1.2: Git and Version Control

Ryan Horne UCLA OARC rmhorne@ucla.edu Twitter: @RyanMHorne

Intro

Today we will be looking at Version control using Git, Bash, and GitHub. If you have not done so already, you need to install Git and create an account on GitHub. This part of the workshop is adapted from data carpentries; if you want more details I recommend starting there.

Agenda

We will:

- Continue a little about project structure
- Talk about bash and Git
- Run Git from the command line
- Setup GitHub account
- Fork a repository

Quick Side Note

The color palette that I am using was developed by Masataka Okabe (Jikei Medical School) and Kei Ito (University of Tokyo). It is colorblind friendly and is under the recommended MAXIMUM of 8 categories. See https://siegal.bio.nyu.edu/color-palette/.

What do YOU do?

Unmute or on chat, tell us!

- How do you organize your projects now?
  - Yes, your writing is a project!

- How do you collaborate with others?
  - What programs do you use, if any?
  - Do you have shared drives? Documents?
- How much thought do you put into the organization and mechanics of your digital projects?

So, You Want to Make a DH Project

- Does your file structure and naming make sense?
- Is your raw data separate from what you are working on?

Some Basic Organizational Points

- Keep raw data in a separate directory from your working data
- **DO NOT TOUCH IT**
  - Copy it, *then* open it
    - Programs *can* modify your data just by opening it!
  - If in doubt, look above


Modify
Raw Data
Copy the
Data and
Modify the Copy
WHICH ONE??
JAKE-CLARK.TUMBLR
imgflip.com

So, You Want to Make a DH Project Part 2

- Does your file structure and naming make sense?
- Is your raw data separate from what you are working on?
- Is your README up to speed?
- Wait a second…..

Why a README?

- First rule of DH: Document, document, document
- Second rule of DH: Document, document, document
- Seriously…
- Document, document, document
- A README file can save **you** countless hours, let alone any collaborators

What is a README?

- You should ***ALWAYS*** have a README.md file. ***ALWAYS!!!!***
  - Think of this as a love letter to yourself. Do you remember everything about your data? Columns? Abbreviations? What you had for breakfast?
  - Where did you get your data form? What were the data formats? What did you do to it? Etc.
- Some great templates out there (I linked to UCSB Research Data Services one)
- There might be field specific things that you need
  - If so, best practice to make some blank README files as templates, then copy over and modify as needed

How to do a README?

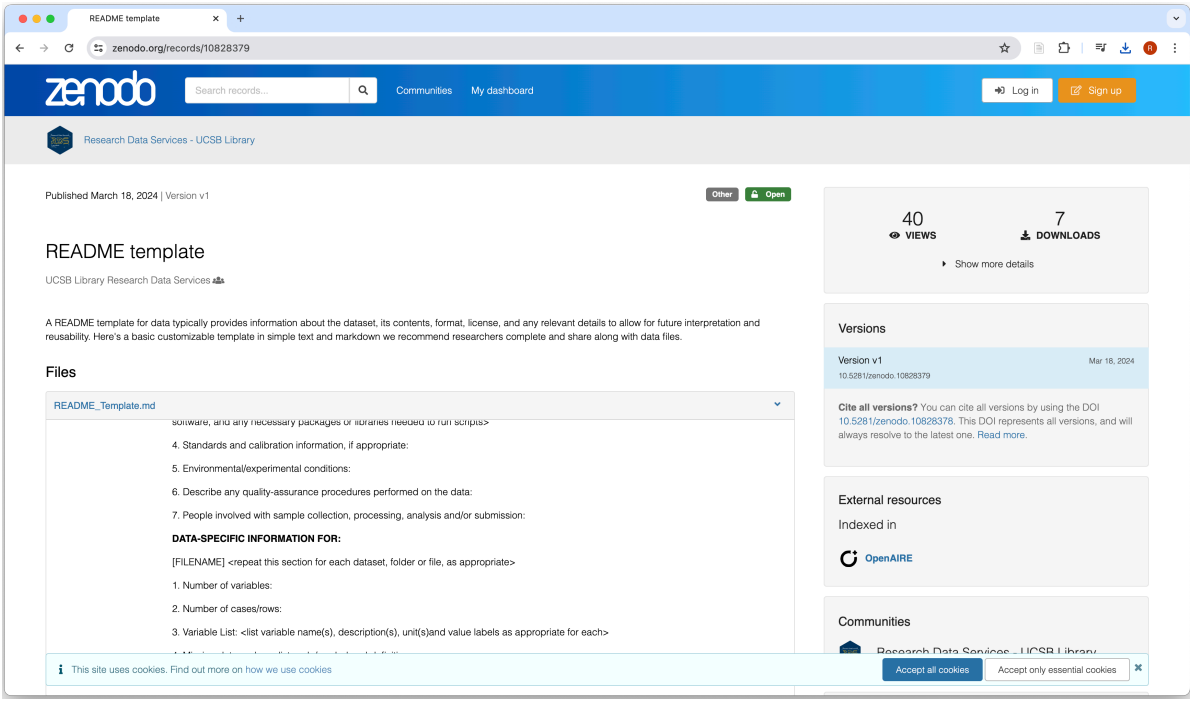


Figure 1: https://doi.org/10.5281/zenodo.10828379

Time to do DH!

Now we are ready to do some serious Digital Humanities!

However…..

