INFO2180 - Lab 1 (20 marks)

Due Date: Monday, September 25, 2023

For this lab you will be getting comfortable with the basics of the terminal/command line and also learn the basics of Git and Github. You will also create a simple HTML page and style it's content.

Exercise 1: Getting familiar with the command line

The command line is a text interface for your computer. It's a program that takes in commands, which it passes on to the computer's operating system to run.

If you are on Windows, I suggest you try to use "Windows Powershell." If you are using Linux or macOS then I suggest you start by using the default "Terminal" application.

When you first open the Terminal you will typically see your *command prompt* and it looks something like:

`C:\Users\{username}>` on Windows or **`yourusername@yourcomputername:~$`** on macOS/Linux.

For the purpose of this lab and the course we will shorten this and simply use `$` before any of the commands we want to execute at the command line. Note: You don't need to add the $ to the command.

1. The first command we want to try is the pwd command. This stands for *Print Working Directory.*

   ```
   $ pwd
   ```

   You may see something like `/home/{username}/` or `C:\Users\{username}>`

2. Next let us try entering the `ls` command (on Windows you can also try the `dir` command). This command will *List* the contents of the current directory.

   ```
   $ ls
   ```

   So you may see something like:

   ```
   Desktop/    Documents/    Downloads/    Music/    Pictures/
   Videos/
   ```

3. The third command we want to try is the `cd` command. This command stands for *Change Directory*. So at the command line type:

   ```
   $ cd
   ```

   and if you know the name of the directory or path you want to go to you can type:

   ```
   $ cd {childFolderName}
   ```

   ```
   e.g. $ cd Documents/
   ```

   You can also navigate up one directory from the one you are currently in by using "`../`". For example:

   ```
   $ cd ../
   ```

4. Some other commands that you will probably find useful:

   - `mkdir` - “Makes” a directory in the file system at the specified file path. e.g.

     ```
     $ mkdir some-folder
     ```

   - `rm` - Deletes (“removes”) the file at the specified file path. e.g.

     ```
     $ rm index.html
     ```

- `rmdir` - Deletes the directory at the specified path. e.g.

  ```
  $ rmdir some-folder/
  ```

- `cp` - copies files or directories. e.g. Here, we copy the file `styles.css` file and place it in the `some-folder/` directory

  ```
  $ cp styles.css some-folder/
  ```

- `mv` - To move a file into a directory, use `mv` with the source file as the first argument and the destination directory as the second argument. e.g.

  ```
  $ mv somefile.txt my-folder/
  ```

Continue to play around with these commands and get familiar yourself more familiar with the command line/terminal.

Exercise 2: Install Git on your computer

Installing Git on Windows

1. Download the latest Git for Windows installer (https://git-scm.com/downloads).
2. When you've successfully started the installer, you should see the Git Setup wizard screen. Follow the Next and Finish prompts to complete the installation. The default options are pretty sensible for most users.
3. Open a Windows Powershell or Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt). Then verify the installation was successful by running the following command:

   ```
   $ git --version
   ```

   You should then see the version printed. For example: `git version 2.42.0`

4. Run the following commands to configure your Git username and email using the following commands, replacing Emma's name and email with your own. These details will be associated with any commits that you create:

   ```
   $ git config --global user.name "Emma Paris"
   $ git config --global user.email "eparis@atlassian.com"
   ```

Installing Git on MacOS

1. Download the latest Git for MacOS installer (https://git-scm.com/downloads).
2. Follow the prompts to install Git.
3. Open a terminal and verify the installation was successful by typing git --version:

   ```
   $ git --version
   ```

   You should then see the version printed. For example: `git version 2.42.0`

4. Configure your Git username and email using the following commands, replacing Emma's name and email with your own. These details will be associated with any commits that you create:

   ```
   $ git config --global user.name "Emma Paris"
   $ git config --global user.email "eparis@atlassian.com"
   ```

