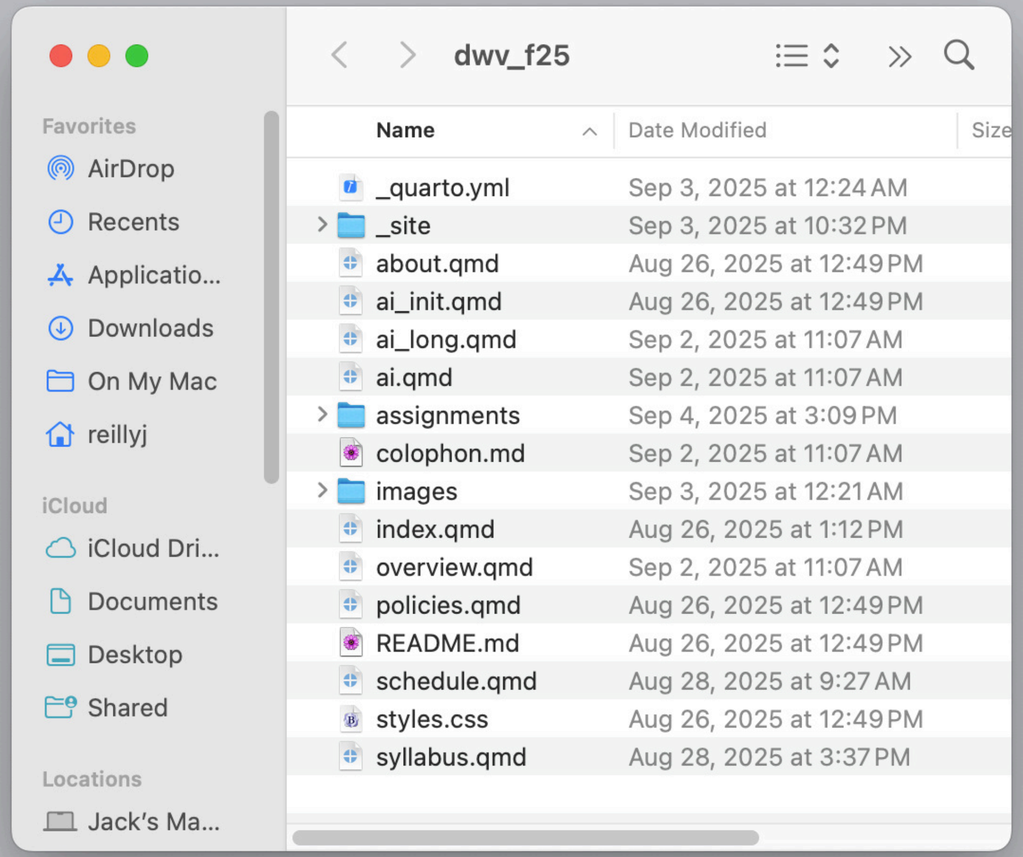
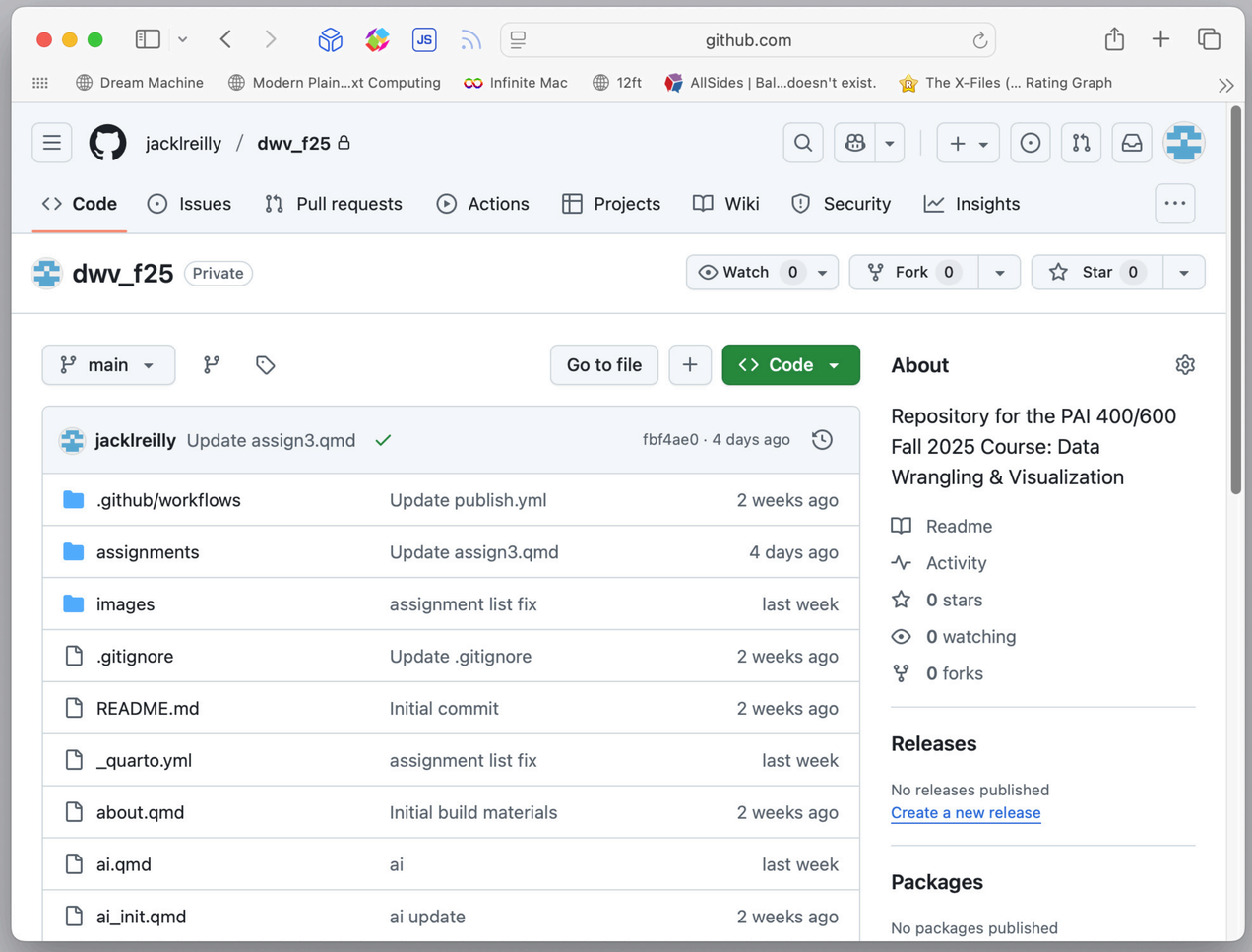




File Management & Version Control

Never lose anything ever again

CONTROL

- and -

Reproducibility

The to-do list

we want to know the **exact code**
that produces the **exact result**
that we **exactly report**

also, we'd like to **keep records**
preferably **open**

What does our workflow look like?

