Workshop V

Version Control with Git CLI

Prerequisites and Background

Prerequisites

- GitHub Account

  - <https://github.com/join>

- Git

  - <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

  - Make sure `git --help` works

- Text Editor

  - Vim, VSCode, etc...

  - I will be using Vim on the Git Bash on the Windows Operating System

GitHub vs Git

- **GitHub** is where your code is stored online

- **Git** tracks changes to your code

- An analogy with Google Drive

  - **GitHub** is like **Google Drive**

  - Your **code and files** are individual **Google Docs**

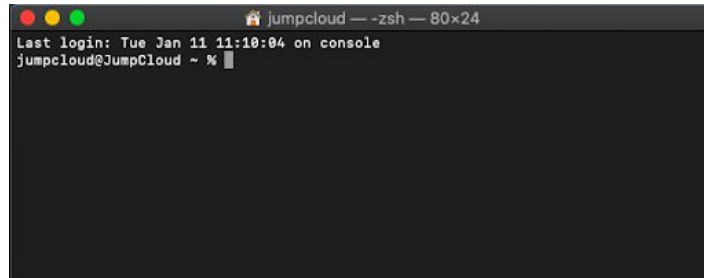 - **Git** is the **saving and version history functionality** of Google Docs

CLI and Cloning

The "CLI"

- **CLI** stands for the **Command Line Interface**

  - Aka the **terminal**

- Enables you to run commands on your computer

Commands and Terminology

- A **directory** is a location in the computer where files are located

- **cd** is the command to change directory

- **ls** lists all files and directories at a location

- **mkdir** creates a folder

- A **repository** is where your code as well as its history is stored
  - Can have a local (on your computer) or remote (online on GitHub) repository

Cloning

- **Cloning** is copying code from GitHub onto your computer
  - We call this creating a **local** copy
- The code is copied from a **repository**
- Done using the git clone command
  - Ex: `git clone https://github.com/dylanvu/OPS-GitHub-Workshop.git`
  - You'll make making your own repo next!
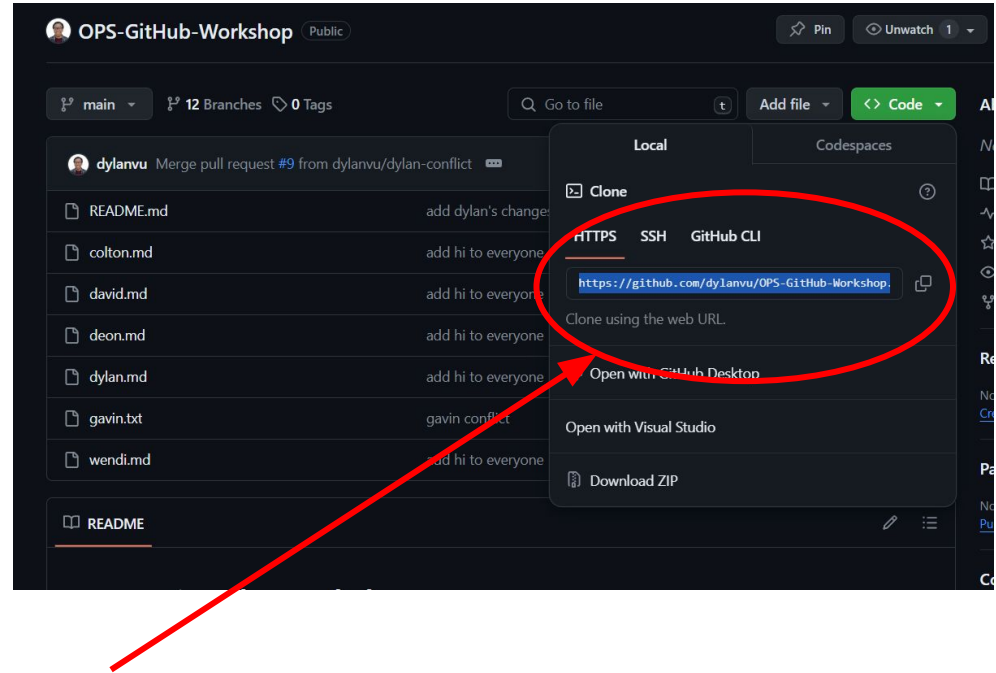
Example: Obtain the URL to clone your repository

Follow Along

Create a repository and clone it!