Installing Git on Linux

1. From your shell, install Git using apt-get:

   ```
   $ sudo apt-get update
   $ sudo apt-get install git
   ```

2. Verify the installation was successful by typing:

   ```
   $ git --version
   ```

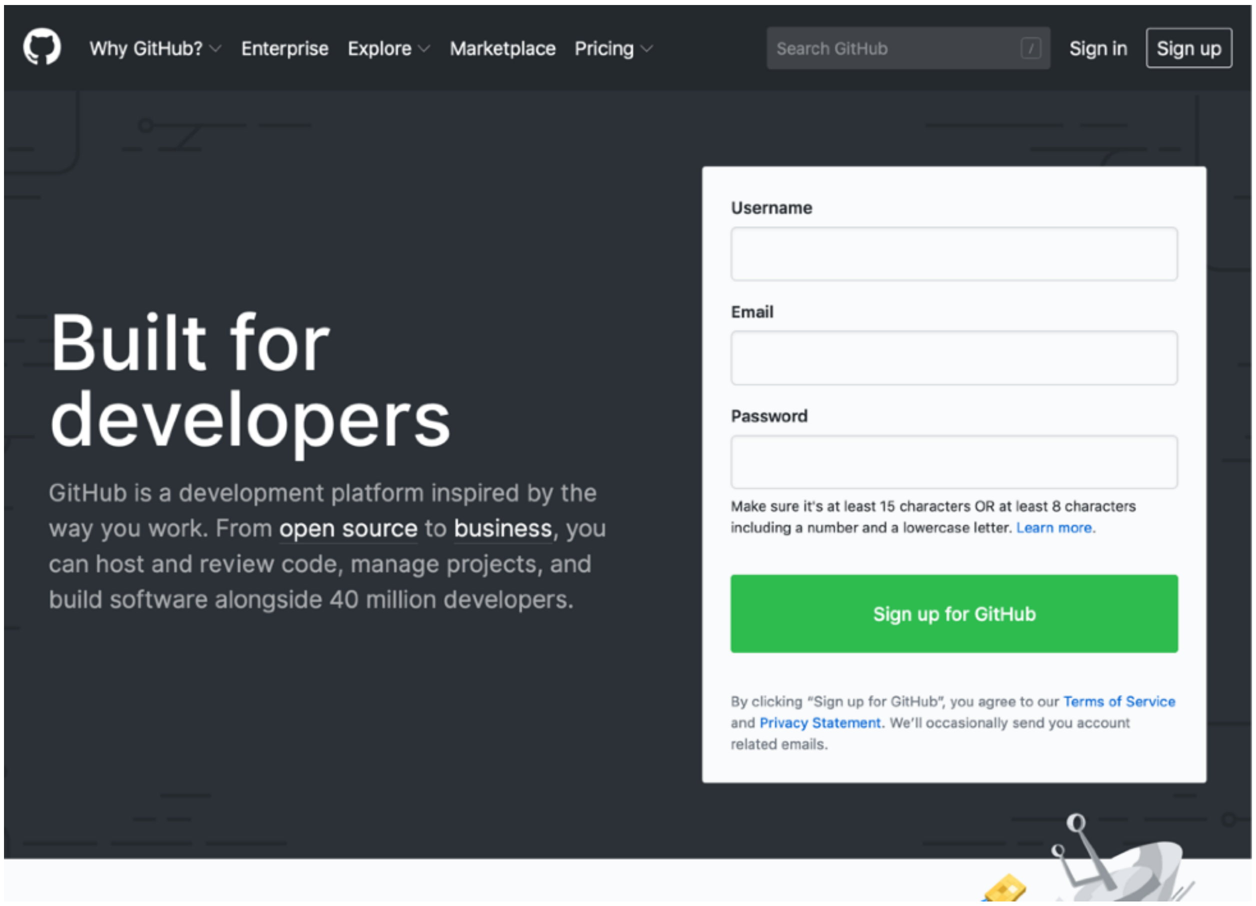
You should then see the version printed. For example: `git version 2.42.0`

3. Configure your Git username and email using the following commands, replacing Emma's name and email with your own. These details will be associated with any commits that you create:

```
$ git config --global user.name "Emma Paris"
$ git config --global user.email "eparis@atlassian.com"
```

Exercise 3: Sign Up for an account on Github

Open your web browser and go to https://github.com and sign up for an account.



