

# GIT

## CSE5006 – LAB 1

## TABLE OF CONTENTS

1. Introduction to Git.....	3
1.1. Why Use Version Control?.....	3
1.2. Configuring Git .....	3
2. Starting a Git Repository .....	5
2.1. Cloning an Existing Repository .....	5
2.2. Starting a New Repository.....	8
3. GitHub .....	11
3.1. Creating your GitHub Account .....	11
3.2. Login on the Command Line .....	11
3.2.1. Creating a GitHub Personal Access Token .....	11
3.2.2. Logging in with GitHub using Git via Command Line .....	13
3.3. Creating a Repository .....	16
3.3.1. Create the Repository .....	16
3.3.2. Clone the Repository .....	16
3.4. Forking a Repository.....	16
3.4.1. Exercise 1.....	18
3.5. Using Branches .....	18
3.5.1. Exercise 2.....	19
4. Summary .....	20
4.1. Git Quick Reference.....	20

## 1. INTRODUCTION TO GIT

### 1.1. WHY USE VERSION CONTROL?

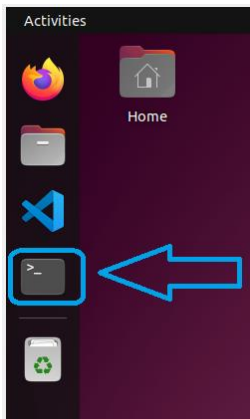
The easiest way to organize a software project is to have a directory with all the source files. When you want to make a change, open the file in a text editor and save it. But what if you introduce a bug and need an old version? Copy-pasting files and folders is tedious.

VCS helps track code changes. Commit as you develop, like saving a snapshot. VCS allows reverting to earlier commits, detailed backups, analysing project history, integrating changes from others, working on different versions, and identifying code contributors. To find out more about what is VCS, you can find it in the [link here](#).

These days, you will find that almost all software is developed using version control, with Git being the dominant tool used to facilitate this. We will be using Git as our VCS in this subject.

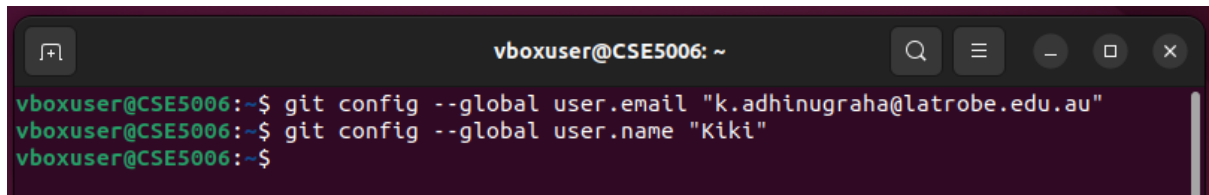
### 1.2. CONFIGURING GIT

Before we proceed, configure Git to identify code editors on this computer. This is helpful for tracking changes made by multiple individuals in a project. Run these commands in the terminal.



Fill in your own email and name instead of the ``<your_student_email>`` and ``<your_full_name>`` placeholders.

```
git config --global user.email "<your_student_email>"  
  
git config --global user.name "<your_full_name>"
```

A screenshot of a terminal window with a dark purple background. The window title is 'vboxuser@CSE5006: ~'. It shows three lines of terminal output: the first line is the prompt 'vboxuser@CSE5006:~\$' followed by the command 'git config --global user.email "k.adhinugraha@latrobe.edu.au"'; the second line is the prompt 'vboxuser@CSE5006:~\$' followed by the command 'git config --global user.name "Kiki"'; and the third line is the prompt 'vboxuser@CSE5006:~\$' with no command. The terminal has standard window controls (search, menu, zoom, close) in the top right corner.

```
vboxuser@CSE5006: ~  
vboxuser@CSE5006:~$ git config --global user.email "k.adhinugraha@latrobe.edu.au"  
vboxuser@CSE5006:~$ git config --global user.name "Kiki"  
vboxuser@CSE5006:~$
```

