Git 100: Basics of the git version control system

https://developer.cisco.com/learning/lab/git-intro/step/1

Introduction

This Learning Lab provides you with the fundamentals of the distributed version control system.

Prerequisites

You need a Git client on your workstation. If you are a DevNet event, such as at CiscoLive!, the workstations are pre-installed with Git, and you must open the Git shell. Find the icon on the taskbar that looks like:

For Windows users there are two ways to interact with Git on the command prompt:

- `git bash` is a UNIX-style command prompt allowing you to interact with the Windows file system.
- `git cmd` is a Windows-style command prompt allowing you to interact with the Windows files system.

Throughout the tutorial we assume that you're either working on a UNIX system or using `git bash` on Windows.

What is version control vs. distributed version control?

Unlike version control systems like Subversion, Git is a **distributed** version control system. What this means is that Git is really great for sharing code with many individuals while enabling you to keep your changes synchronized, properly versioned, and with a complete replica of the commits and repo content.

Note: GitHub is not the same as Git. GitHub is a company and a service that has popularized the Git protocol, and provides a centralized community for discovering code. As a small bit of trivia, Git was largely created by Linus Torvalds, the founder of the Linux kernel.

Getting started with `git config`

So that you can get credit (or blame) for the work that you've contributed, you need to associate your name and email address with your work. To do that in Git, you use the git config command.

```
git config --global user.name "Your Name Comes Here"git config --global user.email you@your domain.example.com
```

Now you're all set to start contributing your code!

Next: Initialize your repository

Initialize your repository

1. On your workstation, make a directory in ~/src called git-intro.
   You should now have a directory structure that looks like ~/src/git-intro. (on Windows C:\Users\<username>\src\git-intro).
2. Change directory into git-intro.
3. Next, you will initialize your repository (used interchangeably with "repo") with the command git init.

   You should see a message that resembles:

   Initialized empty Git repository in <working directory>/.git/

You have created a local repo within your project contained in the hidden directory .git. This is where all of your change history is located on your local workstation.

Checking repository status

As you work on your project, you want to check which files have changed. This is helpful when you are committing files to the repo, and you may only want to commit a few of them.

While in the git-intro directory, type the command git status.

You should see the following message:

```
On branch main

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

This message tells you a few things:

- That you are on branch main. (We'll discuss branches later.)
- That the last commit message was, Initial commit.
- That there is nothing changed to commit.

You will see that the status of your repo will change once we add files and start making changes.

Next: Adding files

Adding files

We have an initialized, but empty repository.

Create a file

1. In our source directory (`git-intro`), create a file `first.txt`.
2. Open `first.txt` in a text editor, and add the message:

   `Our best thoughts came from others.`
3. Save these changes.
4. Type `git status` again. You should see a message like this:

```
On branch main

Initial commit

Untracked files:
(use "git add <file>..." to include in what will be committed)
first.txt
nothing added to commit but untracked files present (use "git add" to track)
```

As you can see, Git recognizes that there is a new file in the directory, and it knows that it's not being tracked.

Commit the changes

Next, we need to add `first.txt` to "stage" your work, and then "commit" your work. An important concept in Git is the "stage". The "stage" is an intermediate phase where you can assemble your changes before you then "commit" your work. This also allows you (if you like) to stage small modifications to the same file as you work on it.

To store the staged changes in your Git repo, you need to commit them.

1. As you can see in the status, you need to execute `git add <file>` to stage your work. Do that, and then execute `git status` again. You should see the following:

```
On branch main

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

        new file. first.txt
```

You'll notice that there are now "staged" changes in the form of `new file. first.txt`

10. Commit the changes by executing the following command:

```
git commit -m "Add inspirational quote"
```

The `-m "<message>"` adds a description of your changes in a human-readable form. As you gain experience with Git, you'll find a lot of information about writing excellent commit messages. For now, we'll just keep it simple and move ahead.

12. Type `git status` again. You will see Git reply with:

```
[main (root-commit) 7be53cc] Add inspirational quote
 1 file changed, 1 insertion(+)
 create mode 100644 first.txt
```

Note that the number and letter combination `7be53cc` contained in `[main (root-commit) 7be53cc]`. Yours will be different than the ones in this tutorial. Git uses a SHA1 hash to file away your changes. When someone asks you, "What's your commit hash?", this is the number they are asking for!

Next: Looking at your commit history

Viewing your commit history

In some cases, you may want to review the commit history for a repo. To view the most recent commits, you use the command `git log`. Try that now.

You should see something like:

```
commit 7be53cc330db1207c9b26fe560704f90405742fd
Author: Ashley Roach <myemail@email.com>
```

```
Date:   Sun Dec 20 07:48:29 2015 -0700

    Add inspirational quote
```

Short hash vs. full hash

In our previous step, we looked at the commit has that Git presented to us: 7be53cc. You should notice that 7be53cc is the first 7 characters of the full commit hash above: 7be53cc330db1207c9b26fe560704f90405742fd.

Making changes

That may seem like a lot of work just to make a commit, but in time it will become routine for you. Additionally, there are many Git GUI clients out there that can automate many of these steps.

1. Moving forward, let's make a change to first.txt. In your text editor, add a new line to your file and save the file. Add the text: -Letters and Social Aims. Quotation and Originality. Ralph Waldo Emreson.

```
The best preparation for tomorrow is doing your best today.
-Letters and Social Aims. Quotation and Originality. Ralph Waldo Emreson
```

4. Now when you perform git status you will see that there are changes in your file.

```
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   first.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

13. You need to stage your file again, by executing git add, followed by another git commit:

```
$ git add first.txt
$ git commit -m "Add quote attribution"
```

```
16. $ git log
```

Note: Since you have only tracked files that you modified, you can do this in one step with `git commit -a -m <message>`.
Your `git log` should resemble the following example:

```
commit e36e13f20c4596ecd7d2a418a35295850146ae5c
Author: Ashley Roach <myemail@email.com>
Date:   Sun Dec 20 08:23:05 2015 -0700

  Add quote attribution

commit 7be53cc330db1207c9b26fe560704f90405742fd
Author: Ashley Roach <myemail@email.com>
Date:   Sun Dec 20 07:48:29 2015 -0700

  Add inspirational quote
```

Now you have two commits.

Next:

Oops! I made a mistake!

If you made a simple mistake in your file, the best option is to fix it and create a new commit. You may have noticed that Emerson was misspelled in our previous commit as Emreson. Fix that using your text editor, then execute:

```
$ git add first.txt
$ git commit -m "Fix typo"
```

Viewing the differences between two commits

Now that you have a few check-ins, you can review changes among commits. You can do that with the command `git diff <commit> <commit>`. Use the `git log` command to find two commits to compare.

```
$ git diff 7be53cc d553bb
diff --git a/first.txt b/first.txt
index d5ff131..86a582c 100644
--- a/first.txt
+++ b/first.txt
@@ -1 +1,2 @@
 The best preparation for tomorrow is doing your best today.
+-Letters and Social Aims. Quotation and Originality. Ralph Waldo Emerson
```

Summary

Good job! You have learned some of the fundamentals of git, including:

- Configuring git
- Checking file status
- Making commits
- Determine differences between files