Built for developers

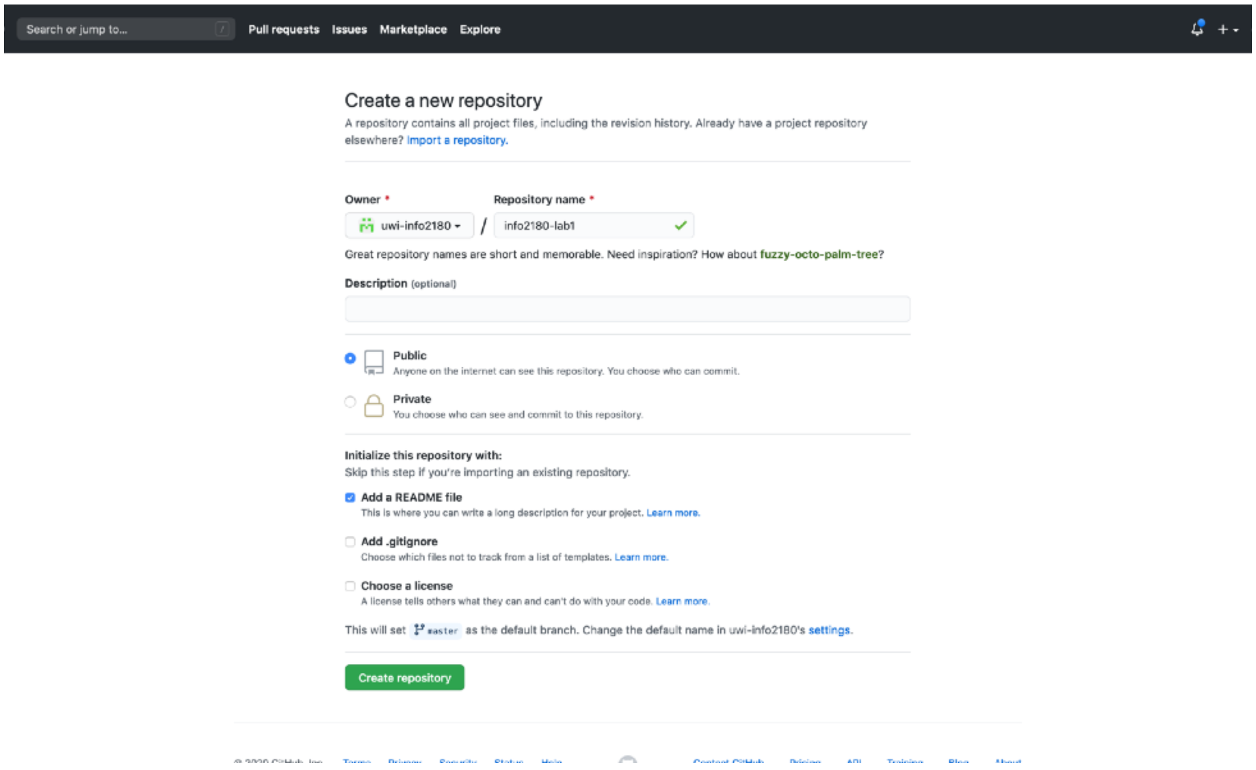
GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 40 million developers.

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.

By clicking "Sign up for GitHub", you agree to our Terms of Service and Privacy Statement. We'll occasionally send you account related emails.

Exercise 4: Create a new Git repository

1. You can create a new Github repository by going to https://github.com/new or by clicking the "+" icon in the top right once you are logged into the website and select "New Repository".
2. For this lab, use the repository name as **`info2180-lab1`**.
3. Ensure that your repository is *Public* and that you select the option to initialize the project with a README file.
4. The remaining options can be left at their defaults. Then click on **Create Repository**.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * uwi-info2180 / Repository name * info2180-lab1 ✓

Great repository names are short and memorable. Need inspiration? How about fuzzy-octo-palm-tree?

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.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☑ Add a README file
This is where you can write a long description for your project. Learn more.

