Git and GitHub

- In this course, we'll heavily use two tools, Git and GitHub, that are incredibly popular in the software development world.

- It's important to note that Git and GitHub are two separate tools, each with its own purpose. We'll review each of these tools here.

Git overview

- If you do any kind of serious, collaborative (or even individual) work as a software engineer, you'll probably use a ***version control system (VCS)***.

- A VCS is a tool (a program) for managing changes to your code and for making it easier to work with many people on the same code.
    - Git is one of the most popular VCSs around.

- Git manages changes in your code by taking a snapshot of your entire codebase every time you tell it to.
    - These snapshots are stored permanently in a ***repository***, which you can think of as a kind of database that lives on the computer where you're working.
    - Storing a snapshot in the repository like this is called ***committing*** your code***.***
    - You can think of a commit as a milepost, of sorts. It's a place in your work you want to remember and make record of, something you don't want to lose.
    - Every new commit records a new ***version*** of your code.
    - Git maintains a ***history*** of all of the versions of a project ever recorded.
        - You can look at (and even revert to) your code at different points in its history, and compare the differences between different points in the history.

- Git is a ***distributed VCS.***
    - Many computers can hold a copy of a repository.
        - Any non-local Git repository is called a ***remote repository***.

    - Every copy of a repository is a full copy of the entire database of snapshots.
        - Like P2P.
    - Git has commands to synchronize copies of a repository between two machines.
    - This allows many people to work on the same piece of code easily.
        - Each person makes changes and commits them to their local repository.
        - Then they use Git's synchronization commands to make sure their repositories are in sync.
            - Changes can be ***pushed*** from the local repository to a remote repository.
            - Changes can also be ***pulled*** from a remote repository to the local repository.

GitHub overview

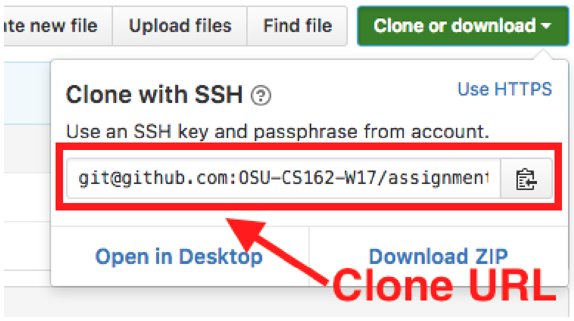
- GitHub is a web application that does several things:
    - Hosts Git repositories on the cloud.
        - These typically serve as a central (master) remote repository for one or more developers.
    - Provides a nice web interface for browsing code in a Git repository.
    - Provides nice web-based tools to collaborate on code (centered around Git repos).
    - Provides tools to link code to external services (e.g. for building, testing, or publishing code).

- In this class, we'll mainly use GitHub mainly for its cloud-based repository hosting service.
    - Specifically, we will use GitHub to host remote copies of the repositories in which we store our programming assignment code.

How we'll use Git and GitHub in this class

- In this class, we'll primarily use one essential Git/GitHub workflow:
    1. For every programming assignment, you'll be provided with a Git repository hosted on GitHub.

2. You will use Git to make a copy of this repository on your development machine. Specifically, you'll use the command `git clone`.
   - Every repository on GitHub has a unique URL you can use to clone that repository:



Clone with SSH

Use an SSH key and passphrase from account.

```
git@github.com:OSU-CS162-W17/assignment
```

   - Note that GitHub provides two different kinds of clone URLs: one that uses SSH for communication (like the one above) and one that uses HTTPS for communication. **Importantly, to use the SSH URL, you must have SSH keys set up with GitHub. If you try to use the SSH URL without SSH keys set up, you will receive an error that says something about a public key being denied.**
     - If you don't know what SSH keys are or don't want to bother with them, just use the HTTPS URL. When performing remote Git operations (e.g. clone, push, pull) using a GitHub HTTPS URL, you'll simply be asked to enter your GitHub username and password to authenticate yourself.

   - Using that URL, you can make a copy of the GitHub repository on any other machine (including your development machine) with a command like this:

```
git clone git@github.com:OSU-CS290-F17/assignment-1-robwhess.git
```

   - The copy of the repository on your machine will simply be a directory full of the files from the repository.

3. Once you have a copy of a repository on your development machine, you can work in that directory as you wish, modifying, adding, or deleting files.
    - At any point, you can use this command to print a summary of the current state of your work on the Git repository:

      ```
      git status
      ```

    - If you want to see the changes you've made to the code, listed line by line (in unified diff format), you can use this command:

      ```
      git diff
      ```

4. Once you've made a certain amount of progress (e.g. you got a function working, you moved code to a new file, etc.), you'll want to commit a snapshot of your code into your local Git repo. First, you'll have to ***stage*** the files whose changes you want to include in the snapshot.
    - In Git, committing is a two-step process:
        - First you stage (i.e. mark as ready for commit) the files you want to commit.
        - Then, you actually commit the staged files.
    - Git works this way so that you can pick and choose which files actually get committed. In other words, you don't *have* to commit every file that's been changed, only a subset.

    - You can stage a file with the `git add` command, like this:

      ```
      git add some_code.cpp
      ```

    - As before, you can always call `git status` to see what files are staged.

5. Once your files are staged, you can execute your commit with the `git commit` command:

   ```
   git commit -m "A short message describing this commit"
   ```

    - The `-m` option allows you to provide a short message to describe your commit, so you can get a quick sense for the commit when you look back on it later.

- If you omit the `-m` option, Git will open a text editor for you so you can write a message to describe your commit.

- If you ever want to look at the history of commits you've made on a given repository, you can use this command:

  ```
  git log
  ```

6. Once you've made a commit (or a few commits), you'll want to make sure they're saved on GitHub. To do this, you can use the `git push` command:

   ```
   git push
   ```

   - This synchronizes the remote repository on GitHub with your local repository, pushing any new commits you've made into the remote repo.
   - You can verify that your code is saved on GitHub by using your browser to navigate to your repository there and checking to see if your changes show up.
     - You should be able to see your new commit messages or see the actual code changes you made.

Summary of particularly useful Git commands

Here's a list and brief description of the Git commands described above:

- `clone` – copies an entire remote repo to the local machine
- `log` – prints the history of all commits made to the local repo
- `status` – prints a brief message describing the working state of the local repo
- `diff` – prints the actual differences between different versions of the local repo
  - By default, `diff` prints the difference between the working (i.e. current) code and the last commit.
- `add` – stages a file for commit
- `commit` – commits (i.e. take a permanent snapshot of) all the staged files
- `push` – synchronizes all commits *from* your local repo *to* a remote repo (e.g. your GitHub remote repo)