

Practical 5 : Working with Git (Additional Topics)

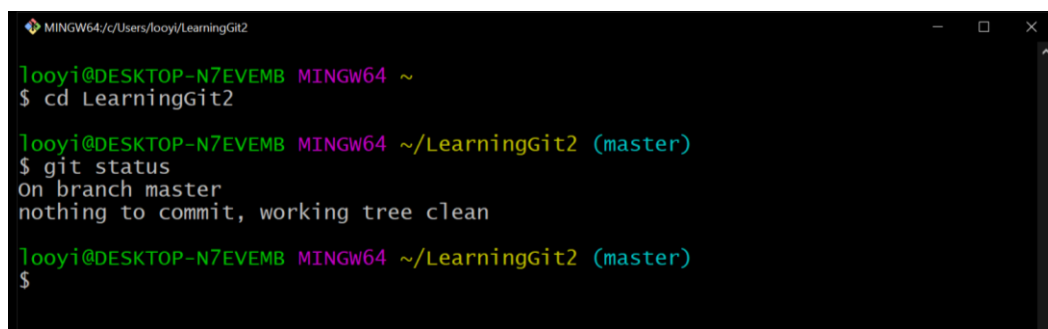
Continuing from the previous practical, this practical is exploring more on Git with some additional topics of Git which are essential to be learnt.

Git Merge vs Git Rebase.

Git Merge was discussed previously to merge changes from specific branch into master or changes from local to remote repo or vice versa. There is however, another way of merging in Git, which is rebase. In order to look into the workflow of rebase, let's work on the LearningGit2 repo. For this exercise, **Git Bash** application will be used rather than the normal or Git command prompt.

Creating a Branch, then Merge.

Thus far, the LearningGit2 local repo has only master branch as shown in Figure 1.



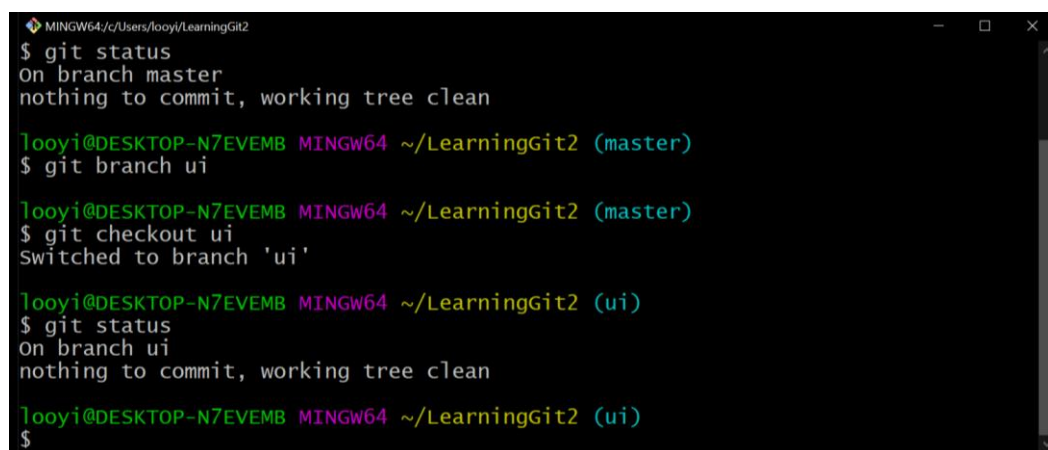
```
MINGW64~/Users/looyi/LearningGit2
looyi@DESKTOP-N7EVE MB MINGW64 ~
$ cd LearningGit2

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git status
on branch master
nothing to commit, working tree clean

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$
```

Figure 1: LearningGit2 status.