What We (Probably) Want to Avoid

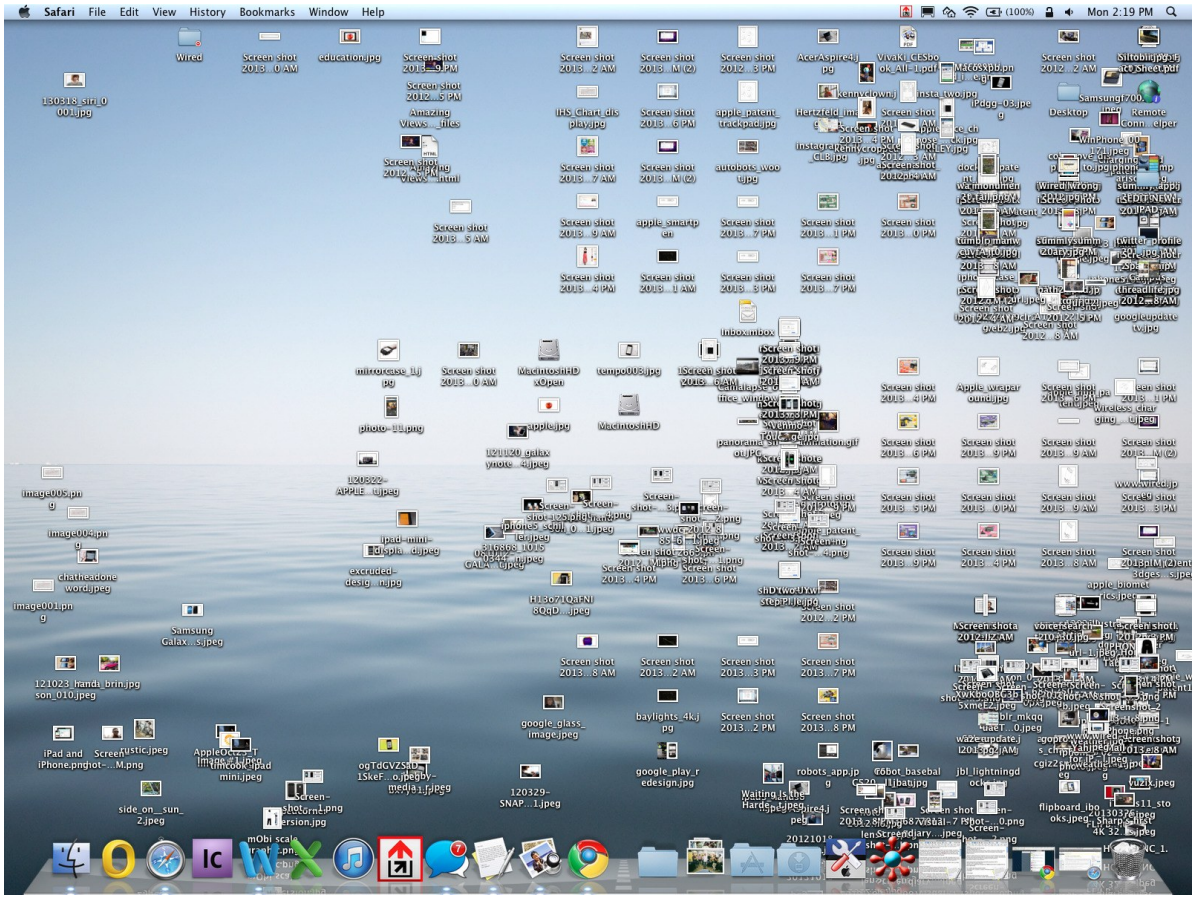
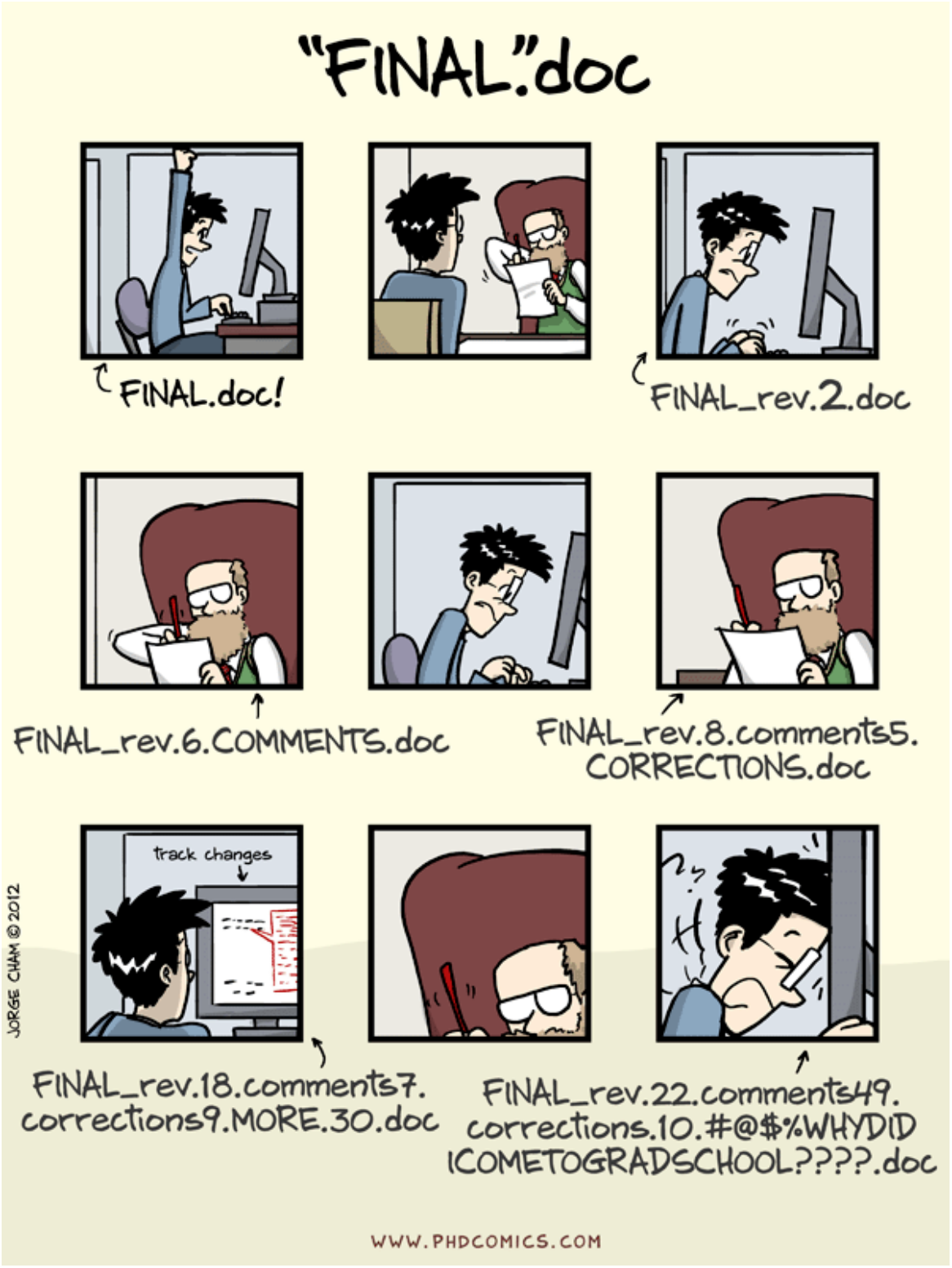