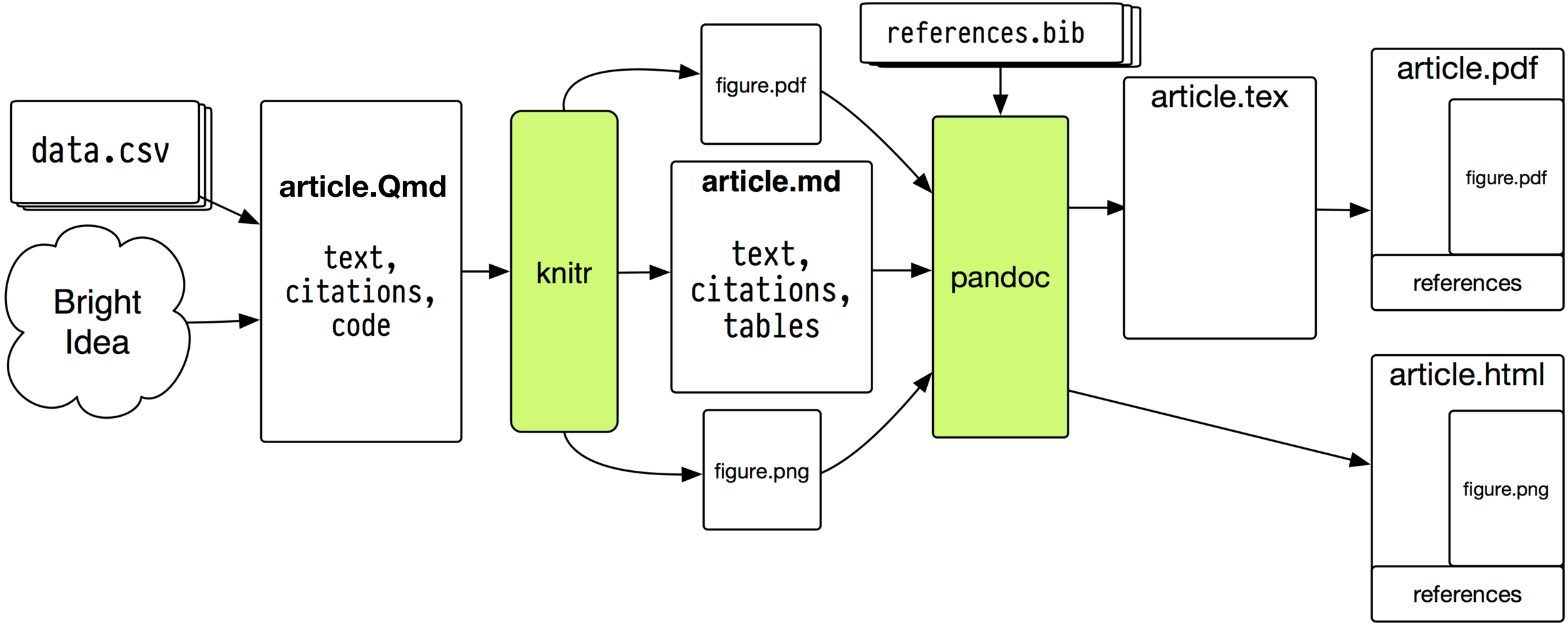


Figure credit: Kieran Healy, Plain Person's Guide to Plain Text Social Science

So far

goal		tools
exact code	✓	R; RStudio
exact result	✓	R results directly via quarto; no copying or typing of intermediate files & results
exactly report	✓	Quarto; programmatic export to alternate formats
keep records	*?*	
open science	✓	R/Rstudio/Quarto are all open source

What is keeping records about?

- Many things:
  - Maintaining original copies of data
    - Not overwriting with new versions (save a working.csv file if you need to)
    - Manipulate data programmatically from original source file
  - Keeping *versions* - allowing you to return, if you like, to prior work
  - Working with others; knowing who contributed what
  - Keeping backups and archives of your own work

What is keeping records about?

- Many things:

Good coding practice

  - Maintaining original copies of data
    - Not overwriting with new versions (save a working.csv file if you need to)
    - Manipulate data programmatically from original source file
  - Keeping *versions* - allowing you to return, if you like, to prior work
  - Working with others; knowing who contributed what
  - Keeping backups and archives of your own work

Stay tuned

What is keeping records about?

- Many things:
  - Maintaining original copies of data
    - Not overwriting with new versions (save a working.csv file if you need to)
    - Manipulate data programmatically from original source file
  - Keeping *versions* - allowing you to return, if you like, to prior work
  - Working with others; knowing who contributed what
  - Keeping backups and archives of your own work

Need a new tool

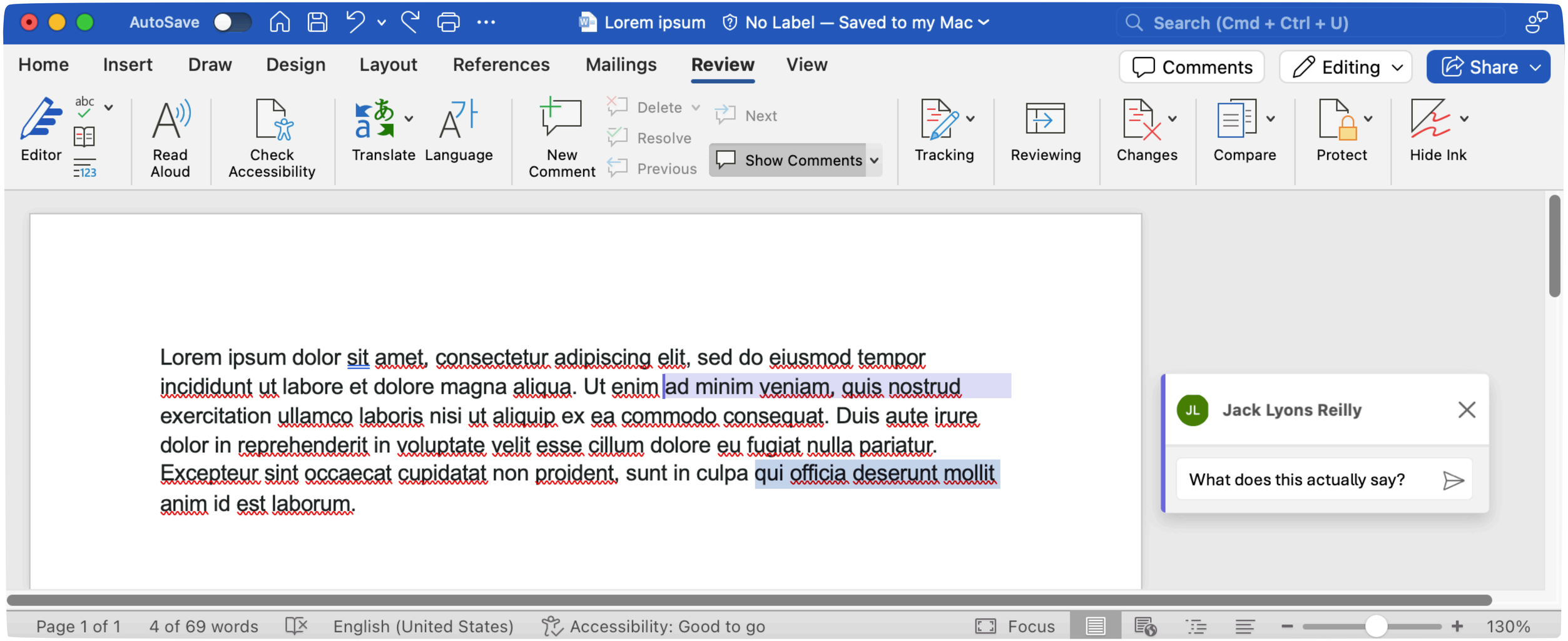
The first step in any reasonable file organization strategy

- **File organization**
  - Project = folder, folder = project
    - All files for a project go in the folder. Data management files, data analysis files, data visualization files, quarto written files, bibliography files, (often but not always) data files
    - **All files go in the folder.** (You can have subfolders)
  - Folders should have readme files at the top level. This is to tell someone else what the project is and how to use it.
- If you break all other sorts of rules, this one will often still save you.

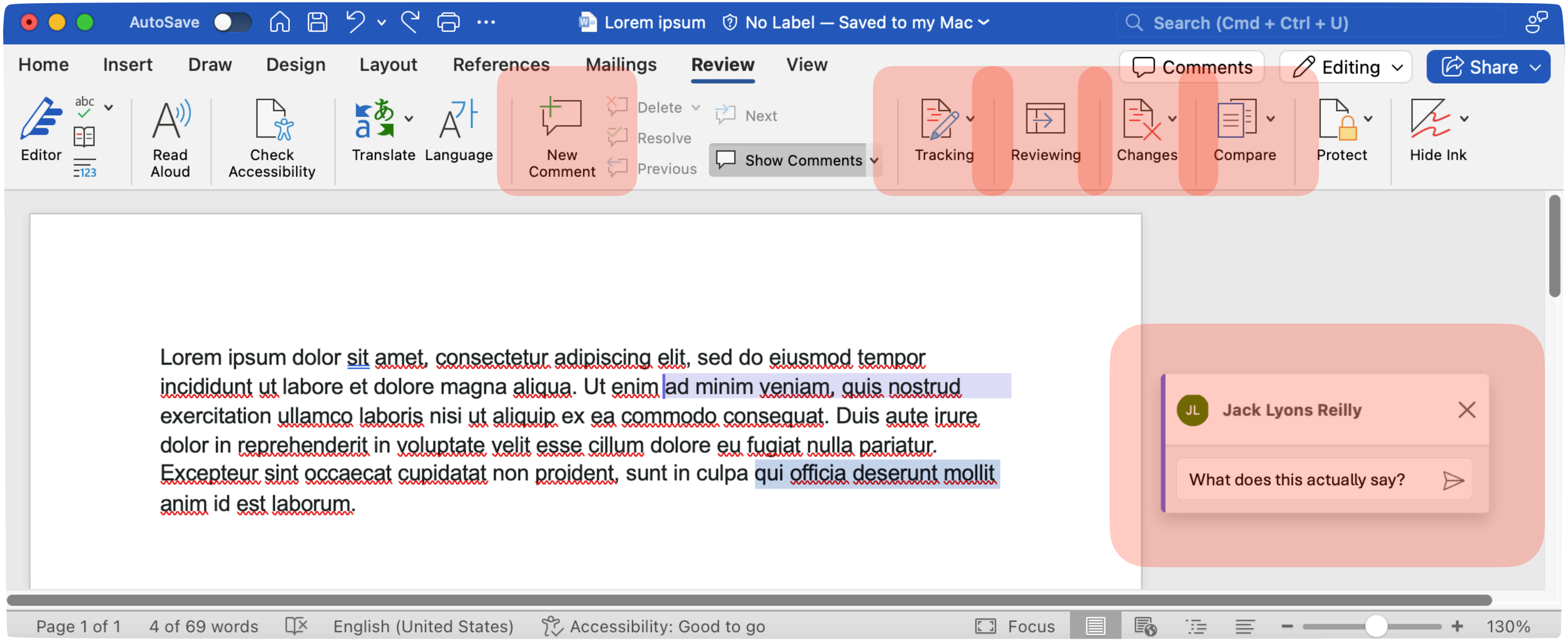
Now we have our project folder. Two main questions remain

- How do we **keep track of file versions** in our project folder?
- If a collaboration, how do we keep track of **who contributed**/wrote/edited what in the project?