In order to create a branch, execute `git branch [branch name]` command and to switch to the branch that was created, execute `git checkout [branch name]` as illustrated in Figure 2.



```
MINGW64~/Users/looyi/LearningGit2
$ git status
on branch master
nothing to commit, working tree clean

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git branch ui

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git checkout ui
Switched to branch 'ui'

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$ git status
on branch ui
nothing to commit, working tree clean

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$
```

Figure 2: Create branch and switch to created branch

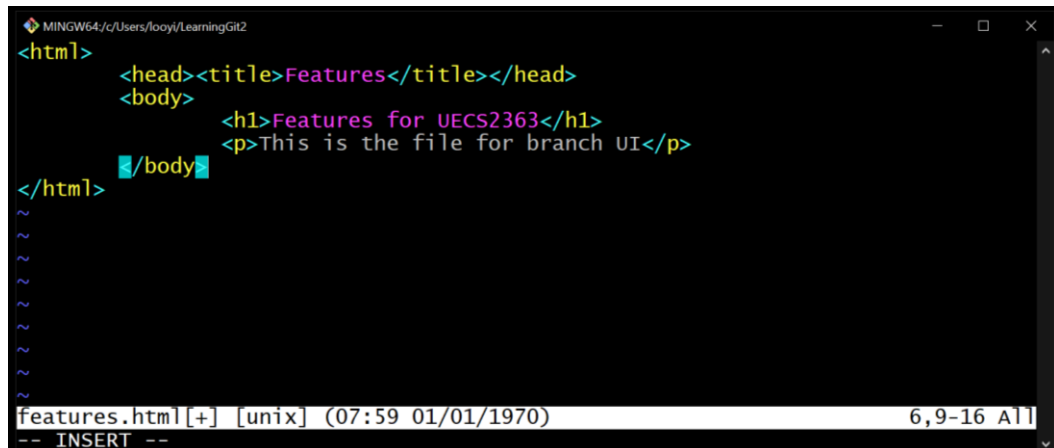
Create some changes in “ui” branch, such as adding a new file with the vim editor by executing `vim [filename.fileextension]` command as shown in Figure 3 and 4.

UECS2363 SOFTWARE CONSTRUCTION AND CONFIGURATION

```
looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$ git status
On branch ui
nothing to commit, working tree clean

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$ vim features.html
```

Figure 3: Vim command for opening editor to create and edit new file.

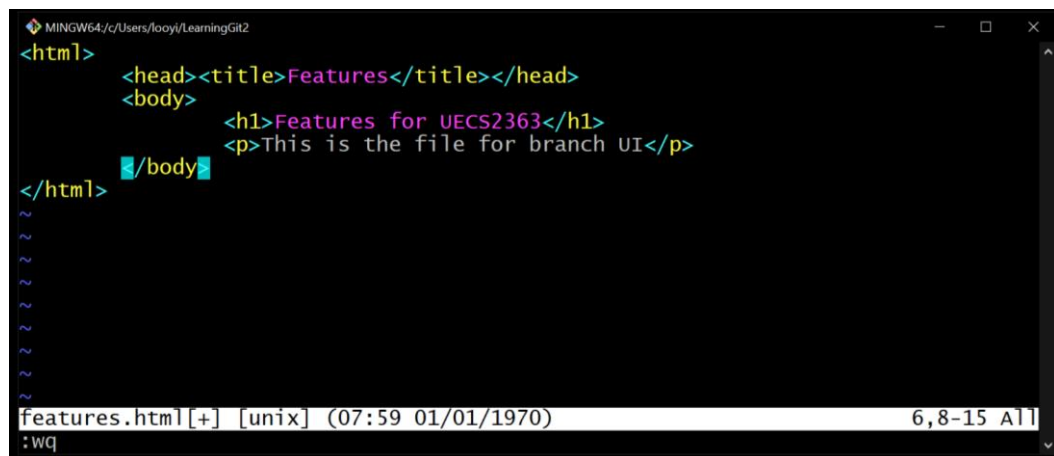


```
<html>
  <head><title>Features</title></head>
  <body>
    <h1>Features for UECS2363</h1>
    <p>This is the file for branch UI</p>
  </body>
</html>

features.html [+], [unix] (07:59 01/01/1970) 6,9-16 All
-- INSERT --
```

Figure 4: Editing new file with vim editor.

