14.4.3 Using a Git Repository

Click one of the buttons to take you to that part of the video.

Using a Git Repository 0:00-0:43

In this demo we'll continue to explore using git.

For this demo, I've created a remote repository on Bitbucket. You can see that the repository contains six files. One is the README.md file. One is a special file named .gitignore, and this .gitignore file contains the names of files that we want git to ignore when it's doing our source control.

And there four other working files that we created, file1, file2, file3, file4.

If we click ‘Clone', up here in the corner, you can see that it will provide us the path to this repository that's on Bitbucket, that we can use to clone the repository down to our local machine.

Clone a Repository 0:32-2:23

So, I'll go ahead and close that.

Now that I've switched back to my Linux machine, I'm currently in my projects folder. So, if I run ‘git clone' from here, it will actually clone the repo that I created to this location, into the projects folder.

The clone command tells it which project on Bitbucket it took the pull from at Bitbucket.org, and then the name of the repo that we want to clone is the ‘my_git_repo.git'.

We need to enter the root password. And now that we've entered the root password the project has been cloned down.

If we do an ls command here, you can see that there's my_git_repo and my_local_repo, that we created in the last demo. If we CD into my_git_repo and we do an ‘ls -a' there, you can see that the files that we saw on the git repo are there, the files 1, 2, 3, 4. The .gitignore, and the README.md file are there. And in addition to that the .git folder which has all of the configuration for the repository is here also. They've all been created when we did the clone. We didn't have to do a separate git init.

I'm going to CD into the .git folder and if we do an ‘ls -a' there, we see that there several files and directories that make up the index and the local repo.

---

gitignore 2:24-2:45

Now let's go back to the main folder and look at the .gitignore file.

I'm going to do a cat on the .gitignore file, and you'll see that the only thing that in that file is *.txt, which means that any file, which is a .txt extension on it will be ignored by git when we try to do an add or a commit.

Configure the Local Repo 2:46-4:15

Next, we'll want to configure some settings for the local repo. If you add a username and email address, these are added to your commit so that others can see who committed the files. We'll say it's user Bob. We can also do a user email and we'll call that bob@testout.com.

One other thing that is a handy configuration to make, is to configure your preferred editor. So, we'll go ahead and call this the core.editor, and we want our core editor to be nano. It's a little easier to use than VI, so I want to use nano.

Okay, now that we've made those settings, we want to see what settings are set up for this local repo we can use the ‘git config -l' command. And you can see the information that we added, the username is Bob, the email for Bob@TestOut.com, the editor for the nano's. So, a lot of the settings that we saw in the local repo are here. Note that the information about the remote repository was added to the configuration automatically, right here, when the repo was cloned.

I'm going to go ahead and clear the screen again.

Using git 4:16-5:58

Now that we have a repo cloned and configured, let's do some work.

Before doing any work, it's always a good practice to do a git pull to make sure that your local repository has the latest version of the files from all of your coworkers on the projects. So, we'll do a ‘git pull' and enter the password, our Bitbucket password, and it tells us that we're already ‘up to date'.

So, let's take a look at the folder again. We see our file 1, 2, 3, 4 and our README.md that was there, the .gitignore file is hidden.

Let's do some work and we'll create some new files, newfile1…2, newfile3. And let's create one called settings.txt, so that I can show you what the .gitignore file does.

Now, if we do a ‘git status' at this point, you should see that the ‘newfiles' that we added are showing as needed to be added to the index but notice it doesn't tell you anything about settings.txt because that file is being ignored.

So, we can go ahead and add these files. We'll do a ‘git add', and we'll tell it to add all of them at once, and it does warn us that this file called settings.txt is being ignored and not being seen.

So, we went ahead and added those, and if we do the ‘git status' again you'll see that the files are now green, they've been added to the index, and they're ready to be committed.

Just to show you some of the other commands we can also do ‘git mv' and we can change the name of a file if we want, we can call file1, we'll call it file5.

---

git mv and git rm 5:47-6:18

We can also remove some files if we want, with ‘git rm' and we'll go ahead and remove file3. If we do ‘git status' again, you'll see that the rename files and the move files are also showing in the index as things that need to be added to the commit.

---

git commit 6:19-6:44

So, with all of our file changes added to the index we can commit the changes to our local repo and to do that we just do our ‘git commit'. Once again, it opens us up. But this time it opens up a nano, so we'll give it a commit message of "Adding our new files", and then we do a Ctrl + o to save that, and Ctrl + x to exit.

And then it tells us that the commit is completed.

---

git push 6:45-7:35

With the changes now in the local repo, we can use ‘git push' to push the changes up to the remote git repository on Bitbucket.

To do that, we just do a simple ‘git push', add our password, and the push is made.

Now I'm going to change back to my Bitbucket page, and I'll do a refresh on Bitbucket, and all of those changes that we made to the files… file2, file4, file5, file3 is gone, newfile 1, 2, 3 are reflected now in our remote repository.

So, the changes that we made to our working directory, and then we added them to the index, committed to the local repository, and then we pushed them to the remote repository, those changes are all now reflected in the remote repository.

Branching 7:36-10:11

All right, there's one other great feature of git, and that's the ability to create branch…I'm going to go ahead and clear the screen again, so can get back to the top…a branch allows you to experiment with changes without having to add them to the main repo files until you're ready to do so. So, you can create all sorts of experimental things, play with them, test them, and they don't affect the main repository until you decide to do so.

I'm going to create a branch called test and we'll need to switch to that branch. The command to do that is ‘checkout test'. So, we'll switch that branch.

We can do an ls here and you'll see because we branched, all of the files that were in the main branch are still in this branch tests but anything that we add is not going to go into the master branch here.

So, let's go ahead and add a few files just so that we can see how this happens. I'm going to create a file called branch1 and a file called branch2. You can see that we've added those files. We'll do a ‘git status', which tells us that we have two files that need to be committed, so we'll go ahead and add those files and commit them. And it warns us, once again, that we got a file with a .txt extension that's being ignored.

And then let's go ahead and do the ‘git commit'.

Once again, "Adding branch files". I'll do Ctrl + o, Ctrl + x, and it went ahead and committed those to the local repo.

Now let's switch back to the master branch, we're going to go ahead and check out the master branch, ‘git checkout master'.

I'm going to go ahead and clear the screen so we can get to the top again.

If we do an ls on the repository now, you'll see that the branch files that we had added are no longer there. That's because there in the branch, so let's suppose that we decide the files we created in the branch are worthy of the project and that we want to have them into the master branch.

So, to do that we can merge them using ‘git merge' so the ‘git merged' command is used to bring those files that we had in the test branch back into the master branch.

Let's go ahead and run that. It tells us that we brought in two files branch1 and branch2 and, if we do an ls now on the repo, you'll see that the branch1 and the branch2 files are showing up in the repo.

---

git log 10:12-10:39

One final command that may come in handy if you need to restore a commit is the ‘git log' command.

We'll run the ‘git log' command.

Here, you see all of the various commit's that have been made to the project.

Now this is also a VI file, and so if you want to get out of this, you need to scroll down a little, scroll down to the end of the file, ESC, and then once again, colon q, and it gets you out.

Summary 10:40-10:54

That's it for this demo.

In this demo we showed you how to clone and a configure repository. We added, moved, and remove files to the staging index. We committed and pushed changes. And we branched the repository and then merged the branch back into the master branch.