1. Create a new repository on GitHub with a README.md
2. Open up the Git Bash CLI
3. Clone the repository
4. cd into the repository
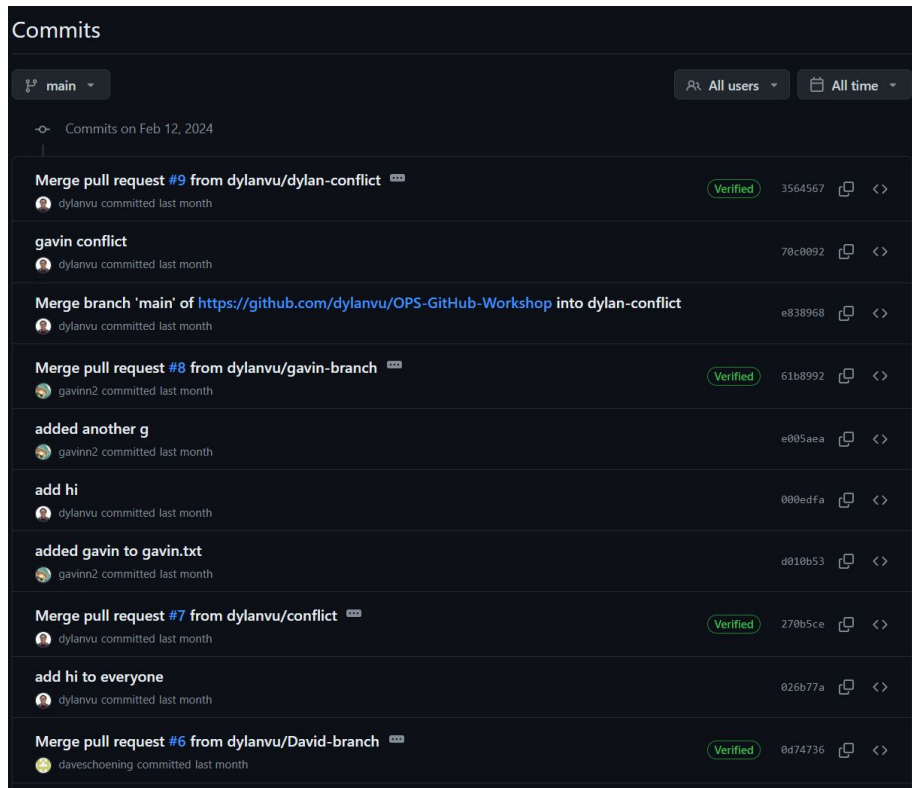
Adding Git to Existing Repositories

- Done using the `git init` command
- Once git is initialized, you can put it on GitHub

Git Workflow: Overview

How Git Remembers

- Git tracks what has changed between the current version and the last **commit**
  - A commit is like a **checkpoint**: you can go back to it
- A commit is identified by its **SHA**: a unique 40-character hexadecimal string

The Git Workflow

- There are several steps to "save" your code changes
  1. **Add**: Tell git what files you want to save
     - `git add <files>`
  2. **Commit**: Create the checkpoint
     - `git commit -m "your commit message"`
  3. **Push**: Put the code on GitHub
     - `git push origin <branch>`
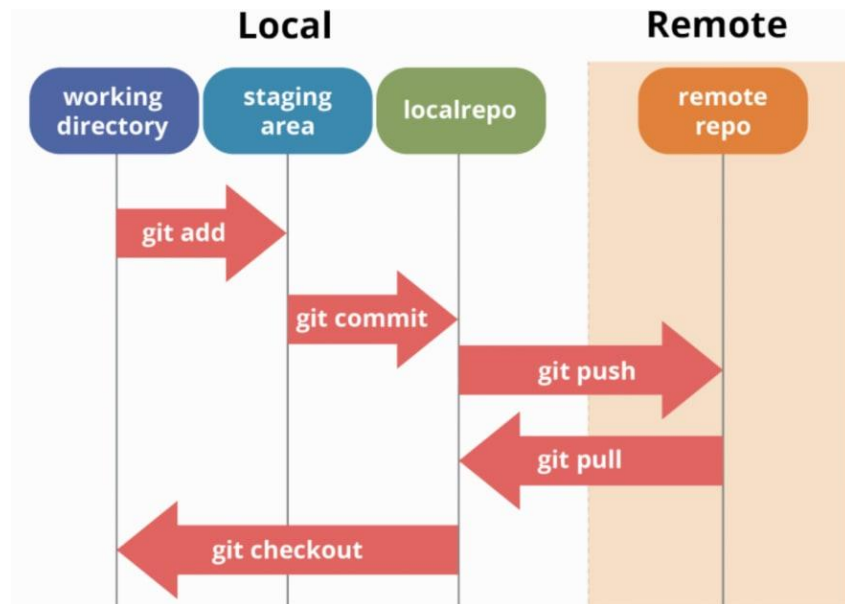
Any Questions?

