby.
  - most popular *open source* version control system out there.
- GitHub
  - GitHub = **online hosting platform** that provides an array of services built on top of the Git system.
    - (Similar platforms include Bitbucket and GitLab.)

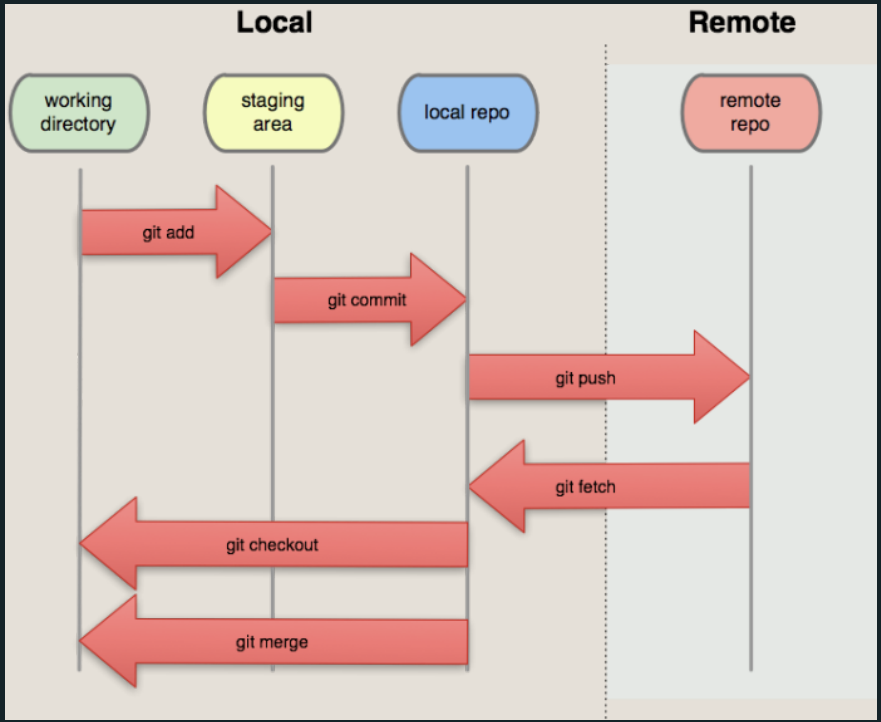
Git vs. Github

- It's important to realize that Git and GitHub are distinct things.
- We don't *need* GitHub to use Git... But it will make our lives so much easier.

→ There is a learning curve, but I promise you it's worth it.

Git model

1. You do work in your **working directory**
2. Then you add it to your **staging area**
3. Once you've staged all you changes for one discrete task, **commit** a snapshot of the staging area
4. If you have a remote repository, **push your commit**

