CS 240: Basic Git Commands Transcript

[00:00:00] The next thing we want to learn is how to run the basic commands that we just talked about and get.

[00:00:07] Now we're going to go back to the sample chess project that we already created.

[00:00:12] And as you recall, I created a folder I ran, get in it inside that folder and that initialized the folder to be a G repository and then I copied in a bunch of chess starter files that are provided um on our uh CS 240 GitHub.

Start visual description. The professor demonstrates how to initialize a Git repository by creating a folder, running the git init command inside that folder, and copying in a bunch of chess starter files from the CS 240 GitHub repository. End visual description.

[00:00:31] Now, if I run to get status, well, let's, let's not do that. Let's run a list command first.

[00:00:40] And so these are all the files and folders that, that I have in my project, I run get status.

[00:00:46] Get says all those things are untracked. I don't know what they are.

[00:00:51] Um That just means we've never added or committed those, those files and folders to the, the repository.

[00:01:01] So the next thing I want to do is I want to add, let's go ahead and add one of these files to the repository.

[00:01:07] So in this case, I'm going to run to get a command and let's pick the notes dot MD file. Let's, let's add that one to the repository.

Start visual description. The professor demonstrates how to add a file to the Git repository by running the git add command and selecting the notes.md file. End visual description.

[00:01:19] So, once I've added, oh, no, that's the wrong name.

[00:01:25] Let's add the notes dot MD file to the repository.

[00:01:31] Now, if we run, get status, you'll notice that.

[00:01:36] Get now says, oh, I know what notes dot MD is. That's a file that you've added to the repository.

[00:01:42] We haven't committed it yet, but we have added it. We staged it, it still doesn't know what the other files are, but we've, we've um we've staged the notes dot MD file.

[00:01:53] Next thing I'm going to do is I'm going to run the get commit command.

[00:01:57] Now, when you run the get commit command, you need to also specify what's called a commit message.

[00:02:03] And so I'm going to use the dash M option that stands for message.

Start visual description. The professor demonstrates how to commit changes to the Git repository by running the git commit command with a commit message describing the changes made to the files. End visual description.

[00:02:07] And so what I want to do is I want to type in a string that describes the changes that have been made to the files.

[00:02:13] So usually when you commit a bunch of changes to the repository, there's some theme um there that those changes are somehow related to each other, maybe

you're adding a new feature to your program and all the changes that you've made since you last committed are related to implementing that new feature.

[00:02:30] And so in this case, you might say um for chess, we might say we implemented move algorithm four king and queen pieces and having typed in the commit message, I just have need to hit, enter and that has now committed the notes that MD file to, to the repository.

[00:02:56] So if I run and get status again, um, it still lists all those untracked files, but you'll notice that notes dot MD is not in that list.

[00:03:08] And it's basically telling me here that you haven't made any changes to the files we know about since the last time that you committed.

[00:03:14] Now, let's go to Intel, which we opened earlier and let's actually make a change to the notes dot MD file.

[00:03:23] Let's just type in something here. This is my first note and save it.

[00:03:31] So now that I've changed that file, now, I can go back and run and get status again and you can see that G is telling me, oh, you've modified the notes dot MD file that's part of the repository and you've modified it since the last time you committed and these other files, we still don't know what they are.

[00:03:50] And so at that point, I can go ahead and just do another commit if I want to.

[00:03:57] Now, rather than do that, what I'm going to do next is I'm going to take all these things that are not in the repository yet.

[00:04:03] And I'm, I'm going to go ahead and add those to the repository. So, I'm going to stage those.

[00:04:09] Actually, I'm going to stage all the changes that I've, I've made.

[00:04:12] So I'm going to run the, the G A and I'm just going to put a dot and the dot Basically means add everything in this folder and all the sub folders below it.

[00:04:24] And so you can see there that now that I've added all those files to my repository and that get now knows what they are. So, they're all green.

[00:04:36] And so what it's saying here is changes to be committed.

