ET0735 - DEVOPS FOR AIOT
SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING, SINGAPORE POLYTECHNIC

LABORATORY 1: INTRODUCTION TO GIT AND GITHUB

Objectives

By the end of the laboratory, students will be able to

- Perform Configuration Management using the Git / GitHub workflow
- Implement basic Git Command Line operations for the following,
  - Create new Git Repository
  - Commit files
  - Create and switch between branches
- Set up a GitHub account
- Connect to a Remote Git Repository in GitHub
- Push files from Local to Remote GitHub repository
- Create Git submodules to reference external Git/Github repositories

Activities

- Installation and Setup of Git Command Line
- Creation of a new local Git Repository
- Commit Files to Git locally
- Branches in Git (list, create, checkout, merge)
- Create Tags in Git
- GitHub account
  - Create a GitHub account
  - Create a new GitHub repository
  - Push Git Local Repository to Remote GitHub repository
- Create a GitHub Readme file
- Create a Git tag and GitHub Release
- Creating Git submodules

Review

- Successfully execute common Git operations to perform commit, merges, create tags, create branches on a local Git repository
- Create your own personal GitHub account
- Push local Git repository changes to a remote GitHub server
- Create Software releases in GitHub based on Git tags

Equipment:

Windows OS laptop

Procedures:

1 Installation and Setup of Git Command Line

1.1 Before installing git, please check that **.Net Framework** (4.7 or later) is installed in your laptop.

1.2 Download the **Git installer** from the link below, and run the installer.

https://git-scm.com/download/win

When running the Git installer, choose default settings, except the following:

Default Editor:

Notepad

Terminal Emulator to use with Git Bash:

Use Windows' default console window.

1.3 Open a **Command Prompt (cmd)** window.

1.4 At the command line, type the following command to setup a new Git user. Add your name between the "< >".

```
git config --global user.name <your name>
```

Example:

```
git config --global user.name "KweeTeck "
```

1.5 To link an email address to the newly created Git user, type the following command.

```
git config --global user.email <your email>
```

Example:

```
git config --global user.email tan_kwee_teck@ichat.sp.edu.sg
```

1.6 After you have created a new Git user and linked an email address to this user, verify that Git has saved the new user settings using the command below.

```
git config --list
```

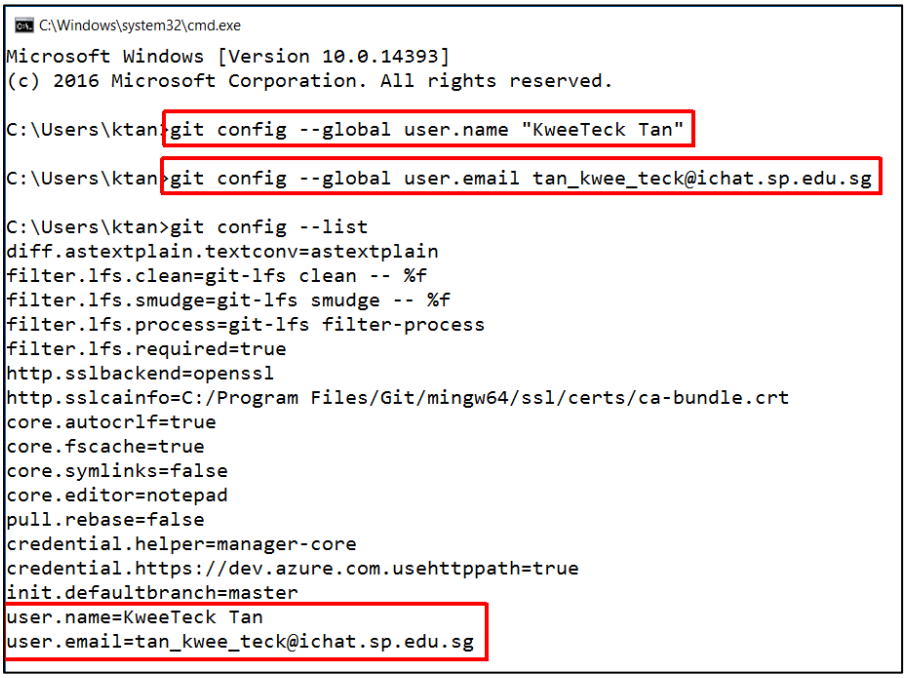


Figure 1 – Initial setup of Git user.

The newly created user name and email are **not** used to authenticate with a remote Git server or services such as GitHub, Gitlab, etc.

Instead Git will append the current user information with user name and email address to each Git "Commit" comment whenever new or modified files are "Committed" to a Git local repository.

You will learn more about Git "Commits" in the next sections of this lab exercise.

2 Creation of a new local Git Repository

2.1 The first step to using Git is to create a new folder to keep files in this directory to be added to the Git repository. Create a new folder in your laptop's hard disk. For example, if you store your data in **C-drive** of your laptop, you may create a new folder "**Local_Git_Repository**" and the directory of this new folder will be **C:\Local_Git_Repository**.

2.2 To create a new Git repository, first change to the directory where you want to create a new Git repository. You may set to a path in your C-drive or other drives in your laptop.

```
cd <new folder's path>
```

Example:

```
cd "C:\Local_Git_Repository"
```

2.3 Once you have changed to the directory where you want to create a new directory for your new Git repository, run the DOS command below

```
mkdir <name of new directory for new Git repository>
```

Example:

```
mkdir lab1
```

```
C:\Local_Git_Repository>mkdir lab1

C:\Local_Git_Repository>cd lab1

C:\Local_Git_Repository\lab1>
```

2.4 After changing to the directory of the new repository, run the following command.

```
git init
```

The "git init" command is used to initialize a new Git repository. It creates a **.git** subdirectory that contains all the necessary files for Git to operate. It's important to note that "git init" only needs to be run once per repository when you are creating a repository. If you've already initialized a repository in a directory, running "git init" again will overwrite your existing repository with a new, empty one.

If the new Git repository has been successfully initialised, you should see the following status in the command prompt:

```
C:\Local_Git_Repository\lab1>
C:\Local_Git_Repository\lab1>git init
Initialized empty Git repository in C:/Local_Git_Repository/lab1/.git/
```

Figure 2 – Initialisation of Git repository.