Upon entering the vim editor, press “i” to enter editing mode of file. Press “Esc” to quit editing mode. Then enter “:wq” in order to quit editing as shown in Figure 5.

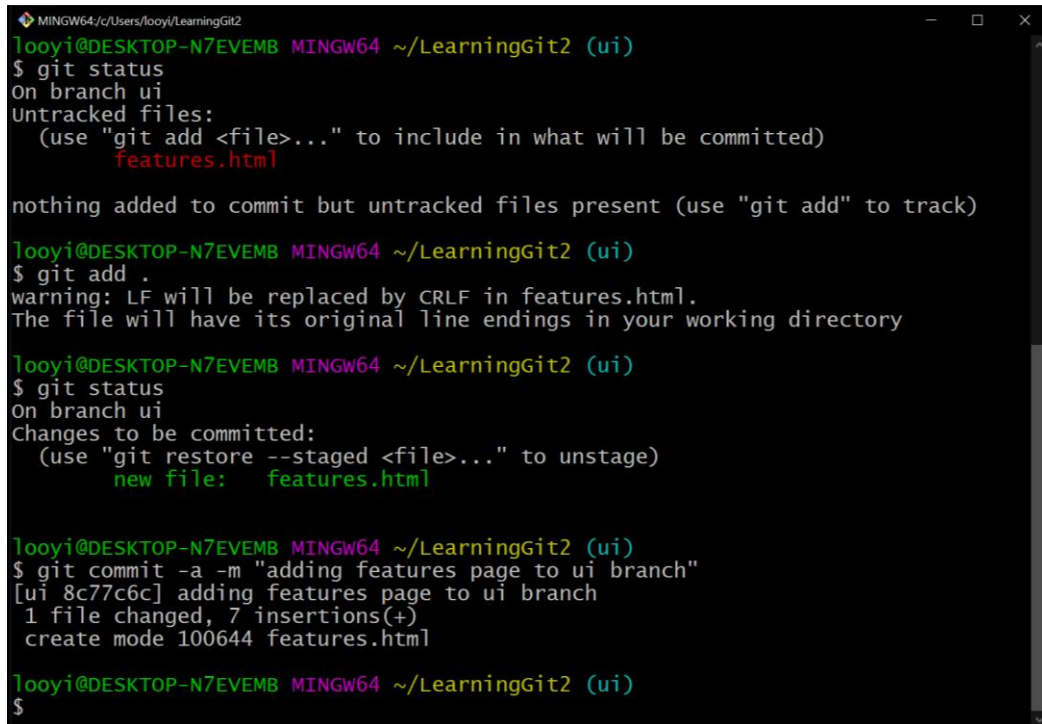


```
<html>
  <head><title>Features</title></head>
  <body>
    <h1>Features for UECS2363</h1>
    <p>This is the file for branch UI</p>
  </body>
</html>

features.html [+], [unix] (07:59 01/01/1970) 6,8-15 All
:wq
```

Figure 5: Quit editing mode in vim editor.

Add the file and commit the adding of new file into “ui” branch using the same way learnt in the practicals before this as shown in Figure 6.



```

MINGW64~/Users/looyi/LearningGit2
looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$ git status
On branch ui
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        features.html

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

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$ git add .
warning: LF will be replaced by CRLF in features.html.
The file will have its original line endings in your working directory

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$ git status
On branch ui
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   features.html

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$ git commit -a -m "adding features page to ui branch"
[ui 8c77c6c] adding features page to ui branch
1 file changed, 7 insertions(+)
create mode 100644 features.html

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$

```

Figure 6: Commit new file into “ui” branch.