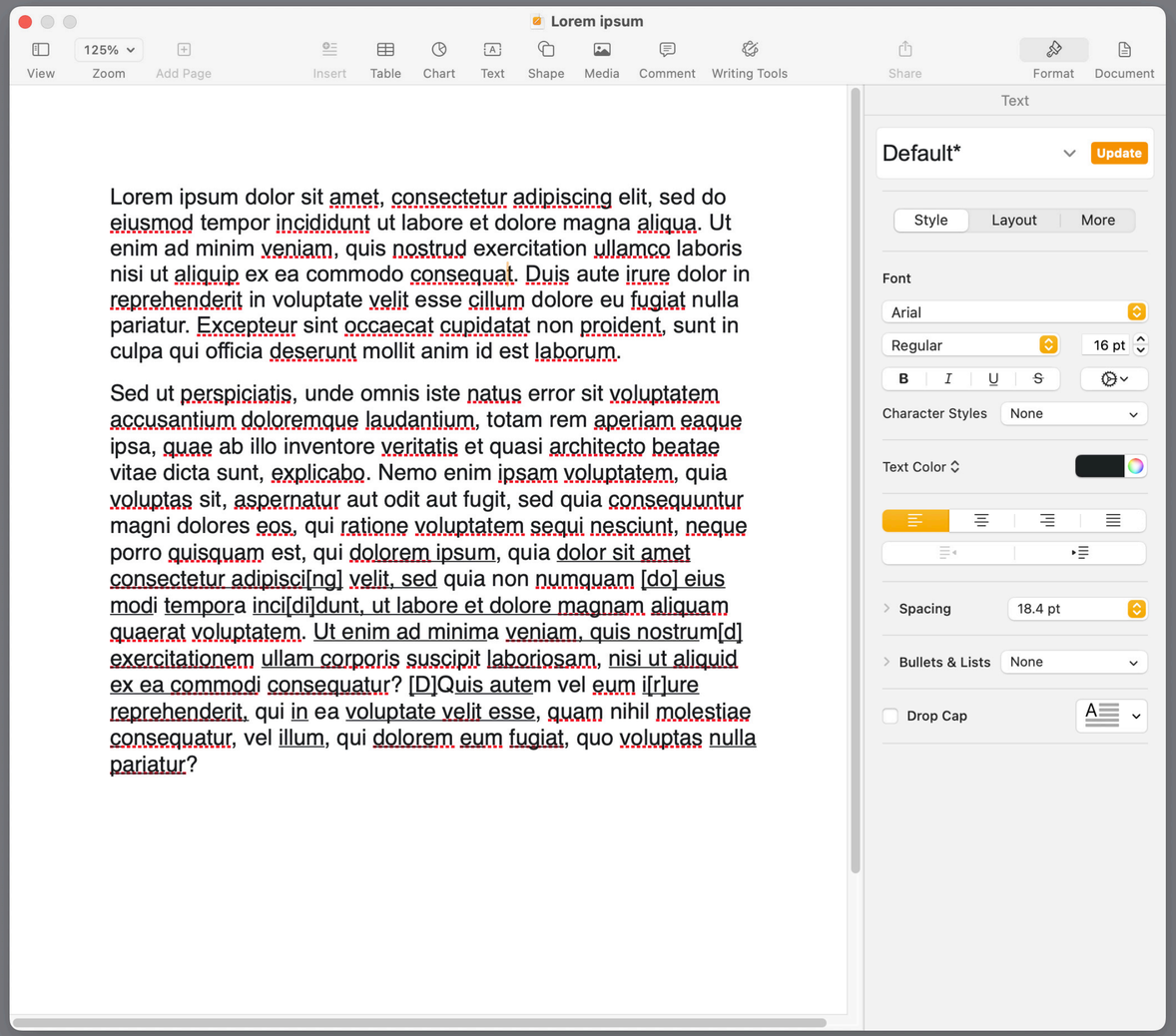
What does this versioning look like in the **office** model?

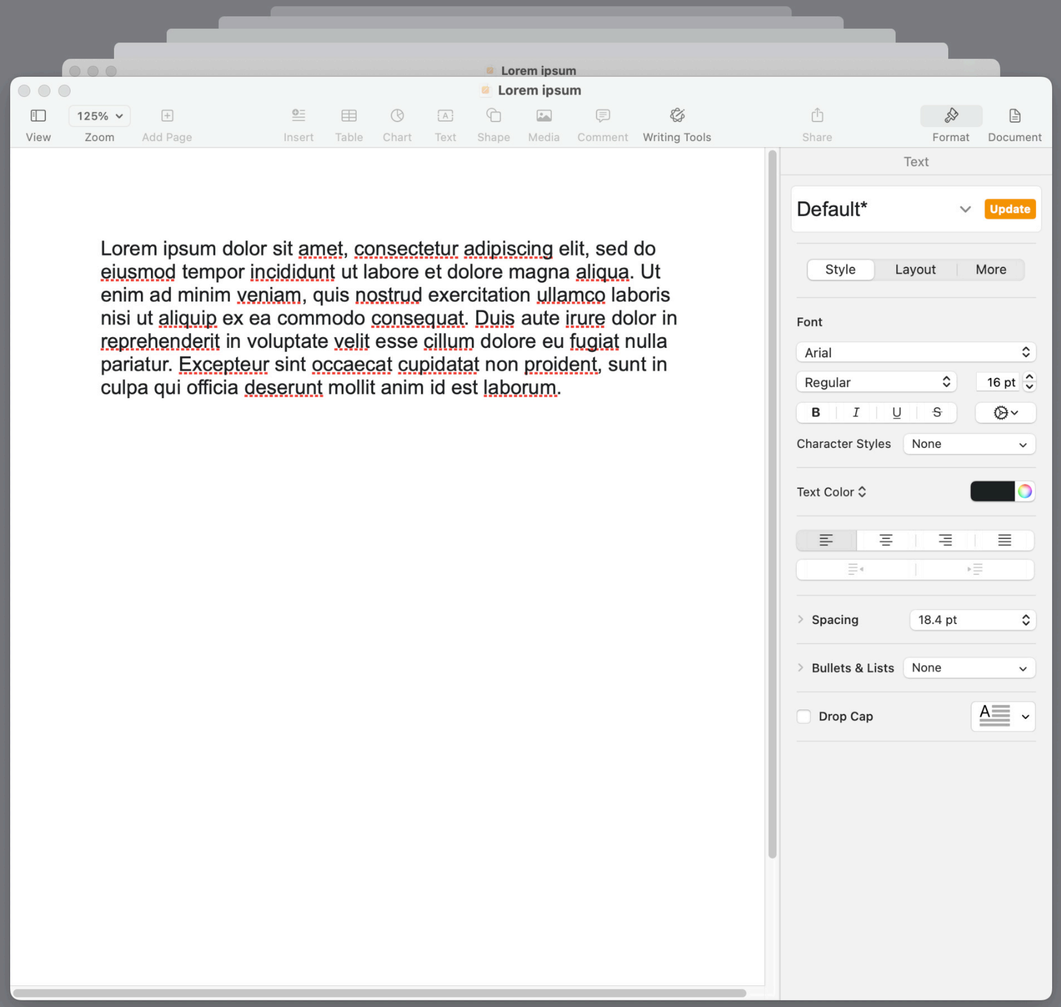


You have a lot of tools here!