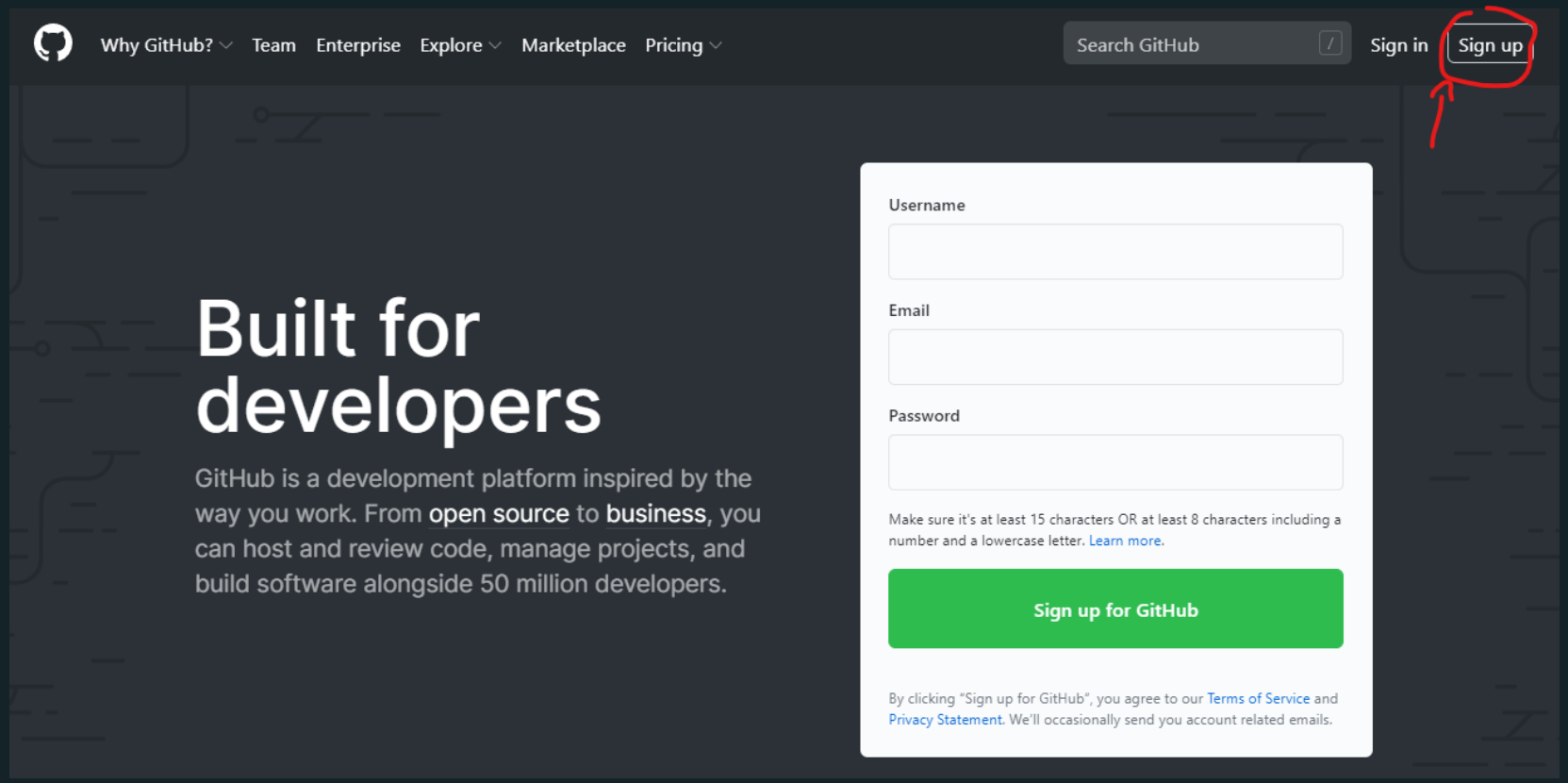


Getting started on a project

Where we create our first repository!

[Task 1] Setup GitHub account

- Navigate to GitHub's homepage + "Sign Up"
  - Go through the account setting steps ("Verify your email address"...)



Navigate to GitHub's homepage. Navigate to "Sign Up" in the top right hand side of the page.

[Task 2] Getting started with Git(Hub)

1. Install Git (Linux, Mac, Windows) if not already installed
2. Git comes with a command line interface (powerful!).
3. You might want to add a *graphical interface* to make things easier:
    - GitHub desktop
    - You can link it with your GitHub account

[Task 3] Your first (local) repository

Let's look at an example using GitHub desktop

1. Open GitHub Desktop and select File/New repository
2. Choose the name and the local directory to use
3. Start working in the directory, i.e.
    - Create some .txt file with some text
    - Commit it
    - Make a modification, and commit again: look at the changes!

[Hint] What actually is the Git repository?

- The Git local repository is associated with a particular directory
- Open the directory in your Git interface to see your options
- Git stores all its workings in that directory in a hidden subfolder called ".git"

3 special options:

- **README.md**: description of the directory
- **.gitignore**: what should be ignored by the tracking systel
- **licence** → open source?

[Hint] What should I include?

1. At a minimum:
    - Code (.do, .py, .R, .m, .jl, and so on)
    - Text files (.txt)
    - LATEX documents (.tex)
2. I also recommend:
    - Raw .csv datasets, if small (<10 MB)
3. These are binary files, so you can't see differences between versions. I recommend including them anyway.
    - PDF files
    - Word, Excel, PowerPoint files
4. Some people also include all datasets.
    - Note that GitHub doesn't allow files larger than 100 MB, or projects with total size larger than 1 GB.

For datasets, look into Git Large File Storage.

[Hint] What should I exclude?

In order to avoid driving your collaborators crazy, you must tell Git to ignore the junk files using a file called .gitignore. It looks like this:

- Junk created by LaTeX: *.synctex.gz, *.out *.log
- Junk created by Python: *.pyc

Best practice: use .gitignore to explicitly exclude everything that you don't want to include, and commit .gitignore like any other regular file.

GitHub maintains a list of standard .gitignore files for many common languages.

Backbone of git: Commits & branches

Where we commit ourselves (locally)!

Commits: saving a snapshot

"One discrete task" = a collection of changes, across multiple files (or not), that does *one thing*.

Examples:

- Change the formatting of a variable from string to numeric, and treat it properly across multiple scripts
- Change your regression specification in code, in the output, and in your paper and supporting documentation
- Add a new function

Before you commit

- Your code should run properly → run tests
- No compilation erros (in Latex for example)
- Output should be consistent inside the commit (including comments)

But it's better to have *frequent commits* (that might have small mistakes) than to have *giant, infrequent* commits.

Viewing changes when committing

minor correction

mguillot 2ba8acb 1 changed file +3 -3

lectures/0-overview.md

```diff
@@ -106,7 +106,7 @@ Belgique

 --

-## Introduction: You are you ?
+## Introduction: Who are you ?


 <div style="position:relative;  text-align: center;" >
@@ -114,7 +114,7 @@ Belgique

 --

-## What do you want to learn during the class?
+## What do you expect to learn during the class?


 <div style="position:relative;  text-align: center;" >
@@ -644,7 +644,7 @@ No general texbook. Specific references will be given when corresponding subject

 - [Introduction](https://pp4rs.github.io/pp4rs-python/intro.html) to python, pandas, plotting

-- [Stackoverflow](https://stackoverflow.com/): all the answers are there, but you have to ask the right question.
+- [Stackoverflow](https://stackoverflow.com/): all the answers are there, but you have to ask the right question.

 ---
```