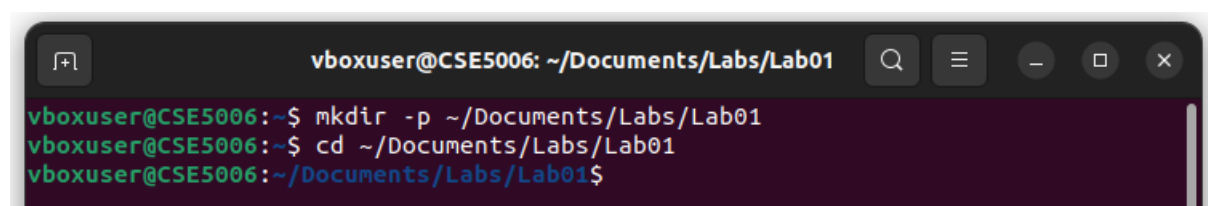
## 2. STARTING A GIT REPOSITORY

In Git, a "repository" refers to all project files and their complete change history. You can get a new repository on your machine by **cloning an existing Git code repository** or **initializing Git in a local project**.

### 2.1. CLONING AN EXISTING REPOSITORY

Open terminal. Create new directory for lab and change into it (using "\$" as prompt):

```
mkdir -p ~/Documents/Labs/Lab01  
  
cd ~/Documents/Labs/Lab01/
```



```
vboxuser@CSE5006: ~/Documents/Labs/Lab01  
vboxuser@CSE5006:~$ mkdir -p ~/Documents/Labs/Lab01  
vboxuser@CSE5006:~$ cd ~/Documents/Labs/Lab01  
vboxuser@CSE5006:~/Documents/Labs/Lab01$
```

To download the "marked" project, you need the Git clone URL: <https://github.com/chjj/marked.git>.

```
git clone https://github.com/chjj/marked.git
```

```
vboxuser@CSE5006:~/Documents/Labs/Lab01$ git clone https://github.com/chjj/marked.git  
Cloning into 'marked'...  
remote: Enumerating objects: 13993, done.  
remote: Counting objects: 100% (2206/2206), done.  
remote: Compressing objects: 100% (156/156), done.  
remote: Total 13993 (delta 2134), reused 2087 (delta 2049), pack-reused 11787  
Receiving objects: 100% (13993/13993), 7.96 MiB | 4.32 MiB/s, done.  
Resolving deltas: 100% (8901/8901), done.  
vboxuser@CSE5006:~/Documents/Labs/Lab01$
```

Great, the repository has now been downloaded to your computer. Let's take a look at what we just grabbed from the Internet.

```
ls
```

```
vboxuser@CSE5006:~/Documents/Labs/Lab01$ ls  
marked
```

The output of `ls` shows that a directory called "marked" has been created and populated with the most recent version of all project files. Let's change into the "marked" directory and explore a little bit more.

```
cd marked
```

```
git log --abbrev-commit --pretty=oneline
```

