Intro to Git and Version Control

Intro

So far, when you work on software, you probably have been editing a file, saving it, compiling it and moving on when it worked. If you are lucky, you might have had no real problems doing this. But at some point, you might find that some decision you made 5 saves back was a very wrong one. Or you are going to want to go through and edit all your code to make it cleaner, but you know this runs the risk of breaking something and don't want to take that risk. Version control can help solve a lot of these problems.

Basics of Git

There is *a lot* of depth one can go into when talking about version control. Git is just one tool to implement version control. Other popular options are SVN and Mercurial. However, Git is what we will be using for this class and we will only scratch the surface.

There are a few terms that will be used throughout the class when talking about assignment deliverables or in videos showing off what Git does. Here are the most important terms and their definitions:

Master >

Master is the main branch of your repository. It should represent your software in its most advanced, complete state. So the most recent commit in Master should always represent the most complete version of your software that you deem to be correct and functional.

Branch >

Often times you need to do work that will either change or temporarily break your software while you make changes. A branch is a separate version of your software project. It often branches off of Main (but can branch off of other branches) and when it does, it basically takes a snapshot of Main when is branches off. Any changes you make will only exist in the branch until you merge them back into main.

Commit >

A commit is sort of like a snapshot of your workspace. Changes are not saved to the repository until they are added and committed. GitHub desktop automates the adding, but be aware if you are using a different tool, adding and committing are two different steps. Every time you make a commit, you will be adding a snapshot in time you can branch off of or roll back to if needed. Committing is probably the most common operation in Git.

Merge >

Merging is the process of taking a branch and applying those changes to the thing it branched off of. Often times this is really straightforward. Git is really smart when it comes to merging. It can tell which lines were changed and where they belong in the file *even if the Master has been changed after you made a branch.* The only time things get messy is if changes were made to the same line of code in both the branch you are merging and the branch you are merging into. In

this case it will ask you which version is right.

Clone ›

Cloning makes a copy of a repository. When the copy is made generally all the branches are cloned as well. You will clone your repository on GitHub to your local computer to do work or you may end up cloning repositories I make to get access to things like automated test suites. The repo you cloned from is referred to as *origin*.

Push ›

Pushing is sort of the opposite of cloning. After you clone you often want to get your changes back to the place you cloned from. To do this you *push* to that repository. Unlike cloning, when you push, you only push a single branch to a single branch. Often you are pushing your current branch to origin's master branch.

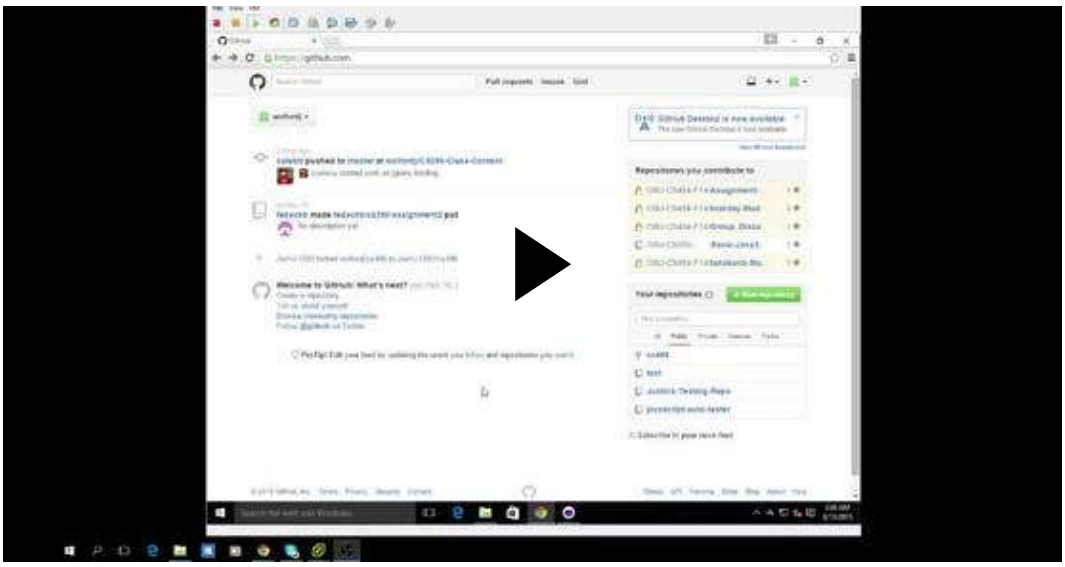
Pull-Request ›

GitHub is a tool that is separate from Git. It is sort of like a wrapper that adds additional features. Its main purpose is to help people work on and share repositories from remote locations. At an office, there is often one repo on the network that everyone can clone and push work to easily. It gets more difficult when everything is not on the same network. So a feature that GitHub created is the pull-request. If I make a change that I think improves software, I can make a pull request. This basically tells the owner of the repository that I want them to pull my changes into their repository. I usually don't use this feature as I am often the only one contributing to my projects, but it is the work-flow that GitHub desktop uses so it is worth mentioning here. Once the repo owner authorizes the pull request, it works like a merge.

Activity ⚡

Get Git Going!

The best way to learn Git is to use Git. The easiest way to get started is with GitHub Desktop. Go to GitHub Desktop (https://desktop.github.com/), download GitHub for desktop for your OS. The following video will go through the steps of making a repository, committing files to it and simple branching.



You should make a repository. Add a file and commit it. Change some files and commit it. Make a branch, do some work on that branch and merge that branch using a pull request. When you are done, when you look at the repo on GitHub you should see all the changes you made in master and all the changes you made in the branch you pulled in.

If you get stuck, first check their getting started guide (https://help.github.com/desktop/guides/). One of their guides deals with getting the software installed. The other talks about the common operations you will be doing with GitHub for desktop. If these don't get your questions answered, ask on the discussion board.

Review

At this point you should be able to clone a repo from GitHub, make some changes and push those changes back to GitHub. If you can do those things, you know all you need to know for this class. That said *a lot* of jobs want you to have version control experience and currently Git is the *in* tool to use for version control. If you can learn more advanced concepts and start using Git on the command line, those will be extremely valuable skills for the future.