# Hyperiondev

**Build Your Brand**

# Building a Technical Portfolio

Visit our website

**Take note!**

**Important** – By **2 February 2025**, you should record an **invite to an interview** (if seeking employment) **or** a self-employment declaration (if seeking to be self-employed) at **www.hyperiondev.com/outcome8**. Additionally, you should record an **offer of a job** (if seeking employment) or evidence of obtaining **new work/contracts** (if self-employed or seeking to be) at **www.hyperiondev.com/finaloutcome8** by **13 April 2025**. Please record these milestones as soon as you reach them.

A **limited** number of co-certifications will be awarded based on your achievement of all **four criteria** for successful completion of this Skills Bootcamp.

# Introduction to Build Your Brand tasks

In the course of your Skills Bootcamp, you will encounter a number of "Build Your Brand" (BYB) tasks. What are these and what value do they offer you?

Let's put this in context. Remember that the Department for Education (DfE) has selected HyperionDev as a provider to offer a variety of fully funded coding Skills Bootcamps. The Skills Bootcamps are intended for people who wish to change their careers to tech, either by gaining employment in the tech sector or by starting or growing their own businesses using the learnings from the bootcamp to attain contracts for new coding work.

The DfE are entirely funding your studies on your Skills Bootcamps, with the requirement that you fulfil certain criteria. These criteria include completing a certain amount of study within the first week, completing the overall program with a certain minimum number of Guided Learning Hours, achieving an offer to an interview (for those seeking employment) or new work opportunities (for self-employed people), and finally either a new job or promotion based on the new skills, an apprenticeship on the same basis, or proven new contracts or work opportunities (for self-employed people) that directly utilise the coding skills learnt on the Skills Bootcamps.

Getting interviews and job offers, or finding new self-employed work opportunities or contracts can be tricky, and can feel intimidating to many students. The purpose of the BYB tasks is to support you in **incrementally building** up skills and resources that will equip you to meet these DfE requirements. The tasks walk you through:

- considering and formulating career goals to help you find your feet and plan how you will derive the most value from your time on your Skills Bootcamp (this task),
- creating a top-notch technical CV (which can be used for job applications or to establish your credentials as a professional when applying for work contracts if you are self-employed or intend to become self-employed),
- writing a cover letter (a skill that can also be applied, with slight variation, to writing bids for new work as a self-employed person),
- creating and polishing a LinkedIn profile (as LinkedIn is an indispensable tool in today's strongly networked business environment),
- searching for and applying to jobs or bidding for new work contracts, and
- creating a technical portfolio with which to showcase your new skills as a coder (this can be shared with prospective employers or business contacts, providing concrete evidence of your capacity and experience).

Along the way we share examples, tips, and tricks for everything we are teaching you to do. Additionally, we support your journey as a professional moving into the field of tech with our Career Services support program.

Are you ready to get started? Let's dive in!

# Part 1 – Introduction to version control and Git

## WHY VERSION CONTROL?

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 first part of the task will introduce you to the basics of version control. It focuses on the Git version control system and the collaboration platform, GitHub.

## 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 were not 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 which is not only the most widely used modern version control system but also free, open-source, and 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.

## Git components: The working directory, staging area, and git repository

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

In Git your files can have three main states: **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.