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/marked$ git log --abbrev-commit --pretty=oneline
cb54906 (HEAD -> master, origin/master, origin/HEAD) fix: Migrate to Typescript (#2805)
929d235 (tag: v5.1.2) chore(release): 5.1.2 [skip ci]
a33ed06 Qbuild v5.1.2 [skip ci]
e465ce4 fix: Add support for Node 16 (#2886)
ab23e19 chore(deps-dev): Bump jasmine from 5.0.2 to 5.1.0 (#2887)
c69b62b chore(deps-dev): Bump @babel/preset-env from 7.22.7 to 7.22.9 (#2880)
76a2103 chore(deps-dev): Bump @semantic-release/github from 9.0.3 to 9.0.4 (#2882)
af881c1 chore(deps-dev): Bump @babel/core from 7.22.8 to 7.22.9 (#2881)
c68be89 chore(deps-dev): Bump rollup from 3.26.2 to 3.26.3 (#2884)
f59fd2a chore(deps-dev): Bump eslint from 8.44.0 to 8.45.0 (#2883)
48daab1 docs: Add raito to the list of tools (#2878)
9a3d089 chore(deps-dev): Bump @babel/core from 7.22.5 to 7.22.8 (#2874)
3851a71 chore(deps-dev): Bump rollup from 3.26.0 to 3.26.2 (#2876)
5a53a95 chore(deps-dev): Bump semantic-release from 21.0.6 to 21.0.7 (#2875)
53ae7bd chore(deps-dev): Bump @babel/preset-env from 7.22.5 to 7.22.7 (#2873)
13cbdf5 chore(deps-dev): Bump @semantic-release/release-notes-generator from 11.0.3 to 11.0.4 (#2872)
19b8ced (tag: v5.1.1) chore(release): 5.1.1 [skip ci]
eaa232d Qbuild v5.1.1 [skip ci]
e6a7184 fix: fix typo (#2870)
16533f5 chore(deps-dev): Bump semantic-release from 21.0.5 to 21.0.6 (#2866)
:...skipping...
cb54906 (HEAD -> master, origin/master, origin/HEAD) fix: Migrate to Typescript (#2805)
929d235 (tag: v5.1.2) chore(release): 5.1.2 [skip ci]
a33ed06 Qbuild v5.1.2 [skip ci]
e465ce4 fix: Add support for Node 16 (#2886)
ab23e19 chore(deps-dev): Bump jasmine from 5.0.2 to 5.1.0 (#2887)
c69b62b chore(deps-dev): Bump @babel/preset-env from 7.22.7 to 7.22.9 (#2880)
76a2103 chore(deps-dev): Bump @semantic-release/github from 9.0.3 to 9.0.4 (#2882)
```

The `git log` command prints a summary of commits that were made to the repository. More on commits later, for now, just understand that we have access to the entire project history, not just the current files.

```
16533f5 chore(deps-dev): Bump semantic-release from 21.0.5 to 21.0.6 (#2866)
a99ca4f chore(deps-dev): Bump rollup from 3.25.2 to 3.26.0 (#2867)
900ff10 chore(deps-dev): Bump eslint from 8.43.0 to 8.44.0 (#2868)
884c782 chore(deps-dev): Bump eslint-plugin-n from 15.7.0 to 16.0.1 (#2859)
805aa9a chore(deps-dev): Bump jasmine from 5.0.1 to 5.0.2 (#2858)
2964347 chore(deps-dev): Bump rollup from 3.25.1 to 3.25.2 (#2857)
21917da chore(deps-dev): Bump @semantic-release/release-notes-generator from 11.0.2 to 11.0.3 (#2856)
3d44697 chore(deps-dev): Bump eslint from 8.42.0 to 8.43.0 (#2855)
17ed99d chore(deps-dev): Bump @semantic-release/npm from 10.0.3 to 10.0.4 (#2854)
):
```

Hit "q" to quit the log.

We don't need a copy of the "marked" project, so you can delete it.

```
cd ..
```

```
rm -rf marked
```

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/arked$ cd ..  
vboxuser@CSE5006:~/Documents/Labs/Lab01$ rm -rf marked  
vboxuser@CSE5006:~/Documents/Labs/Lab01$ ls  
vboxuser@CSE5006:~/Documents/Labs/Lab01$
```

## 2.2. STARTING A NEW REPOSITORY

Now we'll create a new project repository from scratch. Begin by making a new directory for the project and changing into it.

```
mkdir my-awesome-project  
  
cd my-awesome-project
```

Create a new file so that the project isn't empty.

```
echo "My project is awesome, and I'm awesome." > README
```

Now we have a README file with some humble text inside.

```
vboxuser@CSE5006:~/Documents/Labs/Lab01$ mkdir my-awesome-project  
vboxuser@CSE5006:~/Documents/Labs/Lab01$ cd my-awesome-project  
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ echo "My project is  
awesome, and I'm awesome." >README  
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ ls -l  
total 4  
-rw-rw-r-- 1 vboxuser vboxuser 40 Jul 31 15:01 README
```

Let's create a new Git repository so that we can start tracking changes.

```
git init .
```

Read this command as "initialize a Git repository in this directory".

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git init .  
hint: Using 'master' as the name for the initial branch. This default branch nam  
e  
hint: is subject to change. To configure the initial branch name to use in all  
hint: of your new repositories, which will suppress this warning, call:  
hint:  
hint:   git config --global init.defaultBranch <name>  
hint:  
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and  
hint: 'development'. The just-created branch can be renamed via this command:  
hint:  
hint:   git branch -m <name>  
Initialized empty Git repository in /home/vboxuser/Documents/Labs/Lab01/my-aweso  
me-project/.git/  
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$
```

Now we'll ask Git about the current status of our newly created repository.



