How To Setup Your Own Computer

Basic Git workflows

When you set out to accomplish something *that you have done before*; whether you are aware of it or not, you have workflows that you follow to get that thing done. Developers (which now includes you) are the same. When they need to get something done, they have workflows that they follow. While these workflows may be new to you, they will quickly become your own.

Cloning a repository

When you want to work with someone else's code, you first have to get it on your machine. We do this with the `git clone` command. For example, when you were preparing your machine for this *Python Fundamentals* module, we instructed you to clone (download) our `dne-dna-code` or `dne-security-code` or `dciv2-code` sample code repository to your machine. See the "How To Setup Your Own Computer" link if you have not completed the setup.

> **Note:** You should have already cloned the sample code to your computer. Please don't do it more than once. Having multiple copies of the code samples on your computer can cause confusion as to which one you are working with.

When you clone a repository, files are copied to a folder in the location where you run the command.

```
$ git clone https://github.com/CiscoDevNet/dne-dna-code
```

or

```
$ git clone https://github.com/CiscoDevNet/dne-security-code
```

or

```
$ git clone https://github.com/CiscoDevNet/dciv2-code
```

This example shows the resulting output in your terminal window:

```
Cloning into 'dne-dna-code'...
remote: Counting objects: 274, done.
remote: Compressing objects: 100% (115/115), done.
remote: Total 274 (delta 145), reused 266 (delta 139), pack-reused 0
Receiving objects: 100% (274/274), 1.12 MiB | 3.41 MiB/s, done.
Resolving deltas: 100% (145/145), done.
```

When you ran this command, the git client on your laptop connected to the URL provided and cloned a full copy of the repository to your machine. By the way, when we say "clone," we mean it. When you clone a repo you are indeed getting a complete copy of the repository and all of its contents. From the first commit to the last, you now have locally on your

(Note: if you do this in a repo with many commits, you can press `space` to page through the commits and `q` to quit).

Now that you have our repo on your computer, let's get it ready for you to edit and commit your changes. Change to the `dne-dna-code` or `dne-security-code` or `dciv2-code` repository:

```
$ cd dne-dna-code
```

or

```
$ cd dne-security-code
```

or

```
$ cd dciv2-code
```

Prepare the repo for your changes

When you first clone a repository, you should be viewing and working on the "master" branch, which just by convention of naming is the default branch for a repository.

You can verify what branch you are working on, and the current status of your working tree with the `git status` command:

Run this on your workstation:

```
$ git status
```

Click for Sample Output

If you started editing files on the master branch and committing your changes, your local repository would immediately be out of sync with the remote server, which isn't a problem... until it is. If someone else commits some changes to the master branch, and pushes their changes to the server before you push yours... You will have to merge (reconcile) their changes with your own before you can push your changes to the server. In fact, you cannot even get updates from the server on this branch until you reconcile the discrepancy. *Sigh.*

Wait... I thought this whole version control things was supposed to make collaborating on files easier!?! It does, but you need to create a branch!

Run this on your workstation:

```
$ git checkout -b mycode
```

Click for Expected Output

Use the `git checkout` command to switch between branches. Adding the `-b` option to the checkout command causes Git to create a new branch – and then switch to it.

When you create a branch on your local repository, you are creating a safe place for you to make changes to the repo. Any changes you make and commit are committed locally to this new branch. Since you aren't making any commits to the branches that are synced with the remote server, any updates made to these other branches ("master" or otherwise) on the remote server, can easily be pulled down to your computer.

Keeping your local repository up-to-date

Over time, your local Git repository will get out of sync with the remote repository. As people push commits to the server, you

command.

> **Note:** If you recently cloned your repo, there might not have been any updates to the remote for you to retrieve.

Run this on your workstation:

```
$ git fetch
```

Sample Output

Now that you have cloned our repository, created a local branch for your edits, and know how to keep your local repo in sync with our remote, you are ready to make and commit your changes!

Making changes

If you make any changes to a file under git's version control, git will know.

On your workstation, open the `intro-python/git-basics/change_me.txt` file on your computer and make a change - add, edit or delete anything you like. Then run the following commands:

```
$ git status
```

Expected Output

```
$ git diff
```

Sample Output

From `git status` output you can see that Git detected the modification to the `change_me.txt` file, and from the `git diff` output you can even see the what changed in the file (in my example I added " Let's see if I can sneak this change in..." to the end of the file).

Reverting changes

Sometimes you make changes and then decide that you simply want to go back to some past commit. Here are the simple workflows that can help you back out changes:

Need: *I made changes to a file, and now I want to revert those changes and get the original (last committed version) of the file back.*

Solution: You can " checkout" the last version of the file to overwrite the changes you have made and restore the file to its last committed version.

```
$ git checkout <file path>
```

Need: *I have made changes to several files, and I want to revert **all** of those changes and get back to my " last known good state" (last commit).*

Solution: You can " reset" your working directory to the last commit. **Note: You will lose all the changes you have made since your last commit.**

```
$ git reset --hard
```

Need: *I created a branch to experiment with some changes, and now I just want to throw the whole thing out.*


```
$ git branch --delete --force <branch name>
```

> **Note:** Always be careful when you see options like `--hard` or `--force`. These should be keywords that cause you to pause and think about what you are doing as you will lose some work when you run these commands. If that is your intention, proceed. If not, think twice (or three times) before running these commands.

Committing your changes

While the change we made to a trivial text file is, well trivial, later you will edit code to make it functional. When you get your code working *(insert awesome feeling of accomplishment)*, you are going to want to save that accomplishment and lock it away indelibly in the repository. To do this, you will "commit" your changes to the repository.

Commit authors and messages

When you make a commit to a Git repository, Git automatically includes (in the indelibly hashed and stored commit) the name and e-mail address of the person that made the commit and a non-optional commit "message." When done right (which as you would expect doesn't happen all the time), commit messages can serve as descriptive change logs that record the history of the changes in the project.

Telling git who you are *(one-time setup)*

You need to tell Git who you are before you can commit any changes to a repository.

Run this on your workstation, inserting your own name and e-mail address:

```
$ git config --global user.name "Your Name"
$ git config --global user.email your@email-address.com
```

Making a commit

Committing your changes to a repository is a two-step process:

1. **Add**: Stage files to be added to the commit with `git add`.
2. **Commit**: Commit the files to the repository with a commit message `git commit`.

Try this out by running the following commands on your workstation:

Feel free to make your own commit message if you think mine is lame.

```
$ git add intro-python/git-basics/change_me.txt
$ git commit -m "Git Commit - Done."
```

The `-m` option lets you supply a short commit message right there on the command line. Without the `-m` option, Git would open your environment's default text editor (which on many systems is `vim`) so that you can enter a detailed and potentially lengthy commit message.

computer. Committing often means that you are capturing more of the history of your code. You can go back and view commits and past changes anytime.

Yeah! My code (or some portion of it) is working!! **Commit.**

Next step: The DevNet sample-code workflow

(/learning/tracks/devnet-beginner/programming-fundamentals/git-basic-workflows/step/2)

(/learning/tracks/devnet-beginner/programming-fundamentals/git-basic-workflows/step/4)

3 / 5