Figure 2: From MIT Broad Research Communication Lab

One Problem


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Another Problem

images/homer.mp4

Yet Another Problem

What is the Worst that Can Happen With This Mess?

https://www.youtube.com/watch?v=q0POXW4V1_k

Seriously Though - Our Issues

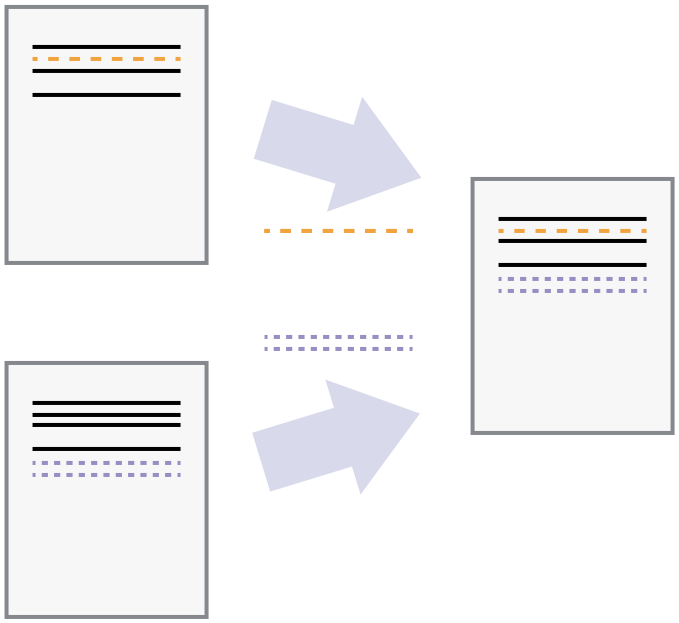
- How do you keep track of changes, modifications, etc?
  - How can you roll things back to a previous state?
  - How can you recover from a spilled coffee mug?
  - How do you recapture that wonderful paragraph that is *waaaaay* past undo?
  - How do you effectively collaborate?
  - How do you share your data and findings?
  - How can you setup free web space?

Version Control

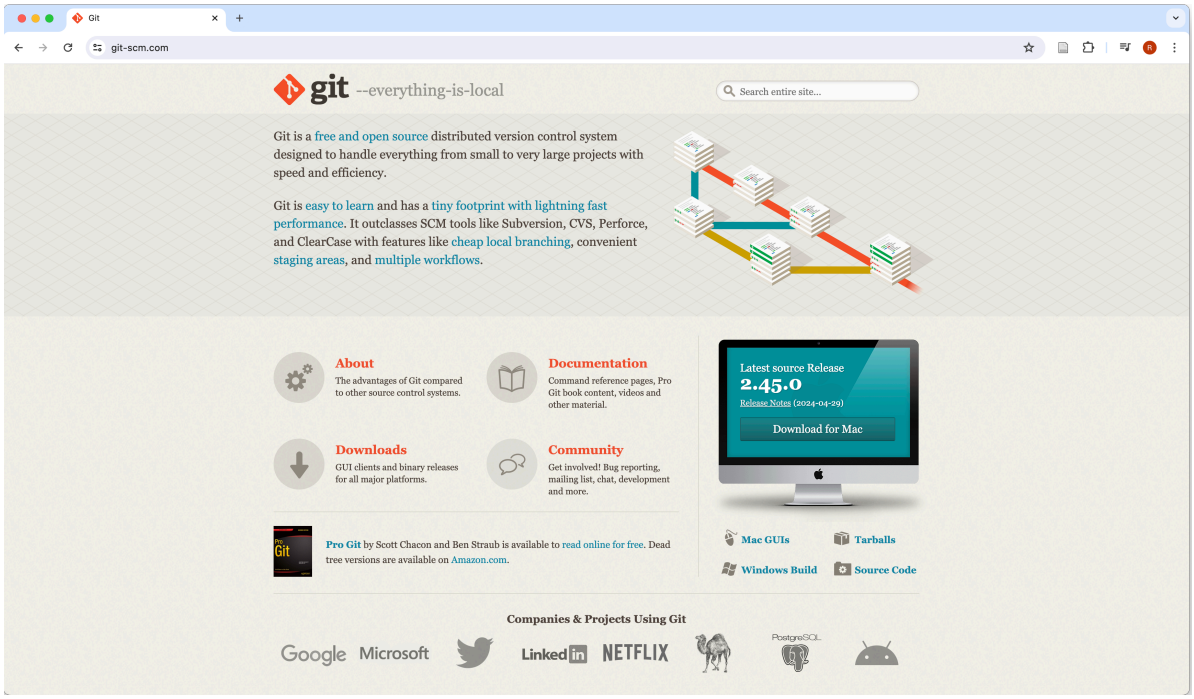
- Basic idea: track changes in a file / directory
  - Also to reconcile different versions of a file
  - There is some functionality for this built into Word and Google Docs
- Think of changes as separate from the document / file
  - This is a common theme in the workshop: ***presentation and data are different***
- Helps with project organization, reusability, reproducibility, and portability