```
git status
```

Oh no, red text!

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README

nothing added to commit but untracked files present (use "git add" to track)
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$
```

Don't worry, it's not that bad - Git is simply telling us that the README file is untracked. What we need to do is tell Git that we are interested in storing this file in our repository.

```
git add README
```

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git add README
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$
```

Note that you can also add entire directories using "\$ git add .", which is handy if your initial project has a lot of files.

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git add .
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$
```

```
git status
```

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git status
On branch master

No commits yet

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

vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$
```

Ah, better - green text. Changes highlighted in green are "in the staging area". Ready to be committed into repository's history. Now, commit the git and run the command as follows:

```
git commit -m "First commit"
```

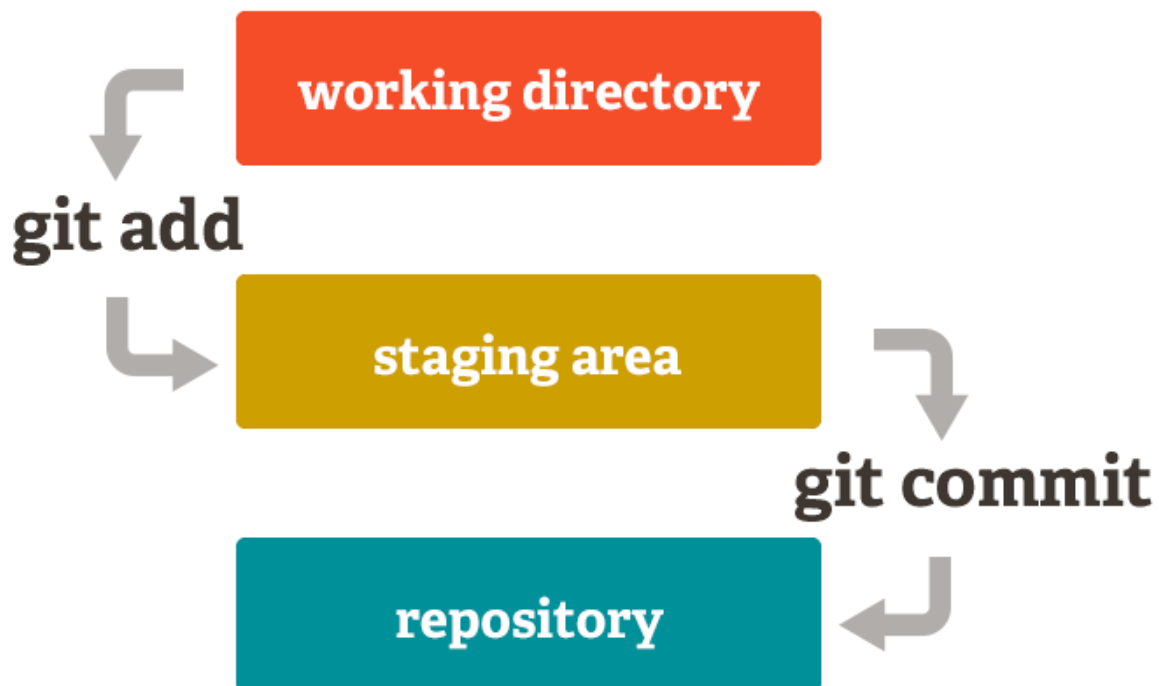
```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git commit -m "First commit"
[master (root-commit) 0497101] First commit
1 file changed, 1 insertion(+)
create mode 100644 README
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$
```

Read this command as "commit all changes in staging area into repository with message 'First commit'." Commits update repository and describe file additions, removals, or modifications. Write concise and informative commit messages summarizing changes since previous commit. No previous commit in this case, so it's not important. Run the following command:

```
git status
```

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git status
On branch master
nothing to commit, working tree clean
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$
```

Git will report the working directory is clean. Congrats, you saved a snapshot of the project state into the Git repository. If you struggle with Git areas, see the git visualization as a diagram below:



When you edit, create, or delete a file, the change is made in the working directory. To inform Git about a change, add it to the staging area. Adding a change to the staging area does not affect the repository itself. You can use `git add` multiple times as you work. Once you complete a unit of work, commit all changes in the staging area to the repository. This creates a "checkpoint" in the project's history that you can revert to later if needed. Making a commit clears the staging area, allowing you to repeat the process.