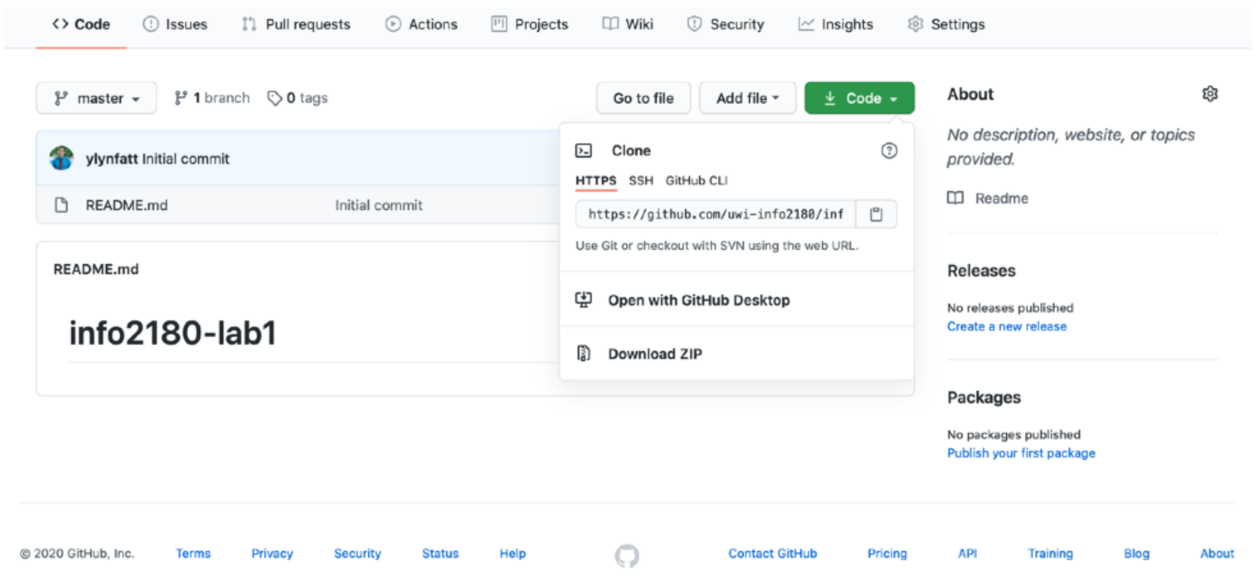
☐ Add .gitignore
Choose which files not to track from a list of templates. Learn more.

☐ Choose a license
A license tells others what they can and can't do with your code. Learn more.

This will set `master` as the default branch. Change the default name in uwi-info2180's settings.

Exercise 5: Clone your repository to your computer

1. On your newly created repository, click on the "Code" button and copy the URL in the box that appears.



2. Open your command prompt, navigate to the folder (e.g. using the **cd** command) where you would like to clone your repository and using the URL you copied in step 1, type the following:

```
$ git clone https://github.com/uwi-info2180/info2180-lab1.git
```

NOTE: Ensure you change the URL and use the one that you copied in Step 1.

3. Then navigate to your newly cloned repository by typing:

```
$ cd info2180-lab1
```

4. You can then type the command `ls` and you should see a `README.md` file.

Exercise 6: Edit the README.md file in the folder on your computer and push it to your Github Repo

1. Edit the file called `README.md` with your text editor of choice.

2. In the file add:

   ```
   # INFO2180 Lab 1

   This is Lab 1 for <Your Name>
   ```

   Of course, change `<Your Name>` to your actual name.

3. Save the file and then at the command line (or Windows Powershell), type:

   ```
   $ git add README.md
   $ git status
   ```

   You should then see that the file has been staged.

4. Commit these changes to your local Git repository:

   ```
   $ git commit -m "Edited README.md"
   ```

5. Check the status of your Git repository:

   ```
   $ git status
   ```

   It should now mention that there is nothing new to commit and that the working tree is clean.

6. Great! Now we should push this information to GitHub.

   ```
   $ git push
   ```

   It may ask you to enter your Github username and password. Enter it and then press enter. If all goes well you can then view your Github repository on the Github website and you should see that the file exists on Github with the changes you made to the file.