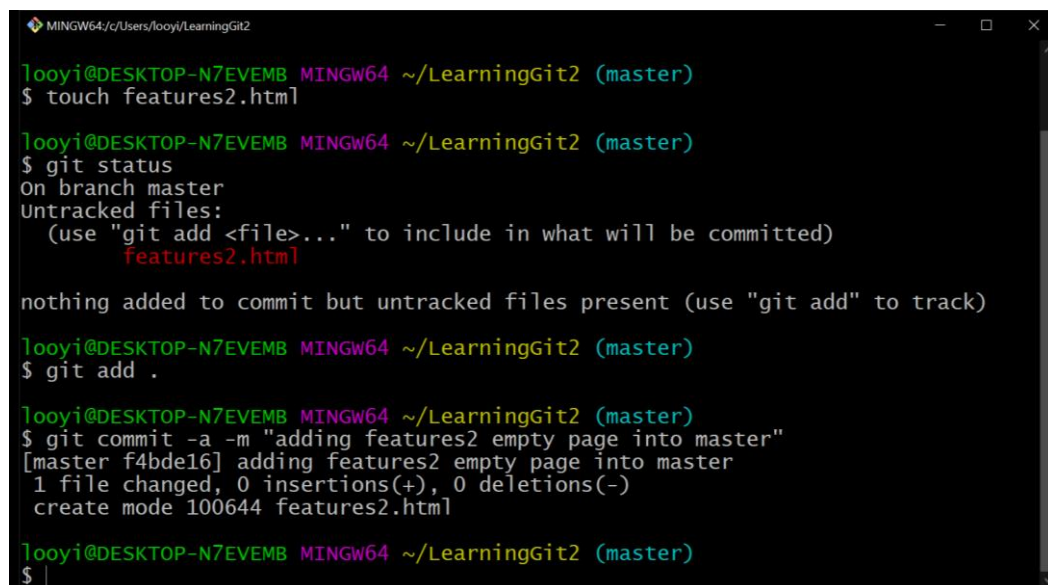
View the commits done thus far to the LearningGit2 repo.

Q: What do you see in your repo?

Now change back to “master” branch.

Q: What do you see in your LearningGit2 folder?

Add a file in “master” branch using `touch [filename.fileextension]` command as illustrated in Figure 7.



```

MINGW64~/Users/looyi/LearningGit2
looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ touch features2.html

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        features2.html

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

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git add .

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git commit -a -m "adding features2 empty page into master"
[master f4bde16] adding features2 empty page into master
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 features2.html

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$

```

Figure 7: Commit a new file to “master” branch.

Switch back to “ui” branch to do the merging of “ui” and “master” by executing `git merge master` command. The command will enter into editor. Press “i” to enter editing mode in order to add a merge message. Press “Esc” to quit editing mode. Then enter “:wq!” as shown in Figure 8.