### 3. GITHUB

In the previous section, we created a new Git repository locally. However, we want to store a copy of the repository online for others to download or collaborate. There are options for hosting Git repositories on the Internet, and we will use GitHub.

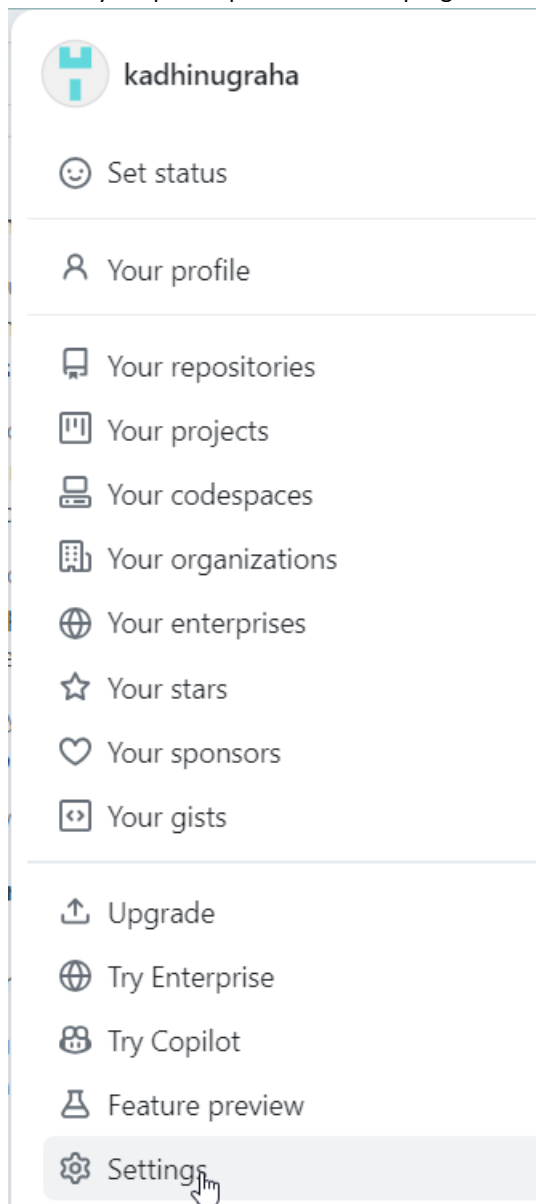
#### 3.1. CREATING YOUR GITHUB ACCOUNT

Go to <https://github.com> and create a new account or use an existing account if you would like.

#### 3.2. LOGIN ON THE COMMAND LINE

##### 3.2.1. CREATING A GITHUB PERSONAL ACCESS TOKEN

1. Go to the GitHub website <https://github.com/> and log in to your account.
2. Click on your profile picture in the top-right corner of the screen and select Settings.



3. On the left sidebar, click on Developer settings.

Codespaces

Packages

Copilot

Pages

← Saved replies

---

Security

Code security and analysis

---

Integrations

Applications

Scheduled reminders

---

Archives

Security log

Sponsorship log

---

<> Developer settings

**Social accounts**

Link to social profile

Link to social profile

Link to social profile

Link to social profile

**Company**

You can @mention your company's GitHub organization to link it.

**Location**

☐ Display current local time  
Other users will see the time difference from their local time.

All of the fields on this page are optional and can be deleted at any time, and by filling them out, you're giving us consent to share this data wherever your user profile appears. Please see our [privacy statement](#) to learn more about how we use this information.

Update profile

4. In the left sidebar of the next screen, click on Personal access tokens and then tokens classic.

Settings / Developer Settings

GitHub Apps

OAuth Apps

Personal access tokens

Fine-grained tokens

Tokens (classic)

**GitHub Apps**

Want to build something that integrates with the GitHub API. You can also read more about GitHub Apps.