2.5 Using Windows Explorer, open the Local_Git_Repository folder. You should see that a new folder ".git" is created, as shown below. Take note that ".git" is a "hidden item", therefore please make sure that you configure your File Explorer to show "hidden items". (Procedure: In File Explorer, click "View" and put a tick to "**Hidden items**". You should also tick the "File name extensions" option, so that the full file name of your files will be shown.)

This new ".git" directory contains the Metadata information that tracks the files contained in the Git repository. It maintains the different versions of each file in the repository and also tracks all commits, branches, etc that are done on this repository.

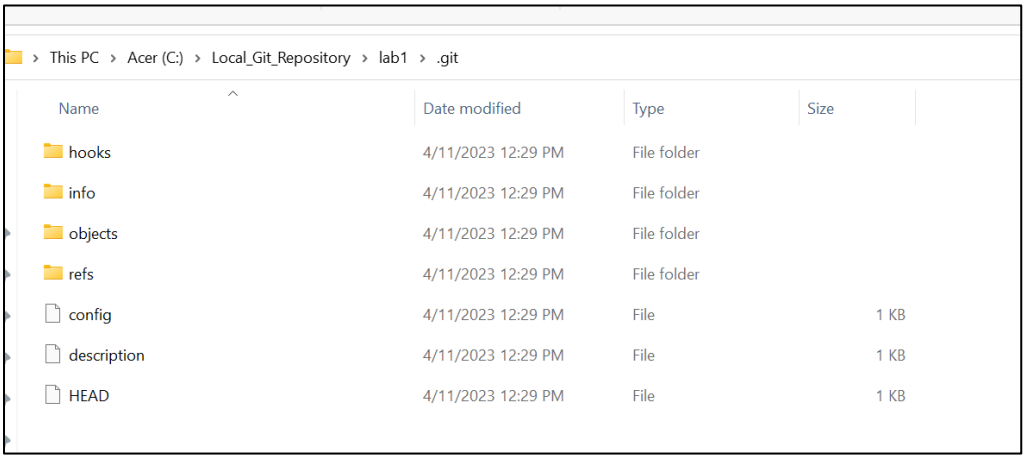


Figure 3 – Folders and files created automatically inside the .git folder.

3 Committing Files to a local Git Repository

After creating a new Git repository, the next step is to add files to this repository.

"Committing" a file to a Git repository means that Git will track the status of this file, if it has been modified, all the different versions of this file that have been previously committed, and also the developers that have committed the specific file versions.

3.1 Using any text editor (e.g. Notepad), create a new Python file "**HelloWorld.py**" with the following Python code in the root directory of your new Git repository.

Figure 4 – A Python file to be created in the Git repository.

3.2 Run the git command below to check the status of the Git repository.

```
git status
```

After running this command, you should see that the new "HelloWorld.py" file is located in the Git repository root directory, but "HelloWorld.py" is listed under "**Untracked files:**". This is because although the file "HelloWorld.py" is physically located in the directory of the Git repository, we have not yet committed this file to the Git repository.

```
C:\Local_Git_Repository\lab1>git init
Initialized empty Git repository in C:/Local_Git_Repository/lab1/.git/

C:\Local_Git_Repository\lab1>git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)

C:\Local_Git_Repository\lab1>
```

Figure 5 – The Python file "HelloWorld.py" is shown as "untracked".

3.3 Before committing a new or modified file (in this case, HelloWorld.py) to the Git repository, you need to add this file to the "staging area" using the following command.

```
git add HelloWorld.py
```

"**Staging files in Git**" is the processing of adding snapshots of the files to the staging area, preparing them for the "Commit" action. Take note that "`git add`" command **does not** commit the file to the Git repository.

3.4 After adding the new file "HelloWorld.py" to the staging area, run the `git status` command to verify that the file has been added to the "Staging Area".

```
git status
```

```
C:\Local_Git_Repository\lab1>git add HelloWorld.py

C:\Local_Git_Repository\lab1>git status
On branch master

No commits yet

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

Figure 6 – The Python file "HelloWorld.py" added to the staging area, but not yet "committed".

Question 1 :

Why is it necessary to have an intermediate "git add" command in the git workflow for a commit?

Now that "HelloWorld.py" has been added to the staging area, we need to "**commit**" this file to the Git repository with a short **text comment** that describes the changes made in this commit.

```
git commit -m "Initial version to display text message
on console"
```

```
C:\Local_Git_Repository\lab1>git commit -m "Initial version to display text message on console"
[master (root-commit) 3aa1907] Initial version to display text message on console
 1 file changed, 1 insertion(+)
 create mode 100644 HelloWorld.py
```

Figure 7 – The Python file "HelloWorld.py" is now committed to the Git repository, with a text message "Initial version...".

Question 2 :

What is the purpose of describing the changes in the commit command?

3.5 After the "commit" is completed successfully, run `git status` again to verify that all files in the current base directory are committed successfully to the Git repository.

```
git status
```

```
C:\Local_Git_Repository\lab1>git status
On branch master
nothing to commit, working tree clean
```

Figure 8 – Nothing else waiting to be committed.

If any existing files in the Git repository is modified, you can use command **`git status`** to see if any files have been modified and not yet committed.

Comparison of Changes in Files

3.6 Let's now modify the HelloWorld.py file and make use of git command to find the differences. Open the HelloWorld.py and append the text " is a DCPE module") to it. The HelloWorld.py file should look like below:

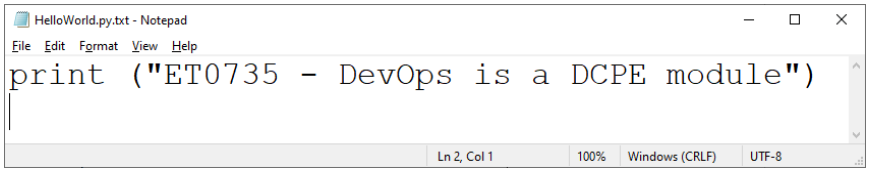


Figure 9 – Modify the HelloWorld.py, appending the text "is a DCPE module" to it.

3.7 Since you made some changes to the already-committed file "HelloWorld.py", then this should be reflected when you run the command `git status` again.