Other applications implement similar features in different ways *(Apple Pages)*

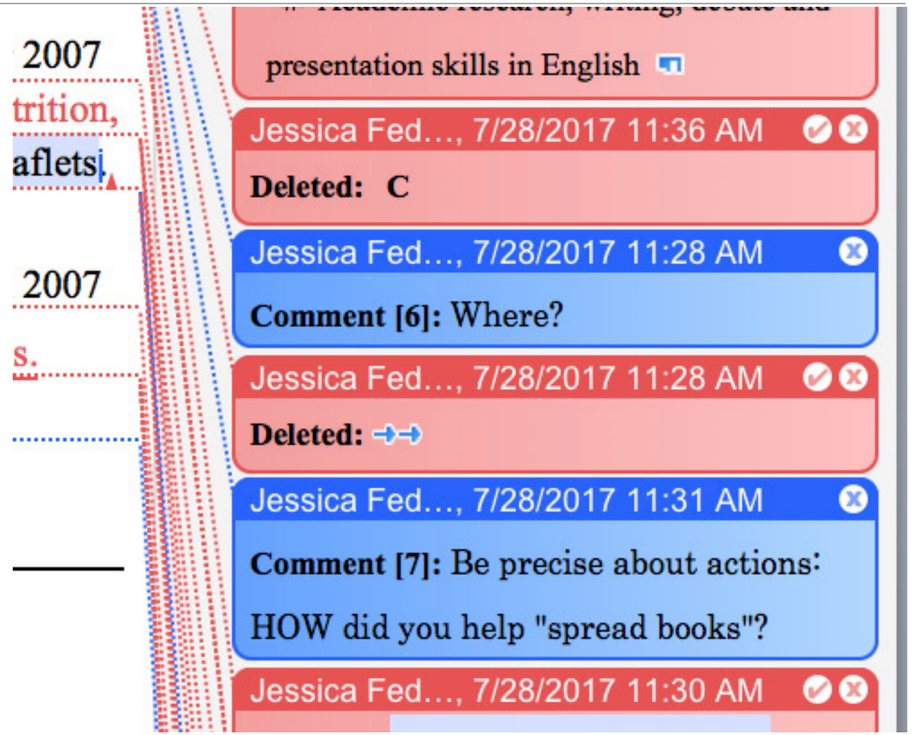




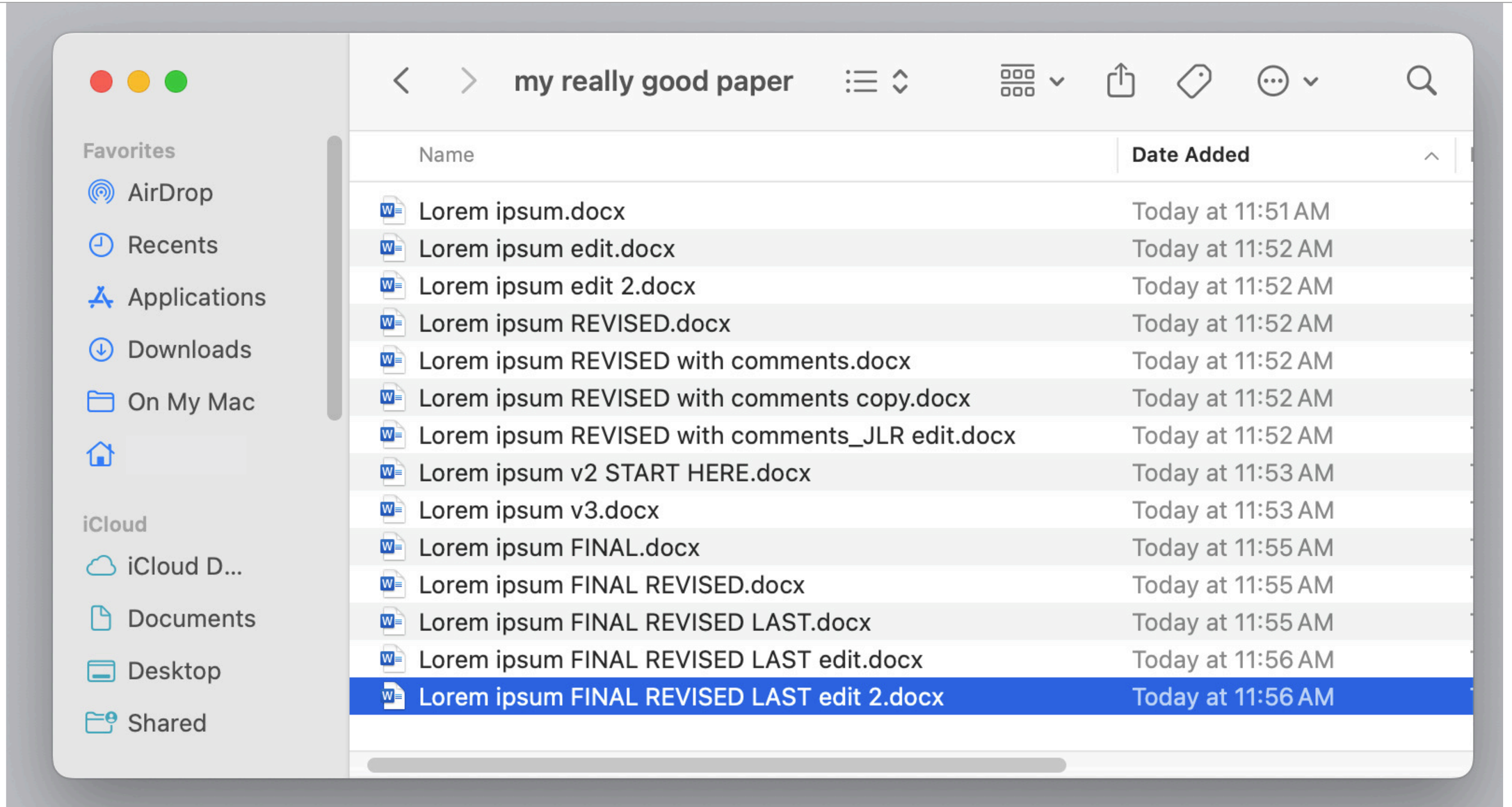
File>>Browse All Versions

A few notes

- Usage of these tools varies significantly.
  - Less so among students
  - More among professionals
- Many don’t even know they exist
- They can also end up pretty quickly, as the stuff of a UI designer’s nightmare