5. Click on the Generate new classic token button.

GitHub Apps

OAuth Apps

Personal access tokens

Fine-grained tokens

Tokens (classic)

Personal access tokens (classic)

Tokens you have generated that can be used to access the GitHub API.

Generate new token (classic)

For general use

6. Give your token a descriptive name in the Note field.

7. In the Select scopes section, select the repo checkbox to add all repo related permissions.

## New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

### Note

CSE3CWA-5006

What's this token for?

### Expiration \*

Custom...

30/11/2023

### Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

8. Scroll down and click on the Generate token button.



9. You will now see your new token. Make sure to copy the token and store it safely; you will not be able to see it again!

## Personal access tokens (classic)

Generate new token ▼

Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp\_

Copy

Delete

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

### 3.2.2. LOGGING IN WITH GITHUB USING GIT VIA COMMAND LINE

Now that you have your personal access token, you can use it to authenticate your Git operations via the command line.


1. Go to <https://github.com/new> to create a new repository.
2. Set the name as 'my-awesome-project'.

3. Make sure the repository is public.

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)



Required fields are marked with an asterisk (\*).

Owner \*  kadhinugraha / Repository name \*

✓ my-awesome-project is available.






Great repository names are short and memorable. Need inspiration? How about **stunning-octo-potato** ?


Description (optional)


- ☒  **Public**  
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**  
You choose who can see and commit to this repository.

4. Click the create repository button.



**Create repository**

 my-awesome-project (Public)  Pin  Unwatch 1  Fork 0  Star 0

 **Set up GitHub Copilot**  
Use GitHub's AI pair programmer to autocomplete suggestions as you code.

 **Invite collaborators**  
Find people using their GitHub username or email address.

**Quick setup — if you've done this kind of thing before**

 Set up in Desktop or **HTTPS** **SSH**  

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

**...or create a new repository on the command line**

```
echo "# my-awesome-project" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/kadhinugraha/my-awesome-project.git
git push -u origin main
```

5. Open the terminal/command prompt.
6. Navigate to your local 'my-awesome-project' Git repository.
7. Add your GitHub repository as a remote repository with the command replacing **username** with your GitHub username:

```
git remote add origin https://github.com/username/my-awesome-project.git
```

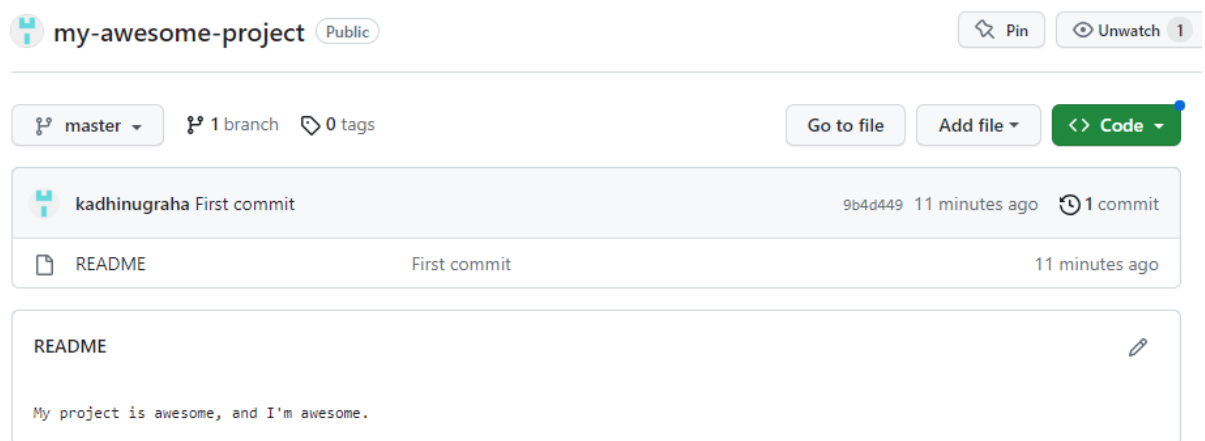
```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git remote add origin https://github.com/kadhinugraha/my-awesome-project.git
```