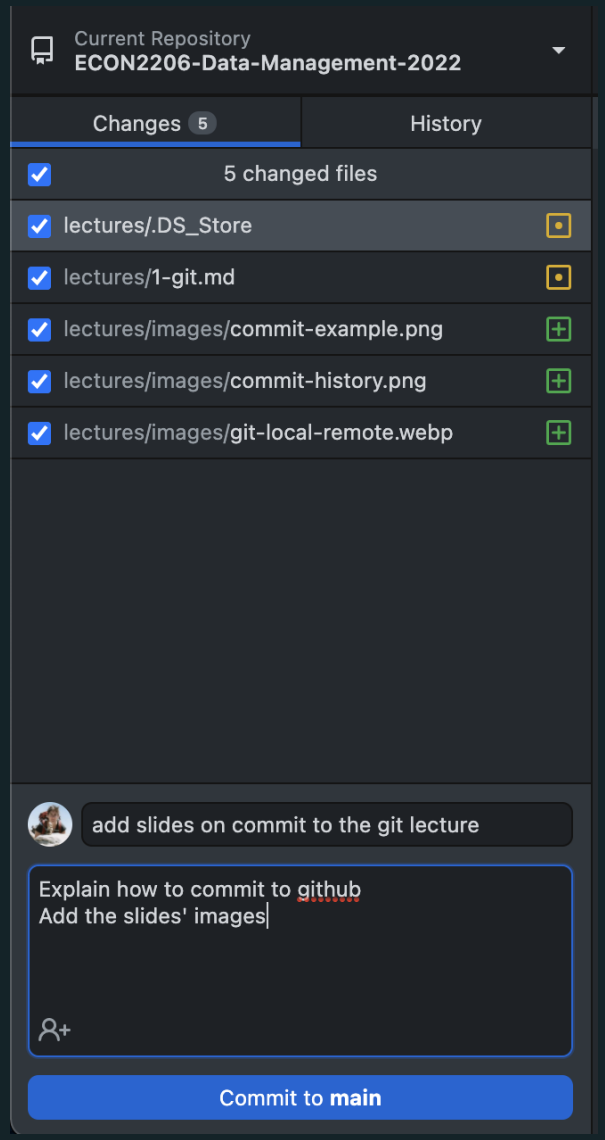
Commit message

Examples:

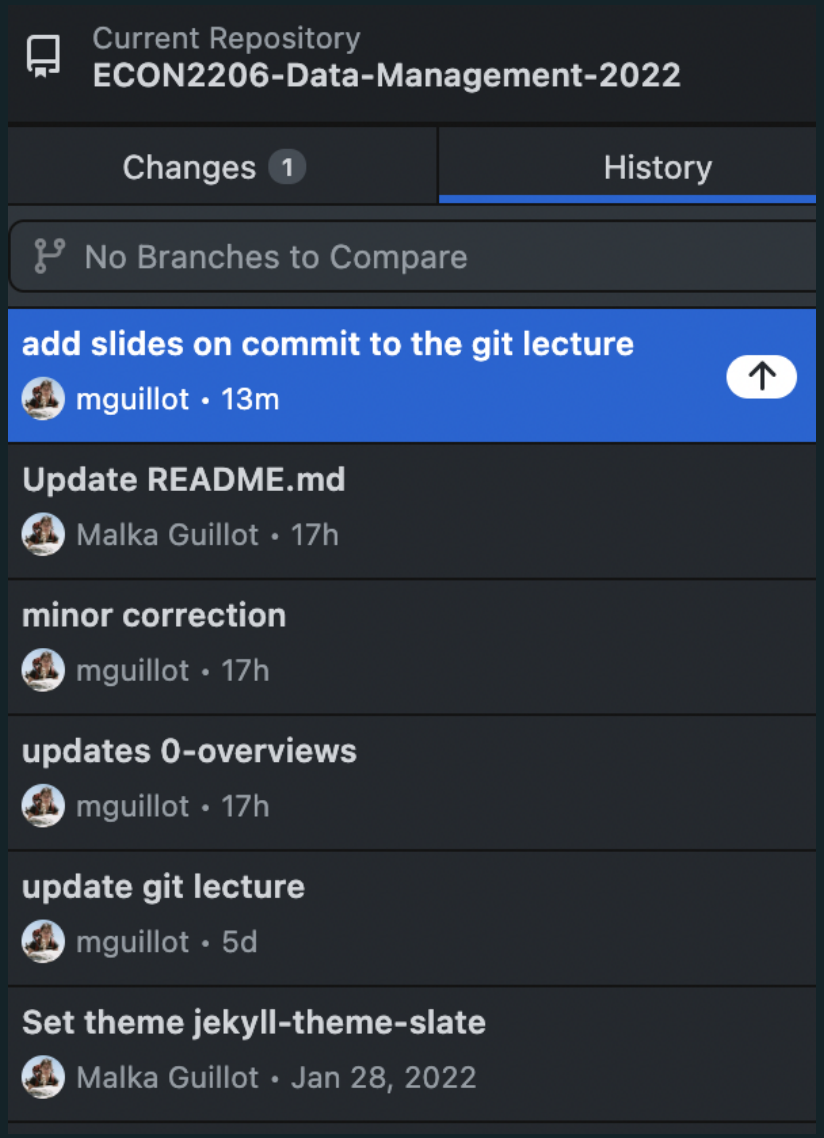
- “Change the formatting of start date variable from string to date format”
- “Add year dummies to regression specification”

→ The more detail, the more your future self will thank you.