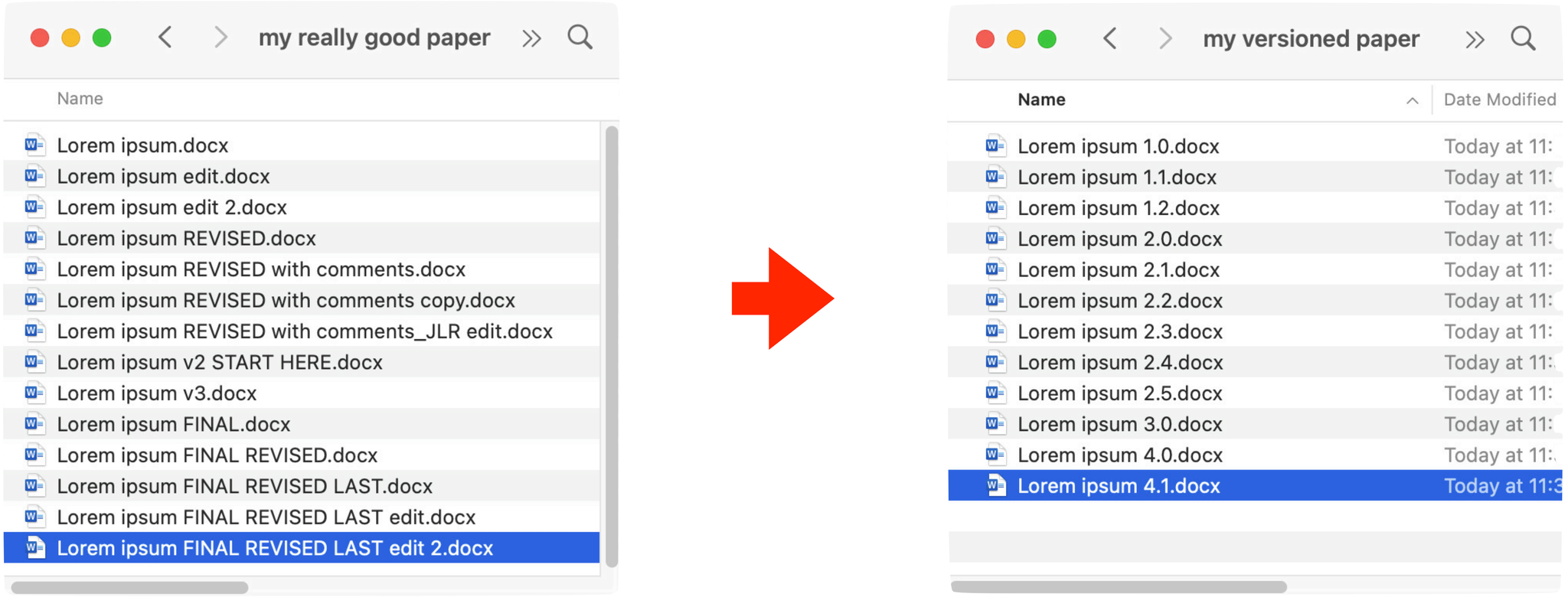


The most common form of version tracking is also the most terrible

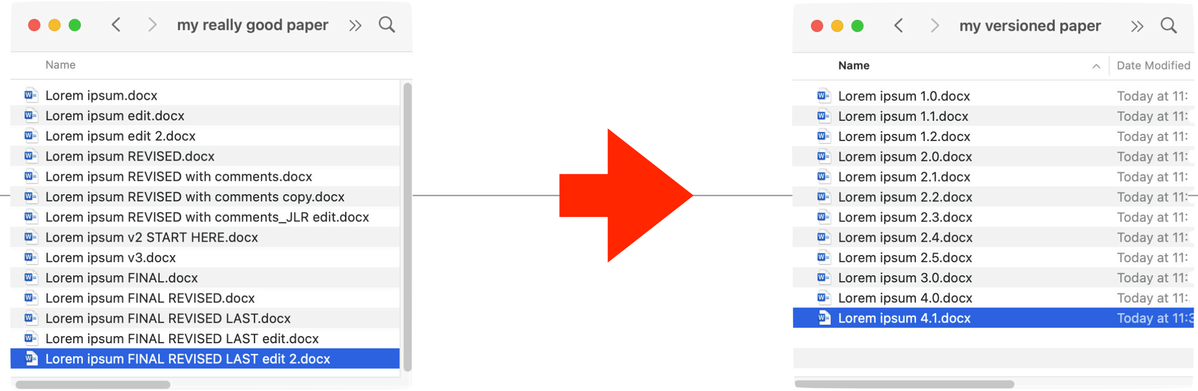


Version Tracking 101

- If you don’t want to do anything fancy, at least version number in your filesystem reasonably



Is this that much better?



- Actually, yes!
  - Let’s be clear: we can still do better
- But:
  - You know what the current file is
  - In plain text document formats, you can leave comments to yourself in the header about what changed in each version
- It’s got some stuff to recommend it. Simple, clear

Better version tracking. What do we want?

- Desiderata
  - Always makes most recent version clear
  - Keeps old variants for easy access
  - Allows for *branching*
  - Allows for tracking of contributions by different contributors
  - Tracks at the *project* level, not within particular files
  - Open

Version tracking systems

desiderata	track changes/ office model
most recent version clear	
old variants for easy access	
allows for *branching*	
tracking contributors	
tracks at the *project* level	
open	

Version tracking systems

desiderata	track changes/ office model	filesystem model (bad)
most recent version clear	✓	✗
old variants for easy access	✓	✓
allows for *branching*	✗	✗
tracking contributors	✓	✗
tracks at the *project* level	✗	✓
open	✗	✓

Version tracking systems

desiderata	track changes/ office model	filesystem model (bad)	filesystem model (better)
most recent version clear	✓	✗	✓
old variants for easy access	✓	✓	✓
allows for *branching*	✗	✗	✗
tracking contributors	✓	✗	✗
tracks at the *project* level	✗	✓	✓
open	✗	✓	✓

Introducing Github (with Git)

- So what are we to do?
  - Git & Github (two separate things, actually!)
- **A word of warning**
  - This is definitely the nerdiest, techy-est thing we're introducing on a computer level
  - It requires patience

Is it worth it?

- If you use it judiciously and correctly, with an eye toward your interests: yes
    - This kind of version control is **mandatory** for programmers and professional data scientists
    - For applied researchers, it certainly has its virtues
        - But you can also get yourself into corners
- Do you work with many collaborators? On large, unstructured data sets? With lots of code, analysis, and reporting files in plain text? That are prone to change and require management over time?
    - The more you said “yes” to the above, the more this is worth it for you. If you only said “yes” once (or not at all) it may be less.

Some userbase guesses*

tool	wild guesstimate*	notes
scripting statistical software	75%	*Notable age and field of study curve*
open statistical software	50%	*Notable age and field of study curve*
open data	50%	*More journals are requiring, so more is happening; definitions matter. Some data cannot be shared; some is inherent open but with limits (Census), etc*
plain text writing/ automated reporting	30%	*LaTeX heavily dominant; Quarto less so. Economics & QSS sorts more likely. Definitions also matter: using LaTeX doesn't always mean in an automated way*
project level version control	10%	*Git is dominant but not the only way; see Subversion*

**I tried to find a better source for this and failed, so this is all anecdata; ie, wild guesses*

So why are we covering it?

- A couple reasons
  - It is the final piece to the plain text puzzle. You can (and will) pick and choose what you use after this class, but you should know all the tools
  - There are lots of little bonuses to knowing Git/Github
    - Even if you don’t use it for *everything;* being able to use it for *some things* can be very nice
      - Collaboration, public file access for reproducibility, website hosting, etc

What is GitHub, and what is GitHub not?

Git(Hub) is for versioning code and plain text files!	True!	*This is its primary purpose!*
GitHub is for sharing data!	Not really	*While it can be used for data, it's not well suited for it. git is focused on plain-text code documents*
Git is primarily for software developers	False!	*It has distinct benefits for applied researchers because it tracks whole **projects** (folders) not just an individual file*
Version control is only useful for collaborative projects	False!	*Solo work can also benefit from project tracking and versioning*
Git is open!	True!	*It is!*
GitHub is open!	False!	*GitHub is owned by Microsoft. There are alternative options for using Git, however - gitlab, bitbucket, rolling your own (not recommended)*

Terminology

- A **version control system**: software that tracks changes in file content for a whole folder

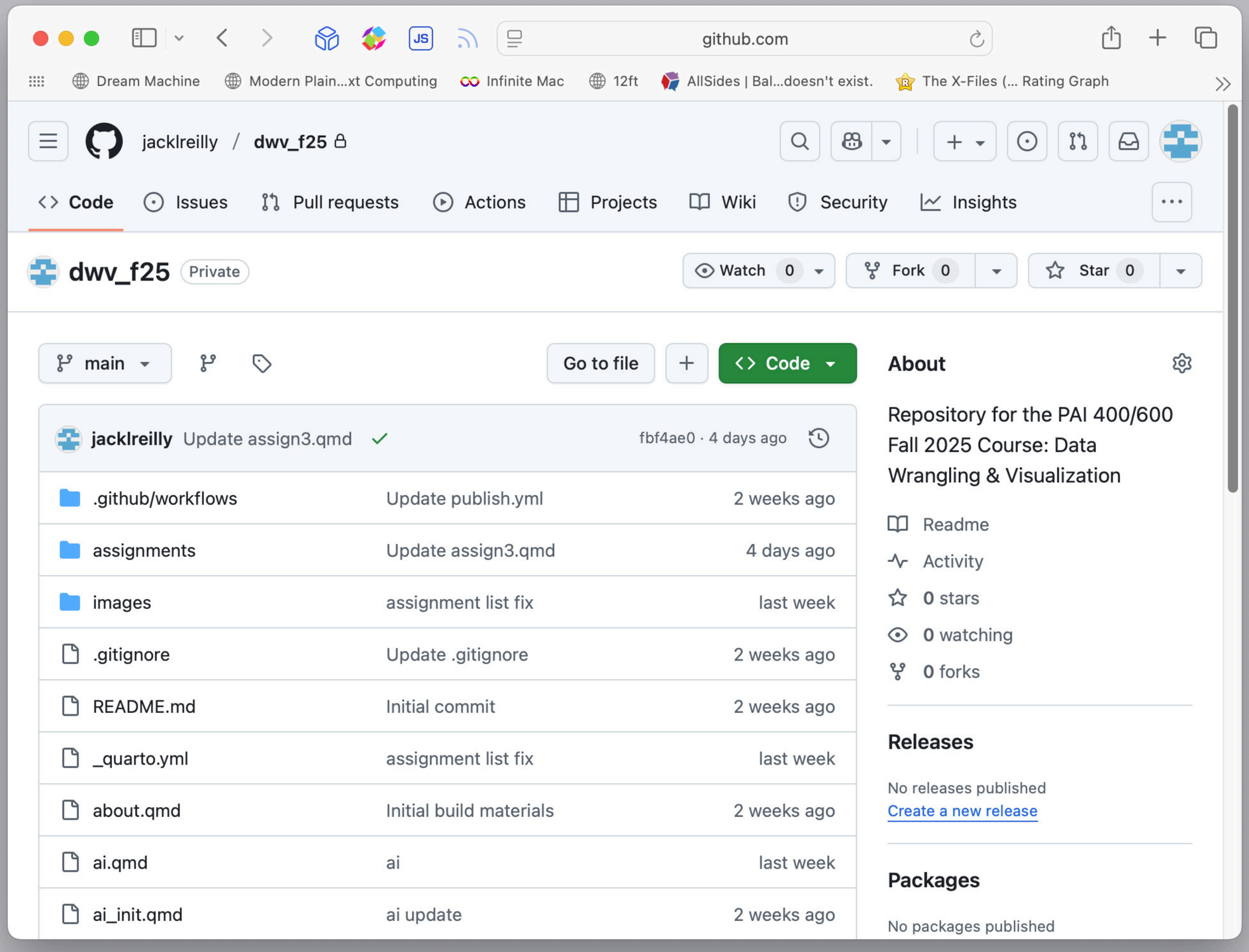
- **Git** - a particular type of version control software you can use on your local computer