- When you need to perform an operation that requires authentication (e.g., git push), you will be asked for your username and password:

```
git push --set-upstream origin master

vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git push --set-upstream origin master
Username for 'https://github.com': kadhinugraha
Password for 'https://kadhinugraha@github.com': 
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 248 bytes | 248.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/kadhinugraha/my-awesome-project.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

- Enter your GitHub username as the username.
- Use your personal access token as the password.
- Refresh your browser to see the content of your first repository



Remember, your personal access token should be kept secret, just like your password. If you believe your token has been compromised, you can easily create a new one and delete the old one from the GitHub settings.

#### Notes:

- If you use your github password, you might see the following output.

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git push --set-upstream origin master
Username for 'https://github.com': kadhinugraha
Password for 'https://kadhinugraha@github.com': 
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repo
sitories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/kadhinugraha/my-awesome-project.git/'
```

Do not use your account password to login.

- If you didn't create the remote repository in the first place, you might see the following output

```
vboxuser@CSE5006:~/Documents/Labs/Lab01/my-awesome-project$ git push --set-upstream origin master
Username for 'https://github.com': kadhinugraha
Password for 'https://kadhinugraha@github.com': 
remote: Repository not found.
fatal: repository 'https://github.com/kadhinugraha/my-awesome-project.git/' not found
```

### 3.3. CREATING A REPOSITORY

#### 3.3.1. CREATE THE REPOSITORY

Just like before, except with a few steps to help us with cloning!

1. Sign into your GitHub account at <https://github.com/>.
2. Once you're logged in, click the '+' icon in the upper right corner next to your profile picture and select 'New repository'.
3. Name your repository in the 'Repository name' field. This name will be used in the URL for the repository on GitHub. Optionally, you can add a description in the 'Description' field.
4. Decide if you want your repository to be public (visible to everyone) or private (visible only to you and people you invite).
5. Initialize your repository with a README.
6. Click the 'Create repository' button.

#### 3.3.2. CLONE THE REPOSITORY

Now that you have a repository on GitHub, you can clone it to your local machine using Git from the command line.

1. Navigate to the main page of your repository on GitHub.
2. Above the list of files, click the 'Code' button. This will open a dropdown.
3. In the 'Clone' section, click the clipboard icon to copy the clone URL for the repository. This URL should look like `https://github.com/username/repo.git`, where 'username' is your GitHub username and 'repo' is the name of your repository.
4. Open your terminal/command prompt.
5. Navigate to the location where you want the cloned directory to be made.
6. Type `git clone`, paste the URL you copied in step 3, and press enter. The command should look like `git clone https://github.com/username/repo.git`.
7. Your local clone will be created.

Remember, if the repository is private, you'll need your GitHub credentials to clone it. If you're using token-based authentication, as described in the previous instructions, you'll use your personal access token as the password when prompted.

### 3.4. FORKING A REPOSITORY

**Forking a repository on GitHub** creates a copy of that repository under your GitHub account. This allows you to freely experiment with changes without affecting the original project.

Here are the instructions on how to fork the "`https://github.com/CSE5006/lab-1`" repository:

1. Open your web browser and navigate to the repository's URL: <https://github.com/CSE5006/lab-1>.
2. At the top-right corner of the page, you'll see a button labelled Fork. Click on this button.
3. Wait for GitHub to create the fork. This process usually takes a few seconds.
4. Once the forking process is complete, you'll be automatically redirected to your new forked repository. The web address will look something like "`https://github.com/YourUsername/lab-1`", where "YourUsername" is your GitHub username. You can confirm that the fork was successful by looking at the description above the file list, which should say "forked from CSE5006/lab-1".



Now you have a forked copy of the original repository in your GitHub account. You can clone it to your local machine, make changes, and push updates just like any other repository you own.