```
C:\Local_Git_Repository\lab1>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   HelloWorld.py

no changes added to commit (use "git add" and/or "git commit -a")
```

Figure 10 – The "HelloWorld.py" file has been modified, so the "git status" command reports it.

3.8 You can use the git command **`git diff`** to view the file differences before deciding to either commit the file again to the Git repository, or continue modifying the file and commit the changes to the Git repository later.

```
C:\Local_Git_Repository\lab1>git diff
diff --git a/HelloWorld.py b/HelloWorld.py
index b4b7362..91c9077 100644
-- a/HelloWorld.py
-- b/HelloWorld.py
@@ -1 +1 @@
-print("ET0735 - DevOps")
+print("ET0735 - DevOps is a DCPE module")
```

Figure 11 – The "git diff" command reports the changes in the HelloWorld.py file.

Question 3 :

What does the green and red colour coded line means when you issue `git diff`?

red - before

green - after

4 Branches in Git

4.1 To view all the branches created in the current Git repository, use the command **`git branch`** without any parameters.

```
git branch
```

This command lists all the available Git branches in the current repository and the current selected branch is indicated with a "*" asterisk prefix and highlighted in green colour below. In the example shown below, "master" is the selected branch.

Figure 12 – The "`git branch`" command lists all the Git branches in the Git repository.

Create new Git branches

You can create new branches in Git repository. Before creating a new branch, first run the `git branch` command to view all the available branches and check that you are in the correct branch that you would like to create a new branch from.

To create a new branch, use the **`git branch`** command **with a parameter** specifying the name of the new branch:

```
git branch <new branch name>
```

4.2 Using the "master" branch as the base branch, create a new branch "bug-fix1" using the command below.

```
git branch "bug-fix1"
```

4.3 After creating the new "bug-fix1" branch, use the `git branch` command to verify that the new branch creation is successful.

Checkout (switch) branches

To switch to a different Git local branch, you need to "**checkout**" that branch, using the **`git checkout`** command:

```
git checkout <branch name>
```

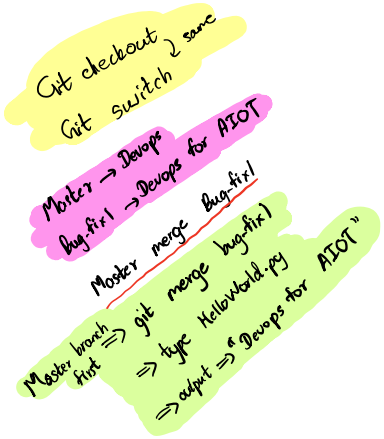
4.4 Now, try switching to the new branch using the `git checkout` command. Then use the `git branch` command to confirm that you have switched to the new branch.

4.5 You should see that the "*" is now placed at the line of "bug-fix1".

```
git branch

git checkout "bug-fix1"

git branch
```




```
C:\Local_Git_Repository\lab1>git branch
* master


C:\Local_Git_Repository\lab1>git branch "bug-fix1"


C:\Local_Git_Repository\lab1>git branch
  bug-fix1
* master


C:\Local_Git_Repository\lab1>git checkout "bug-fix1"
Switched to branch 'bug-fix1'


C:\Local_Git_Repository\lab1>git branch
* bug-fix1
  Master
```

Figure 13 – The "`git`" commands to list, create, and checkout branches.

Question 4 :

Under what circumstances will you use branching in a software development project?

Branching in Git enables developers to work on new features, bug fixes, or experiments without affecting

Merging Git Branches

Git branches can be used for several different reasons, for dedicated bug fixes, new features, software releases, etc.

However, sometimes we will also need to merge the branches into new or existing branches if we need the code changes implemented in one branch in another branch.

To merge a branch into the current branch, use the command:

```
git merge <branch to be merged>
```

4.6 You will now try merging branches. First, check again that you have checked out the "bug-fix1" branch.

4.7 Next, make the small code change in the file HelloWorld.py as shown below.

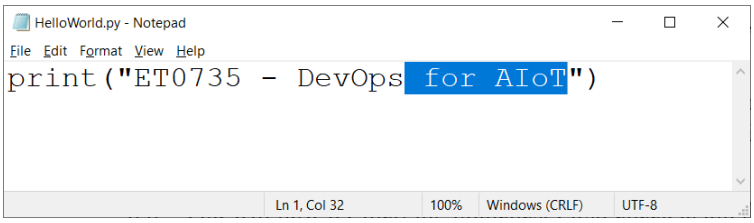


Figure 14 – Add text " for AIoT" to the HelloWorld.py file.

4.8 Commit the changes to the new branch "bug-fix1". Write down the git commands you use using the space provided below. Use the text "Extend the module name" as your commit command's short message.

.

4.9 Now, you are going to merge this change done in the "bug-fix1" branch into the "master" branch. You need to switch back to the "master" branch first.

```
git checkout "master"
```

4.10 Before merging the branch "bug-fix1" to the "master" branch, let's review the original file : HelloWorld.py. Open the HelloWorld.py in any text editor. Take note that the changes you made in "bug-fix1" branch is not reflected here. Close the file after reviewing.

4.11 Merge the branch "bug-fix1" into the current "master" branch using the **`git merge`** command.

