What is git? What is GitHub?

Git is a version control system. GitHub is a platform for hosting git repositories. You will have a local copy of the code on your machine as well as a remote copy on GitHub.

Git & GitHub setup

You should:

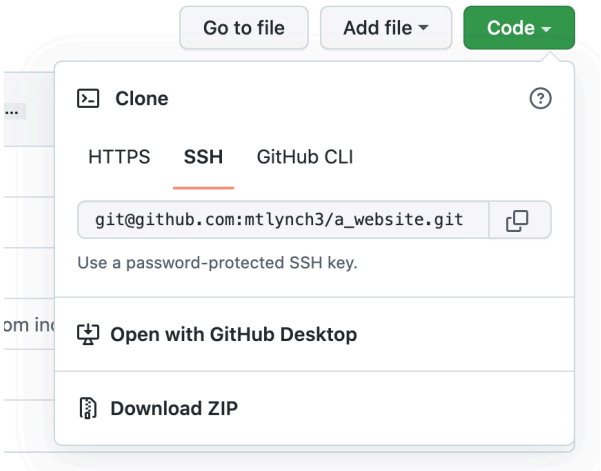
- Have git installed locally
- Have a GitHub account
- Be able to connect to GitHub from your machine via SSH

Once you have the above, you can create a repository on GitHub. You should initialize the repo with a README.

To get a copy of this repo locally, you need to clone it. Open your terminal and use the git clone command along with the URL of the remote repository:

git clone git@github.com:username/repo-name.git

The remote URL can be found on the GitHub page for the repo:



Remember to always use the SSH link when cloning a repo.

Cloning a repository to your local machine will create a folder with the same name. You should know where in your file system you are cloning the repo. For example, if you run **git clone** from your Desktop, that's where the folder containing the local repo will be.

Basic git workflow

Once you have a local git repo, you can start making changes. You can edit existing files, create new files, or delete files. Use any text editor or IDE (Sublime, Atom, VS Code, etc) to edit your files.
As you are making changes, you should be making commits frequently to keep track of your progress.

To make a commit, you first need to stage the files you want to commit:

git add filename

To add multiple files, you can list filenames separated by spaces (**git add filename1 filename2**), add all files in the current directory (**git add .**), or add all files in the repo that have been modified (**git add -A**).

Once you have the relevant files staged for commit, run the following command:

git commit -m "descriptive message"

You should be committing frequently and providing descriptive commit messages. As you commit, you are updating the git project history but this is only reflected locally. To update the remote, we need to push our commits. If you are pushing from the main branch, you can do:

git push

If you are pushing from a new branch for the first time, you should set the upstream with the **-u** flag:

git push -u origin branch-name

After this, you will be able to push commits from the new branch using only **git push**. Note: **origin** is an alias for our remote URL

Branch management

When creating a new feature branch, you should be creating it from the most updated version of the main branch. First, make sure you are on the main branch: **git checkout main**

To make sure your local main branch is the same as the remote you can run **git pull**.

Once your local main branch is up to date you can create a feature branch:

git checkout -b feature-branch

The **-b** flag tells git to create a new local branch called **feature-branch** and switch to that new branch (since you are running this command from the main branch). To switch back to the main branch at any time you can do **git checkout main**. Likewise, once the branch is created you can switch back to that branch with **git checkout feature-branch**.

Make changes to your repository while on the feature branch; you will stage, commit, and push as reviewed above. When you are done with the branch and ready to merge the changes into the official project history (the main branch), you should first open a pull request. An

option to do so will appear on your remote repo once you push the branch to the remote.
Once the pull request is created you can close it by merging the feature branch. After the branches are merged you can delete the feature branch.

At this point, your remote repo is more up-to-date than your local repo. To update your local repo so that it is the same as your remote, switch to the main branch and run:

git pull -p

This will update your local main branch so it has the same commit history as the remote. The **-p** flag is there to remove (or "prune") any "stale" branches you have locally. Stale means the branch has been deleted on the remote but it still shows up as a remote branch when running **git branch -a**.

Even after deleting the feature branch on the remote and doing git pull with pruning, your local feature branch still exists. To delete it use the following command:

git branch -d feature-branch

Useful commands

- git diff branch1 branch2
    - Show differences between branch1 and branch2
- git branch -a
    - Show all branches, local and remote
- git checkout -t origin/branch-name
    - Check out a remote branch locally
- git restore filename

    - Undo non-committed changes
- git reset --hard HEAD~1
    - Revert repo to previous commit (undo most recent commit)
- git reset --staged filename
    - Unstage file (i.e., undo git add filename)
- git remote -v
    - Show remote url (origin)
- git log
    - View commit history

More detailed info on undoing changes to a repo

What is HEAD? It is a pointer to the most recent commit on a branch. Each branch has its own HEAD.

The remote (origin) HEAD is usually pointing to the most recent commit of the main branch, as that is the default branch. This tells git what version of the project to use when cloning a repo.

git show HEAD

This command will display the commit ID that HEAD is currently pointing to (usually the most recent commit of the current branch), the commit message of this commit, and the changes that were implemented by this commit.

Questions from class:

Can you have multiple remote repos? Yes.

Are there other kinds of git workflows? Yes.

In this course, we will only be using feature branch workflow which uses one central remote repository.