Basic Functionality

Multiple Users



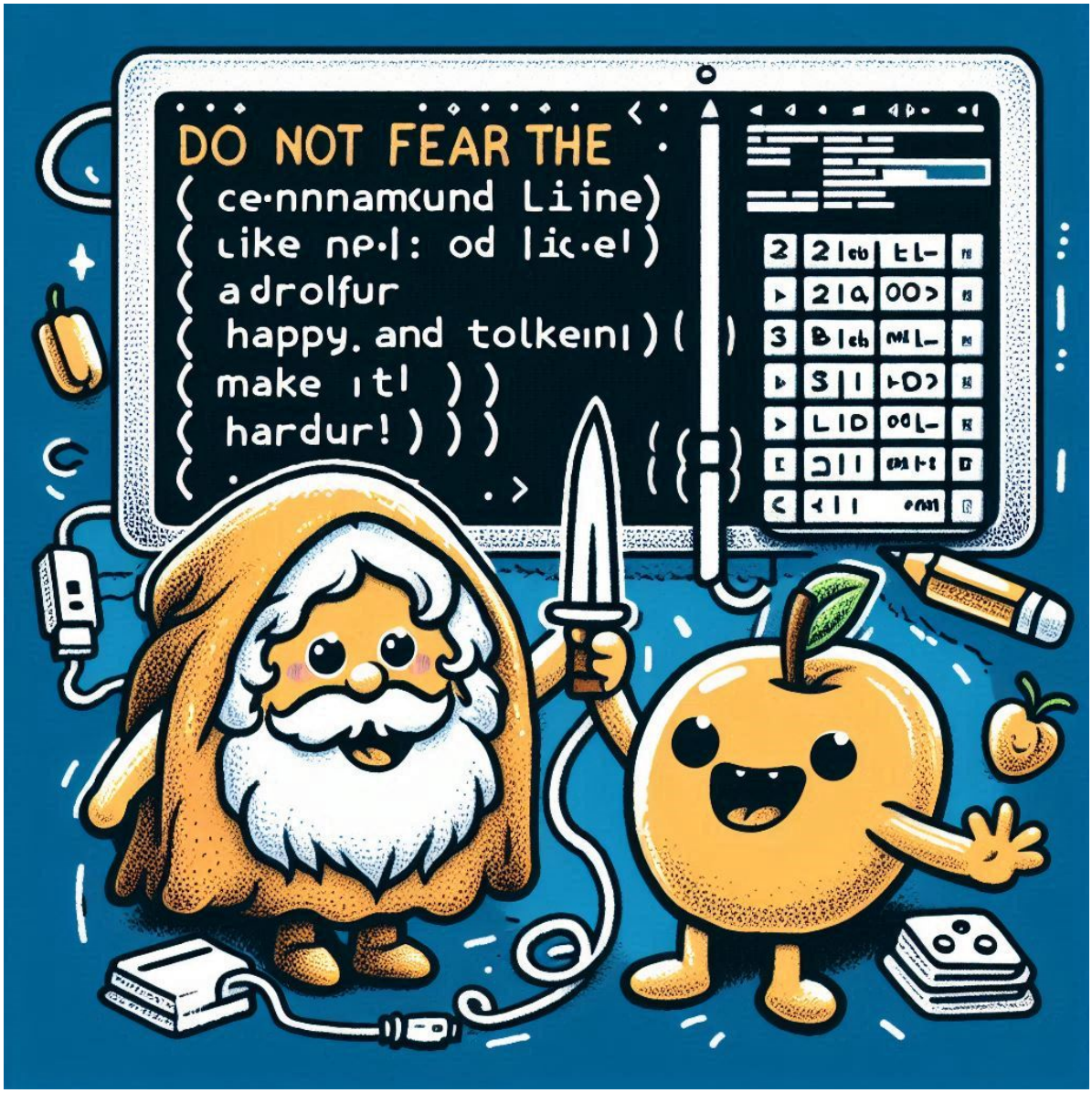
Git



Git

- First made in 2005 to track development of Linux kernel
- Free and open source distributed version control system
- ~ 95% of developers reporting it as their primary version control system as of 2022
- You can use it for all kinds of computer files
- Small and lightweight
- Command line and applications
  - Deployments in GitHub, SourceForge, Bitbucket and GitLab
- We are going to learn things the hard way, then shift to the easy way

Time For the Command Line!



Get Started: The Command Line

- Terminal for mac
- Command line for windows

- Using the same account info as your GitHub account, you configure Git by typing in the following:

```
$ git config --global user.name "Vlad Dracula"
$ git config --global user.email "vlad@tran.sylvan.ia"
```

- If you elect to use a private email address with GitHub, then use that same email address for the user.email value, e.g. *username@users.noreply.github.com* replacing username with your GitHub one.

More Configurations

We will be using VS Code as our example; we set this with the following command:

```
$ git config --global core.editor "code --wait"
```

Name the Default Branch

Now we will set Git to use the name "main" for default branches. More on branches later….

```
$ git config --global init.defaultBranch main
```

Historical note: You may encounter `master` instead of main in documentation, files, projects, etc. This was very problematic as a naming convention, and so many of us are switching to `main`.

See Our Configurations

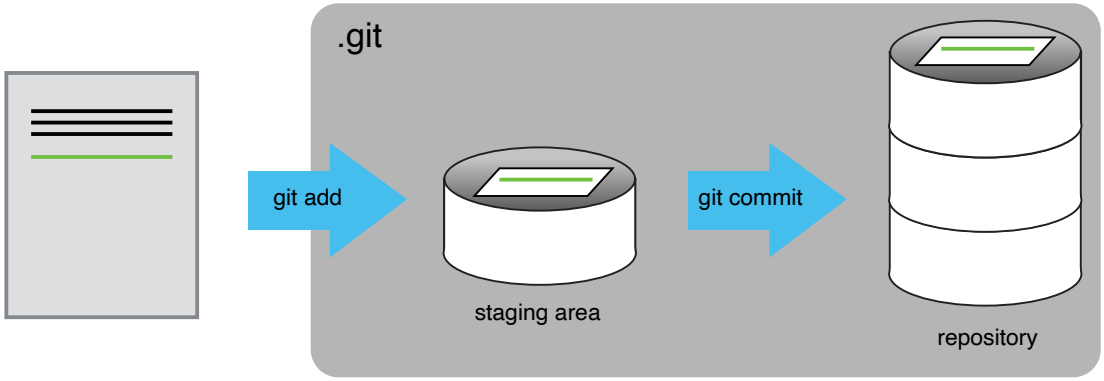
```
$ git config --list
```

Help!