```
git merge "bug-fix1"
```

4.12 Run the `git branch` command to list the branches in the Git repository. Notice that the branch "bug-fix1" remains (does not disappear) even after the branch have been merged. Now open HelloWorld.py in any text editor again. The changes you made in "bug-fix1: branch should be reflected in the file now. You have successfully merged the "bug-fix1" branch to the "master" branch.

5 Creating a GitHub account

GitHub is a popular cloud-based **remote Git repository** where developers can upload their software projects for collaboration and sharing with other developers.

In Git, you can use GitHub to "push" your code to GitHub where your code is uploaded to the cloud and can be shared with others.

5.1 First, you will create a free GitHub account. Using the link below, sign up for a free GitHub account.

5.2 Please register the new GitHub account using your **SP iChat email account**.

https://www.github.com/

Write down your Github credentials in the box below for future reference

E-mail	
Github user name	

Create a new GitHub Repository

GitHub organizes code projects based on repositories, so before you can push your code from your local Git repository to GitHub, you first need to create a repository in GitHub.

5.3 After logging into GitHub in the main page, click +∇ and select "**New Repository**" to create a new GitHub repository.

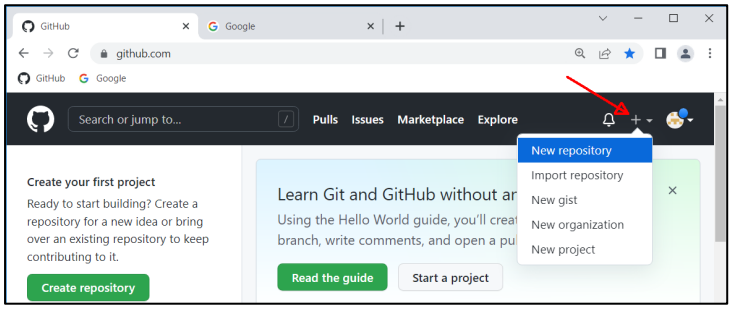


Figure 15 – Create a new GitHub repository.

5.4 Fill in the field for the "Repository name" (use **Lab 1**) and set the repository to "**Public**" (radio button). Click the "**Create Repository**" button at the bottom of the page to continue.

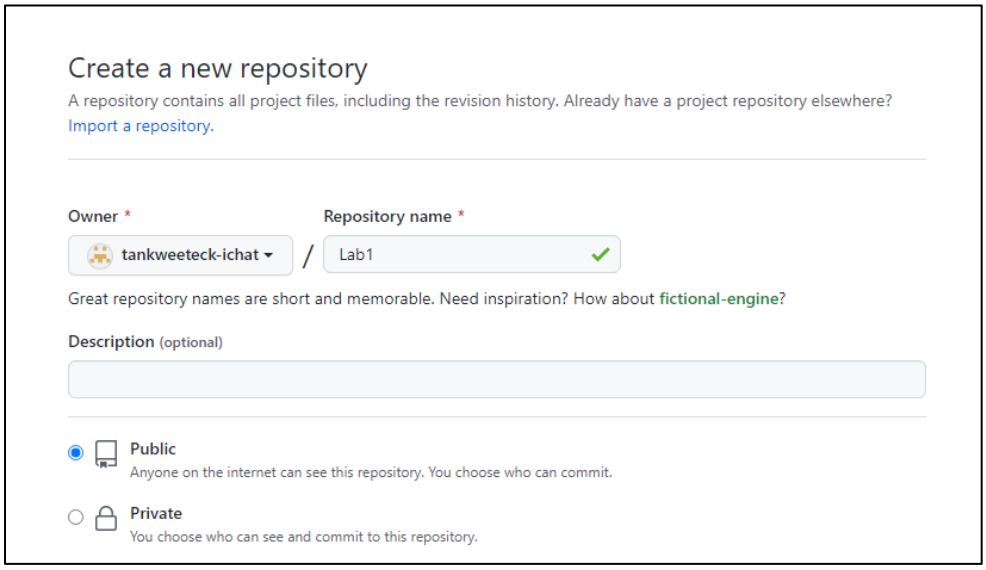


Figure 16 – Fill in the fields of the new GitHub repository.

Push contents from Local Git Repository to Remote GitHub repository

After creating the remote GitHub repository, you can "push" (upload) your Local Git Repository to the new remote GitHub repository.

5.5 In the GitHub repository, under the "Quick setup…"section (see Figure 17), click the copy button  to copy the URL link of the repository. The link should be something like this:

https://GitHub.com/<your name>/Lab1.git

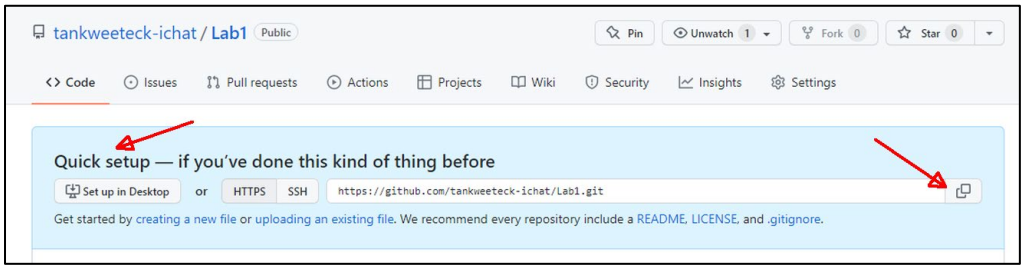
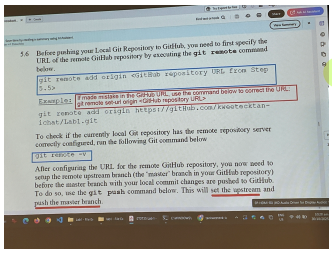


Figure 17 – Copy the URL of the GitHub repository.

5.6 Before pushing your Local Git Repository to GitHub, you need to first specify the URL of the remote GitHub repository by executing the **`git remote`** command below.



```
git remote add origin <GitHub repository URL from Step 5.5>
```

Example:

```
git remote add origin https://gitHub.com/kweetecktan-ichat/Lab1.git
```

To check if the currently local Git repository has the remote repository server correctly configured, run the following Git command below

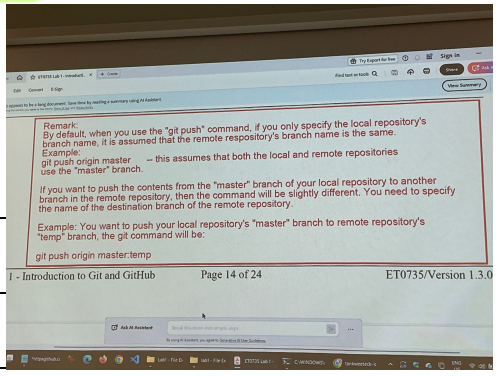
```
git remote -v
```

After configuring the URL for the remote GitHub repository, you now need to setup the remote upstream branch (the 'master' branch in your GitHub repository) before the master branch with your local commit changes are pushed to GitHub. To do so, use the **`git push`** command below. This will set the upstream and push the master branch. to the remote repository.

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

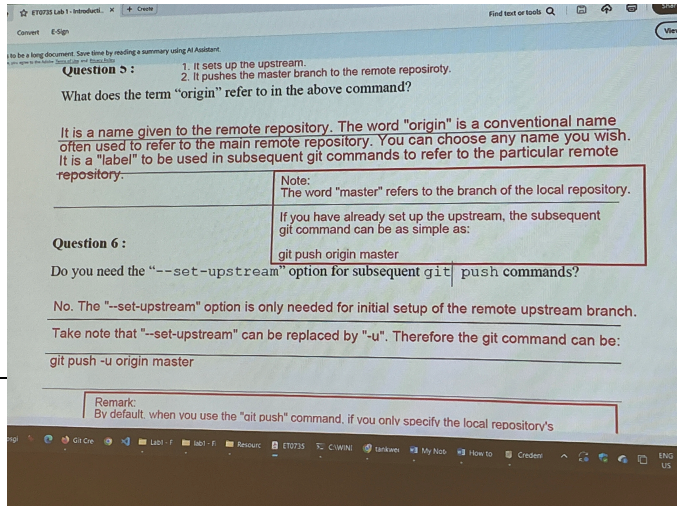
Question 5 :

What does the term "origin" refer to in the above command?



Question 6 :

Do you need the "`--set-upstream`" option for subsequent `git push` commands?



If this is the first time that you push files from any Git repository to a GitHub remote repository, then the following popup might appear (Figure 18). To continue, select "**Sign in with your browser**" option.

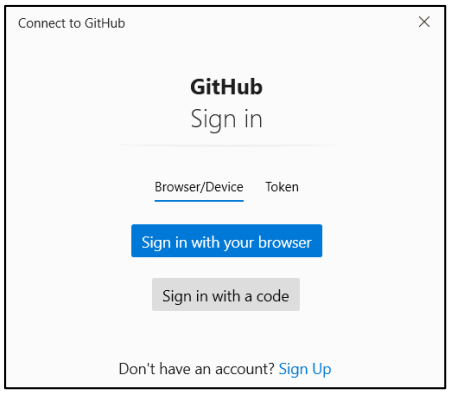


Figure 18 – Choose how you would sign-in to your GitHub repository.

In your browser (Figure 19), enter your GitHub user name and password that you used to register your new GitHub account in Step 5.2.

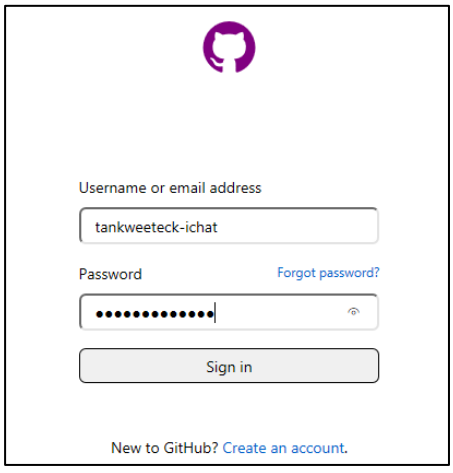


Figure 19 – Sign-in to your GitHub repository via browser.

After successfully logging into GitHub, you should see the web page below.

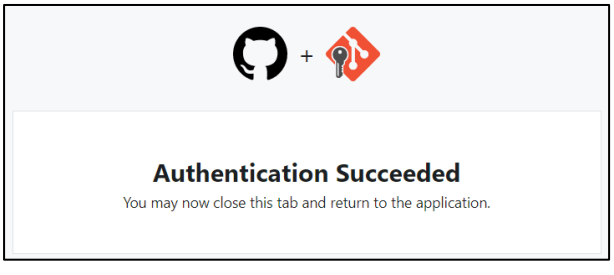


Figure 20 – Sign-in to your GitHub repository via browser.

For subsequent changes in your master branch, you push the Local Git Repository to your remote GitHub repository using the **`git push`** command below.

```
git push -u origin
```

The `git push` command should be successfully completed as shown in the command window below.

```
C:\Local_Git_Repository\lab1>git push -u origin
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 315 bytes | 315.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/mdilyasf/lab1-ay2320.git
   14f23e5..9b71fb1  master -> master
branch 'master' set up to track 'origin/master'.
```

Figure 21 – Completion of `git push` command.

6 GitHub Readme file

For each GitHub repository, it's usually a good practice to create a Readme file which then appears in GitHub as the main webpage for a repository.

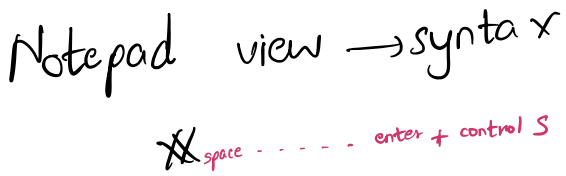
The GitHub Readme file syntax is based on a custom Markdown syntax which is documented in the link below.

https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

In this part, you will create a text file "Readme.md" and push it to your remote GitHub repository.

6.1 Using Notepad, create a new text file "Readme.md" in your c:\Local_Git_Repository\Lab1 directory. Remember to name the file as **Readme.md**, not **Readme.md.txt**. You will be warned that changing the filename extension from might make the file unusable. Ignore the warning.

6.2 In the Readme.md file, add a H1 header with the text "ET0735 – Lab 1 (Introduction to Git and GitHub)" (See Figure 22). Make sure that there is a space between "#" and "ET0735 – Lab 1 (Intro….)". Save the file and close the text editor.



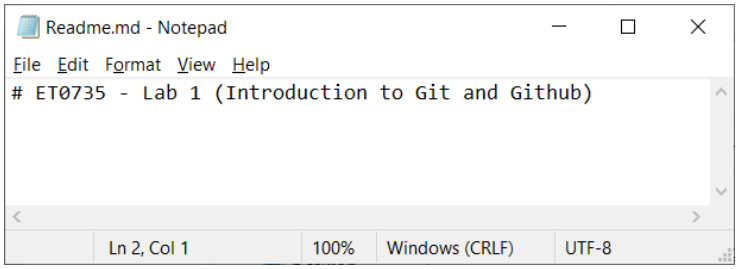


Figure 22 – Content of the "Readme.md" file.

6.3 Commit and push the new Readme.md file to GitHub. Write down the git commands using the space provided below.

Git add Readme.md
git commit -m "Add Readme.md file"
git push origin

6.4 Reload the URL for your GitHub repository page and check that your Readme.md file is now the main HTML page for your Lab1 GitHub repository.

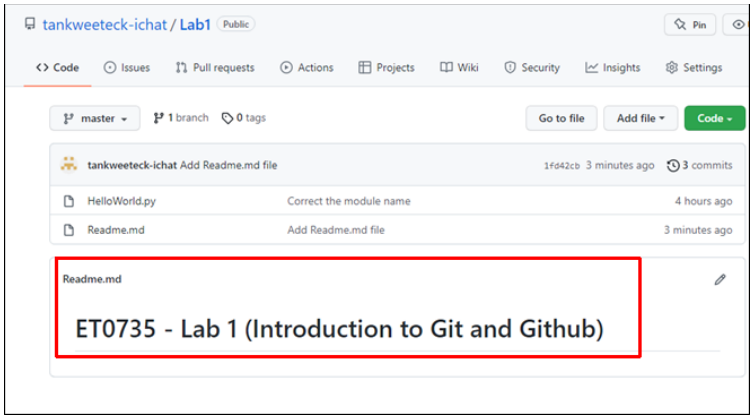


Figure 23 – The Readme.md file is used as the main HTML page for your "Lab1" GitHub repository.

Explore further syntax for readme.md in the following link

https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

Experiment with various syntax codes such as changing the font style, adding links and images to your project.

7 Creating a Git tag and GitHub Release

Create tag in local Repository and upload to github

Tags are used in Git usually to create Software Releases. A Software Release is basically a collection of files with each file having fixed version.

In Git, we use "Tags" to mark these collection of files in a specific branch and these "Tags" are basically a text string such as "v1.0", "v1.1", etc which typically describes the Software Release number.

To create a tag in Git, first check that you are in the correct branch, else switch to the branch (`git checkout`) you want to create the tag on. Once you are in the correct branch, use the **`git tag -a <tag> -m <comment>`** command to create a new Git tag on the current file versions in this branch.

Example:

```
git tag -a v1.0 -m "Initial release v1.0"
```

Now, you are going to create a new tag, and apply the new tag to your Local Git Repository. You will then push the tag to your GitHub repository.

7.1 Before creating a new tag, verify that your GitHub repository shows that it has no tag (i.e. 0 tags).

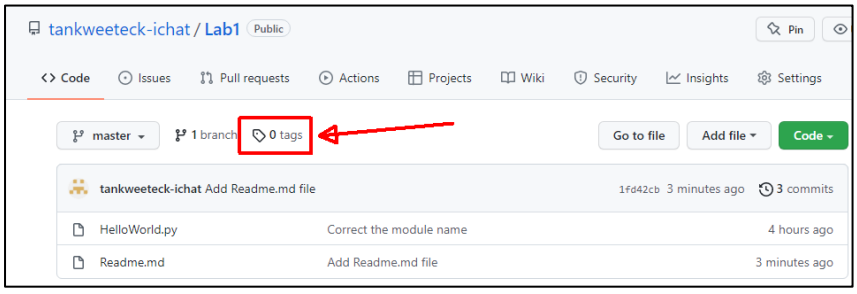


Figure 24 – The "Lab1" GitHub repository has no tags.

git tag
git tag -n

7.2 Check that you are at the "master" branch of your Local Git Repository.

7.3 Create a new git tag "v1.0" using the `git tag` command below.

```
git tag -a v1.0 -m "Initial release v1.0"
```

7.4 You will now push the newly created tag from the local git repository to the remote GitHub repository. The `git push` command used so far for pushing files needs to be modified slightly to push git tags. The modified git push command is shown below.

```
git push origin v1.0
```

```
C:\Local_Git_Repository\lab1>git tag -a v1.0 -m "Initial release v1.0"