7. To see the history of the commits you can run the following command:

   ```
   $ git log
   ```

Exercise 7: Create a branch and Merge it

1. Next, we will learn to create a branch. Branching allows us to create a copy of our code that we can work on in isolation without necessarily affecting main branch of our code (typically called the 'master' or 'main' branch). There are two ways to create a branch:

   ```
   $ git branch <branch-name>
   $ git checkout <branch-name>
   ```

   or

   ```
   $ git checkout -b <branch-name>
   ```

2. Let's create a "`update-readme`" branch

   ```
   $ git checkout -b update-readme
   ```

3. In this branch let us update the `README.md` file with a paragraph on Branching. It should look similar to below:

   ```
   # INFO2180 Lab 1

   This is Lab 1 for <Your Name>

   ## Branching

   Branching allows you to isolate development work without
   affecting other branches in the repository. Each repository
   has one default branch (usually called master), and can have
   ```

```
multiple other branches.
```

4. Now let's see what happens when you type *git status*. Git will inform us whether or not we have new files, and files updated. But, what's the difference between the old file and the new file.

   ```
   $ git diff README.md
   ```

   It shows us what we add and what we change.

5. Let's now go ahead and commit those changes:

   ```
   $ git add README.md
   $ git commit -m "Added new section on Branching"
   ```

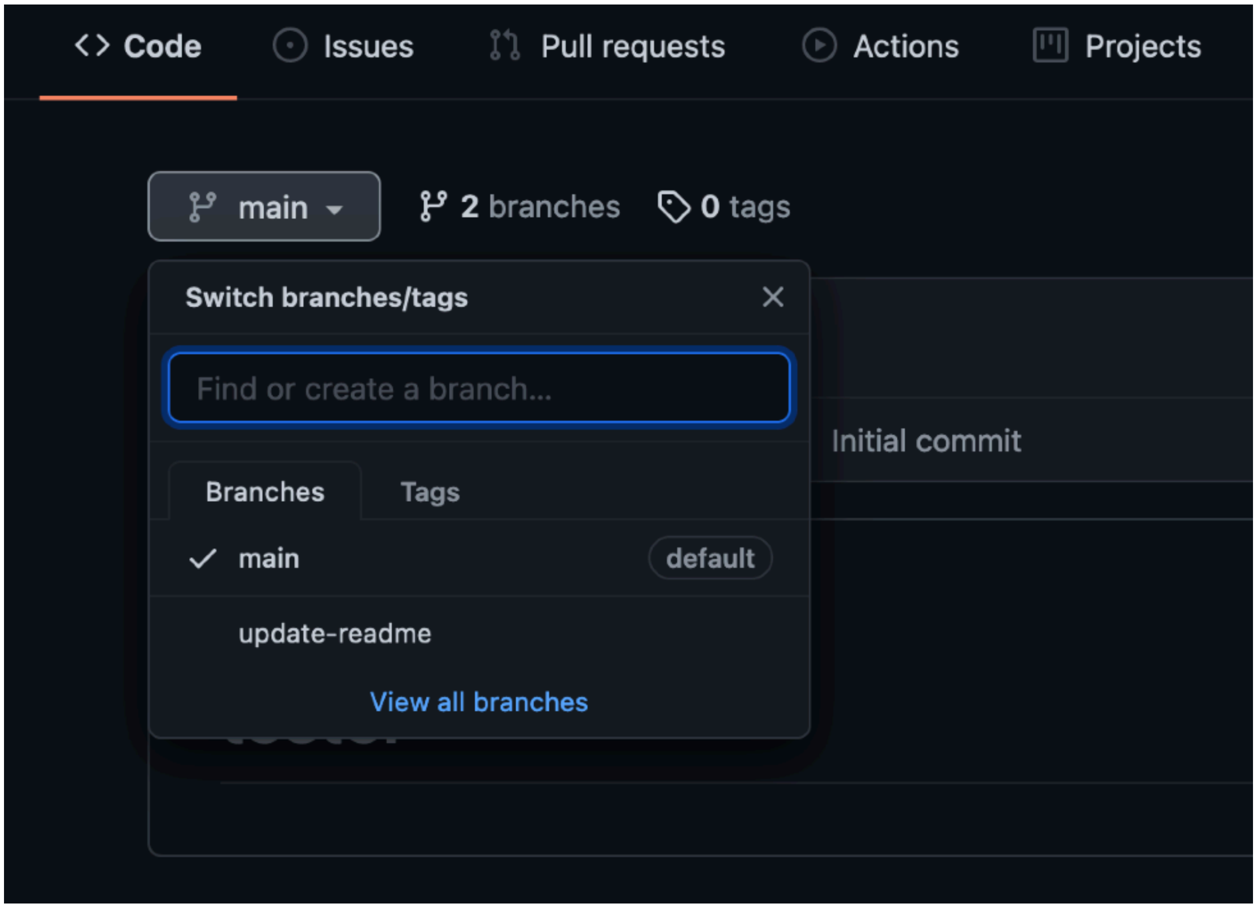
6. Now let us push these changes to Github