Figure 8: Commit a new file to “master” branch.

```
MINGW64: c:/Users/looyi/LearningGit2
looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$ git merge master
Merge made by the 'recursive' strategy.
 features2.html | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 features2.html

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (ui)
$ git log
commit 56f1f35113e11b631701ca78e5c94d59701322ae (HEAD -> ui)
Merge: 8c77c6c f4bde16
Author: YimLingLoo <looyimling@gmail.com>
Date: Mon Jul 27 03:36:09 2020 +0800

    Merge branch 'master' into ui
    merging development in master into ui branch

commit f4bde162f5dbb8a238337edd1da26e9c5e02a994 (master)
Author: YimLingLoo <looyimling@gmail.com>
Date: Mon Jul 27 03:31:40 2020 +0800

    adding features2 empty page into master

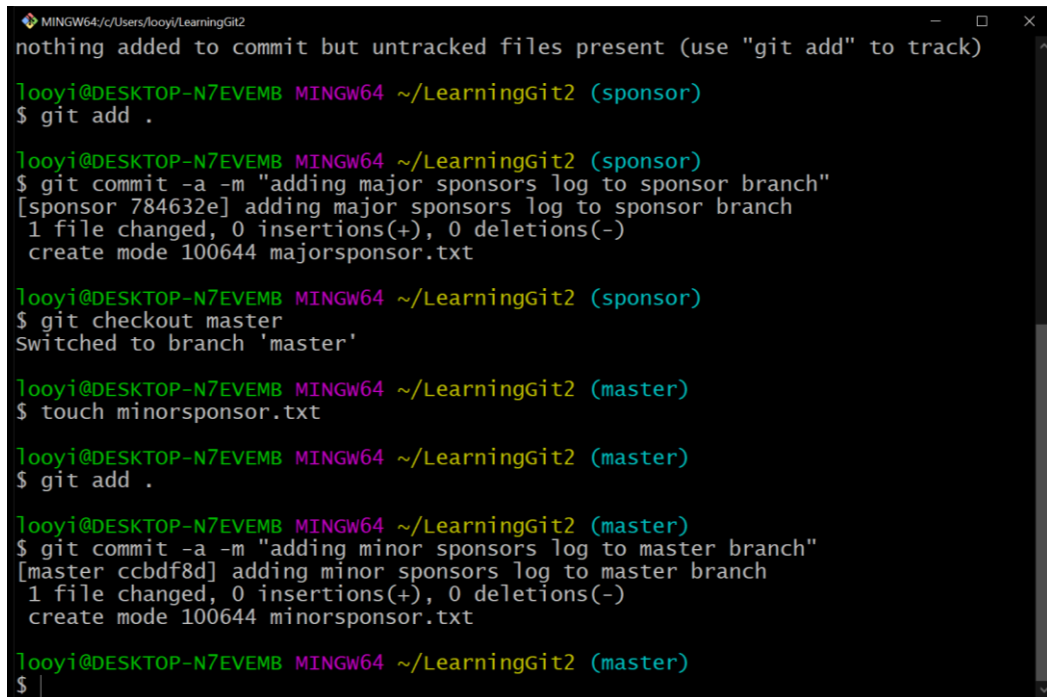
commit 8c77c6cc93bd380dfe54d4da535d569070e35b03
Author: YimLingLoo <looyimling@gmail.com>
Date: Mon Jul 27 03:11:30 2020 +0800

    adding features page to ui branch
```

Figure 9: All recent commits inclusive of HEAD; merge commit.

Creating a Branch, then Rebase.

In order to see the difference between Merge and Rebase, let's repeat some of the steps taken in Git Merge (section above); add a new branch, add a file to the branch, then add a file in master branch as illustrated in Figure 10.



```

MINGW64~/Users/looyi/LearningGit2
nothing added to commit but untracked files present (use "git add" to track)

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (sponsor)
$ git add .

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (sponsor)
$ git commit -a -m "adding major sponsors log to sponsor branch"
[sponsor 784632e] adding major sponsors log to sponsor branch
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 majorsponsor.txt

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (sponsor)
$ git checkout master
Switched to branch 'master'

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ touch minorsponsor.txt

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git add .

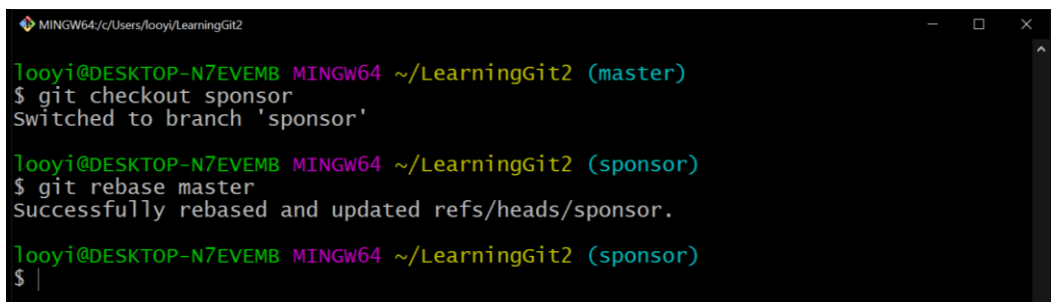
looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git commit -a -m "adding minor sponsors log to master branch"
[master ccbdf8d] adding minor sponsors log to master branch
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 minorsponsor.txt

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$

```

Figure 10: Add “sponsor” branch, add a file to both branches.

Similar to Git Merge, switch to “sponsor” branch then merge the branch with “master” by executing `git rebase [branch name]` command as shown in Figure 11.



```

MINGW64~/Users/looyi/LearningGit2

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git checkout sponsor
Switched to branch 'sponsor'

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (sponsor)
$ git rebase master
Successfully rebased and updated refs/heads/sponsor.

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (sponsor)
$

```

Figure 11: Merge changes in “master” branch into “sponsor” branch using Git Rebase.