## Git components: The 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.

## 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 Linux:

### 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 Linux

1. Git is usually pre-installed on Linux distributions. Please follow the instructions at **https://github.com/git-guides/install-git** if Git is not installed.
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"
```

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.

## WHAT IS GITHUB?

Git is the foundation of many services that work on version control. The most popular and widely used of them all is GitHub. GitHub is an online Git repository hosting service. GitHub offers all of the functionality of Git and a lot more. While Git is a command-line tool, GitHub provides a web-based graphical interface. It offers access control and many features that assist with collaboration, such as wikis and basic task management tools for all projects. It is free to use for open-source projects and offers paid plans for private projects

GitHub is used to distribute bootcamp content to students on many of our bootcamps, and also used by the students to share code with mentors and lecturers, and to submit tasks for grading. For this reason, you should have received a guide to GitHub as part of your onboarding into your bootcamp. This walks you through the very basics of logging into and interacting with GitHub.

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. You can even host websites for free directly from your repository!

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.

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

# Part 1: Practical task 1

First, create a Google doc with a uniquely identifiable filename that includes your name and email address and a task identifier for this task (**Tech_Portfolio**). For example, if your name was John Smith and your email address was john_smith@gmail.com, your filename would be **John Smith – john_smith@gmail.com – Tech_Portfolio**. As you progress through this Practical Task you will fill your answers into this Google doc, which you will save as a PDF and upload to GitHub at the end.

In this practical task, you will be installing Git. Follow these steps:

- Install Git by following the instructions given in Part 1.
- 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 the displayed Git version and place it into your Google answers doc.

# Part 1: Practical task 2

In this practical task, you will be creating a free GitHub account if you have not done this already.

Follow these steps:

- Ensure you have read through the document entitled 01-012_GitHub Guide provided to you at onboarding.

- If you have not already made one, create a free GitHub account by visiting **https://github.com/join**.

- After you have created your account, copy the URL of your GitHub profile and add it to your Google answers doc.

# Part 2 – Creating a technical portfolio with GitHub

## BUILDING YOUR PROFESSIONAL PORTFOLIO

Many tools help build a professional brand online, but to showcase your skills as a developer, few are more important than GitHub. Your GitHub repository is a place where you can share your code to demonstrate your skills. This second part of the task focuses on helping you to understand how to showcase your newly acquired development skills to peers, potential clients, and employers, so that as you move through the course, you can build up a portfolio.

## WHAT ARE REPOSITORIES?

We will start Part 2 by introducing 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.

## TYPES OF REPOSITORIES

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 (such as GitHub) or in your local network. Teams of developers use remote repositories to share and exchange data, as you will use GitHub during your Skills Bootcamp. These serve as a common base where everybody can publish and receive changes.

### Local repositories

You can get a repository on your local machine in one of two ways:
- Initialise a new repository that is not yet under version control for a project on your local computer.
- 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.

### Initialising a repository

To create a new local repository, you have to initialise it using the `init` command. To do this, firstly open your terminal (or command prompt if you are using Windows) 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 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.py** or **newFile.js** (depending whether the language you are learning is Python or JavaScript). For the sake of this task, we will continue to refer to files with Python (**.py**) file extensions as the majority of students are learning Python. If you are a web development/JavaScript student, just keep in mind that the file extensions for all the files you use will be **.js** instead.
To add **newFile.py** 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.py
```

## 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. As mentioned in the Introduction to Version Control and Git, a snapshot enables you to compare the current version of your project to a previous version. 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.py**, you should see the following code, showing that **newFile.py** 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.py
```

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

## Committing your changes

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.

Once you've added a file to your repository, you should 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.py"
```

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 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`:
```
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.

## AUTHENTICATING GIT WITH YOUR GITHUB CREDENTIALS

One of the objectives of this task is for you to synchronise your local repository with a remote repository hosted by GitHub. To do this, you will need to be able 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, for this task and academic programme, we will only focus on installation and authentication. You are, however, welcome to experiment once you have completed the task.

To install GH CLI, please follow either one of the following options based on the supported operating system you use:
- macOS options:
  - Download and install GH CLI directly from **https://cli.github.com/**.
  - If you have **Homebrew** installed, run the following command on your terminal: `brew install gh`
- Windows options:
  - Download and install GH CLI directly from **https://cli.github.com/**.
  - 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 **https://cli.github.com/**.

- 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 in this task. 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 technical mentor call from your dashboard.

`gh auth login`

You will receive a prompt that looks like the one shown below, to which you need to select the GitHub.com option. You may select the options shown in the prompts by pressing Enter.



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 in the prompt similar to the one below.



You will then be prompted if you wish to authenticate Git with your GitHub account. You may simply press enter and GH CLI will automatically input Yes for you. Before you press enter, your screen should look like the screenshot below.



Once you've accepted the request to authenticate Git with your GitHub credentials, you will be prompted as in the screenshot below whether you wish to login with a web browser. Please select that option and follow the instructions that follow.



Once you've logged in with GH CLI, you've successfully authenticated Git with your GitHub credentials and are ready 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, you may run the command below.

```
gh auth logout
```

## SYNCING YOUR LOCAL AND REMOTE REPOSITORIES

Now that you understand how to create and add files to a local Git repository you should also learn how to **push** your repository from the command line to a remote GitHub repository. Have a look at this **excellent video tutorial** that explains how you can do exactly that.

Here is a summary of the command line prompts when you "`push an existing repository from the command line`":