- **GitHub** - a service (free for most students and academics) run by Microsoft, which hosts your GitHub repositories online, publicly or privately

- **Repository** - a folder that you have entered into a version control system. It's just a regular folder with a hidden subfolder (.git) inside it that tracks and stores all changes

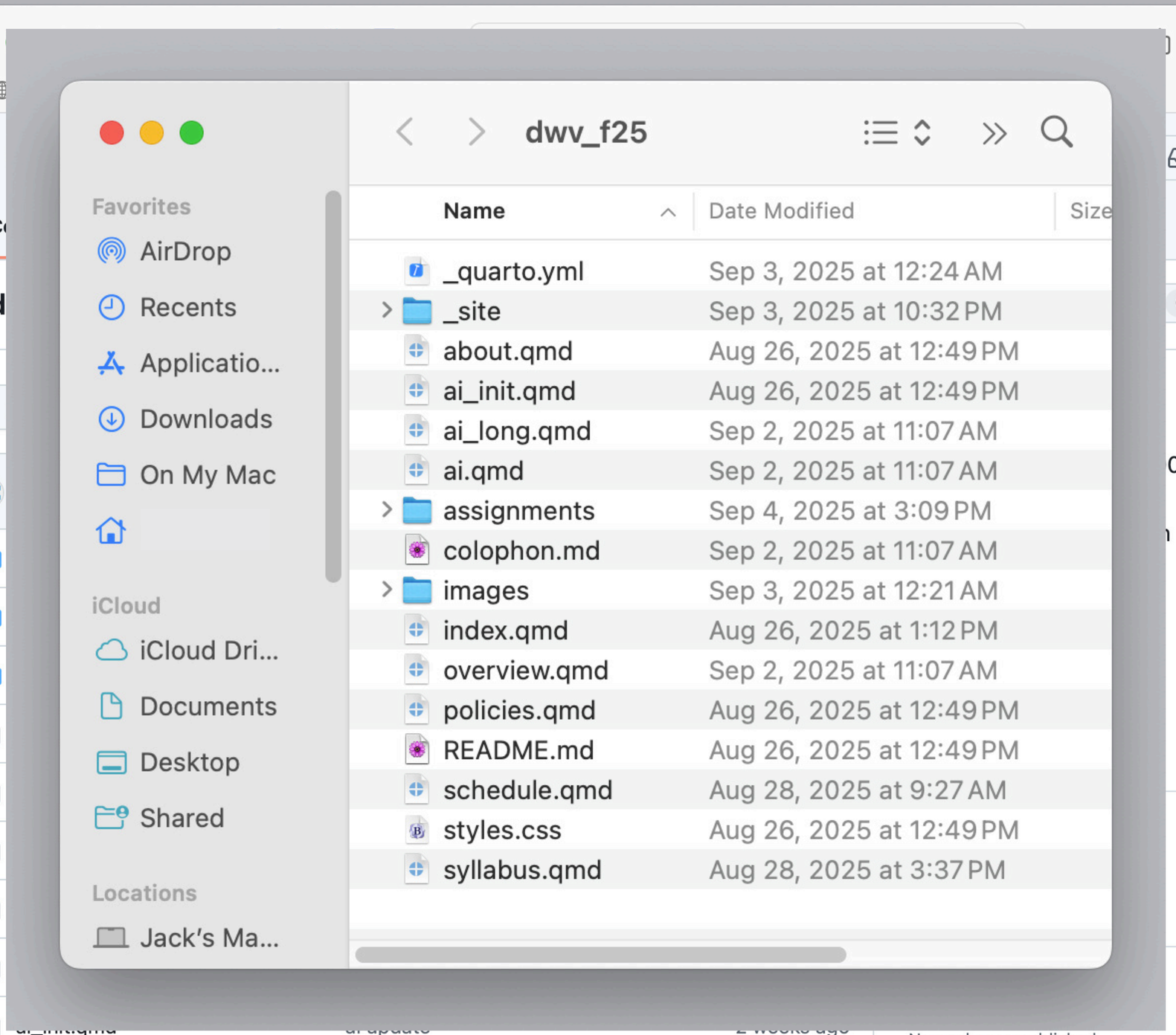
Show me

- The (private) GitHub repo for the course



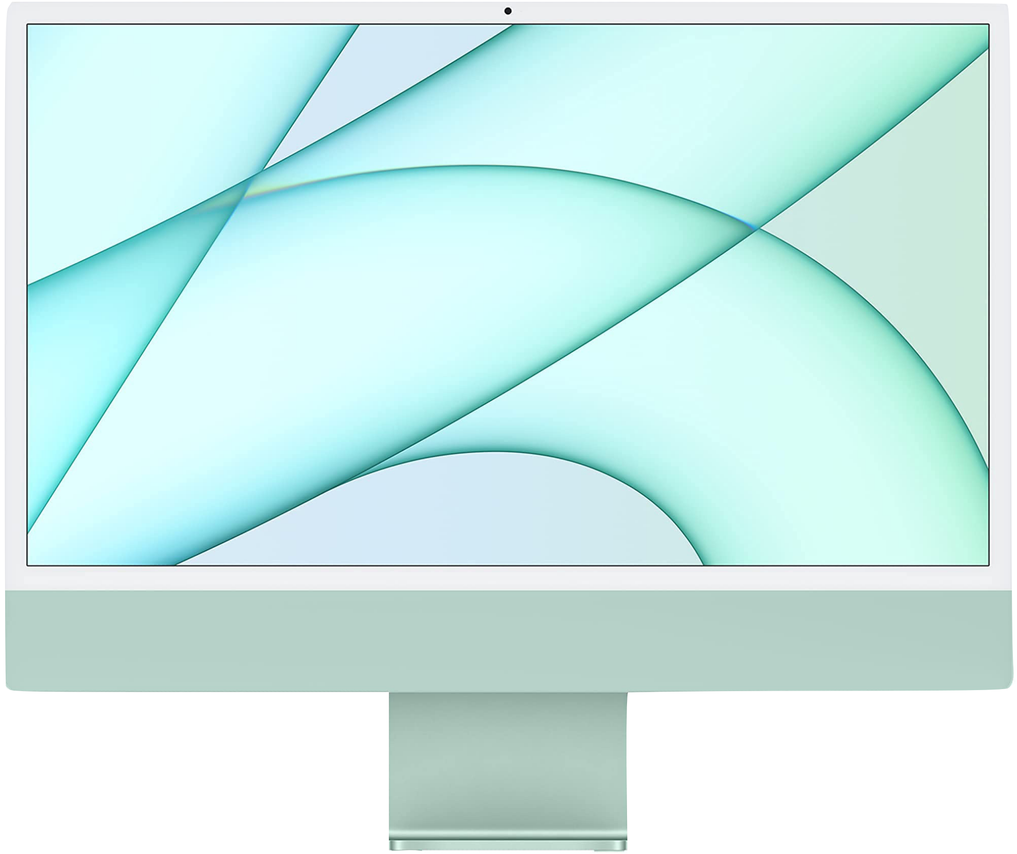
Show me

- The (private) GitHub repo for the course
  - It mirrors (exactly) a folder on my computer
  - It also serves the website we use



A fundamental file syncing problem: **what is truth**?

- Let’s imagine you have a common problem: email and files on your phone, tablet, and computer



A fundamental file syncing problem: **what is truth**?

- Now let's imagine you're walking around all day with your phone, while your tablet and computer sit home

  - You edit a file on your tablet away from a WiFi network
  - Unbeknownst to you, your six year old goes on your desktop and edits the same file. "Hi Daddy!"
  - Your phone sits unchanged.
- You return home with your tablet and phone.
  - What file is **the truth**? How does a sync service decide?

A fundamental file syncing problem: **what is truth**?

- In GitHub, “truth” is not what is on your local machine
  - **Truth exists** in the centralized GitHub repository online
- When you want to work on your project, you “check it out” (or **pull**) from the online repository (ie, download it)
  - Then you do stuff to it
  - Then you “check it back in” to the local repository, with a note explaining what you’ve done (**commit**), and then send it back to the server (**push** it), where it lives until you are ready to work with it again
- *Note: you’ll still have the local version of the files on your computer! But always pull changes before starting to work, or you’ll end up with conflicting copies* = **merge conflict**; one of the scariest phrases in the git lexicon

Workflow

Cloud

Local

pull

push

dwv_f25

add

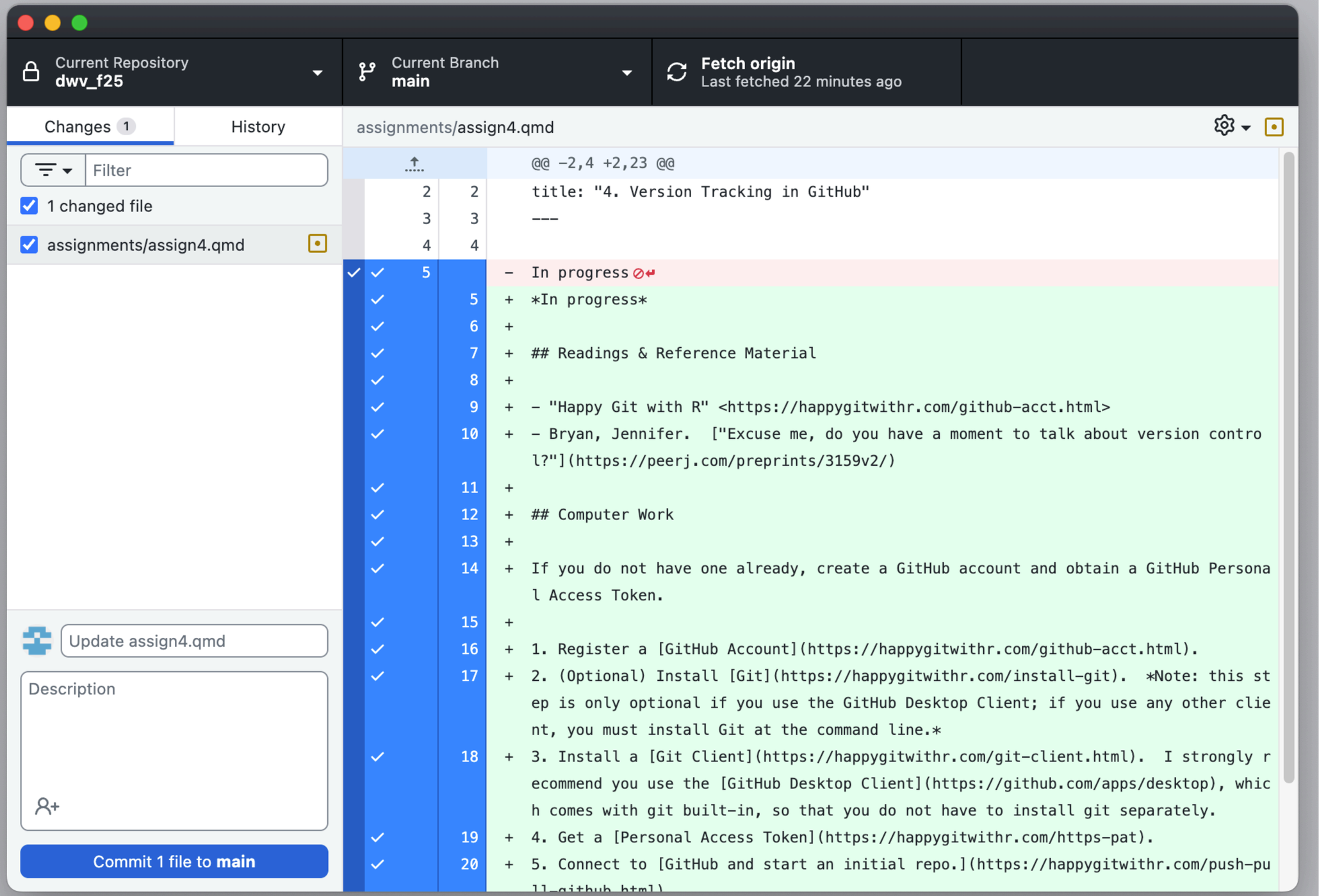
commit

dwv_f25

How do you check in and out?

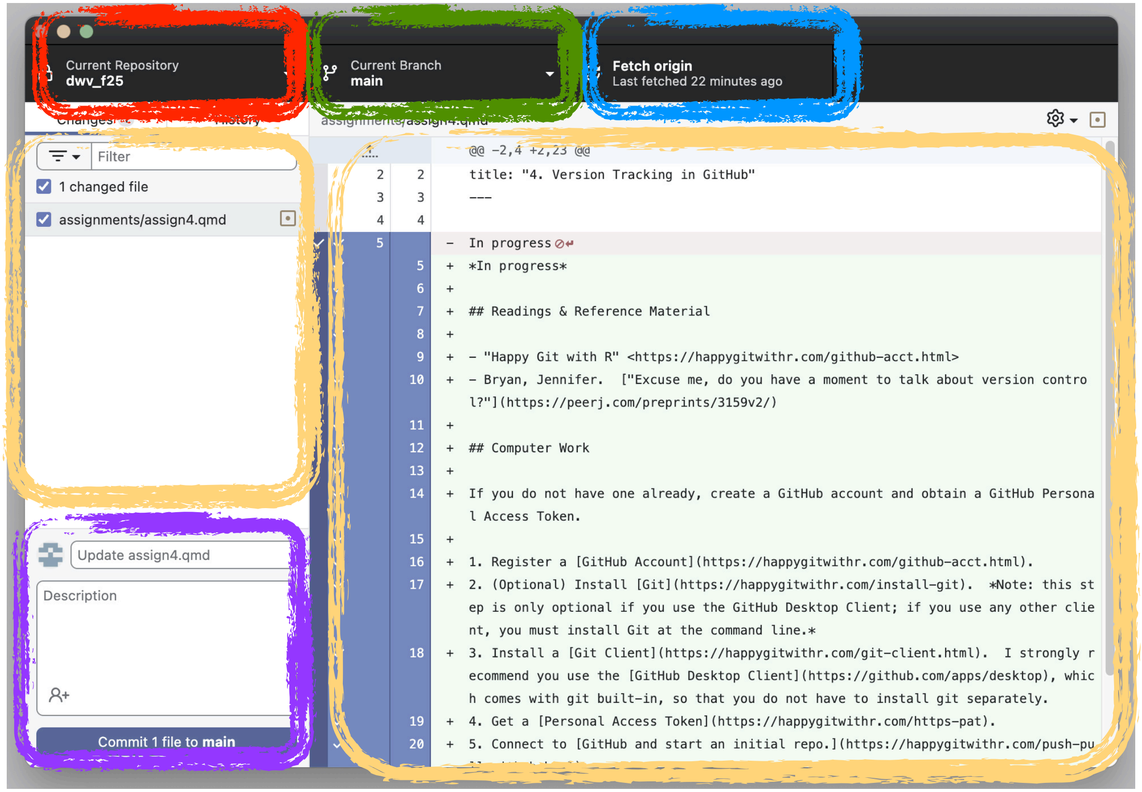
The application keeps track of all file changes.

When you're ready, you *commit* your changes to version tracking and then *push* back to the server



And what is all of this business?

- A **repository switcher**, for using multiple repositories
- Controlling **branches** - different variants of the same project
- **Committing changes** to the local repo
  - Including change messages
- **Fetching** changes, **pulling**, and **pushing**
- Text and file change **viewers**



What have we gotten ourselves into here?