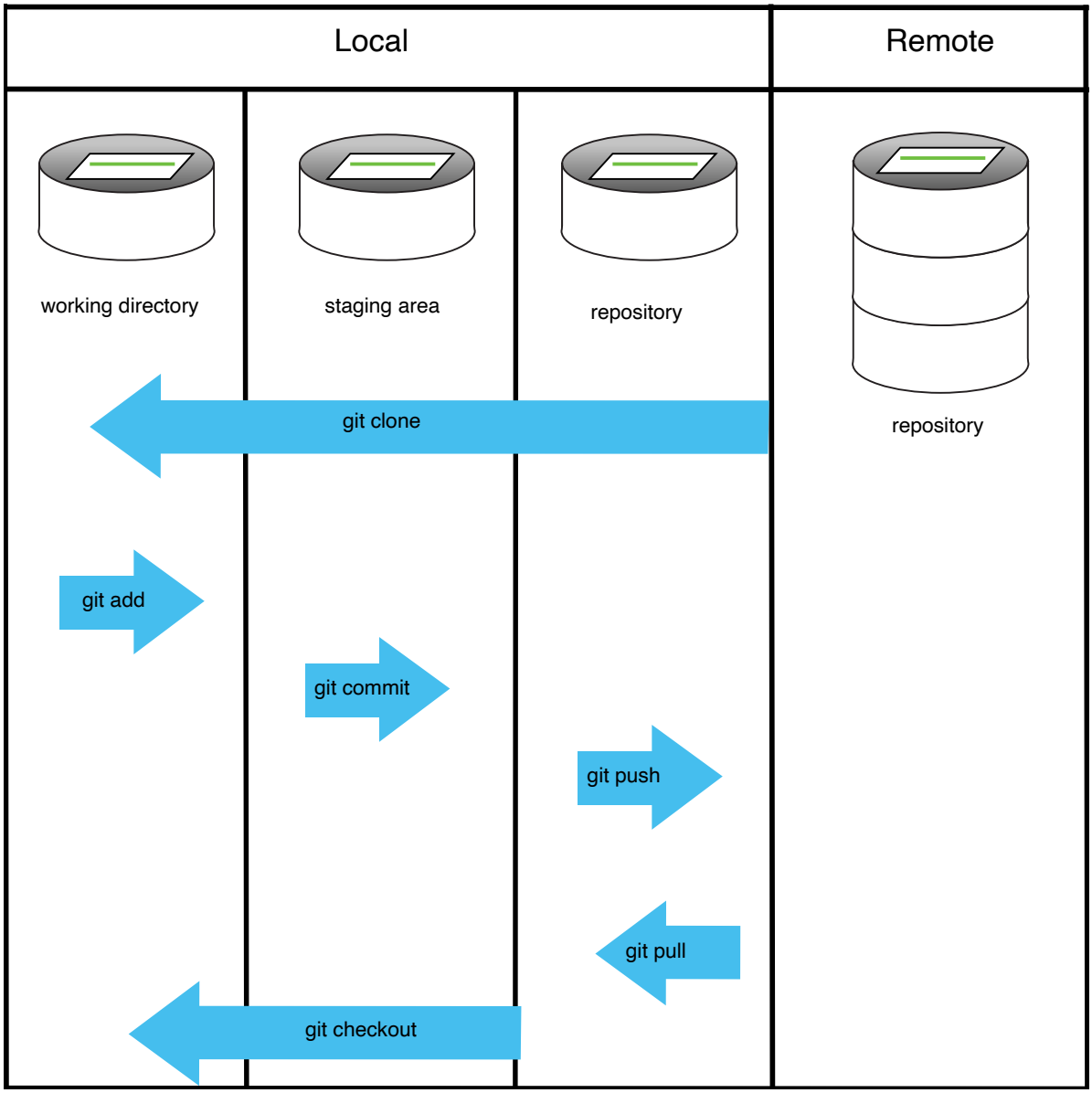
```
$ git config -h
$ git config --help
$ git help
```

Now we are ready to use Git!

Basic Idea



Git Overview



Git Overview

Create

Git Add

Git Commit

GitHub

Branch

Pull Request

Create

First step: Are we making something new or cloning?

- If it is cloning, then clone in a working directory that makes sense
  - Cloning makes a *local* copy of a repository
  - Still has references to that repository
  - Might not be owned by you
- If we are creating something new, then we can create it directly on the local machine OR create a new remote repository
  - We will start by making something local

Create

- First, navigate to a place where you can put files. The *Desktop* can be a good choice; *Documents* or some other directory is probably better.
- Once there, let's make a directory and go into it

```
$ cd ~/Desktop
$ mkdir bsana2024
$ cd bsana2024
$ ls
```

Create

Now we need to create the empty repository for git

```
$ git init
```

Now list the directory

```
$ ls
```

Create

We do not see anything in the directory…but it is hidden

```
$ ls -a
```

Create

We Need a README.md!

- Grab it from what we saw before: https://doi.org/10.5281/zenodo.10828379
- Open it in your favorite text editor (or *Visual Studio Code*)
- Make some modifications to the file!

Git Add

Tell Git What is Going On

- Just adding a file is not enough for the command line; now we need to tell Git it is there
- This is done with the `git add` command