```
git remote add origin https://github.com/[REPO-OWNER]/[REPO-NAME]
git branch -M main
git push -u origin main
```



Extra resource

Have a look at the Git cheat sheet in your GitHub repository for more useful Git commands.

## BRANCHING

It is common for several developers to share and work on the same source code. Since different developers will have to be able to work on different parts of the code at the same time, it is important to be able to maintain different versions of the same codebase. This is where **branching** comes in.

Branching is one of the fundamental aspects of working with Git. A branch represents an independent line of development. It allows each developer to branch out from the original codebase and isolate their work from others. By branching, you diverge from the main line of development and continue to work without messing up or disrupting the main line.

Branches are essential when working on new features or bug fixes. You create a **new branch** whenever you add a new feature or fix a bug to encapsulate your changes. This ensures that unstable code is not committed to the main codebase and also enables you to clean up your feature's history before merging it into the main branch.
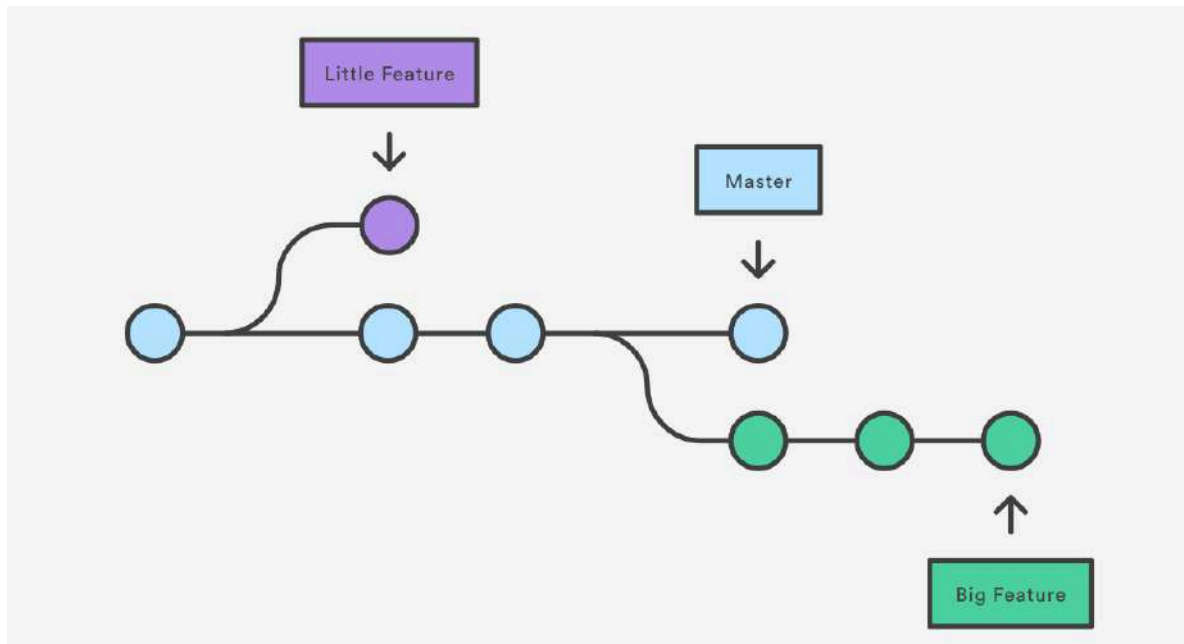
*A repository with two isolated lines of development (__Atlassian__ under CC BY 2.5 Australia; from __Atlassian__)*

The image above visually represents the concept of branching. It shows a repository with two branches; one for a small feature and one for a larger feature. As you can see, each branch is an isolated line of development which can be worked on in parallel and keeps the main branch, known as the master branch, free from dubious code.

Git creates a master branch automatically when you make your first commit in a repository. Until you decide to create a new branch and switch over to it, all following commits will go under the master branch. You are, therefore, always working on a branch.

The HEAD is used by Git to represent the current position of a branch. By default, the HEAD will point to the master branch for a new repository. Changing where the HEAD is pointing will update your current branch. You can check where the HEAD is currently pointing with the `git status` command which will display this information in the first line of output.

## Creating a branch

To create a new branch, use the `git branch` command, followed by the name of your branch. For example:

```
git branch my-first-branch
```

## Switching branches

Using the `git branch` command does not switch you to the new branch; it only creates the new branch. To switch to the new branch that you created, use the `git checkout` command.

