CSI6203 Scripting Languages: Workshop 1

Aim:

To introduce you to a Unix-Like operating system, the Bash command line Interface and the concepts of scripting.

Task 1: Running the environment

Refer to the document in Module 0 for how to connect to the Azure online environment. To complete this workshop task, you will need to access a Unix-like operating system with bash.

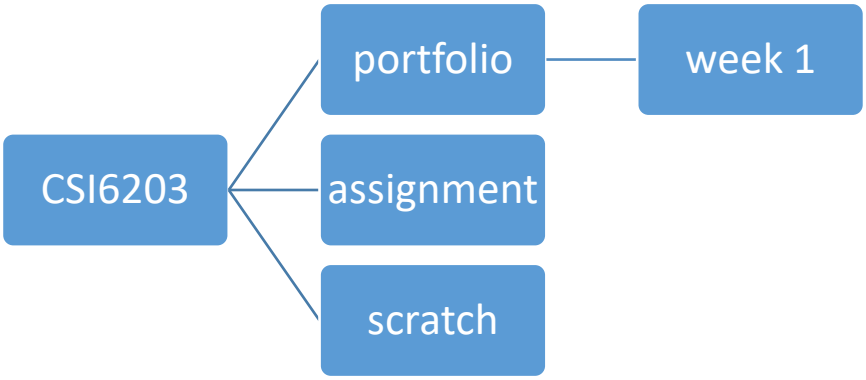
Task 2: Navigating and creating directories using the bash shell

Any task that can be done using a mouse, can also be done (often faster) using the Bash command line interface. You may wish to refer to the "bash command cheat sheet" available on blackboard for help with commands.

In the terminal program, create a new folder in your "home" directory named "CSI6203" using the "mkdir" command.

```
$ mkdir CSI6203
```

Now create the following directory structure using the command line. You may want to use the "cd" command to navigate through the directories to the folder you want.



These are the folders we will be using for this unit.

The "portfolio" directory will be used to store the portfolio assessment tasks which will comprise Assessment Task 2 of the unit.

The "assignment" directory will be used to store your assignment work which will comprise Assessment Task 3 of the unit.

The "scratch" directory will be used for any miscellaneous scripts that are needed.

Do not create any other folders as these may be created via scripts in later weeks.

Task 3: Writing the first script.

Navigate to the “week 1” directory and type

```
$ code .
```

To start Visual Studio Code and open the current directory for editing.

In the code editor, click on the “new file” button in the Explorer Pane and create a file named “first.sh”

This will be our first shell script. A shell script contains a list of bash commands to be executed in order.

write the following script into the file, including the shebang comment.

```
#!/bin/bash
echo "Hi there!"
exit 0
```

When you are done, save your work and return to the terminal.

Task 4: Executing the shell script

Before executing the script, we need to make sure it has the appropriate permissions set. By default, most files will be created without the “execute” permission set. To change the permissions to allow execution. Use the chmod command.

```
$ chmod +x /home/user/CSI6203/portfolio/week1/first.sh
```

Now that it’s executable, to execute the shell script we can write the path of the script into the command line:

```
$ /home/user/CSI6203/portfolio/week1/first.sh
```

This is a lot to type… You can use tab completion to make it easier.

You can also use the built-in bash directory expansions

```
$ ~/CSI6203/portfolio/week1/first.sh
```

or (if the file is located within the current working directory)

```
$ ./first.sh
```

Task 5: getting the exit code

The exit status of a program or script can be used to see whether it successfully completed, or to retrieve information from the script after it has completed. The exit code of the last script to be run in bash is stored in a special variable called `$?`

After running the script, print out the exit code using

```
$ echo $?
```

You should see that it returns the value `0`

0 is generally used to indicate that a script finished successfully, however you can change the exit code by returning different values from the "exit" command. This can allow your scripts to report errors.

Task 6: Different exit codes

Try changing the exit code to something else and then printing the $? variable again. Exit codes must be a number and generally refer to some kind of error (such as a missing file or invalid operation).

```
#!/bin/bash
echo "Hi there!"
exit 22
```

Task 7: Command line arguments

Scripts can receive input from the command line as they are executed. Each argument typed into the command line is referred to by a special numeric variable.

The first argument typed is $1 the second is $2 etc.

Try creating a new script named "second.sh" with the following code.

```
#!/bin/bash
echo "Hi there!"
echo "It's good to see you $1!"
exit 0
```

When you execute this code, you can provide the value for $1 by typing it into the command line after the path to the script. Don't forget to set the execute permissions with chmod first!

```
$ ./second.sh Rob
```

You can type your own name in place of "Rob" when executing the script!

Task 8: git and github

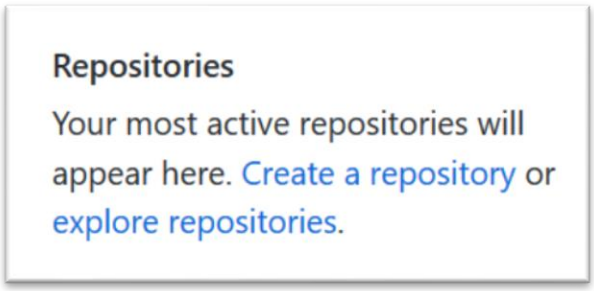
To save your work and keep track of changes, We're going to use git for version control.