Git Commands

Git Status

- Git status tells you the state of the repository
  - What files have changed
  - What files have been added
  - What files are not being tracked
  - What branch you are on
  - Etc…

Git Add

- Git needs to be told what files to save
  - This is done through the `git add` command
- Without adding, you cannot save your files!
- Use `git status` to check what files have been added or not


```
Dylan@Dylan-ASUS-Laptop MINGW64 ~/VSCode/ops/OPS-GitHub-Workshop (main)
$ git status
On branch main
Your branch is behind 'origin/main' by 3 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

Dylan@Dylan-ASUS-Laptop MINGW64 ~/VSCode/ops/OPS-GitHub-Workshop (main)
$
```

```
Dylan@Dylan-ASUS-Laptop MINGW64 ~/VSCode/ops/OPS-GitHub-Workshop (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md

Dylan@Dylan-ASUS-Laptop MINGW64 ~/VSCode/ops/OPS-GitHub-Workshop (main)
$
```

Git Commit

- Git needs to explicitly know when to make a commit
- `git commit -m "your message here"`
- Every commit needs a message
  - Specified in the command with a -m flag
  - Without a -m brings up an editor (usually in vim) to create a message

```
Dylan@Dylan-ASUS-Laptop MINGW64 ~/VSCode/ops/OPS-GitHub-Workshop (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md

Dylan@Dylan-ASUS-Laptop MINGW64 ~/VSCode/ops/OPS-GitHub-Workshop (main)
$ git commit -m "update readme with to demonstrate committing"
[main c7c1a0a] update readme with to demonstrate committing
 1 file changed, 3 insertions(+), 1 deletion(-)

Dylan@Dylan-ASUS-Laptop MINGW64 ~/VSCode/ops/OPS-GitHub-Workshop (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Dylan@Dylan-ASUS-Laptop MINGW64 ~/VSCode/ops/OPS-GitHub-Workshop (main)
$
```
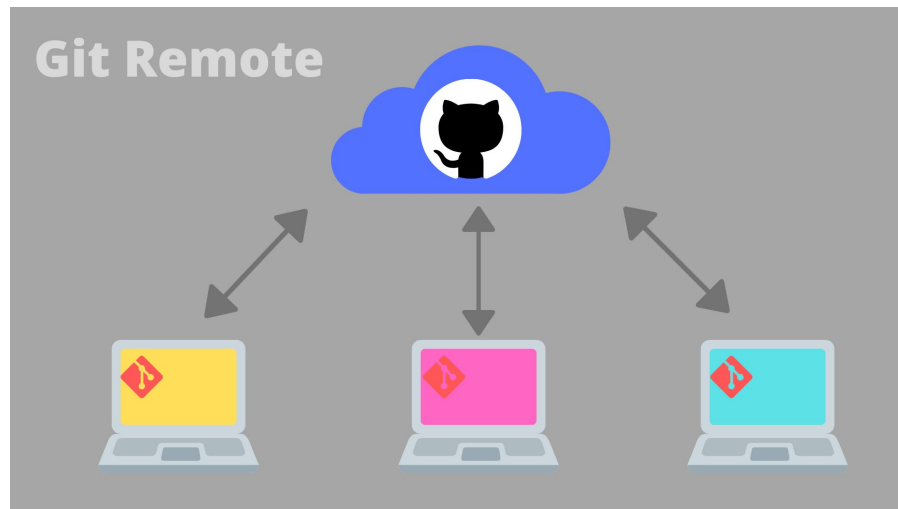
Git Push

- The code on your computer is all **locally saved**
- Use git push to "push" your code to GitHub
  - We call GitHub the **remote**
    - The **remote** is where versions of your project live on the internet or network somewhere
  - Useful for other teammates to view and obtain code change

Any Questions?