```
git checkout my-first-branch
```

Using this command moves the HEAD to the my-first-branch branch.

Alternatively, you can run the `git checkout` command with a -b switch to create a branch and switch to it at the same time. For example:
```
git checkout -b my-second-branch
```

This is short for:
```
git branch my-second-branch
git checkout my-second-branch
```

## Saving changes temporarily

When you make a commit, you save your changes permanently in the repository. However, you might find that you would like to save your local changes temporarily. For example, imagine you are working on a new feature when you are suddenly required to make an important bug fix right away. Obviously, the changes you made so far for your feature don't belong to the bug fix you are going to make. Fortunately, with Git, you don't have to deploy your bug fix with the new feature changes you have made. All you have to do is switch back to the master branch.

Before you switch to the master branch, however, you should first make sure that your working directory or staging area has no uncommitted changes in it otherwise Git will not let you switch branches. Therefore, it is better to have a clean working slate when switching branches. To work around this issue we use the `git stash` command.

## Using git stash

The `git stash` command takes all the changes in your working copy and saves them on a clipboard. This leaves you with a clean working copy. Later, when you want to work on your feature again, you can restore your changes from the clipboard in your working copy.

To restore your saved stash you can either:
- Get the newest stash and clear it from your stash clipboard by using `git stash pop`.
- Get the specified stash but keep it saved on the clipboard by using `git stash apply <stashname>`.

## Merging

When you are done working on your new feature or bug fix in an isolated branch, it is important to merge it back into the master branch. The `git merge` command allows you to take an independent line of development created by `git branch` and **integrate** it into a single branch.

To perform a merge, you need to:

- Check out the branch that you would like to use to receive the changes.
- Run the git merge command with the name of the branch you would like to merge.
```
git checkout master
git merge my-first-branch
```

The above example merges the branch, my-first-branch into the master branch. Please keep in mind that Git has updated the default branch name from 'master' to 'main'. Read more about this **here**.

## CLONING A REPO TO QUICK START

You know how to initialise a Git repository from scratch, but in practice, this is very rarely how you start developing. Typically when starting as a software engineer, you will be tasked with closing one or more issues on a specific repository. We'll get onto exactly what "closing an issue" means, but for the time being, let's first focus on getting the repo cloned so that you can start developing.

For this task, you'll be using your last Python repository. If you have not put it on GitHub yet, go ahead and do that now.

We will be pretending your repo is a team project, and you will be playing the role of developer, code reviewer, and even project manager. To start off, pretend you are a developer seeing the project for the first time and clone the repo:

```
git clone https://github.com/[username]/[repo].git
```

If you were working in a development job, at this point you would usually spend a while compiling the code, resolving dependencies and fixing issues on your computer to get the project running. This should, however, not be a problem for you in this learning exercise as you recently wrote that very code.
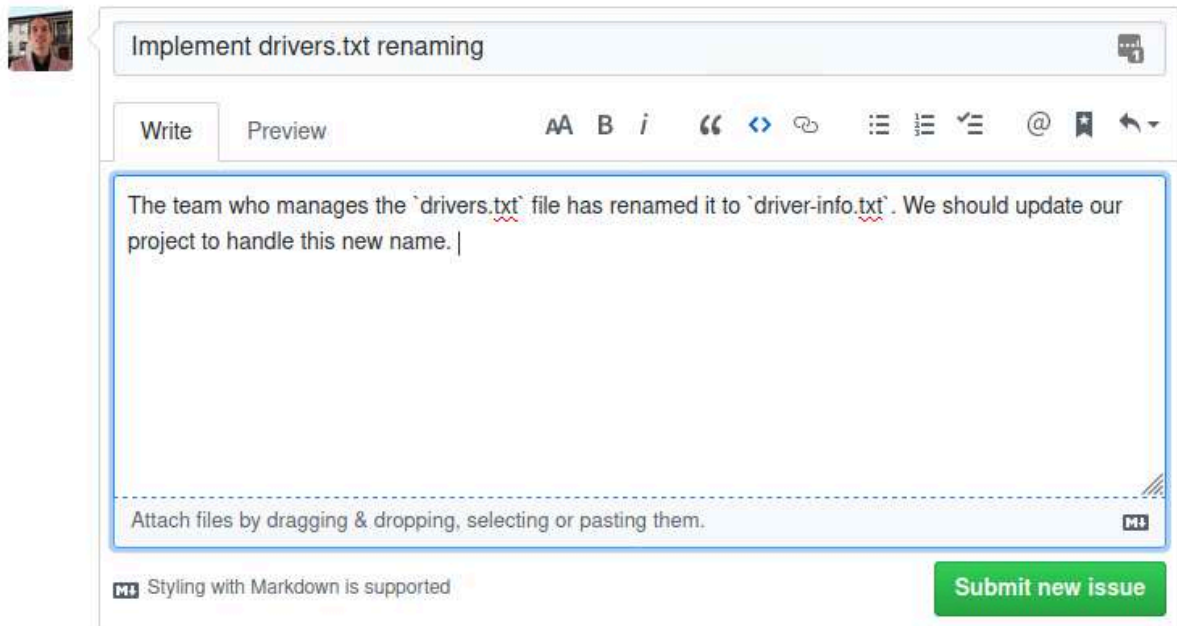
## WHAT IS AN ISSUE?

On GitHub (and many other platforms) an *issue* is a plain-English description of some improvement a codebase needs. This could be a bug fix, a performance improvement, or a new feature. Software engineers almost never do any work unless there's an issue attached to it. This makes sure everyone knows who's doing what and helps line managers understand what's being done to the codebase and what the progress is.

On the GitHub website, go to your repo and click on "Issues" (on the top bar, next to Code). You should see a plain greeting message since your repo doesn't have any issues yet (yay!).

Pretend you're a project manager and you just came back from a meeting where another team asked you to make a change to your project. In order to let your team know what

needs to be done, you make an issue on the repo. Click on "New Issue" (top right) and fill in the following details:



When finished, click "Submit" to officially create the issue. You will see the issue has been given an ID (**#1**).

To the right of your issue description, click on the gear icon next to "Assignees". Type in your username and click on it to assign yourself to the job. If you were a real project manager, you would be assigning one or more of your team members.

## WORKING ON YOUR OWN BRANCH

Now that you (as the developer) have been assigned to the issue, you should implement it. But before doing any coding, be sure to go to the directory and create a new branch. This ensures you don't accidentally work on a co-worker's branch and their code gets overwritten later on. You can name the branch something descriptive indicating what you will be doing on it (like `driver-file-fix`), or to ensure uniqueness you can name it according to the issue ID (`issue-1`). Each company will have its own policy in this respect. As you may recall, the code below creates the branch `issue-1` and switches to it at the same time.