C:\Local_Git_Repository\lab1>git push origin v1.0
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 182 bytes | 182.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/mdilyasf/lab1-ay2320.git
 * [new tag]         v1.0 -> v1.0
```

Figure 25 – Push git tag to remote GitHub repository.

7.5 Reload the URL for your GitHub repository page, and observe that now it shows that there is "1 tag".

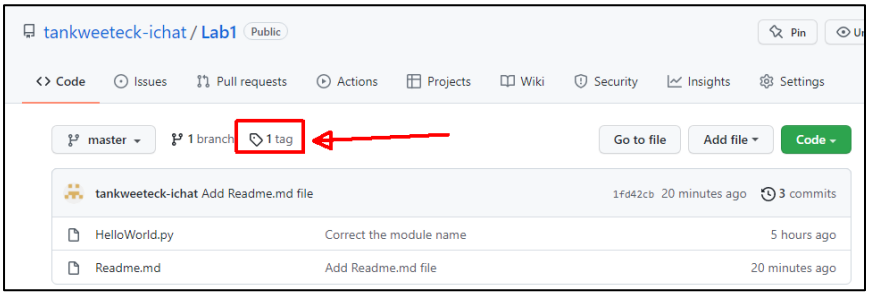


Figure 26 – The "Lab1" GitHub repository now has 1 tag.

7.6 Next, you are going to create a software release v1.0. Click the "1 tag" icon at your GitHub repository, then click "Releases". The page will show "There aren't any releases here" since you have not created any release yet.

7.7 Click the "Create a new release" button. Click "Choose a tag", and select "v1.0". In the "Release title" field, enter "ET0735_Lab1_Code_v1.0" (or whatever text you prefer). In the "Describe this release" field, enter two lines of text: "Initial release v1.0", and "Date: {*today's date*}".

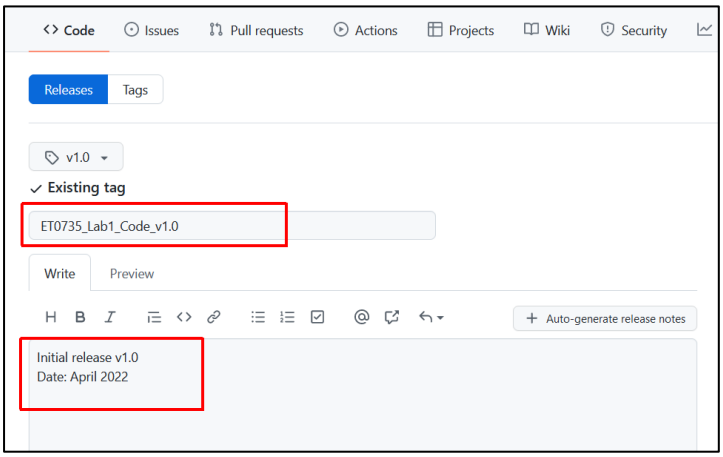


Figure 27 – Fill the "Release title" and "Describe this release" fields.

7.8 Click the "Publish release" button at the bottom.

7.9 Go back to the main page of GitHub repository. You should see a new release.

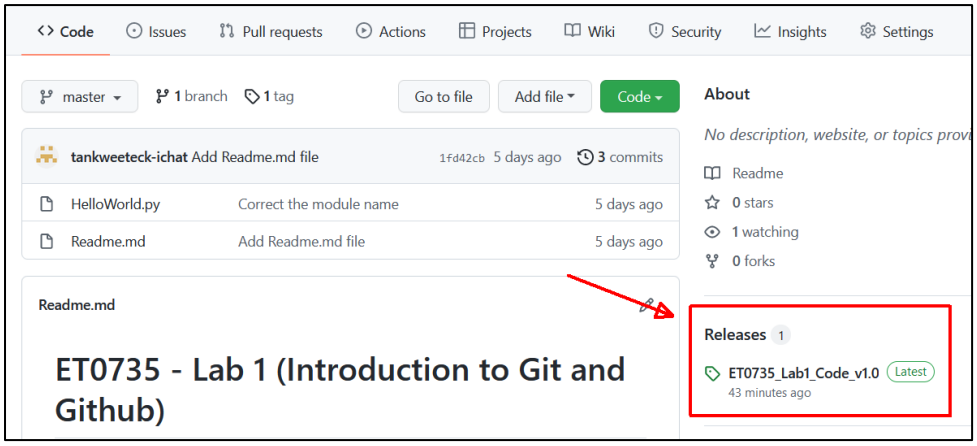


Figure 28 – The "Lab1" GitHub repository now has 1 release.

8 Git Submodules

Until now we have learned how to create a single Git repository which you can use to add, commit and push your files to Git and Github.

However, when working on larger projects such as the Mini-Project it's sometimes necessary to reference external shared Git/Github repositories that are maintained centrally by another team but used in the repositories of other teams.

8.1 Creating Git submodules

To create a Git submodule you first need an existing local Git repository to add the Git submodule into. Git submodules are basically existing Github repositories which are then referenced by your local Git repository.

The Git command syntax is shown below and should be executed at the root level of an existing repository.

```
git submodule add <Git submodule repository URL>
```

Example:

You want to add a Git Submodule from your friend John's github repository https://github.com/JohnProject.git to your local Git repository created at C:\Local_Git_Repository\lab1. You would use the two git commands below:

```
cd C:\Local_Git_Repository\lab1
```

```
git submodule add https://github.com/JohnProject.git
```

8.2 Creating Git submodules from existing Github repositories

In this exercise, we will create a new Git submodule by adding an existing Github remote repository.

Change to the root directory of the Git repository you've created earlier for Lab 1.

Using the command below, we will add a new Git submodule from an existing Github repositttory

