HyperionDev

Version Control and Git Basics

Task

Visit our website

Introduction

Knowing how to use version control is a crucial skill for anyone working on a coding project, especially when working with 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 lesson will introduce you to the basics of version control. It focuses on the Git version control system and the collaboration platform, GitHub.

Version control systems

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. You may wonder why your project would benefit from a version control system. Let’s get into some of the main benefits:

- **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. Therefore, you will have multiple people who need to work on the same file. Without a version control system in place, you would have to work together in a shared folder on the same set of files. 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 computer 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 computer. 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.

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.

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 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.

Git installation

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 to install Git on Windows, Mac, and Ubuntu. Make sure to verify the version, and configure your username and email address.

Installing Git

Windows

1. Go to **Git – Downloading Package** to download the official build from the Git website. Depending on your system architecture, choose the appropriate installer:

    - **64-bit**: Most modern systems use a 64-bit architecture. Select the 64-bit Git installer.

    - **32-bit**: If you have an older system with a 32-bit architecture, choose the 32-bit Git installer.

    Click the download link for the installer that matches your system architecture. The download should start automatically.

2. Once the download is finished, find the installer file in your Downloads folder or wherever you saved it. Double-click the file to launch the Git Setup wizard. Follow the on-screen instructions to complete the installation process.

macOS

1. Download the latest **Git for macOS installer**

2. Follow the prompts to install Git.

Ubuntu

1. Install Git by typing the following commands into your terminal:

    ```
    sudo apt-get update
    ```

    ```
    sudo apt-get install git
    ```

    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.

Verifying the installation

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

```
git --version
```

Configuring your username and email

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"
```

Note that the email you use for configuring Git should be associated with your GitHub account. If you want to use a different email or an anonymous no-reply email, troubleshoot this step using the **Git documentation**.

Before we dive into using Git, you 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: 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.

Git's three stages

Git keeps track of your project's files through three stages: new or modified, staged, and committed. A committed file is safely stored in your local database, representing a specific version of the file. Modified files have been changed since their last committed version, but these changes haven't been saved permanently yet. Staged files are those modified files that you've specifically chosen to include in your next commit.

These stages correspond to three key areas within your Git project. The working directory is where you actively work on your files. The staging area acts as a temporary holding zone, storing a list of files you've selected for the next commit. Finally, the Git repository is the permanent archive that stores all the historical versions of your project, including committed files and their metadata.

Here's a simplified workflow for managing your project with Git:

1. You **create or modify** files in your working directory.
2. You select the specific changes you want to keep by adding them to the **staging** area.
3. Finally, you perform a **commit** operation, which permanently saves the staged changes to the Git repository.

GitHub

GitHub simplifies how you share and manage code. It offers free version control for open-source projects, with paid plans available for private repositories.

Combining the power of Git with a user-friendly web interface, GitHub streamlines collaboration. You can easily track changes, assign tasks to your team, and share wikis to keep everyone on the same page.

GitHub lets you host your own project repository, and even static websites directly from it. This makes it a great platform to showcase your work. Build a **developer portfolio** by highlighting your contributions to various projects on GitHub.

Now, let’s get started with a basic workflow for Git.

Git in practice

Now that you know what Git is and its three main stages, let’s go through the basics of how to use it. There are two ways to get a Git project. You can either initialise a new repository or clone an existing repository. Here, we will go through the steps you would take to initialise and populate a new repository. This typically applies to personal projects, as there is usually an existing code base that you will work with as a developer in a work context. You will learn about cloning an existing repository from GitHub later.

Initialising a repository

To create a new repository, you have to initialise it using the `git init` command. To do this, open your terminal and go to your project’s directory. To change your current directory, 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 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 `new_file.txt`. To add `new_file.txt` to the repository staging area, you would need to enter the following into your terminal:

```
cd /Users/user/your_repository

git add new_file.txt
```

If you want to add all changes in the current directory and its subdirectories, use the following command:

```
git add .
```

The `.` represents the current directory. This command stages all new, modified, and deleted files within the current directory and its subdirectories for the next commit.

Checking the status of your files

A **snapshot** is a complete copy of the entire project's state at a specific point in time, including all files and their respective versions. This snapshot is used for version control and tracking changes. Files can either exist in a **tracked** state or in an **untracked** state in your working directory. Tracked files are files that were added in the last snapshot, while untracked files are any files in your working directory that were not added in your last snapshot and are not currently in the staging area. The staging area is an intermediate area where changes to files are prepared before committing, allowing you to selectively choose which modifications to include in the next commit (we'll explain more about committing in the next section).

To determine the state of files in the repository, we use the `git status` command. The `git add` command is used to start tracking a new file. After adding `new_file.txt`, executing the `git status` command will display the following output, indicating that `new_file.txt` is now being tracked:

Command:

```
git status
```

Output:

```
On branch main

Your branch is up-to-date with 'origin/main'.

Changes to be committed:

  (use "git reset HEAD <file>..." to unstage)

    new file:   new_file.txt
```

You can tell that `new_file.txt` 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 `git commit` command. A commit is a wrapper for a set of changes. 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. 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 new_file.txt"
```

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 instance of what you might see if you run `git log`:

```
commit a9ca2c9f4e1e0061075aa47cbb97201a43b0f66f

Author: Coding Student Bob <bob_can-code@gmail.com>

Date: Mon Sep 4 6:49:17 2024 +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 commit 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`:

```
a9ca2c9f4e1e0061075aa47cbb97201a43b0f66f  Initial commit.
```

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

Now that you've mastered the basics of managing your project with Git locally, let's explore how to connect it with GitHub. Note that you will create new repositories outside of your coursework repository for this task and other projects you would like to share.

GitHub in practice

Having a local Git repository is a great way to manage your project's versions. But to truly leverage the power of version control, you'll want to share your work with others. This section dives into connecting your local Git repository to GitHub, the world's leading platform for hosting code. We'll guide you through the process of authentication and pushing your local project to GitHub, allowing you to collaborate, share your code, and build a strong developer portfolio.

Authenticating Git with your GitHub credentials

One of the objectives of this task is for you to synchronise a local repository with a remote repository hosted by GitHub. To do this, you will need to gain authenticated access to your GitHub account using the GitHub Command Line Interface (GH CLI), a program developed by GitHub Inc. for performing GitHub operations unavailable via Git. While GH CLI offers a range of features, we will focus on installation and authentication for now. You are, however, welcome to experiment once you've completed the task.

To install GH CLI, please follow one of the following options based on the supported operating system you use:

- macOS options:
  - Download and install GH CLI directly from **GitHub CLI**.
  - If you have **Homebrew** installed, run the following command on your terminal: `brew install gh`
- Windows options:
  - Download and install GH CLI directly from **GitHub CLI**.
  - If you have **Chocolatey** installed, run the following command on your terminal: `choco install gh`
  - If you have **Scoop** installed, run the following command on your terminal: `scoop install gh`
  - If you have **WinGet** installed, run the following command on your terminal: `winget install gh`
- Linux options:
  - Download and install GH CLI directly from **GitHub CLI**.
  - If you have **Homebrew** installed, run the following command on your terminal: `brew install gh`

- Follow the instructions in the **official installation documentation**.

Once you have installed GH CLI, you can run the command below and follow the instructions that follow. If the command results in an error, please check your Internet connectivity and try running it again in a fresh terminal window. Otherwise, please schedule a mentor call from your dashboard.

```
gh auth login
```

You will receive a prompt that looks like the one shown below. Select the **GitHub.com** option. Select the options shown in the prompts by pressing Enter.

```
? What account do you want to log into? [Use arrows to move, type to filter]

> Github.com

  Github Enterprise Server
```

Once you've selected GitHub.com, the prompt will ask you for the network application protocol you wish to use with Git. Please choose **HTTPS**, as shown below.