Commit message: example

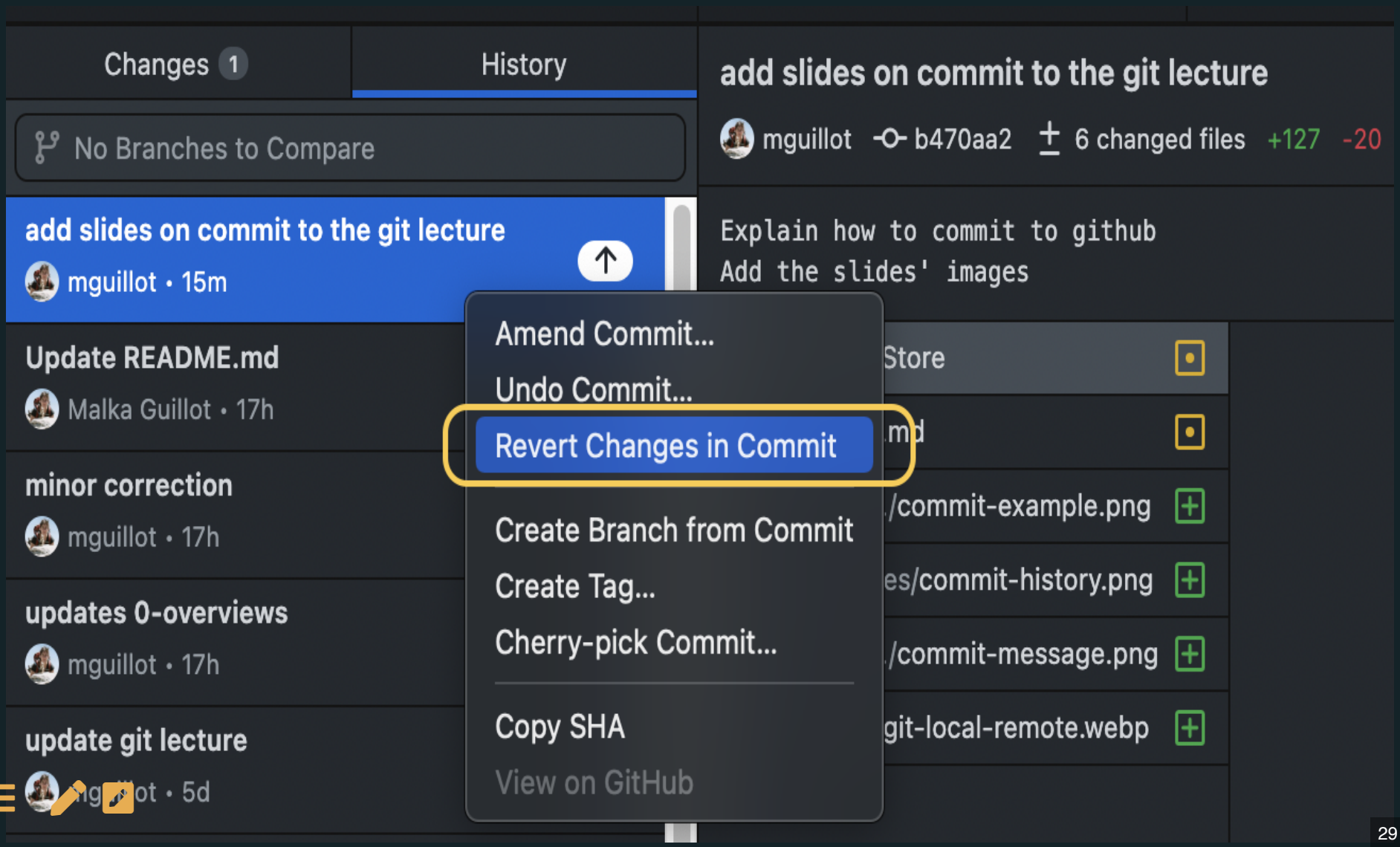


Viewing commit history



When things go wrong: go back in time

What happens when a commit was a mistake? **Revert it**, to make a new commit that undoes it.



This can happen!

Source: https://xkcd.com/1296/

Branches: trying things out

Branches are the most powerful part of Git

- By default, all the work you do goes into the “master” branch
- Want to experiment? Start a new branch
  - You can switch between branches, and make commits to either branch
- If your experiment works out, commit and merge back into the master branch
  - If there are conflicts between the commits you’ve made on the two branches, Git will ask you to resolve them
  - This is easiest with a graphical interface like GitKraken
  - Only works with binary files
  - If your experiment doesn’t work out, delete the new branch painlessly

Keeping it local vs. using a remote repository

Git doesn't require a remote repository. You can run it 100% on your computer, with no connection to an outside server.