```
git checkout -b issue-1
```

Now implement the change. As described in the issue, you need to make the program use `driver-info.txt` instead of `drivers.txt`. You would also rename the actual text file and run your program to ensure there are no problems with the new change.

Now commit your changes, giving a good description of what you changed, and then push your branch to the remote repo.

```
git commit -m "Changed program to use driver-info.txt"
git push origin issue-1
```

You can make multiple commits and pushes on your branch as needed until you are satisfied the change has successfully been implemented.

## Making a pull request

A pull request (**PR**) is a request asking that your branch be merged with the master branch. If accepted, this will make your code changes part of the main codebase. Depending on how the repo is set up, this might even deploy the new version of the program to your users.

It is the responsibility of the developer who made the changes (in the new branch) to create the PR. Go to your repo on GitHub, and click on "branches" (next to commits, just above your code). Find your branch in the list, and to its right, click on "New pull request".



Write a good description of your PR, explaining everything that has been changed in it. It's good practice to mention the issue that originally motivated the changes.
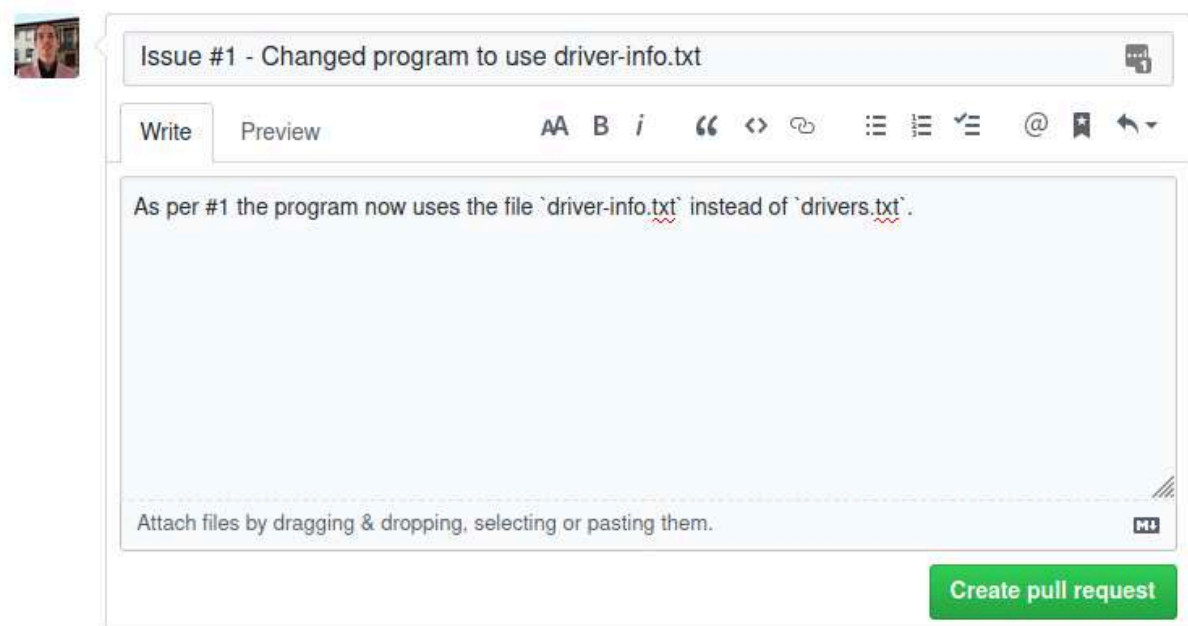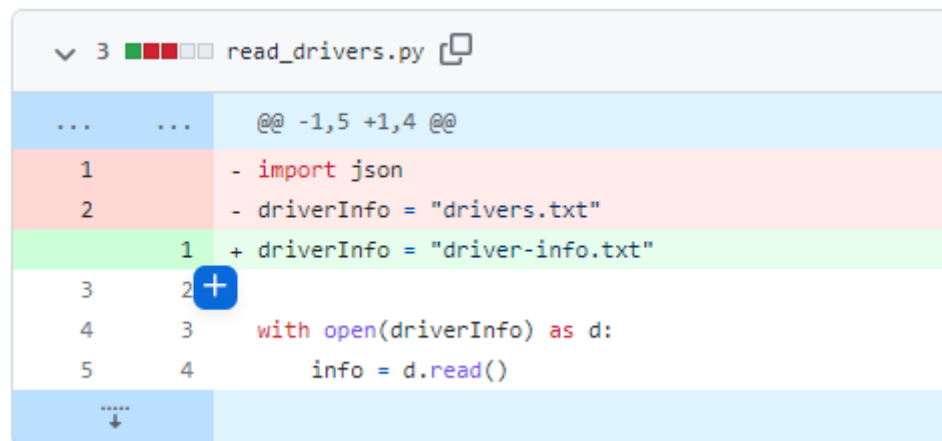
Click on "Create pull request". The PR will be created with an ID (**#2**).

Imagine now you are the project manager and click on the gear icon next to "Reviewers" to assign someone to review the PR. For this task, assign yourself, but, of course, in practice, your project manager would assign a co-worker instead.

## Code review

Because getting a PR accepted can have so many implications for the project, it's important that your code is reviewed by another developer. This is often how easily overlooked mistakes are spotted.

Imagine now you are a code reviewer and click on "Commits" in the PR on GitHub. Here you will see a list of commits made to the branch the PR is for. For an all-encompassing view of changes made to the project, click on "Files changed" on the same top bar. This will show all changes made to the project in a human-readable **diff** view. Here's an example:

```
  3 ▣▣▣▫▫ read_drivers.py

...      ...      @@ -1,5 +1,4 @@
  1             -  import json
  2             -  driverInfo = "drivers.txt"
          1     +  driverInfo = "driver-info.txt"
  3       2 ➕
  4       3        with open(driverInfo) as d:
  5       4            info = d.read()
```

As a reviewer, you might remark that the method name violates project standards (should be `snake_case` not `camelCase`), and that you need to add comments. If you find a mistake in your PR, hover over the offending line and click on the blue "+" button to its left. This will let you write a comment on the code. Once submitted, this will be visible to the original developer.

You as the developer can now make the requested fixes and commit and push them to have the PR updated automatically.

Once pushed, as the reviewer you should go back to the "Files changed" tab and click on review changes (top right) to accept the changes. GitHub restricts review usage so that PR authors can't review their own code, so don't worry about doing this formally right now.

## FINALLY MERGING

After all this work, the branch is finally ready to be merged. Either the code reviewer or the project manager will merge the pull request. Head to your PR on GitHub, click on "Conversation" and scroll to the bottom where you should see a big green "Merge pull request" button. Note that continuous integration may pose further limits (beyond code review) before allowing a merge to happen. This may include successful building and passing of tests for your code. Setting this up is quite involved and typically not the software engineer's responsibility. You can ignore it for now.

Click "Merge pull request" and change the description if you want to. GitHub will now automatically merge your branch into master.

Now head back to the original issue, scroll to the bottom, and make some closing remarks to show that the problem has been resolved. It's a good idea to mention the PR that officially closes the issue.



Click on "Close and comment" to officially mark the issue as closed.

To have the changes reflected on your computer, switch to the master branch and pull the update:

```
git checkout master
git pull origin master
```

## GITHUB AND YOUR DEVELOPER PORTFOLIO

As previously stated, a **developer portfolio** (a collection of online programs that you have developed) allows you to demonstrate your skills rather than just telling people about them.

GitHub provides one of the most industry-recognised ways of sharing your code with others, including peers, prospective employers, or clients. A well-organised and documented GitHub repository can serve as a core component of a developer portfolio.

Even before seeing your work, prospective employers may also be impressed with the fact that you have experience in working with Git and 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.

A README file is essential for all software projects and learning to write clear, easy-to-read, and 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. Keep in mind that if you add screenshots to your readme file, you will have to push those image files into your repository too.
- 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, which stands for **m**ark**d**own. 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 example, if you want to italicise text, you would surround the text with asterisks, like this: `In this paragraph *this text would be in italics*.`

Below is a summary of **Markdown syntax** taken from this **GitHub Guide**:

| | |
|---|---|
| **Headers** | Headings are indicated with hash symbols that range from H1 to H6<br>`# This is the biggest heading you get, used for Main Titles`<br>`## This is a slightly smaller heading, used for Subtitles`<br>`###### This is the smallest heading` |
| **Emphasis** | `*This text will be italics*`<br>`_This will also be italics_`<br>`**This text will be bold**`<br>`__This will also be bold__`<br>`_You **can** combine them_` |
| **Unordered lists** | `* Item 1`<br>`* Item 2`<br>`  * Item 2a`<br>`  * Item 2b` |
| **Ordered lists** | `1. Item 1`<br>`2. Item 2`<br>`3. Item 3`<br>`   a. Item 3a`<br>`   b. Item 3b` |
| **Images** | `![alt text](image url)`<br>Example: ![GitHub Logo](/images/logo.png) |
| **Links** | `[link text](link url)`<br>Example: [GitHub](http://github.com)<br>OR use the shortcut syntax which doesn't specify the link text:<br>`<link>`<br>Example: <http://github.com> will automatically display as http://github.com |
| **Blockquotes** | `> This is a block quote. It can span over multiple lines.`<br>`> It will display the quote with a grey line on the left-hand side.`<br>Example:<br>As Kanye West said:<br>> We're living the future so<br>> the present is our past. |

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



For more information about markdown, see the markdown cheatsheet (additional reading) provided by GitHub **here**.

**Take note!**

**A reminder of the important dates** – By **2 February 2025**, you should record an **invite to an interview** (if seeking employment) **or** a self-employment declaration (if seeking to be self-employed) at **www.hyperiondev.com/outcome8**. Additionally, you should record an **offer of a job** (if seeking employment) or evidence of obtaining **new work/contracts** (if self-employed or seeking to be) at **www.hyperiondev.com/finaloutcome8** by **13 April 2025**. Please record these milestones as soon as you reach them.

A **limited** number of co-certifications will be awarded based on your achievement of all **four criteria** for successful completion of this Skills Bootcamp.

# Part 2: Practical task 1

For these tasks, use the same Google doc you created in Part 1 when you need to submit answers.

Follow these steps:

- Login to GitHub using the account you created in Part 1 of this task.
- Create a new repository by selecting the 'New' button as shown in the image below.
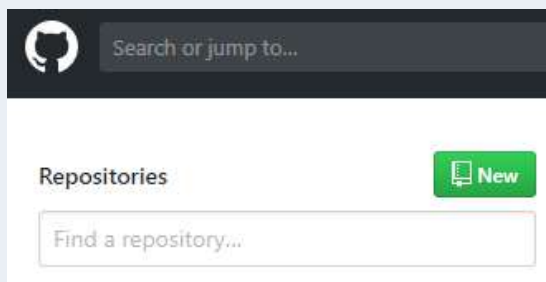


- Name the repository '**byb_project**' and make sure that it is public.

- Next, create an empty folder called **byb_project** on your local machine.
- 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 **byb_project** folder called **helloWorld.py** or **helloWorld.js** (depending on whether the language you're learning is Python or JavaScript, as explained earlier; please keep this in mind for the remainder of the practical tasks and align your file extensions to the coding language relevant for you. Python for software engineering/Data science and Javascript for web development).
- Write a program in the file you have created that prints out the message "Hello World!" using the code below. Only copy into your file the code relevant to the programming language you are learning.

Python
```python
# Python program to print "Hello, World!"
print("Hello, World!")
```

JavaScript
```javascript
// JavaScript program to print "Hello, World!"
console.log("Hello, World!");
```

.

- Run the `git status` command again. You should now see that your **helloWorld.py** or **helloWorld.js** file is untracked.
- Enter the `git add` command followed by the name of your file (**helloWorld.py** or **helloWorld.js)** to start tracking your new file. For example:
  `git add helloWorld.py` or `git add helloWorld.js`
- Once again, run the `git status` command. You should now see that your **helloWorld.py** or **helloWorld.js** file is tracked and staged to be committed
- Now that it is tracked, let us modify the code in the file **helloWorld.py** or **helloWorld.js**. Change the message printed out by the program to "Git is Awesome!"

- Run `git status` again. You should see that **helloWorld.py** or **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 helloWorld.py` again. If you are coding in JavaScript run `git add helloWorld.js`