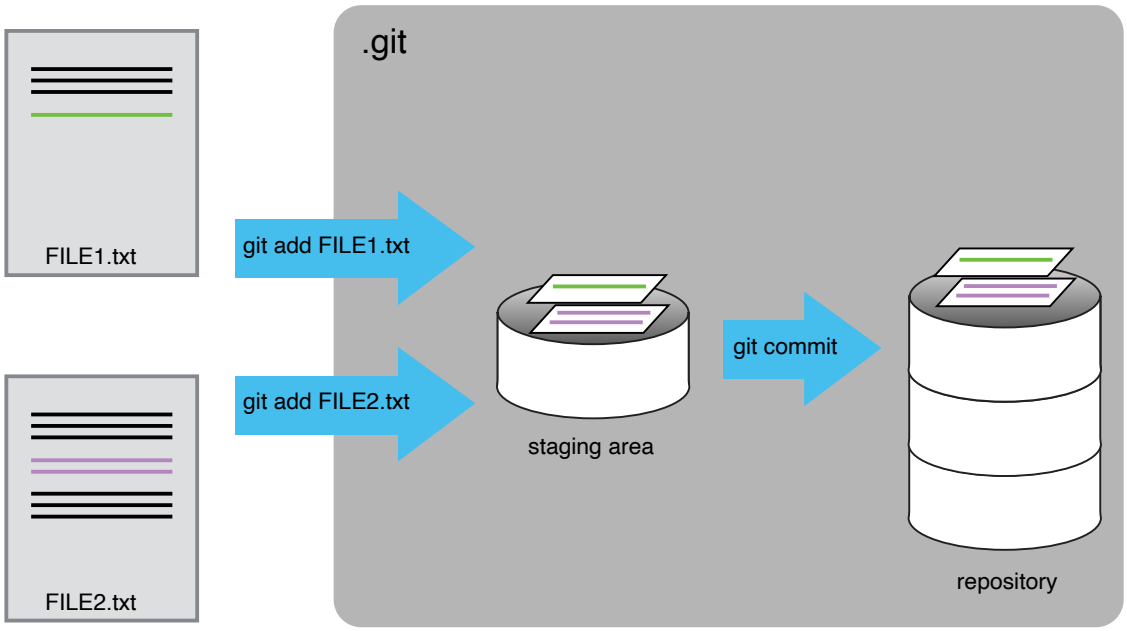
Git Add

This ensures that your files are now seen by Git. This allows you to experiment with data, etc and only add it to be tracked when you are ready to do so.

```
$ git add file_name
```

Now we need to tell Git what we did

Git Commit



Git Commit

- Think of a commit point as a "save point"
- Git tracks changes *BETWEEN* commits
- You should *always* include a message on commits

– Remember what we should do in DH projects?

```
$ git commit -m "valuable_but_short_message_here"
```

Git Commit

What did we do? Let's check!

```
$ git status
```

Git Commit

Let's see the history about this commit

```
$ git log
```

Git Commit

Quick exercise: Can you add a directory with files into the repository?

Git Commit

What about files I do *not* want?

- If I am adding directories
- Files that are huge, there for testing, etc
- Also hidden system files
- This is the `.gitignore` file
- Create a `.gitignore` file, another text file, then add the name of the file to `.gitignore`.

Git Commit

What if we want to *revert* a commit?

First, let's list our commits

```
$ git log --oneline
```

Then find the one we want and revert it

```
$ git revert id
```

GitHub

Now What?

We setup a local repository, everything is going smoothly. We now want to:

- Backup our repository
- Be able to access / change / contribute to our work from anywhere
- Show the world if we want to

We need a remote repository server. The most used / famous is GitHub

GitHub

- GitHub is a for-profit service that hosts remote git repositories
- HTML interface
- Does lots of things easily
- Free service for many projects (public *and* private!)

GitHub

- Create a new repository on GitHub
- Link Our local repository
- Get changes from GitHub to our local repository

GitHub : New Repository

- Login
- Click on “Repositories”
- Click on the green button “New”

- WAIT before going further!

GitHub : Link the Local

- Instructions on the page!
- Navigate to your repository on the local machine
- Check with our favorite command

```
$ git status
```

GitHub : Link the Local

- Copy paste the lines to the terminal and execute. We will discuss what these commands are in a moment.

```
$ git remote add origin https://github.com/<user>/<repository>.git
$ git branch -M main
$ git push -u origin main
```

Now reload the site!

GitHub : Link the Local

```
$ git remote add origin https://github.com/<user>/<repository>.git
```

This command adds a reference to a remote repository which we named *origin*

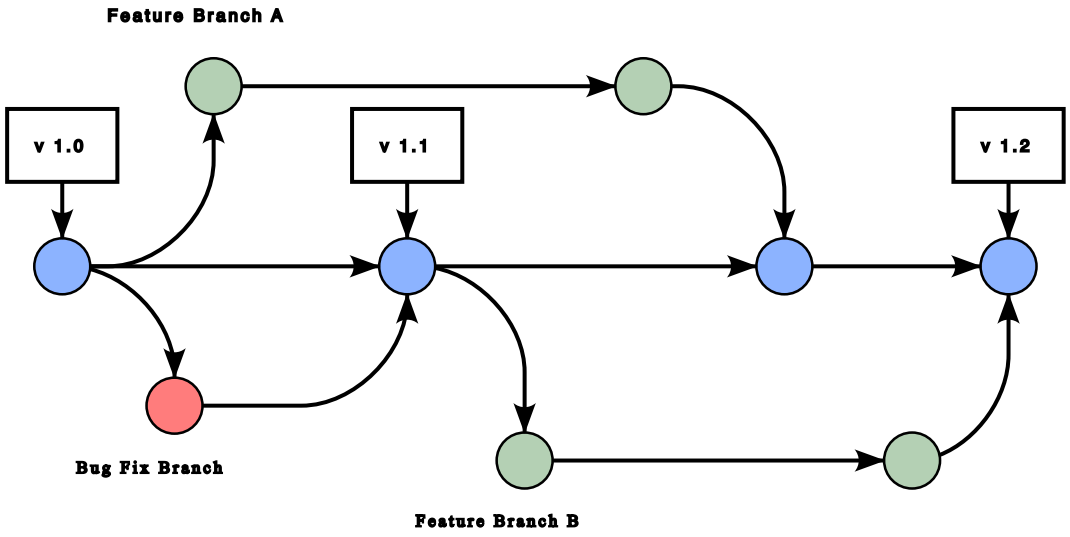
GitHub : Link the Local

```
$ git branch -M main
```

With this, we are creating a new *branch*. The -M option is technically a shortcut for two options: `--move --force`

- `--move`: Move/rename a branch
- `--force`: Resets a branch to start point, allows renaming when used with `--move`.
- Essentially we are overwriting any main branches on GitHub in this repository and pushing our changes there

What is a Branch?



What is a Branch?

