Chapter 0: Introduction to Git

Throughout this course and most future programming endeavors, you will be using version control systems to manage your code. Git is one common version control system and will be the primary focus for this course.

0.1. What is Version Control?

Version control systems are just that: a way for us to control the versions of our code. This means tracking the changes that we've made, and control our revisions.

Think of how you would work together with others (or with yourself across multiple machines) on a coding-based project: do you send around a `.zip` file of the latest version of your code? Do you send snippets of the changes you want made to someone else, who has the master copy? Do you work using a collaborative software (like Google Docs), updating your code in the same place?

All of the mentioned possibilities have shortcomings: either it's hard to keep track of your current version or find out what revisions were made at each step, or it might be hard to work independently. If you make a mistake and need to roll back your changes, you'd either have to find an older version or your code or manually revert things! That's awfully tedious!

Version control doesn't have any of these shortcomings: everyone works on their own independent version and the version control system lets you merge your changes in as needed, keeping track of revisions as they're made.

0.2. How version control works

In version control systems, there is a **master** repository: a copy of the latest versions of all files. People **clone** the repository to get their own local copy, which they work on independently.

As people make changes and reach a state that they want reflected in the master repository, they **push** their changes in. Similarly, anyone who wants the latest version of the repository will **pull** the changes.

There are many other features of version control, but the above is the general concept and the simplest explanation for how we use version control systems.

0.3. Git

Git is the specific type of version control that we'll be using in this course.

In git, the master repository is also known as the **origin**. We use the command `git clone <url>` to get a local copy of the repository. This is similar to just downloading all of the files, but with the fancy addition of having a way to track which files we've altered via `git status`.

0.3.1. Making changes to the origin

After modifying, adding, and removing files as needed, we likely want these changes to be made in the master repository. To do this, we use `git add <files>` to add all of the files that we've changed and that we want to change in the master repository.

Following the `add` command is `git commit -m "<message>"`, which we use to describe what changes we've made (e.g. `git commit -m "Fixing a bug in X"`). Finally, to make these changes in our repository, we finish with `git push`!

Thus the general workflow for making changes is as follows:

- `git status`: Lets us see what files have changed in our local copy
- `git add <files>`: Lets us add ("stage") files that we want to modify in the origin.
- `git commit -m "<message>"`: Saves our changes to the local repository, labelling the added changes with a message to describe our changes.
- `git push`: Pushes changes to the origin repository

One caveat to this workflow: if there are changes in the master repository that you don't have yet in your local version, you'll need to **pull** these revisions.

0.3.2. Pulling new revisions

As you work with a repository, you'll likely need to copy over any changes into your local repository. To do this, you can use a simple `git pull` command.

At this point, you may have to do a bit of additional work: adding/committing local changes or manually handling some merge conflicts.

0.3.3. Branches

Sometimes you'll want to work on an entire feature in a separate repository: committing your changes directly to the master branch might not make sense, as you'll only want to merge your changes in once you finish the feature.

In cases like this, it's best to make a new **branch**: this is a spinoff of the master repository, in which you can commit changes without worrying about the master branch until you're ready to merge it in.

To create a new branch and use it, you can use `git checkout -b <branch name>`. This is shorthand for `git branch <branch name>` followed by `git checkout <branch name>`: the first creates a new branch, and the 2nd switches your local repository to the new branch. The commits you make will now go to `<branch name>` instead of the origin.

If you want to switch branches, just use `git checkout <branch>` to change to that branch.

When you want to finally merge your branch into the master repository, you simply checkout your master branch, and then run `git merge <branch name>`. However, a nicer method is to make a pull request, such that people can see and review the changes you're trying to push in.

0.4. More resources on Git

There is extensive documentation surrounding git, which you'll likely discover as you use it. Instead of rewriting everything, we've collected a few such resources for you:

- BetterExplained: A visual guide to version control
- GitHub: Quickstart