- Useful if you have restrictions on your code (e.g. confidential health data)
- A remote repository helps
  - keep things backed up seamlessly,
  - collaborate with others
- You can push all your branches to the remote repository, or only some of them
- Big companies often have an internal git server

Collaborating

Where we open ourselves to others and go remote!

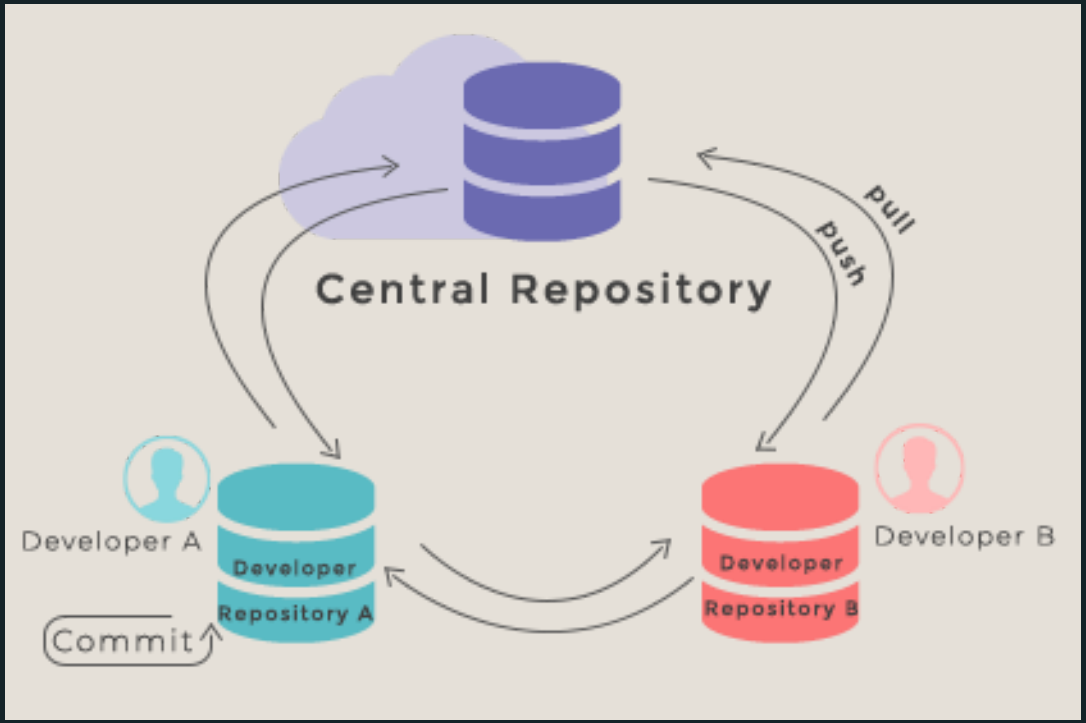
Interacting with the remote directory

The remote repository is on a server, and holds a record of your commits and branches

You push to the remote repository to save all your commits

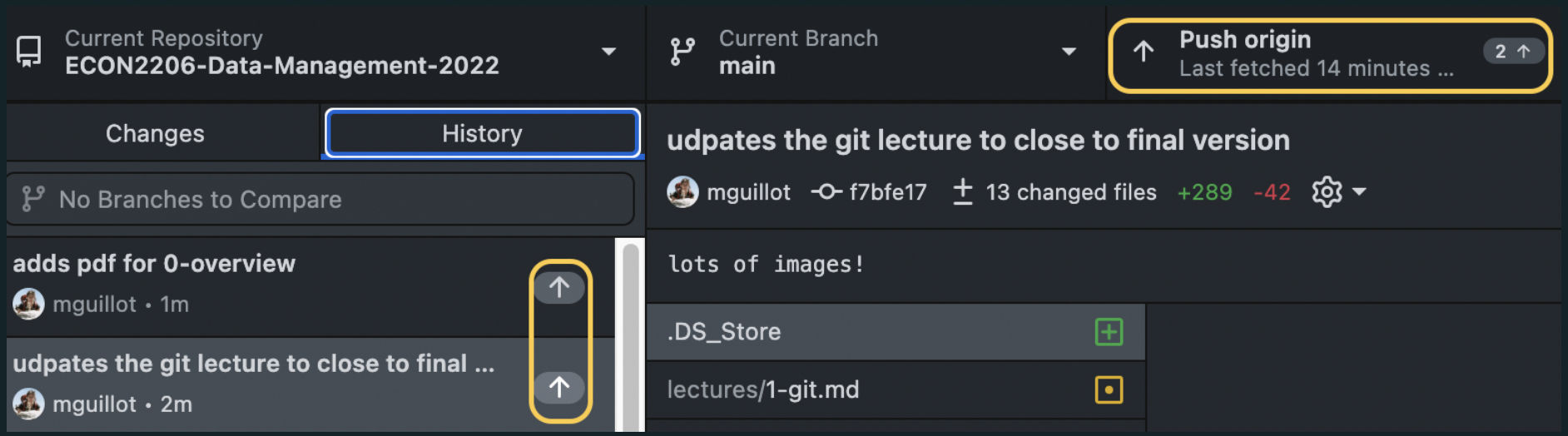
- You pull from the remote repository to load all new commits
- Always commit before pushing or pulling
- If what you're doing is an experiment, make a new branch to avoid any trouble for your coauthor
- If there are conflicts between your commits and your colleagues's commits, Git will ask you to resolve them