# Follow Along

Let's go through the Git workflow!

1. Create any change to the README.md file
2. Add, commit, and push the change
3. View the change on GitHub!

Branches and Merging

Branches

- **Branches** are named pointers/copies to a specific commit
    - Modifying it is like working in a parallel world
- Typically holds one specific feature or fix
- Important
    - If something happens, you can delete/purge a branch without affecting other versions/parts of the codebase
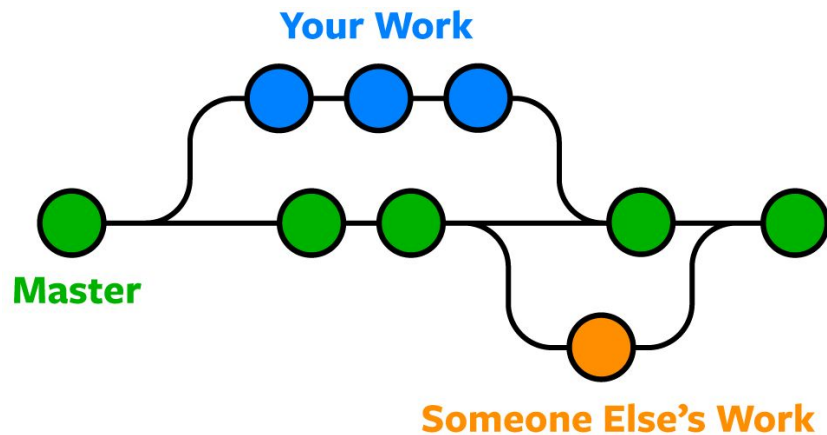
Main/Master Branch

- Most repositories have a **main** or **master** branch
  - We call this the **default branch**
- All code is descended from the default branch
  - Breaking the default branch essentially breaks all branches
- The code on the default branch generally is what is in **production**
  - What other users will see

Git Checkout

- Used to create new branches and/or swap to another branch
- `git checkout -b <new-branch-name>` to create a new branch
- `git checkout <existing-branch>` to switch to a new branch
- Check which branch you are on using `git status`
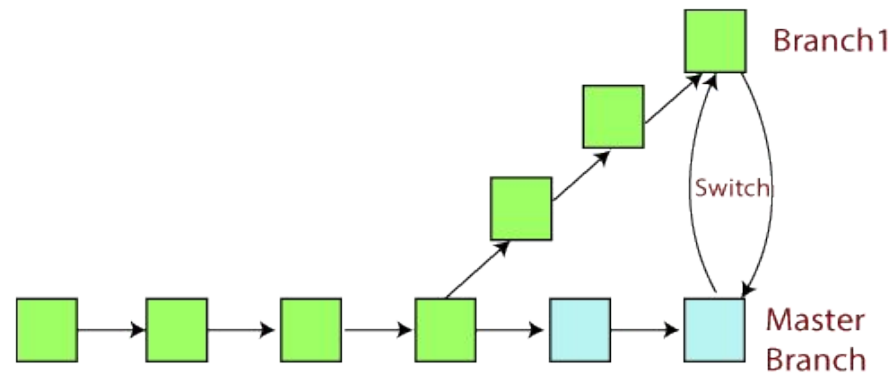
Git Checkout

Follow Along

Let's go through Git branches!

1. Create a branch using the Git CLI named `branch-demo`
2. Create a change to the README.md
3. Push to the branch
4. View the new branch in GitHub!

Merging

- Combines code together
- Command: `git merge <branch-name>`
- Example:
  - You write a function in a file, and your teammate writes a function
- Does not delete any branches!