```
git submodule add https://github.com/ET0735-DevOps-AIoT/Lab1_submodule.git
```

8.2.1 Additional Git files created for Git submodules

After you've created and added the Git submodules in the previous step, observe that Git has automatically created a new hidden file ".gitmodules" in the root directory of your Lab 1 local repository.

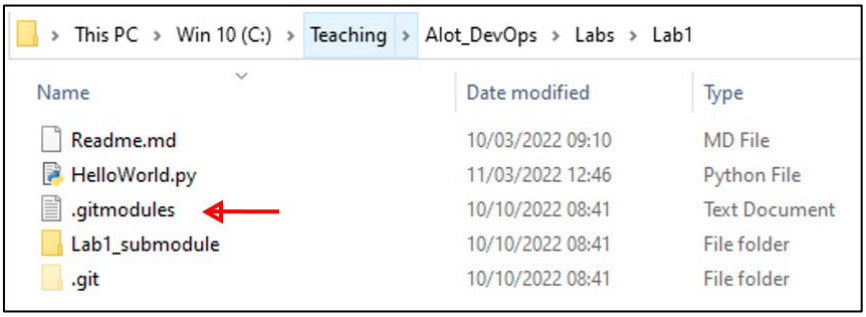


Figure 29 – Local Git repository structure with submodules

.gitmodules is basically a text file which lists the local directory path of the submodule and also its corresponding remote URL. Open the .gitmodules file with NotePad to observe its content.

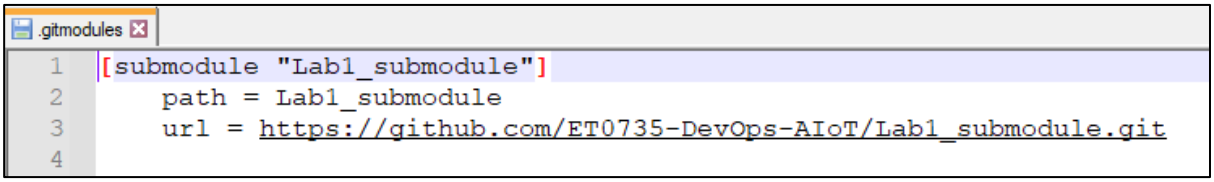


Figure 30 – Content of Git submodules configuration file

8.2.2 Adding Git submodules to Github

At this point all the changes you have done to create and add the Git submodule have been done locally only. The Git Submodule is only added to your local repository, but not yet to your remote repository.

We now need to **commit** and **push** all the changes we have done locally to our remote Github repository.

In the box below, write down the Git commands required to upload your Lab 1 local Git repository containing the new Git submodule you've just created and added.

git add *
git commit -m "Added submodule"
git push origin

After you successfully pushed your updated Lab 1 Git repository to Github, you should also see that the new Git submodules "Lab1_submodule" folder has been added into Github

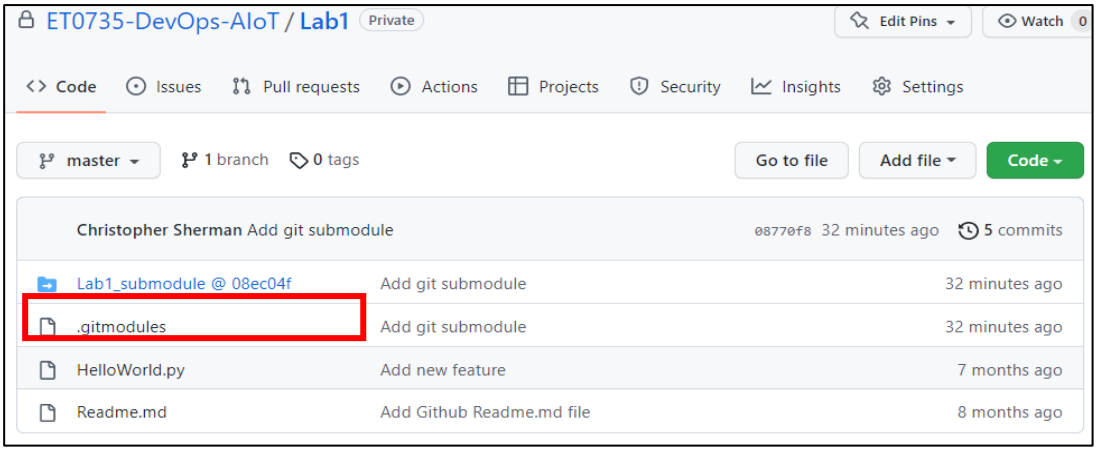


Figure 31 – Git submodules in Github

Also notice that there is a string of hex numbers appended to “Lab1_submodule”

In the box below, write down where this string of hex numbers is derived from.

lab_1 submodule @ 20726d2

20726d2 is a hash code.

8.2.3 Working in a Github repository with multiple collaborators

We have learned how to use Git and Github for a single user to locally and remotely respectively to archive our source code files and perform basic configuration management tasks to control the file versions, repository branches, etc.

However, most software projects are organized as teams of more than 1 developer, therefore Git and Github are usually used to manage the software codebase for a team of software developers.

8.2.4 Adding collaborators in Github repositories

Before we can start working with other developers, we need to grant the developers read and write access to allow them to commit and push files to our github repository.

In your Lab 1 Github repository, go to “Settings” and select “Collaborators and Users”

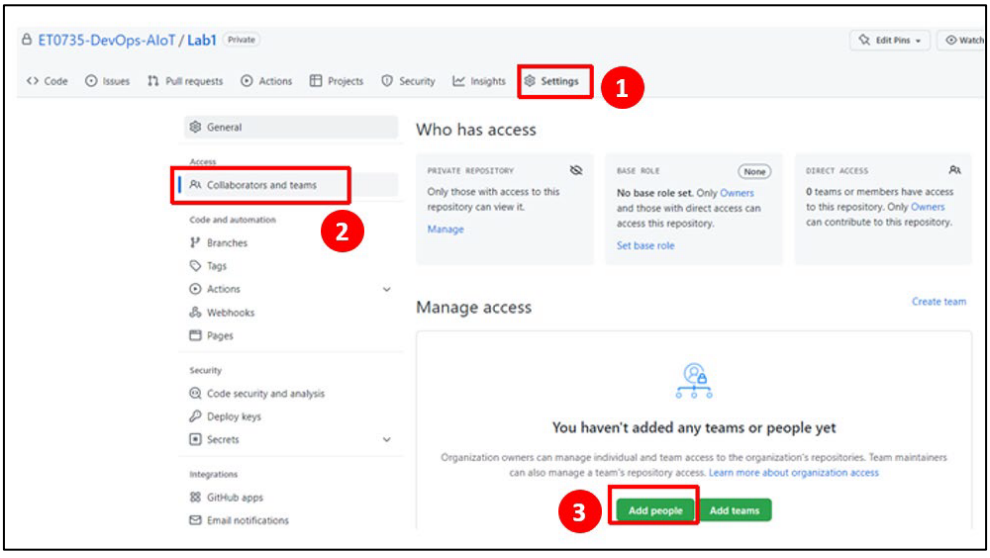


Figure 32 – Adding collaborators into Github

Click “Add people” and then add 1 of your classmates into your Github Lab 1 repository.

8.2.5 Committing and Pushing changes to shared Github repositories

After your classmate has added you as a collaborator into their Lab 1 Github repository, you can already modify and add files to their Github repository.

The first step is to first clone your classmate's Lab 1 Github repository locally onto your laptop. Change to C:\ drive and create a new empty folder "**clone_repo**" (i.e. the directory is "c:\clone_repo"). Open a CMD prompt, and change directory to this new folder using the command:

```
cd c:\clone_repo
```

Next, clone the classmate's Lab 1 Github respository with the Git command below.

```
git clone <URL of repository to clone>
```

Write down in the box below the full git command to clone your classmate's Lab 1 Github repository

git clone - link

After cloning your classmate's Lab 1 repository, modify the Python file "HelloWorld.py" locally to add a second line to the file:

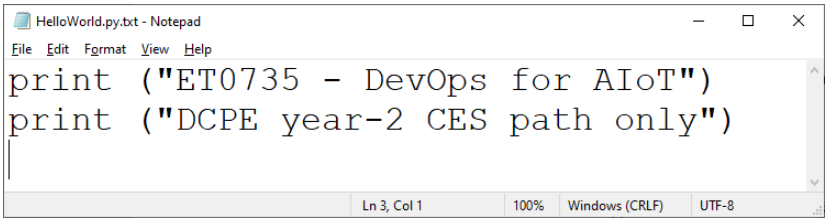


Figure 34 – Add a second line to the HelloWorld.py.

Add, commit and Push your modifications the "master" branch in your classmate's Github repository for Lab 1 and write down all the commands used in the box below

git add HelloWord.py

git commit -m "Added second line to the file"

git push origin

Finally you use the Git command below to view and verify the commit history in your classmate's Lab 1 Github repository.

```
git log
```