Basic workflow: push - pull



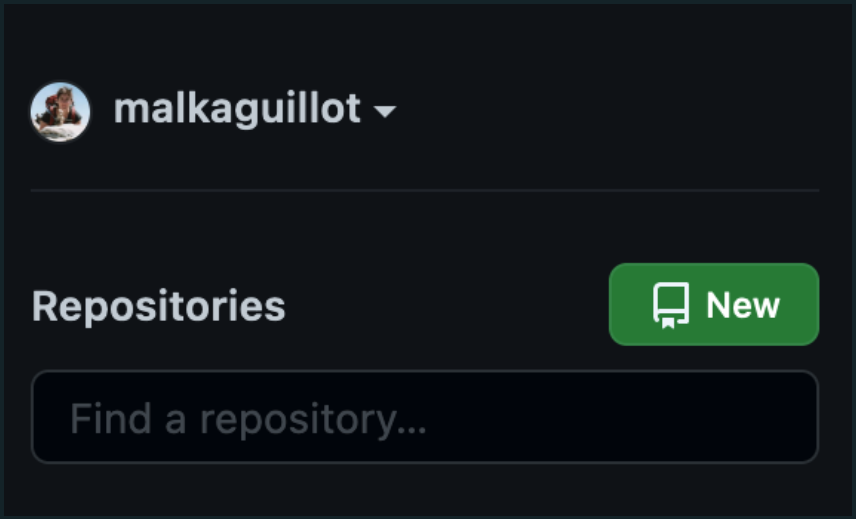
This is what happens between your computer (local) and your repository (remote).

Pushing to the remote repository (GitHub Desktop)



Sending my commits to the internet!

Create a remote repository



- Make sure you click the box to initialize it with a README
- gitignore → python template
- licence

Create a remote repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Repository template

Start your repository with a template repository's contents.

No template

Owner * malkaguillot / **Repository name** * Data Management

Great repository names are Your new repository will be created as **Data-Management**. automatic-barnacle?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

- [x] **Add a README file**
  This is where you can write a long description for your project. Learn more.
- [x] **Add .gitignore**
  Choose which files not to track from a list of templates. Learn more.
  .gitignore template: Python
- [ ] **Choose a license**
  A license tells others what they can and can't do with your code. Learn more.

This will set `main` as the default branch. Change the default name in your settings.