- The devil is in the details
- Workshop this week: actually installing and using GitHub
  - In the meantime, start your installs. (See instructions in Week 4 assignment)
    1. **Sign up for GitHub**
    2. **Install the GitHub Desktop Client**
    3. **Link the two**
- We'll take it from there in the workshop, with an expected level of confusion and standard computer error messages

Version tracking systems

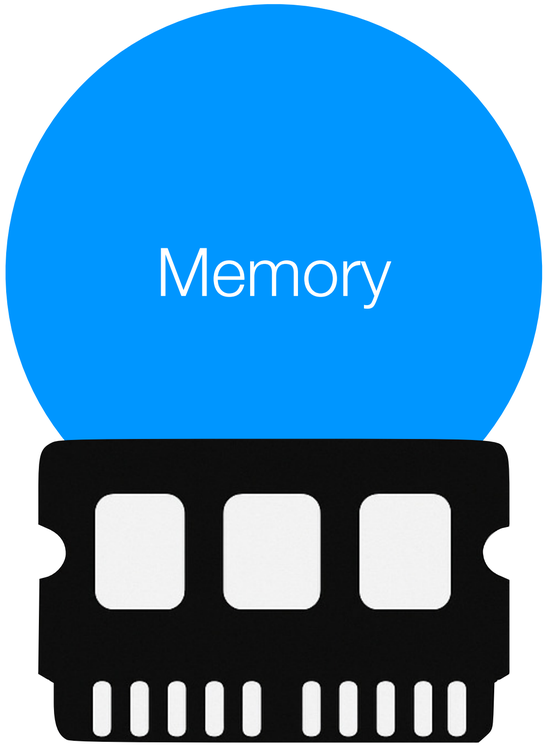
desiderata	application/ office model	filesystem model (bad)	filesystem model (better)	Git/github
most recent version clear	✓	✗	✓	✓
old variants for easy access	✓	✓	✓	✓
allows for *branching*	✗	✗	✗	✓
tracking contributors	✓	✗	✗	✓
tracks at the *project* level	✗	✓	✓	✓
open	✗	✓	✓	✓

A few final words on Git/Hub

- Git (and GitHub) is a rabbit hole. It goes down deep.
    - You can go down deep too! Resolving merge conflicts, branching, rebasing, pull requests, command line options . . . etc. It goes on, especially when you start actually working with others in git repos
    - Very powerful and complicated. Large engineering companies manage their entire codebase in git.
        - The entire source code for Microsoft Windows is in a 3.5 million file git repo
- But we're not trying to be complicated. The goal we have is to use Git in the *simplest possible way,* building a base for potential complexity later
    - Single user, pull, add/commit, push

File Storage

Data in three places



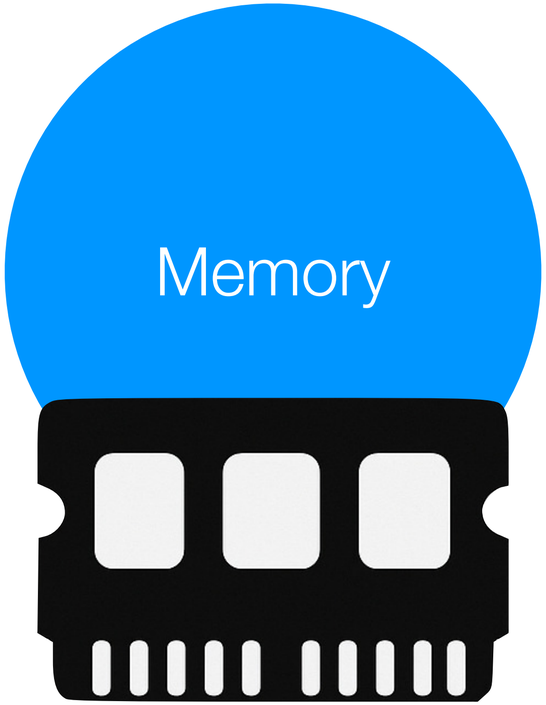
Memory

- Computers, remember, are fundamentally about manipulating items in memory according to processes followed by the CPU

- Memory (RAM) is *temporary storage* for immediate manipulation by the CPU

    - There are multiple *levels* to memory - RAM, cache, registers

- When you read data into R, you are very literally *moving* from somewhere else into working memory



Local Storage

- Where things are kept, long term, in your computer
  - Old style: hard drive (HD)
  - New style: solid state disk (SSD)
  - Removable discs/disks (CDs, tapes, floppies, etc)
- When you turn your computer off, both SSDs and HDs keep their contents (unlike memory, which requires power)
- Local storage organizes your data in a **filesystem**

Filesystems

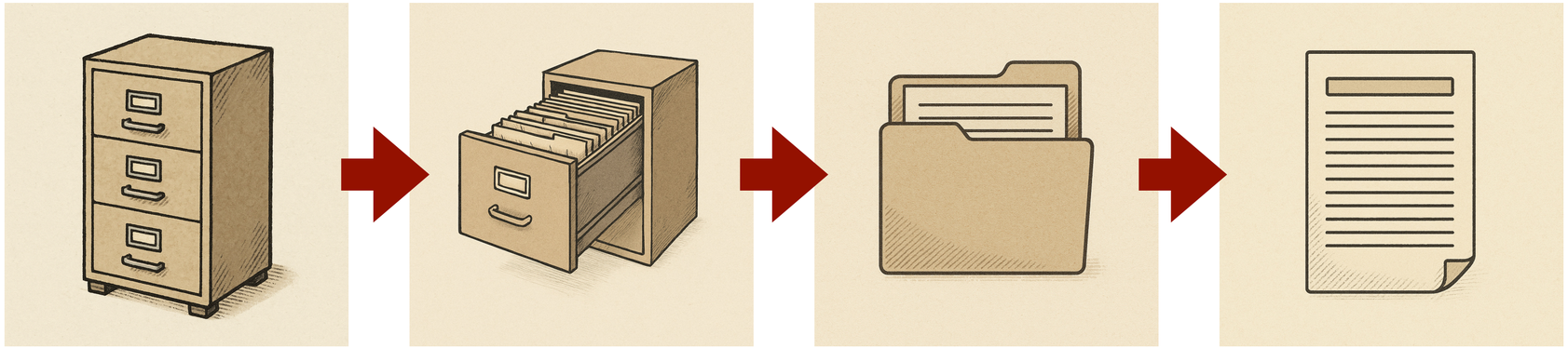
- Provides an organizational system for the location of the files
  - When you access a file, what happens?
    - The operating system looks up where your file is stored in a *directory tree*.
    - The directory tree points to where the actual data is on the drive.

Directory Tree

Storage Platters

Filepaths

- Tell you where in the tree to go. We often represent trees like the right.
  - PAI600 is in Syracuse is in Courses is in Work is in Documents is in Home is in Users is in HD
  - Or: `~/documents/work/courses/syracuse/pai600`



```
├── Applications
├── Desktop
├── Documents
|   ├── datasets
|   ├── github
|   ├── home
|   ├── work
|   |   ├── courses
|   |   |   ├── syracuse
|   |   |   |   ├── pai600
|   |   ├── research
|   |   ├── service
|   ├── logistics
|   ├── miscellanea
├── Downloads
├── Dropbox
├── Local
├── Movies
├── Music
├── Library
├── Pictures
```

Local storage is **flaky** and **unreliable**

- It has no built in **resiliency**
- How valuable, would you say, the data on your computer is to you?
  - Homework?
  - Projects?
  - Work?
  - Pictures of your family?
- If you care about any of this, you need to **back up your data**

How to back up your data

copies of your data

different media

copy off-site

Why three?

- You should always assume something should fail.

“Three is two, two is one, one is none”

- Common failures:
  - Any drive can fail mechanically
  - Something could happen in any one particular physical location
  - **You** could mistakenly do something to your data and need to recover

Things that are not sufficient backups by themselves

- **Dropbox, OneDrive, Google Drive, iCloud**
  - Protects your computer, but deletes files as soon as you do
- **GitHub**
  - Tracks versions, but not large file storage
- **Emailing yourself a file**
  - We’ve all done it in a pinch, but . . . just no

Good 3-2-1 systems

- Typically have a local versioned backup drive
  - In case you delete a file
- Some kind of online backup
  - In case something happens to your physical location
  - Preferably not Dropbox/OneDrive/etc,
    - *In truth, this is a sliding scale. Dropbox online sync is better than no online sync*

Network (remote) storage

- Files that exist elsewhere, either
  - Directly addressable (like a web link)
  - Stored in some more complicated database structure
- Sometimes accessible via API
- You can draw a distinction between *network storage* (physically at your location, but over the local network) and internet storage (anywhere in the world)
- We'll talk more about these later

Next week

- Data formats, importing, exporting, and basic management

Workshop this week

- Github
- More R, Quarto, etc