### 3.4.1. EXERCISE 1

Currently, "red" is displayed as blue text on the web page. Open **styles.css** in a text editor (double click on it) and fix this "bug" by changing the color to red. Save the file and refresh the web page to ensure that the color is correct. If all is well, use the following commands to commit the change and push it to the remote repository:

```
git add styles.css

git commit -m "Fixed red text colour"

git push
```

### 3.5. USING BRANCHES

One of Git's powerful features is branches. Branches are parallel universes for code, allowing different versions of a project. They represent alternate realities, not past versions like commits. The main branch is "master" or "main". Branches make it easy to manage adding new features, especially with multiple developers. Let's create a branch for adding a new web page section.

```
git checkout -b new_section
```

Read this command as "check out a new branch called 'new\_section'." "Checking out" is Git terminology for switching to a different branch.

Right-click on `index.html` and select "Open With" and then choose `gedit`. Immediately below the line containing the comment `TODO: Add new section here`, insert the following HTML:

```
<div id="cool-section"></div>
```

This HTML will add a new container element to the page with ID "cool-section". We can use this ID from within the CSS file to give the section some visual styles. Add the following code to the styles of the file.

```
#cool-section {
    border: 4px solid gray;
    background-color: yellow;
    height: 300px;
}
```

Refresh the web page in the browser and ensure that the new section is visible. Now we can use a variant of the `git add` command which allows us to review all file modifications and selectively add them to the staging area, ready to commit.

```
git add -p
```

When asked whether to "Stage this hunk," enter "y" for yes - this will tell Git that you want to add each change to the staging area. Now that the changes are staged, you can commit them and then push them to the remote repository on GitHub.

```
git commit -m "Added a cool new section"
```

```
git push -u origin new_section
```

Voilà! The `new_section` branch has been uploaded to the remote GitHub repository. Now we are going to switch back to the `main` branch.

```
git checkout main
```

Note that we do not need the `-b` option because the `main` branch already exists. Reload the page in the web browser again.

Oh no! Where are my changes! Git overwrote my work! My boss is going to fire me! I'm going to be a nerf herder for the rest of my life! Woah, slow down there buddy, there's no need to panic. Your changes are still all there, they just exist in a different branch.

```
git checkout new_section
```

Refresh again. See? No changes were lost. Now back to the `main` branch.

```
git checkout main
```

In a team scenario, fellow project collaborators would now take a look at the code in the `new_section` branch, and hopefully approve the changes. Once the new feature is approved, it then needs to be merged back into the main branch, `main`. Run the following command to merge `new_section` into `main` (this does not need to be specified explicitly since `main` is the current branch):

```
git merge new_section
```

Confirm that the `<div id="cool-section">...</div>` is in `index.html`, then push to GitHub.

---

### 3.5.1. EXERCISE 2

Create a new branch called ``add_textarea`` and checkout that branch. Make the following change to ``index.html`` so that a text area is shown inside the ``<div id="cool-section">...</div>``:

```
<div id="cool-section"><textarea></textarea></div>
```

Once you have confirmed that the web page now shows a text area within the yellow section, use Git to add and commit the change. Checkout the ``main`` branch, merge the ``add_textarea`` branch into it, and push to GitHub.

## 4. SUMMARY

You can ask Git to print out a visualization of the history of your project, including branching and merging (don't forget that you can hit 'q' to quit when you're done).

```
git log --graph --abbrev-commit --pretty=oneline
```

Congratulations on reaching the end of today's Git journey!

### 4.1. GIT QUICK REFERENCE

Command	Description
<code>git clone &lt;url&gt;</code>	Download a local copy of the Git repository.
<code>git init .</code>	Initialise a new Git repository in the current directory.
<code>git status</code>	Display the current status of the working directory. Shows files that are unstaged or uncommitted.
<code>git add &lt;file&gt;</code>	Stage a file.
<code>git add -p</code>	Go through all modifications and choose whether to stage them.
<code>git commit -m &lt;message&gt;</code>	Commit staged changes.
<code>git remote add origin &lt;url&gt;</code>	Tell Git where the remote repository is located.
<code>git push -u origin &lt;branch&gt;</code>	Update the remote branch specified to match current local branch state.
<code>git push</code>	Same as above after the above command has been used once.
<code>git pull</code>	Update the current local branch to match the remote counterpart.
<code>git merge my_branch</code>	Merge <code>my_branch</code> into the current branch.
<code>git log</code>	Show the repository's commit history.