Take a good look at all the commits, especially the current commits and HEAD.

Q: Can you spot the difference between using Git Merge and Git Rebase?

The difference between Git Merge and Git Rebase may be illustrated in Figure 12.

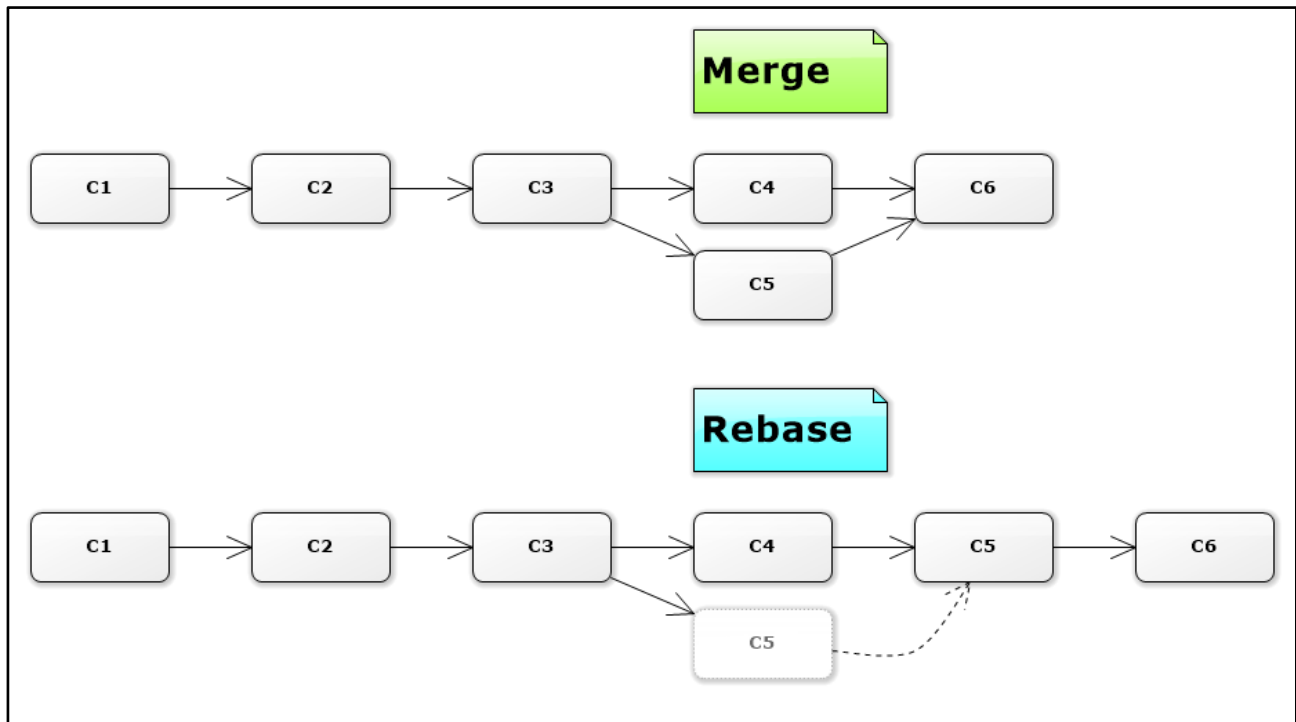


Figure 12: Difference between Git Merge and Git Rebase.

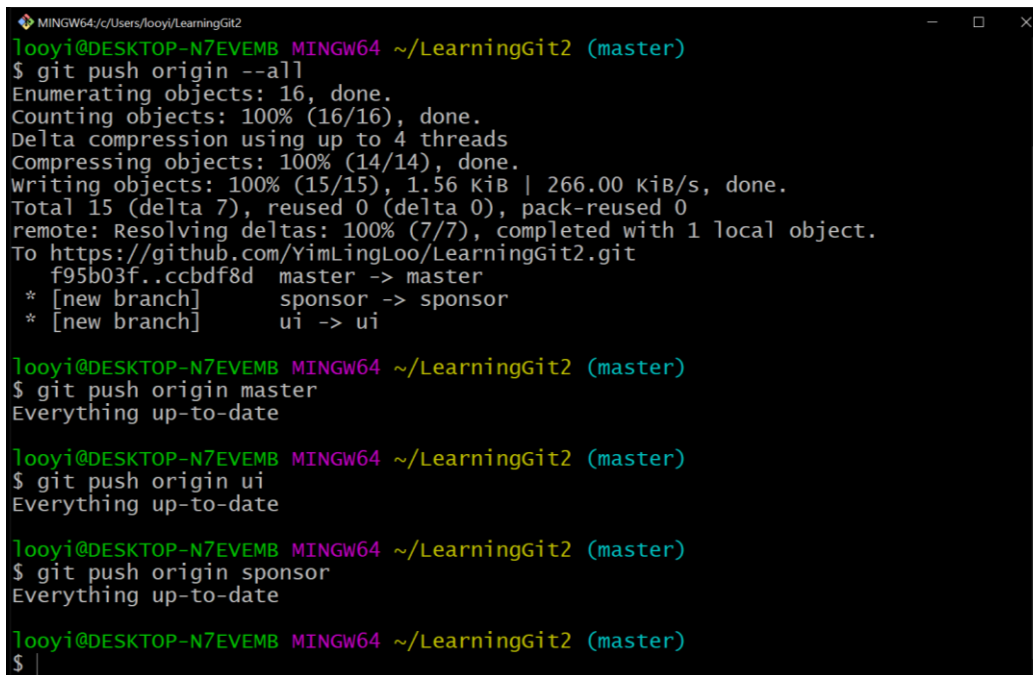
Image source: <https://alexpiccione.wordpress.com/2017/02/06/git-merge-vs-rebase/>

Q: What is the appropriate scenario to use Git Merge or Git Rebase?

Git Tags for Release Versions.

Developers may tag or mark a certain point of software project development for release of the software and put version or release numbers, along with the software release. In such case, Git Tags will be used.

Git Tags are useful to mark software release points as well as historic restore points. Before working on Git Tags, let's push all the changes made in the local LearningGit2 repo into remote repo using the `git push origin --all` and `git push origin [branch name]` commands as illustrated in Figure 13.



```

MINGW64/c/Users/looyi/LearningGit2
looyi@DESKTOP-N7EVEEMB MINGW64 ~/LearningGit2 (master)
$ git push origin --all
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 4 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 1.56 KiB | 266.00 KiB/s, done.
Total 15 (delta 7), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (7/7), completed with 1 local object.
To https://github.com/YimLingLoo/LearningGit2.git
   f95b03f..ccbdf8d  master -> master
   * [new branch]      sponsor -> sponsor
   * [new branch]      ui -> ui

looyi@DESKTOP-N7EVEEMB MINGW64 ~/LearningGit2 (master)
$ git push origin master
Everything up-to-date

looyi@DESKTOP-N7EVEEMB MINGW64 ~/LearningGit2 (master)
$ git push origin ui
Everything up-to-date

looyi@DESKTOP-N7EVEEMB MINGW64 ~/LearningGit2 (master)
$ git push origin sponsor
Everything up-to-date

looyi@DESKTOP-N7EVEEMB MINGW64 ~/LearningGit2 (master)
$

```

Figure 13: Push all changes to remote repo (Github).

In an event where the developer intended to release the outcome of this software project development before adding the remote repo as version 1.0, the developer may look for the git commit hash of the specific commit and put the tag of version 1.0 to it. In this case the tag v1.0 will be put on commit 65ec855 with the `git tag [tag name] [commit hash]` command as illustrated in Figure 14.