For an overview of git and some simple git commands, you can refer to the following reading:

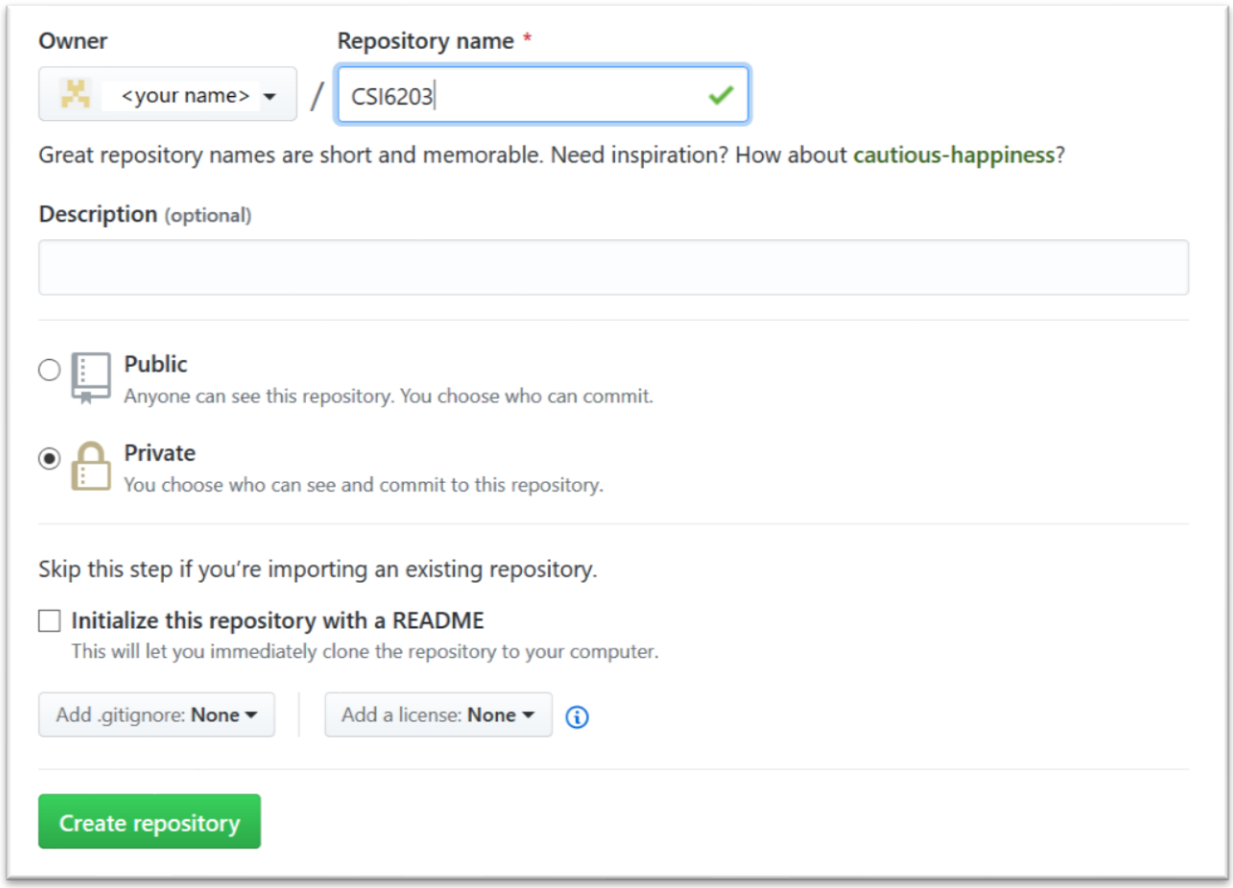
https://guides.github.com/introduction/git-handbook/

To start with, log into your github account at https://github.com. If you have not already, you may need to register for github. Instructions for this can be found in Module 0.

From your github page, click on the “Create a repository” link:



Next, type a name for the repository and select “Private” as in the image below:



When complete, click on the “Create repository” button. This will create a repository for you to store all you work for the unit. Github will keep track of your code, back up your work, and remember any changes you make.

Task 9: committing your work to github

Next, return to the command line and navigate to the CSI6203 directory.

```
$ cd ~/CSI6203
```

To add the workshop work to your github repository, we will follow the instructions in github.

First we'll create a readme file for our project using the echo command:

```
$ echo "# CSI6203" > README.md
```

Next, we create the git repository

```
$ git init
```

Then add the files from this workshop

```
$ git add .
```

Remember that "." Refers to the current directory. The above command will recursively add all files within the CSI6203 directory and subdirectories.

Now that we have added the files, we can commit any changes made to the files.

```
git commit -m "Add folder structure and workshop 1"
```

It's a good idea to include a clear commit message every time you make a change to the repository.

Lastly, we should add a connection to github and push our changes to the internet.

Make sure to use the correct .git URL that includes your own username.

When you push your changes, you'll be asked to type your github username and password. The password field is hidden and will not show your password as you type. Type it in anyway and hit enter to finish the push.

```
git remote add origin https://github.com/My-Username/CSI6203.git
```

```
git push -u origin master
```

When the process is complete, your work should appear saved in github: