Hyperiondev

TASK

Additional Reading - Git Basics

Visit our website

Introduction

Here we delve into using Git and discuss the basic commands you will need in order to use this tool. We will explore how to set up a repository for a new or existing project, use common Git commands, commit a modified file, and view your project's history.

THE GIT VERSION CONTROL SYSTEM

In this bootcamp, we will be using the Git Version Control System. Git is the most widely used modern Version Control System. It is free and open-source and is designed to handle everything from small to very large projects.

Git has a distributed architecture and is an example of a Distributed Version Control System (DVCS). This means that with Git, every developer's working copy of the code is also a repository that contains the full history of all changes instead of having only one single place for the full version history of the project.

As well as being distributed, Git has been designed with performance, security and flexibility in mind.

INSTALLING GIT

Git was installed along with other key packages for you at the start of the bootcamp; however, you may need to configure your Git username and email if you have not already done so. If you already have a GitHub account, use the same email address to configure Git on your local machine.

1. Verify that the installation was successful by typing the following into the terminal:

```
git --version
```

2. Configure your Git username and email using the following commands:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@email.com"
```

Before we dive into actually using Git, we need to understand a few important concepts.

INITIALISING A REPOSITORY

To create a new repository, you have to initialise it using the `init` command. To do this, open your terminal (or command prompt) and go to your project's directory. To change your current directory, you use the `cd` (change directory) command followed by the pathname of the directory you wish to access.

After you have navigated to your project's directory, enter the following command:

```
git init
```

This creates a new, hidden subdirectory called **.git** in your project directory. This is where Git stores necessary repository files, such as its database and configuration information so that you can track your project.

ADDING A NEW FILE TO THE REPOSITORY

Now that your repository has been cloned or initialised, you can add new files to your project using the `git add` command.

Assume that you have set up a project at `/Users/user/your_repository` and that you have created a new file called **newFile.js**. To add newFile.js to the repository staging area, you would need to enter the following into your terminal or command prompt:

```
cd /Users/user/your_repository
git add newFile.js
```

CHECKING THE STATUS OF YOUR FILES

Files can either exist in a tracked state or in an untracked state in your working directory. Tracked files are files that were in the last snapshot, while untracked files are any files in your working directory that were not in your last snapshot and are not currently in the staging area. We use the `git status` command to determine which files are in which state.

Using the `git add` command begins tracking a new file. If you run the `git status` command after you have added **newFile.js**, you should see the following code, showing that **newFile.js** is now tracked:

```
git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   newFile.js
```

You can tell that **newFile.js** is staged because it is under the "Changes to be committed" heading.

COMMITTING YOUR CHANGES

You should now be ready to commit your staged snapshot to the project history using the `commit` command. If you have edited any files and have not run `git add` on them, they will not go into the commit. To commit your changes, enter the following:

```
git commit -m "added new file newFile.js"
```

The message after the `-m` flag inside the quotation marks is known as a commit message. Every commit needs a meaningful commit message. This makes it easier for other people who might be working on the project (or even for yourself later on) to understand what modifications you have made. Your commit message should be short and descriptive, and you should write one for every commit you make.

VIEWING THE CHANGE HISTORY

Git saves every commit that is ever made in the course of your project. To see your repository or change history over time, you need to use the `git log` command. Running the `git log` command shows you a list of changes in reverse chronological order, meaning that the most recent commit will be shown first. The `git log` command displays the commit hash (which is a long string of letters and numbers that serves as a unique ID for that particular commit), the author's name and email, the date written and the commit message.

Below is an example of what you might see if you run git log:

```
git log
commit a9ca2c9f4e1e0061075aa47cbb97201a43b0f66f
```

```
Author: HyperionDev Student <hyperiondevstudent@gmail.com>
Date: Mon Sep 8 6:49:17 2017 +0200

Initial commit.
```

There are a large number and variety of options to the **`git log`** command that enable you to customise or filter what you would like to see. One extremely useful option is **`--pretty,`** which changes the format of the log output. The **`oneline`** option is one of the prebuilt options available for you to use in conjunction with **`--pretty`**. This option displays the commit hash and message on a single line. This is particularly useful if you have many commits.

Below is an example of what you might see if you run **`git log --pretty=oneline`**:

```
git log --pretty=oneline
A9ca2c9f4e1e0061075aa47cbb97201a43b0f66f  Initial commit.
```

For the full set of options, you can run **`git help log`** from your terminal or command prompt or take a look at the reference documentation.

PRACTICE

Feel free to refer to the **Git cheat sheet** as needed for this or any future tasks in which you use Git.

Practice creating a Git repository using these steps

- Create an empty folder called task1_project.
- Open your terminal or command prompt, and then change directory (**`cd`**) to your newly created folder.
- Enter the **`git init`** command to Initialise your new repository.
- Enter the **`git status`** command and make a note of what you see. You should have a clean working directory.
- Create a new file in the task1_project folder called **helloWorld.js** and write a program that prints out the message "Hello World!"
- Run the **`git status`** command again. You should now see that your **helloWorld.js** file is untracked.
- Enter the **`git add`** command followed by **`helloWorld.js`** to start tracking your new file.

- Once again, run the `git status` command. You should now see that your **helloWorld.js** file is tracked and staged to be committed
- Now that it is tracked change the message printed out by the program in the file **helloWorld.js** to “Git is Awesome!”.
- Run `git status` again. You should see that **helloWorld.js** appears under a section called “Changes not staged for commit”. This means that the file is tracked but has been modified and not yet staged.
- To stage your file, simply run `git add` again.
- If you run `git status` again you should see that it is once again staged for your next commit.
- You can now commit your changes by running the `git commit -m “[insert commit message]”` command. Remember to enter a suitable commit message after the `-m` switch.
- Running the `git status` command should show a clean working directory once again.
- Now run the `git log` command. You should see your commit listed.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we’ve done a good job?

Click here to share your thoughts anonymously.