- A new/separate version of the main repository
  - You can make copies of the main file, work on them, etc without changing the main file itself
  - Remember our thing about raw data?
  - Someone else can fix an issue with a different part of the file while you work on another part
  - Git allows you to *merge* these

More About Branches

You can issue the following command to make a branch IF you have an origin

```
$ git branch new_branch_name
```

After this, you can *checkout* the branch

Branch: Git Checkout

- Lets you navigate between the branches created by `git branch`
- Updates files in your working directory to match the branch
- You can also create a branch directly with the checkout command:

```
$ git checkout -b new_branch_name
```

Branch: Git Checkout

Next, let us see what the status of our git environment is:

```
$ git status
```

We can also list all of the branches:

```
$ git branch
```

Working with Branches

Create (or add) a new file to the directory

- How do we add this to staging?
- What *else* do we need to do?

Working with Branches: Where are we?

View our log to see what is happening

```
$  git log --all --graph --oneline
```

Working with Branches: Merge

If we think our changes are a good idea, we need to merge our branch back into the main, then clean it up (delete it)

First we make sure we are on the branch we wish to merge *into*:

```
$  git branch
```

Then we merge them

```
$  git merge <branch name>
```

Working with Branches: Merge Results

We can see everything that merged

```
$ git branch --merged
```

Working with Branches: Delete the Old Branch

```
$  git branch -d test1
```

GitHub : Link the Local

From above (and what we just did): We need to *push* to the remote server

```
$ git push -u origin main
```

This *pushes* our changes to the repository. The -u essentially adds a tracking reference that points to the upstream repository (what is the origin of your code)

GitHub : Link the Local

- If we do not push, they simply sit in the staging area of our computer and do nothing
- We can push to our branch, or even another one (main)
  - This can be dangerous - you do *not* to write to main until you are ready!

GitHub : Push

In this code the origin simply means to push the code to the repository you got it from; the branch_name is whatever branch you are pushing to

- If we are the owner this will simply update our code.
- If we are not, we can issue a *pull request*.

GitHub : Pull

- Create a new branch
- Modify README
- Add / commit
- Switch to main
- Push

```
$ git push origin <branch>
```

- What do you see?

GitHub : Pull

- Copy the address in your status message
- We will then create a *pull request*
- We can review changes before allowing them into the code
  - Add more information if you want, then click the green button
- Now refresh the main repository page and we will finish this process

Where We Are

- We created a repository
- We created a branch
- We issued push / pull requests
- What about updating our local code from the origin?

GitHub : Local Pull Request

- Create a new file in your repository through the web interface
- Now we need to pull that information back to our local repository
- What branch should we be in?
  - Try our branch….what happened?

- Now switch to main

```
$ git pull origin main
```

GitHub : Local Pull Request

How do we update our branch with changes in main (or another branch?)

- We are using merge!
- Checkout the branch
- Merge with `<branch_name>` (in our case main)

```
$ git checkout <branch_name>
$ git merge main
$ git checkout
$ git push origin <branch_name>
```

Take a Mental Break!

While We Are At GitHub...

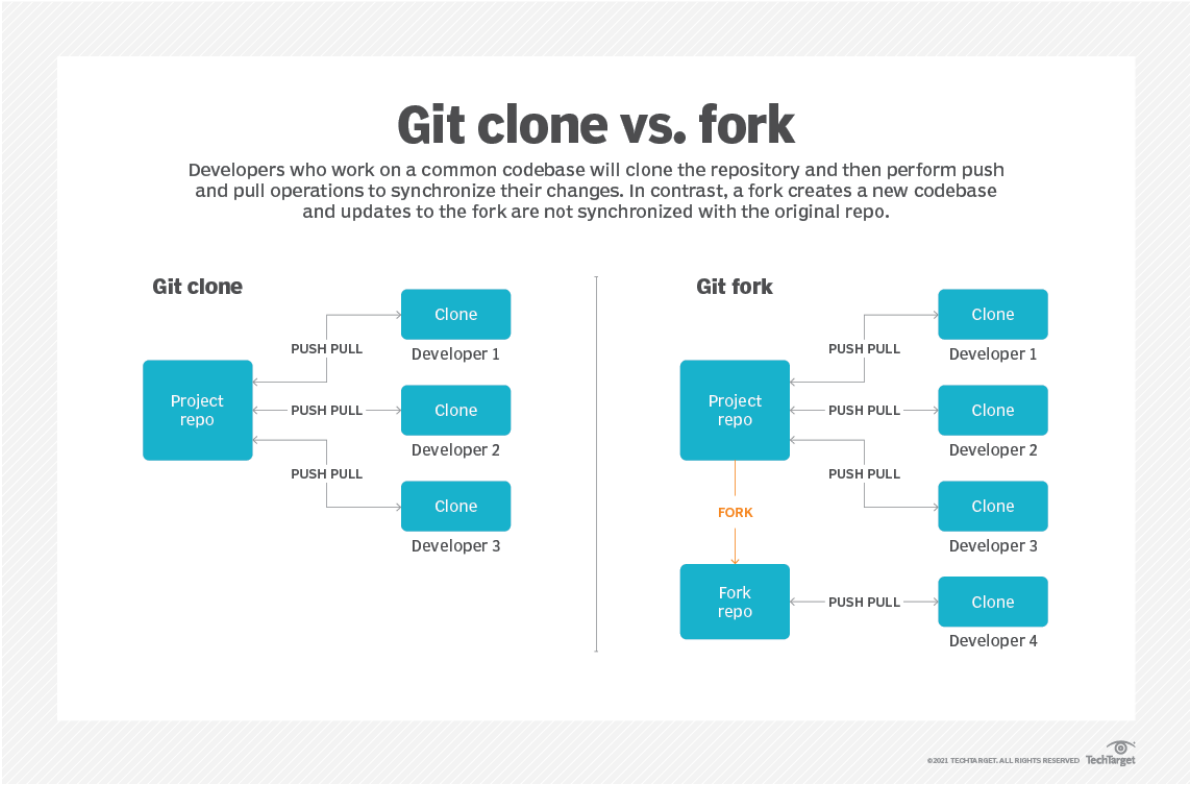
Time to do things the easy way!

- Did it through the command line as it does not abstract the process
- You now know what is going on; We can simplify things
- We are going to go through git on GitHub / GitHub desktop

What About Remote Repositories?