Deleting a Branch

- Delete it locally
  - `git branch -d  <branch_name>`
- Delete it on remote
  - `git push origin -d <branch_name>`

Pulling

- Gets the latest code from GitHub and then combines it with your current code
- Command: `git pull origin <branch-name>`
- You must get the latest version on GitHub (pull) before you push
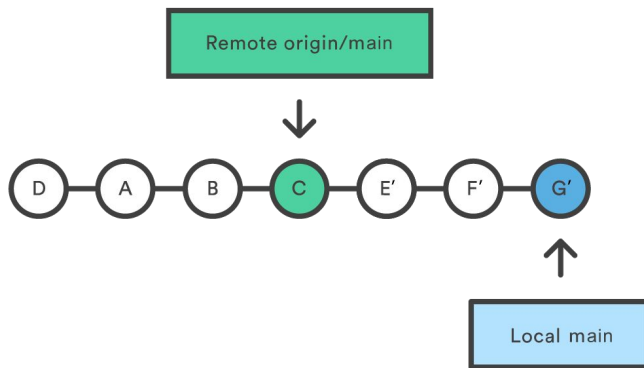
Working with a Team: Pull Requests

- Git merge enables you to merge code to other branches without any reviews or restrictions
  - Putting bad, unreviewed code into main can be dangerous
  - The process of obtaining permission and awaiting code review is done through a **pull request**
  - Done through the GitHub website
  - The base is where the code will go into, and compare is where the code is coming from
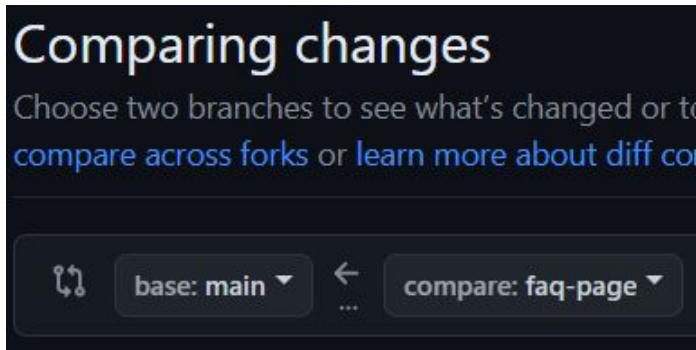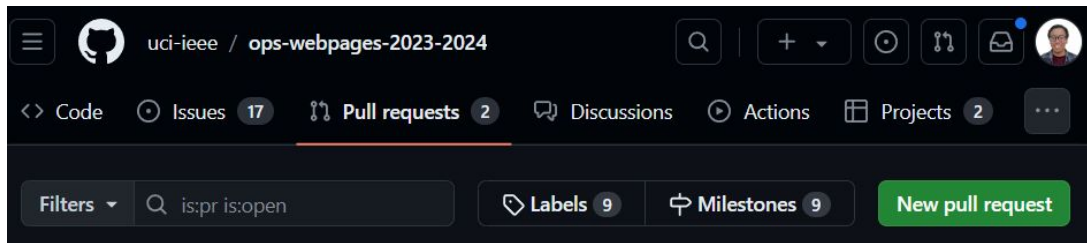
Any Questions?

Follow Along

Let's merge your code from your `branch-demo` branch into main!

1. Checkout to the main branch
2. `git merge branch-demo`
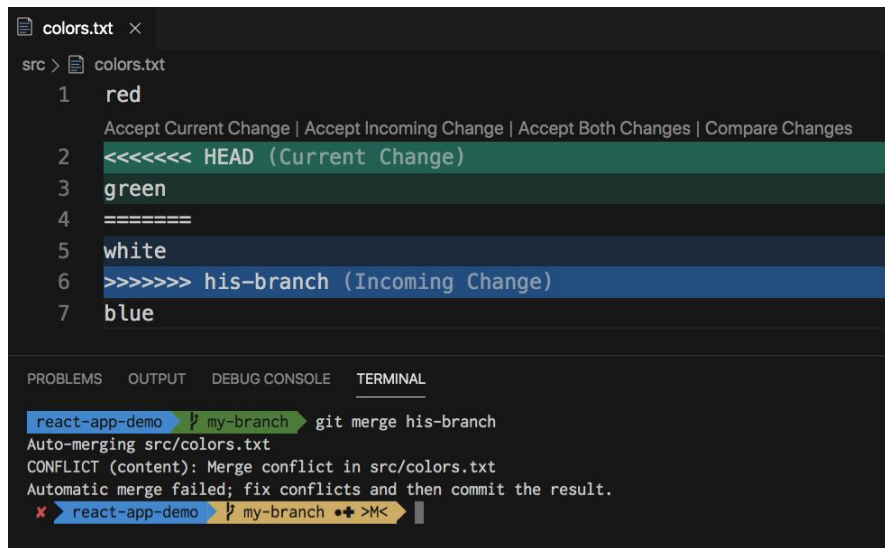3. Push to remote and view the change on GitHub!

