TASK

Version Control I: Introduction to Version Control and Git

Visit our website

Introduction

WELCOME TO THE INTRODUCTION TO VERSION CONTROL AND GIT TASK!

Knowing how to use version control is a crucial skill for any Software Developer working on a project, especially when working in a team of developers. The source code of a project is an extremely precious asset and must be protected. Version control software tracks all changes to the code in a special kind of database. Therefore, if a developer makes a mistake, they can compare earlier versions of the code to the current version to help fix the mistake while minimising disruption to the rest of the team. This task will introduce you to the basics of version control. It focuses on the Git version control system and the collaboration platform, GitHub.

A note from the

HyperionDev Team

A version control system is one of the most important tools for any software developer! Check out this **HyperionDev blog post** to see the other 4 tools essential for all developers.

WHAT IS A VERSION CONTROL SYSTEM?

Version control systems record modifications to a file or set of files so that you can recall specific versions of it later on. A version control system can be thought of as a kind of database. You are able to save a snapshot of your complete project at any time. Then, when you take a look at an older snapshot (or version) later on, your version control system shows you exactly how it differs from the current one.

Version control is independent of the kind of project, technology, or framework you are working with. For example, it works just as well for an Android app as it does for an HTML website. It is also indifferent to the tools you work with. You can use it with any kind of text editor, graphics program, file manager, etc.

WHY DO YOU NEED A VERSION CONTROL SYSTEM?

Below are some of the benefits of using a version control system for your projects:

- **Collaboration:** when working on a large (or even medium-sized) project, more often than not you will find yourself working as part of a team of developers. Therefore, you will have multiple people who need to work on the same file. Without a version control system in place, you will probably have to work together in a shared folder on the same set of files. It is therefore extremely difficult to know when someone is currently working on a file and, sooner or later, someone will probably overwrite someone else's changes.

  By using a version control system, everybody on the team is able to work on any file at any time. The version control system then allows you to merge your changes into a common version, so the latest version of the project is stored in a common, central place.

- **Storing versions:** it is especially important to save a version of your project after making any modifications or changes. This can become quite confusing and tedious if you do not have a version control system in place. A version control system acknowledges that there is only one project being worked on, therefore, there is only one version on the disk you are currently working on. All previous versions are neatly stored inside the version control system. When you need to look at a previous version, you can request it at any time.

- **Restoring previous versions:** being able to restore older versions of a file enables you to easily fix any mistakes you might have made. Should you wish to undo any changes, you can simply restore your project to a previous version.

- **Understanding what happened:** your version control system requires you to provide a short description of the changes you have made every time you decide to save a new version of the project. It also allows you to see exactly what was changed in a file's content. This helps you understand the modifications that were made in each version of the project, even if you weren't the one who made them.

- **Backup:** a version control system can also act as a backup. Every member of the team has a complete version of the project on their disk. This includes the project's complete history. If your central server breaks down and your

backup drive fails, you can recover your project by simply using a team member's local repositories.

THE GIT VERSION CONTROL SYSTEM

In this course, 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

Before you start learning how to use Git, you must install it. Even if you already have it installed, you should ensure that you update it to the latest version. Below are the instructions on how to install Git on Windows, Mac and Ubuntu:

Installing Git on Windows

1. Go to **http://git-scm.com/download/win** to download the official build from the Git website. The download should start automatically.
2. After starting the installer, you should see the Git Setup wizard screen. Click on the Next and Finish prompts to complete the installation.
3. Open a Command Prompt.
4. 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"
```

Installing Git on Mac

1. Download the latest **Git for Mac installer**
2. Follow the prompts to install Git.
3. Open a terminal.

4. Verify that the installation was successful by typing the following command into the terminal:
   ```
   git --version
   ```
5. 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"
   ```