```
? What account do you want to log into? GitHub.com

? What is your preferred protocol for Git operations on this host? [Use
arrows to move, type to filter]

> HTTPS

  SSH
```

You will then be prompted to authenticate Git with your GitHub account or not. You may simply press Enter and GH CLI will automatically input "Yes" for you. Before you press Enter, your screen should look like this:

```
? What account do you want to log into? GitHub.com

? What is your preferred protocol for Git operations on this host? HTTPS

? Authenticate Git with your GitHub credentials? (Y/n)
```

After accepting the request to authenticate Git with your GitHub credentials, you will be prompted as to how you wish to login. Please select the web browser option and follow the instructions which follow, as shown below.

```
? What account do you want to log into? GitHub.com

? What is your preferred protocol for Git operations on this host? HTTPS

? Authenticate Git with your GitHub credentials? Yes

? How would you like to authenticate GitHub CLI? GitHub.com  [Use arrows to move, type to filter]

> Login with a web browser

  Paste an authentication token
```

After selecting the web browser option, you will receive a one-time code. Copy this code, as it will be needed for authorisation. Press Enter to open GitHub.com in your browser. Once GitHub opens and you have signed in, paste the code when prompted and follow the instructions on the screen to complete the authorisation.

Once you've logged in with GH CLI, you've successfully authenticated Git with your GitHub credentials. This allows you to sync your local and remote repositories.

If you wish to log out of GH CLI to log into a different GitHub account or, for other reasons, run the command below.

```
gh auth logout
```

Pushing a local repository to GitHub

Time to share! Now that you've got authenticated access, let's push your code to GitHub. We'll create a new repository on GitHub, grab its URL, and specify the branch to upload code to GitHub. This unlocks collaboration, version control across devices, and a platform to show off your skills.

Follow these steps to push a local repository to GitHub.

- Login to GitHub and navigate to your **home page** or repositories tab.
- Create a new repository by selecting the “New” button, as shown in the image below.

- On the “Create a new repository” page, you will choose a name for your repository and add an optional description. If you are creating a GitHub repository for an existing local repository, it is common to use the same name, but it is not a prerequisite. There are several settings you may adjust, such as choosing to make your repository private or public, adding a README file, and selecting which files not to track.

- Next, GitHub will give you a URL that you can use to push a local repository to GitHub and some guidance depending on whether you need to create the local repository or have an existing repository. The format of the URL is: `https://github.com/[username]/[repository_name].git`

- Let’s assume you have an existing repository. Open your terminal and change the directory (`cd`) to the folder where you created your repository. Then enter `git remote add` followed by `origin` and the URL provided by GitHub. Origin is the default name used for remote repositories. For example:

  ```
  git remote add origin https://github.com/HyperionDev/git-task.git
  ```

  Now you can use the short remote name (`origin`) in the command line instead of the whole URL.

- To push your local repository, you also need to specify the branch. For now, use the `main` branch. More on branches later. For example:

  ```
  git push -u origin main
  ```

  Notice the format of the command is `git push -u [remote_name] [branch-name]`.

  The `-u` instructs Git to set the specified remote branch (`origin main`) as the default upstream branch for the current local branch.

- In summary, the two commands you need to push a local repository to a remote repository you have created will look something like this:

```
git remote add origin https://github.com/HyperionDev/git-task.git

git push -u origin main
```

Extra resource

Explore **working with remote repositories** to learn more about managing your remote repositories.

Spot check 1

Let's see what you can remember from this section.

1. What are the two ways you can get a Git repository?
2. What are the three key areas of a Git project?
3. What is one of the main benefits of GitHub?

README.md files

When you add your code to GitHub, you can and should create README files. A README file is usually the first file that anyone interested in your code will look at. This file should describe your code. It should tell the reader what the project does, why the project is useful, who maintains and contributes to the project, and how a user can get your code to work.

As a README file is essential for all software projects, learning to write clear, easy-to-read and appropriately detailed README files is an essential skill.

According to **this GitHub guide**, README files should contain the following:

- The project name.
- A clear, short, and to-the-point description of your project. Describe the importance of your project and what it does.
- A table of contents, to allow other people to quickly navigate especially long or detailed READMEs.
- An installation section, which tells other users how to install your project locally.
- A usage section that instructs others on how to use your project after they've installed it. Include screenshots of your project in action.
- A section for credits, which highlights and links to the authors of your project if the project has been created by more than one person.

README files have a **.md** extension. Here, "md" stands for Markdown. Markdown is a syntax that lets you style text. If you write text in a program like MS Word, you usually use the toolbar to select appropriate options to style your text (e.g. make certain text bold, underlined, or formatted in another way). When creating Markdown files, you style your text using keywords and characters instead. For instance, if you wanted to italicise text, you would surround the text with asterisks: In this sentence *this text would be in italics*. Explore Markdown syntax in this **GitHub Guide**.

If you would like to experiment with README.md files, start by creating a **profile README** for your GitHub profile.

Extra resource

To see an example of a README file, go to the **18F Open Source Style Guide**. Notice how the README file is rendered in the browser. Now click on "Raw" to see the Markdown for this file.

Instructions

Feel free to refer to the **Git cheatsheet** as needed for this or any future tasks in which you use Git. You may also refer to an open-source book, **Pro Git**, by Scott Chacon and Ben Straub as an additional resource.

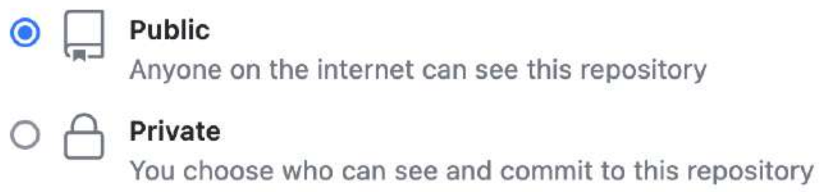
Practical task 1

- Create an empty folder called **git-task**.
- Open your terminal and then change the 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 code file in the git-task folder called **hello_world** and write a program that prints out the message "Hello World!"
    i. e.g. `hello_world.py` or `hello_world.js`
- Run the `git status` command again. You should now see that your hello_world file is untracked.

- Enter the `git add` command followed by `hello_world.py` or `hello_world.js` to start tracking your new file.
- Once again, run the `git status` command. You should now see that your hello_world file is tracked and staged to be committed.
- Now that it is tracked, let us change the hello_world file. Change the message printed out by the program to “Git is awesome!”
- Run `git status` again. You should see that hello_world 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` 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.
- Take a screenshot of the output, name the image file **gitlog**, and upload it to the task folder.

Practical task 2

- Log in to GitHub and navigate to your **home page** or repositories tab.
- Create a new repository named **git-task** and choose the setting to make the repository **public**, as shown below:


[✓ Public]
Anyone on the internet can see this repository

Private
You choose who can see and commit to this repository

- Push the git repository on your local machine containing the hello_world file to the remote git-task repository on GitHub.
- Ensure the repository is public. Then, put a link to the git-task repository in a text file called **git-task.txt** and submit it.
- Once you have completed this bootcamp, you can **delete the repository** that you created here, since it doesn't store any meaningful application code.

Practical task 3

- Create a Github repository called **example-repo**.
- Push any code that you created previously to this remote repository.
- Add a README file for the code that you have pushed to GitHub that contains:
    - The project name.
    - A clear, short description of your code and what it does.
- Add a URL to your public GitHub repository to a text file named **example-repo.txt** and submit it.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.

Spot check 1 answers

What are the two ways you can get a Git repository?

- 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.

2. What are the three key areas of a Git project?
    - The working directory, staging area, and the Git repository.
3. What is one of the main benefits of GitHub?
    - It streamlines collaboration and/or it provides a platform to share your developer portfolio.

Reference list

Chacon, S., and Straub, B. (2014). *Pro Git*. Git SCM. **https://git-scm.com/book/en/v2**