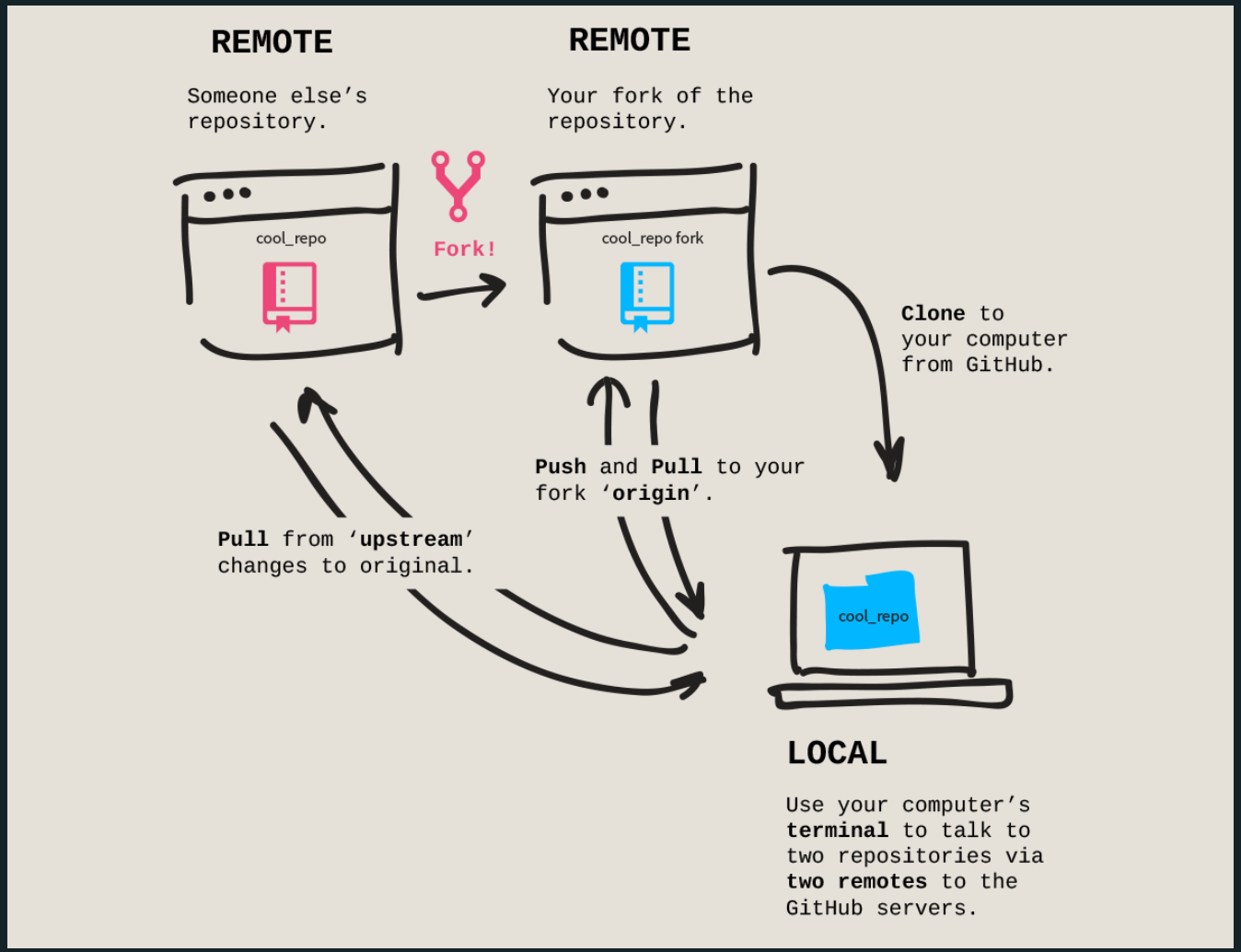
Create repository

The README.md

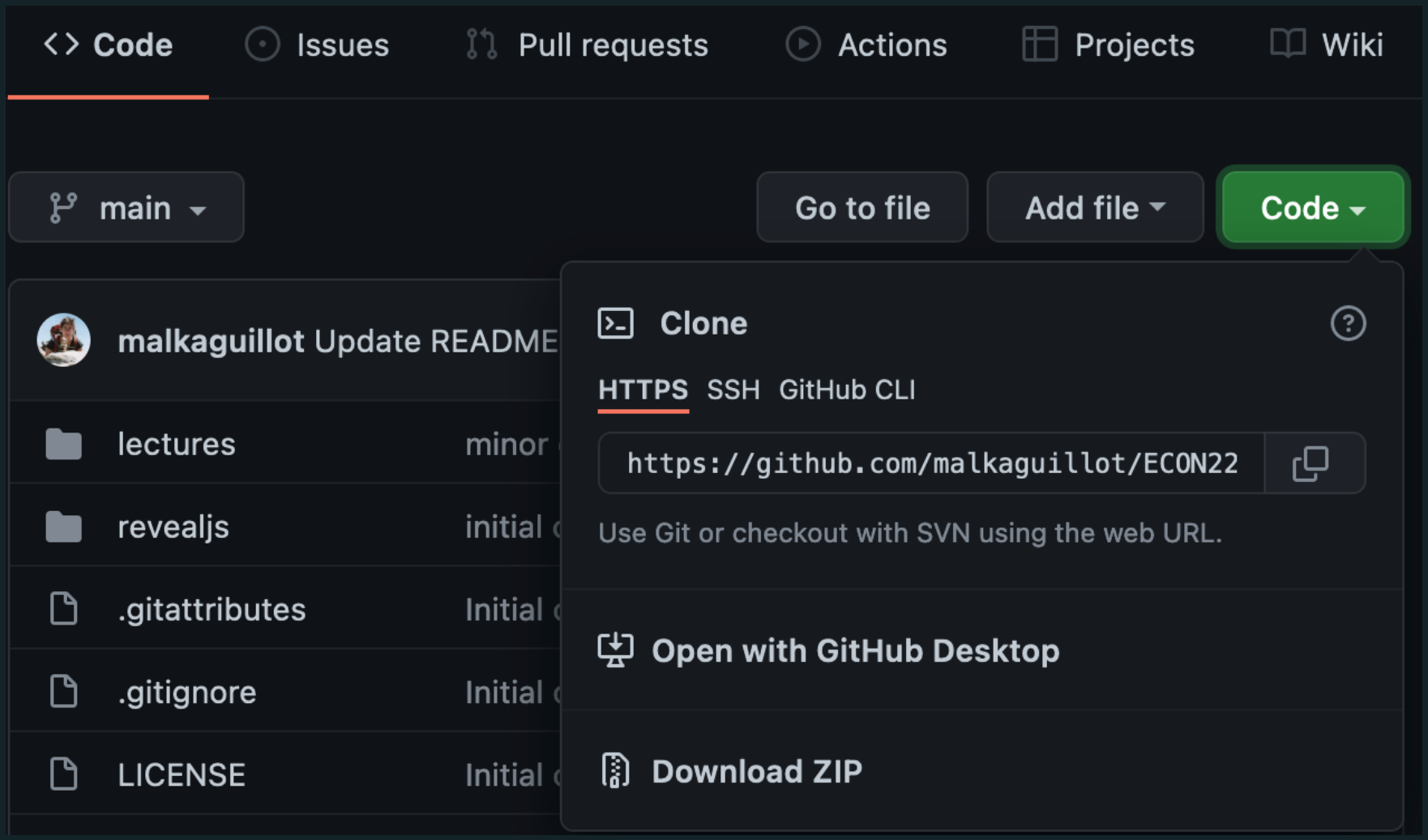
- Very important file!
  - Objective: communicate important information about your project
- A markdown file
- Markdown?= lightweight markup language
  - The guide
  - The syntax

Not only useful for README: for eg., these slides are written in markdown!