Merge Conflicts

Working in a Team

- Git is super useful for working in a team
    - Other people can modify the same file, and Git tries to merge it all together
- What happens if two people modify the same line of code at the same time?
- We get a **merge conflict**
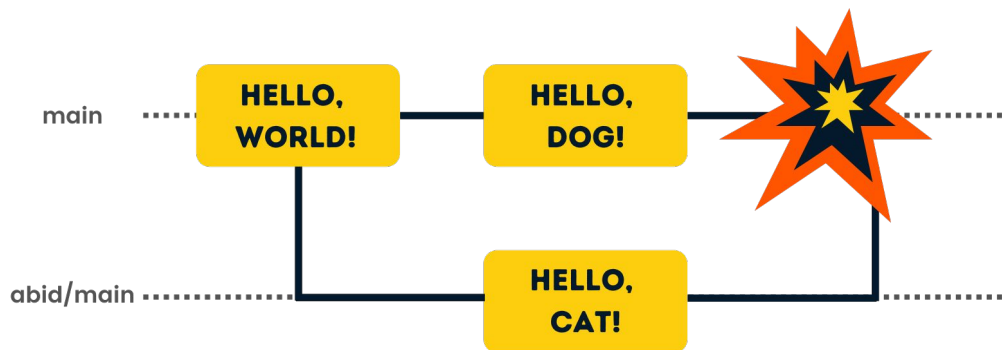
Merge Conflicts - Part 1

- Caused by simultaneous changes by two people
  - Same line of code modified
  - A file deleted while someone edited it
- Typically, conflicts are resolved by the one who triggers the conflict in the code
  - If conflict is too complex, both owners of the code are involved and resolve the conflict **together**.

A merge conflict on VSCode

Merge Conflicts - Part 2

- Identified when you merge or pull code through a bunch of arrows
- After resolving the conflict, must add and commit again

A merge conflict on VIM

Any Questions?

Follow Along

Let's create + resolve a merge conflict!

1. Create a new branch called <u>merge-conflict</u>
2. Modify a line in the README.md, commit, and push it
3. Checkout to main
4. Make an edit to the same line modified in step 2
5. Merge <u>merge-conflict</u> into main
6. Resolve the merge conflict through vim

Reverting Changes

Reverting a Commit

- Undoes the changes associated with a specific commit through another commit
- Example:
  - Say you create a commit A, then make 2 more commits (commit B and C) after
  - To revert to commit A, grab the SHA
  - `git revert <SHA>`
    - This reverts the changes associated with commit A only, and not commit B and C

Reverting Multiple Commits

- What if you wanted to revert many commits?
- Example
  - A → B → C, and you wanted to revert the commits from the current from C and B
- Many ways: [Stack Overflow post here](#)
  - Each method is not equivalent and varies in amount of work
    - Some don't work for merge conflict
    - Some don't work for new files created
  - Suggestion: git reset method

Follow Along

Let's create a commit and revert it!

1. Modify a file
2. Commit and push it
3. Obtain the SHA and revert the change

Any Final Questions?
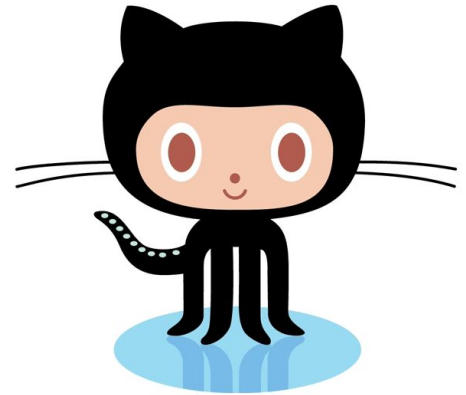
Becoming a Git Wizard

If we had more time…

- Educational Cheat Sheet: https://education.github.com/git-cheat-sheet-education.pdf
- Useful Additional Topics and Commands
  - .gitignore
  - Unstaging
  - Rebase
  - Stashing
  - Forcing your way through Git & permanently altering history
  - Git large file storage (LFS)

FAIR USE DISCLAIMER