   ```
   $ git push
   ```

   It should give you an error, that the current branch does not have an *upstream branch*. All we need to do is setup an association with our local branch with our *origin*, which in this case is Github.

   ```
   $ git push --set-upstream origin update-readme
   ```

7. If we go on Github, you might not see your changes, this is because on Github you are still viewing the *main/master* branch. So you can switch that branch by clicking on the "Branch: main" (or *master*) button and choosing the branch "update-readme". You should now see your changes to the README.md file.



8. Lastly, let us now merge the changes we made in the "update-readme" branch into our "main" branch (or it may be called the "master" branch). To do this on the command line, switch back to your master/main branch using:

   `$ git checkout main` (or `git checkout master`)

   then run:

   ```
   $ git merge update-readme
   ```

   Once that is done, it should merge your changes from the update-readme branch into the main/master branch.

9. Let us now push those changes to Github using:

   ```
   $ git push
   ```

Notice this time we didn't need to do the `--set-upstream` as Github already knows about this branch.

10. If you now look back at your Github repo and switch back to the master branch you should see all the changes that were on your `updated-readme` branch now appearing on your master branch as well.

Exercise 8: Create an HTML file and a CSS file and push to your lab 1 repository.

1. Create a new branch from your `main/master` branch called "`webpage`".
2. Create a HTML file called "`index.html`". In your HTML file ensure you at least have the following (ensuring that you update the sections with your name and the paragraph about yourself):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>INFO2180 Lab 1</title>
</head>
<body>
    <header>
        <h1>INFO2180 Lab 1</h1>
        <p>This is a simple webpage for Lab 1 of INFO2180.</p>
    </header>
    <main>
        <h2>[Your Name]</h2>
        <p>[Paragraph about myself]</p>
    </main>
    <footer>
```

```
        <p>Copyright &copy; 2020 <Your Name>.</p>
    </footer>
</body>
</html>
```

3. Next, ensure you *add*, *commit* and *push* that code to your Github repository.
4. Next, create a CSS file called "`styles.css`" and link it within your `index.html` file using the appropriate `<link>` tag. You are to ensure you style your:
   - Level 1 heading (ie. <h1>) to have a `font-size` of 24px and font `colour` to be #38B2AC.
   - Level 2 heading (ie. <h2>) to have a font-size of 21px and font color to be `#5A67D8`.
   - And your paragraph (ie. <p>) should have a `font-size` of `18px`.
5. Also change the text between the <h2> heading within the <main> tag to have your name and write a paragraph about yourself and why you are interested in Web Development.
6. Ensure you you *add*, *commit* and *push* that code to your Github repository.
7. Next add an image to your folder and add it to your `index.html` file using the appropriate image tag. Please use a relative path (ie. DO NOT use an absolute system path like C:/foldername/myimage.jpg). Also ensure that you name the file in the proper case (ie. any letters that are uppercase should be uppercase and any that are lowercase should be lowercase).
8. Ensure you *add*, *commit* and *push* that code and your image file to your Github repository.
9. Next, switch back to your `main` (or `master`) branch and then merge the changes you made in your "`webpage`" branch back into your `main` branch. Then ensure you *push* those changes back to your Github repository.

Submit your Github repository

1. Locate your Github repository URL and Copy the URL. It should be in the format https://www.github.com/your-user-name/info2180-lab1. For example, if your username is **johndoe**, then the URL you will need to submit is https://www.github.com/johndoe/info2180-lab1.

2. Submit your information via the **“Lab 1”** link on the VLE, with your Github repository URL. (e.g. https://www.github.com/your-user-name/info2180-lab1)