Installing Git on Ubuntu

1. Install Git by typing the following commands into your terminal:
   ```
   sudo apt-get update
   sudo apt-get install git
   ```
2. Verify that the installation was successful by typing the following into the terminal:
   ```
   git --version
   ```
3. 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"
   ```

Extra resource

If you're using a different Linux/Unix distribution, you can use **Git-SCM** to download Git using the native package manager on numerous platforms.

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

REPOSITORIES

A repository can be thought of as a kind of database where your version control system stores all the files for a particular project. A repository in Git is a hidden folder called '**.git**', which is located in the root directory of your project. Fortunately, you do not have to modify anything in this folder. Simply knowing that it exists is good enough for now.

There are two types of repositories, namely, local repositories and remote repositories. A local repository is located on your local computer as the '**.git**' folder inside the project's root folder. You are the only person that can work with this

repository. A remote repository, however, is located on a remote server on the internet or in your local network. Teams of developers use remote repositories to share and exchange data. These serve as a common base where everybody can publish and receive changes.

You can get a repository on your local machine in one of two ways:

- You can initialise a new repository that is not yet under version control for a project on your local computer.
- You can get a copy of an existing repository. For example, if you join a company with a project that is already running, you can clone this repository to your local machine.

COMMIT

A commit is a wrapper for a set of changes. Whenever someone makes a commit, they are required to explain the changes that they made with a short commit message so that, later on, people looking at the project can understand what changes were made.

Every set of changes creates a new, different version of your project. Therefore, every commit marks a specific version. The commit can be used to restore your project to a certain state as it's a snapshot of your complete project at a certain point in time.

THE WORKING DIRECTORY, STAGING AREA, AND GIT REPOSITORY

Your files can have three main states in Git: **committed**, **modified** and **staged**. If your file is committed, its data is safely stored in your local database. If it is modified, the file has changed but has yet to be committed to your database. If it is staged it means that you have marked a modified file to go into your next commit in its current version.

This leads to the main sections of a Git project: the **working directory**, **staging area** and **Git repository**. A working directory consists of files that you are currently working on. The staging area is a file that is contained in the Git repository. It stores information about what will go into your next commit. The staging area acts as the interface between the repository and the working directory. All changes added to the staging area will be the ones that actually get committed into the Git repository, which is where Git stores the metadata and object database for your project.

A version of a file is considered committed if it is in the Git repository. If it has been changed and has been added to the staging area, it is considered staged. If it has changed but has not been staged, it is modified.

BASIC GIT WORKFLOW

Below is the basic Git workflow:

1. Modify a file from the working directory.
2. Add these modified files to the staging area.
3. Perform a commit operation to move the files from the staging area and store them permanently in the Git repository.

GITHUB

GitHub is an online Git repository hosting service. It is free to use for open-source projects and offers paid plans for private projects.

GitHub offers all of the functionality of Git while also adding its own features. While Git is a command-line tool, GitHub provides a web-based graphical interface. It provides access control and many features that assist with collaboration, such as wikis and basic task management tools for all projects.

Each project hosted on GitHub will have its own repository. Anyone can sign up for an account on GitHub and create their own repositories. They can then invite other GitHub users to collaborate on their project.

GitHub is not just a project-hosting service, it is also a large social networking site for developers and programmers. Each user on GitHub has their own profile, showing their past work and contributions they have made to other projects. GitHub allows users to follow each other, subscribe to updates from projects, or like them by giving them a star rating.

A note from the

HyperionDev Team

Check out this **HyperionDev blog post** about Git and GitHub. It provides a simple guide to GitHub for beginners.

Compulsory Task 1

Follow these steps:

- Install Git by following the instructions given above.
- Once you have successfully downloaded and installed Git, enter `git --version` into your terminal or command prompt to display your current version of Git.
- Take a screenshot of what is displayed on your terminal or command prompt and send it to us to show that your Git installation was successful.

Compulsory Task 2

In this task, you will be creating a free GitHub account.

Follow these steps:

- Create a free GitHub account by visiting **https://github.com/join**.
- After you have created your account, submit a file named github.txt with a URL to your GitHub profile.

Completed the task(s)?

Ask an expert code reviewer to review your work!

Review work

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.