[00:04:39] These are, these are all the files that have been staged and they'll be included in the next commit.

[00:04:45] And most of these are new files because these have been added for the first time.

[00:04:48] You'll notice that the notes do MD file is listed as modified because it's already part of the repository, but it's been changed.

[00:04:55] And so having added those to the, to the index or stage, I can go ahead and commit all those changes.

[00:05:11] So we're importing a bunch of files that we were given.

[00:05:14] So I'm just going to call that my initial chess commit and run that.

[00:05:21] And so all those files have now been added to the repository.

[00:05:24] And so now if I get status, get saying, hey, I don't see anything that's changed.

[00:05:30] We're aware of all the files that are here.

[00:05:31] You haven't changed any of them since you last committed. And so, um it's clean.

[00:05:37] So that just shows you the basic use of the, the get a command for staging files and then the get commit uh command for actually committing changes to the repository.

[00:05:54] Now, having done some commits, there's another uh command that we're, we're ready to use the next command we want to learn is to get log command.

[00:06:03] So what the GG log command does is it shows you uh a commit history of your project.

Start visual description. The professor demonstrates how to use the git log command to view the commit history of the project, showing a log of all the commits made to the repository. End visual description.

[00:06:09] So it shows you a log of all the, the commits that you've made to the repository. And so, in this case, um I've only made two commits here and it shows them in reverse order. So, the most recent ones at the top, so you can see here that the last commit I made was uh made at this day and time, this was the commit message.

[00:06:28] Um Here's the person who made the commit.

[00:06:30] And you can see the previous commit as well. You see the message for that.

[00:06:35] So just imagine overtime in a project, you're going to make hundreds and thousands of commits.

[00:06:41] Um You won't get that many commits in CS 240.

[00:06:43] But in a real project, there will be thousands of commits uh that have been made.

[00:06:48] And so a lot of times it's useful to go back and and see the history um of those commits.

[00:06:52] Now, one thing you'll notice here is that each of the commits has a unique identifier.

[00:06:58] So this is the, the commit id.

[00:07:01] And so it looks just like a random string of hexadecimal digits, which it is, and that's the complete identifier for that commitment.

[00:07:11] And so when you run a lot of git commands, you have to specify the id of the commit that you're trying to operate on with the command.

[00:07:18] Now, usually when you run those commands, you can just type in the first few digits of the G or of the commit id because if you can just give me the first few digits, that's probably unique enough that no other commit in the system is going to have a conflict with that. And so, um you never have to type in the, the full commit id, usually just the first few digits is good, but just be aware that every commit has a unique identifier and you'll, you'll frequently need to, to know what those I DS are.

[00:07:44] And so the get log command can be useful for, for finding that out.

[00:07:54] All right. So having, um, looked at our commit log, the next thing I want to do is demonstrate something I mentioned before, which is what if, now that I've committed all my files to the repository. What if I go back? And if I, what, what if I delete all my files? Well, it's maybe I should be careful with that.

[00:08:21] So let's, let's plead everything except the doc git folder.

[00:08:27] If you, if you delete the dot Get folder, you're going to be in trouble.

[00:08:31] But let's say I deleted everything else except the dot Get folder.

[00:08:39] So now if I do get check out and then I do a listing again, you'll see that it, that it has magically restored all of my files and folders to my project.

Start visual description. The professor demonstrates how to restore deleted files from the Git repository by running the git checkout command, which retrieves all

files and folders to the project as they were at the time of the specified commit. End visual description.

[00:08:56] So this is part of the power of GIT, is that at any point you can, you can retrieve your files, and you can go back to any previous version that you, you want. So, if I look at my get log just by typing in the commit ID, I could tell GIT to go back to my project files the way they looked when I did that commit.

[00:09:19] And that's how you would, um, go back in time.

[00:09:22] You, you could run and get checkout command, and you could specify the idea of the commit that you want to see and it'll, it'll revert your, your project files back to what they looked like at that point in time.