Try Git

About
Git allows groups of people to work on the same documents (often code) at the same time, and without stepping on each other's toes. It's a distributed version control system.

Initialize a Git repository `> git init <repo_name>`	**Directory:** A folder used for storing multiple files. **Repository:** A directory where Git has been initialized to start version controlling your files.
Check the Status `> git status`	**The .git directory:** On the left you'll notice a .git directory. It's usually hidden but we're showing it to you for convenience. If you click it you'll notice it has all sorts of directories and files inside it. You'll rarely ever need to do anything inside here but it's the guts of Git, where all the magic happens.
Add to Staging Area `> git add <file>`	**staged:** Files are ready to be committed. **unstaged:** Files with changes that have not been prepared to be committed. **untracked:** Files aren't tracked by Git yet. This usually indicates a newly created file. **deleted:** File has been deleted and is waiting to be removed from Git.
Store Staged Changes `> git commit -m <message>`	**Staging Area:** A place where we can group files together before we "commit" them to Git. **Commit** A "commit" is a snapshot of our repository. This way if we ever need to look back at the changes we've made (or if someone else does), we will see a nice timeline of all changes.
Show Commits Made `> git log`	Use git log --summary to see more information for each commit. You can see where new files were added for the first time or where files were deleted. It's a good overview of what's going on in the project.
Add Remote Repositories `> git remote add <shortname> <url>`	**git remote:** Git doesn't care what you name your remotes, but it's typical to name your main one origin. It's also a good idea for your main repository to be on a remote server like GitHub in case your machine is lost at sea during a transatlantic boat cruise or crushed by three monkey statues during an earthquake.
Push Commits `> git push -u <remote_branch> <local_branch>`	**Cool Stuff:** When you start to get the hang of git you can do some really cool things with hooks when you push. For example, you can upload directly to a webserver whenever you push to your master remote instead of having to upload your site with an ftp client. Check out Customizing Git - Git Hooks for more information.
Pulling Remotely `> git pull <remote_branch> <local_branch>`	**git stash:** Sometimes when you go to pull you may have changes you don't want to commit just yet. One option you have, other than commiting, is to stash the changes. Use the command 'git stash' to stash your changes, and 'git stash apply' to re-apply your changes after your pull.
See Differences `> git diff HEAD`	**HEAD:** The HEAD is a pointer that holds your position within all your different commits. By default HEAD points to your most recent commit, so it can be used as a quick way to reference that commit without having to look up the SHA.
Resetting the Stage `> git reset <file>`	**Commit Etiquette:** You want to try to keep related changes together in separate commits. Using 'git diff' gives you a good overview of changes you have made and lets you add files or directories one at a time and commit them separately.
Undo Changes `> git checkout --<file>`	**The '--':** So you may be wondering, why do I have to use this '--' thing? git checkout seems to work fine without it. It's simply promising the command line that there are no more options after the '--'. This way if you happen to have a branch named octocat.txt, it will still revert the file, instead of switching to the branch of the same name.

Creating a Branch `1 > git branch <branch_name>`	**Branching:** Branches are what naturally happens when you want to work on multiple features at the same time. You wouldn't want to end up with a master branch which has Feature A half done and Feature B half done. Rather you'd separate the code base into two "snapshots" (branches) and work on and commit to them separately. As soon as one was ready, you might merge this branch back into the master branch and push it to the remote server.
Switching Branches `1 > git checkout <branch>`	**All at Once** You can use: git checkout -b new_branch to checkout and create a branch at the same time. This is the same thing as doing: git branch new_branch git checkout new_branch
Remove from Repo `1 > git rm <file>`	**Remove all the things!** Removing one file is great and all, but what if you want to remove an entire folder? You can use the recursive option on git rm: git rm -r folder_of_cats This will recursively remove all folders and files from the given directory.
Merging a Branch `1 > git merge <branch_to_merge>`	**Merge Conflicts:** Merge Conflicts can occur when changes are made to a file at the same time. A lot of people get really scared when a conflict happens, but fear not! They aren't that scary, you just need to decide which code to keep. Merge conflicts are beyond the scope of this course, but if you're interested in reading more, take a look the section of the Pro Git book onhow conflicts are presented.
Delete a Branch `1 > git branch -d <branch_name>`	**Force delete: W**hat if you have been working on a feature branch and you decide you really don't want this feature anymore? You might decide to delete the branch since you're scrapping the idea. You'll notice that git branch -d bad_feature doesn't work. This is because -dwon't let you delete something that hasn't been merged. You can either add the --force (-f) option or use -D which combines -d -f together into one command.