- 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 such as "Modified the message printed".
- Running the `git status` command should now show a clean working directory once again.
- Push the repository on your local machine to the remote repository on GitHub by following these steps:

    - Open your terminal or command prompt. Change directory (`cd`) into the **byb_project** folder you created.
    - **Add your remote repository** using the following command:

      `git remote add [remote-name] [url]`
      e.g. `git remote add origin https://github.com/HyperionDev/byb_project.git`.

      Now you can use the short name (e.g. **origin**) on the command line in lieu of the whole URL. The URL will be indicated under the heading shown below once you have created your repository on GitHub.

      ...or push an existing repository from the command line

    - Push your local repository to your remote repository using the following command:
      `git push -u [remote-name] master`
      E.g. `git push -u origin master`

- Make the repository **public** and put a link to it into your Google answers doc (help for this is available **here**). Make it clear that this link is for *Part 2: Practical Task 1*.
- Once a code reviewer has marked this task as complete (and not before!), you can delete the repository that you have created here since it doesn't store any meaningful application code. Help for this is available **here**.

# Part 2: Practical task 2

Follow these steps:

- Create a second new repository on your GitHub account with the repo name spelt **exactly** the same as your username.
- Create a README.md file inside this newly created repo.
- Use this README.md file to create a landing page for your GitHub account that is attractively styled by using the **GitHub Styling Guide**. This README page will automatically display on your main account page for future recruiters to see.
- Feel free also to add images. This might require research on the usage of the <img> or <picture> tag. You can read more about picture tags **GitHub Docs**.
- Make the repository **public** and put a link to it into your Google answers doc (help for this is available **GitHub Docs**). Make it clear that this link is for *Part 2: Practical Task **2***.

**Useful resource**

- We highly recommend you look for a text called *Cracking the Coding Interview* by Gayle Laakmann McDowell, which offers valuable guidance and practice for tech interviews. This can be **purchased from Amazon**, but may also be available in your local public library or other places online.

# Final Submission Checklist

**Part 1 – Ensure you hand in the following:**
- **Part 1: Practical Task 1 –** A screenshot of the displayed git version after installation, pasted into your Google answers doc.
- **Part 1: Practical Task 2 –** the URL of your GitHub profile, also pasted into your Google answers doc.

**Part 2 – Ensure you hand in the following:**
- **Part 2: Practical Task 1 –** a link to your byb-project repository, pasted into your Google answers doc, **and** your completed profile READ_ME file in the repository.
- **Part 2: Practical Task 2 –** a link to your repository with the repo name spelt **exactly** the same as your username, pasted into your Google answers doc.

Finally, save your Google answers doc as a PDF (using menu options File -> Download -> PDF) and upload the PDF file to your GitHub folder for this Task.

**Remember, if you secure an interview, it is absolutely vital that you notify us** via **hyperiondev.com/outcome8**.

## RUBRIC FOR TECHNICAL PORTFOLIO ASSESSMENT

Your technical portfolio will be marked using the rubric below. It is included here so that you can use it as a guide to completeness if you wish.

If you score below 8/12 for this task, you will be able to resubmit the task. If you score 8/12 or higher, you will not be able to resubmit the task.

| | Unsatisfactory (1) | Acceptable (2) | Outstanding (3) |
|---|---|---|---|
| Screenshot of the displayed git version after installation | · No screenshot of installed git version, or screenshot of something entirely different | · Screenshot provided but indicates git installation was not properly completed | · Screenshot provided and does reflect git was installed |
| URL of GitHub profile | · No GitHub profile URL provided, or URL of incorrect site | · URL provided but (any) problem encountered | · URL provided |
| Pushing repositories to GitHub and ReadMe | · No public link shared to GitHub repositories<br>· No repositories on GitHub<br>· Incorrect local repositories pushed to GitHub | · Public link to at least one GitHub repository shared (either byb_project or the repo with name spelt exactly the same as their username)<br>· At least one of the two required repositories actually pushed to GitHub, corresponding to the link provided | · Public links to both GitHub repositories shared<br>· byb_project and the repo with name spelt exactly the same as their username both on GitHub |
| GitHub profile README | · No profile README created<br>· 3 or more spelling errors, 3 or more grammar errors | · README contains 'landing page' information for GitHub profile but information and clarity of instructions could be improved<br>· Styling or information could be improved<br>· 1-2 spelling errors, 1-2 grammar errors | · README present as appropriate as a 'landing page' for Githug profile, with clear information and instructions.<br>· README has attractive styling<br>· No spelling or grammar errors |

## 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.