There are lots of things out there that you may want to copy

- https://github.com/lawdi/LAWD
- https://github.com/SNAP-DRGN
- https://github.com/AWMC/geodata
- https://github.com/recogito
- https://github.com/pelagios
- https://github.com/LinkedPasts
- https://github.com/isawnyu/pleiades-gazetteer
- https://github.com/WorldHistoricalGazetteer

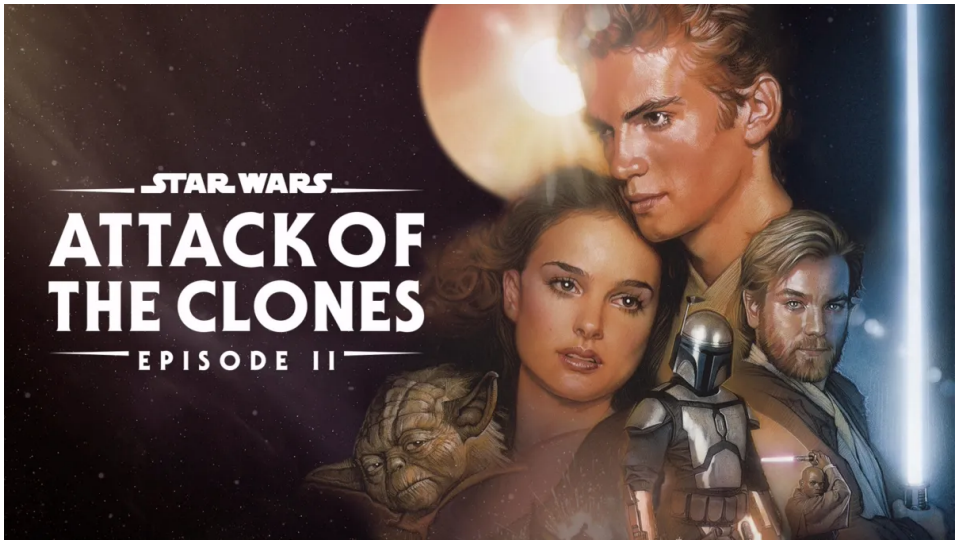


Clones, Forks, and You

- Would you clone or fork the repository for this workshop?
- Why?

Now Clone an Existing Repository

- Makes a local copy of a repository on your machine
- You can pull changes
- You can make pull requests
- Does not impact the original repository
- Hopefully a better time than Episode 2!



Attack of the Clones!

- Navigate to where you want the new code; you do NOT have to make a new directory!
- Go to the workshop repository: https://github.com/2024-5-13-5-17-MJC-BSANA/MJC-BSANA-Data-Workshop-2024
- Click on the green "code" button
- Copy the code
- type `git clone` and paste in your terminal

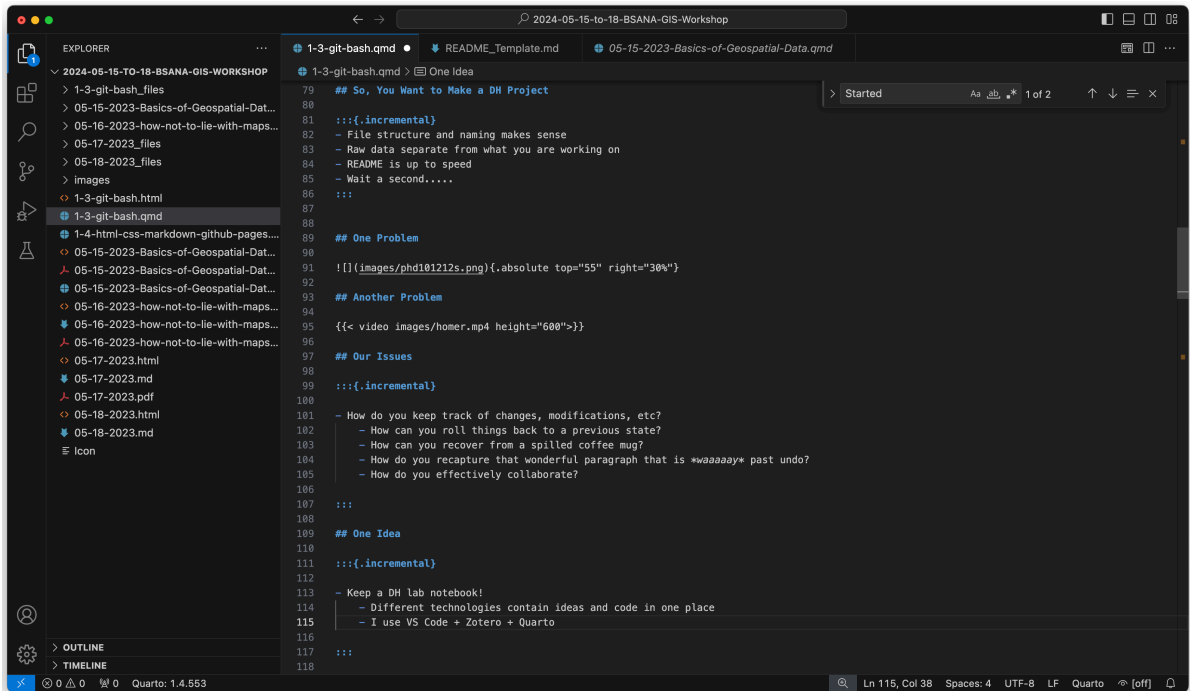
Forking in GitHub

- Easy to do with the web interface
- *YOUR* own copy of a repository; you say what gets pushed in, etc
- Does not impact the original repository
- Back to GitHub!

Even More Tools!

- Keep a DH lab notebook!

  - Different technologies contain ideas and code in one place
  - I use VS Code + Zotero + Quarto
  - Integrates **directly** into GitHub



This Was a Lot!

- You did it!
- This is a technical process that is not *strictly necessary* for DH work
- HOWEVER, this is the *sine qua non* of collaborating with data and code
- Also a great way to structure writing; all of my academic writing is now done with this process
- Also sometimes necessary for getting *your* web pages, writing, etc out there
  - We will go over this later!
  - Don't forget - you want to *communicate* your wonderful DH work
  - Also want to impress / sell your skills on the job market