Basic work: clone or fork ?



Cloning a repo

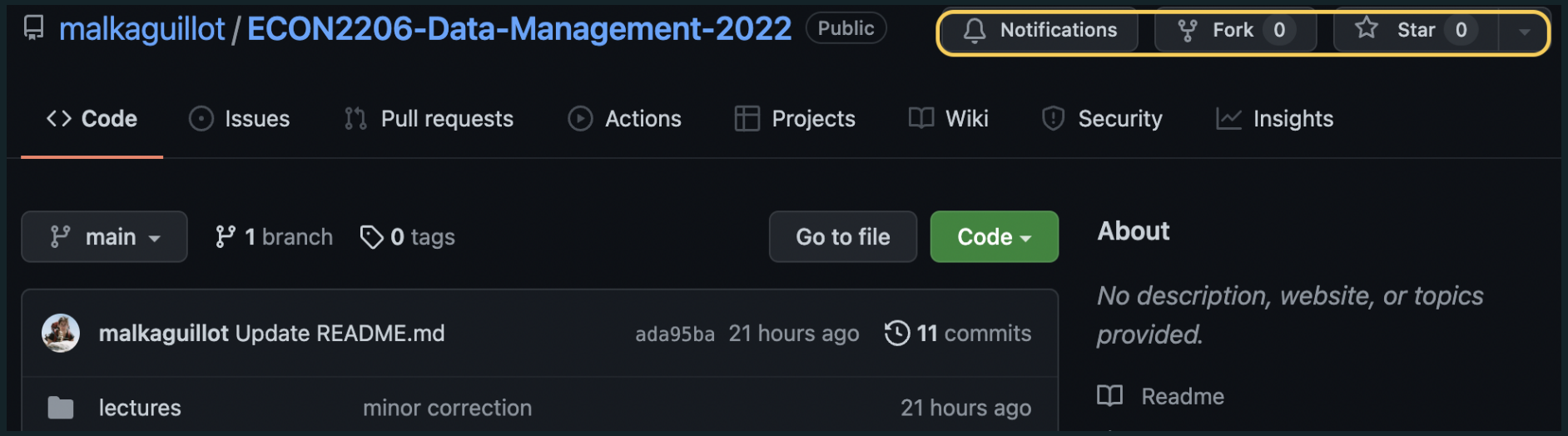


Git Challenge 1

- Create an example repository on your GitHub account (including a readme).
- git clone this repository to your computer. Go to this directory.
- Create three files named file1.txt, file2.txt, and file3.txt in your local repository.
- Stage, commit, and push file1.txt to your remote repository. Refresh the URL on your GitHub page. Do you see your commit?
- Stage, commit, and push file2.txt and file3.txt to your remote repository as a single commit.

Navigating GitHub

Example: our course repository



- **Notification**: Notify you when there are changes or conversations in the repo.
- **Star**: Add this repo to a list of repos that appear in your feed. Think of this as "favoriting" a repo.
- **Fork**: Make a copy of this repository in your own account. → Useful if you are not directly involved with a project but want to build on top of someone else's code.

Git challenge 2 (using GitHub desktop):

- Fork the course repository
- Change the **upstream repository**
  - *In repository settings*: change the "Primary remote repository" to my repo HTTPS addres
- Create a folder **sandbox**: this is were you are going to work!
- Open the .gitignore (you can create it still)
- add on a new line: **sandbox/*** : this will ignore the content of the sandbox when working with the remote => no conflict !
- create a toy file in the sandbox
- In the meantime, I make a commit
- Then can you fetch my commit?

Epilogue

Want more of this?

Let's learn one day how to use the command line interface!

How to interact with the materials?

- Set up GitHub
- Fork the class repository (-> your remote repository)
- Clone your repository on your computer (-> your local repository)
- Add an upstream origin (mine)
- Work in the *sandbox* folder
  - this way, you can fetch my updates

References

- Extensive git manual: https://happygitwithr.com/
- git - the simple guide
- github cheatsheet https://education.github.com/git-cheat-sheet-education.pdf
- interactive tutorial https://gitimmersion.com/index.html
- interactive tutorial on git branching https://learngitbranching.js.org/?locale=fr_FR
- In case it goes wrong: http://ohshitgit.com/

For next week:

- Get confortable with using **Git(Hub)**
  - practice with the challenges
  - go over references
  - work on the interactive tutorials
- **Python** installation
  - Install Anaconda, try out to run python in a Jupyter notebook and spyder
  - See installation guide link
  - Wait for next week's introduction by Michel!
    - Basics of python's syntax: Learn Python
      - less Classes and Objects + Modules and Packages.