UECS2363 SOFTWARE CONSTRUCTION AND CONFIGURATION

```
MINGW64~/Users/looyi/LearningGit2
Author: YimLingLoo <68176566+YimLingLoo@users.noreply.github.com>
Date:   Fri Jul 17 20:12:32 2020 +0800

    Added a paragraph

commit 9d3afdfecd1ede01aea5f2984ee810a46d4e06b4
Author: YimLingLoo <68176566+YimLingLoo@users.noreply.github.com>
Date:   Fri Jul 17 20:11:39 2020 +0800

    Added a new paragraph

commit 65ec8556b3f8690da92232557c74e4125d399881
Author: YimLingLoo <looyimling@gmail.com>
Date:   Fri Jul 17 19:35:08 2020 +0800

    add all existing files

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git tag v1.0 65ec855

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ git push --tag
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/YimLingLoo/LearningGit2.git
 * [new tag]           v1.0 -> v1.0

looyi@DESKTOP-N7EVE MB MINGW64 ~/LearningGit2 (master)
$ |
```

Figure 14: Adding tag “v1.0” to commit 65ec855 and push changes to remote repo.

Changes of release/tag can be pushed to remote repo by executing `git push --tag` command. Take a look in GitHub LearningGit2 repo; it has now one tag under “Releases” section on the right navigation pane as shown in Figure 15.

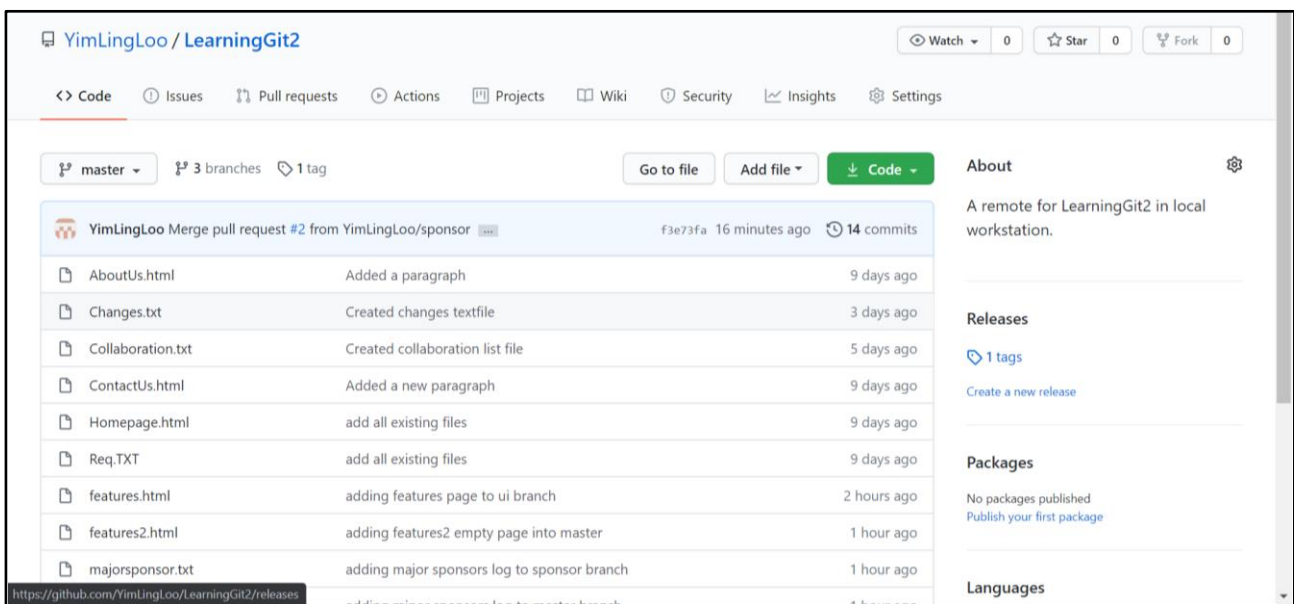


Figure 15: Tag reflected in Github (remote repo).

The release details and files are listed out when clicked on the “Release” link, as shown in Figure 16.

UECS2363 SOFTWARE CONSTRUCTION AND CONFIGURATION

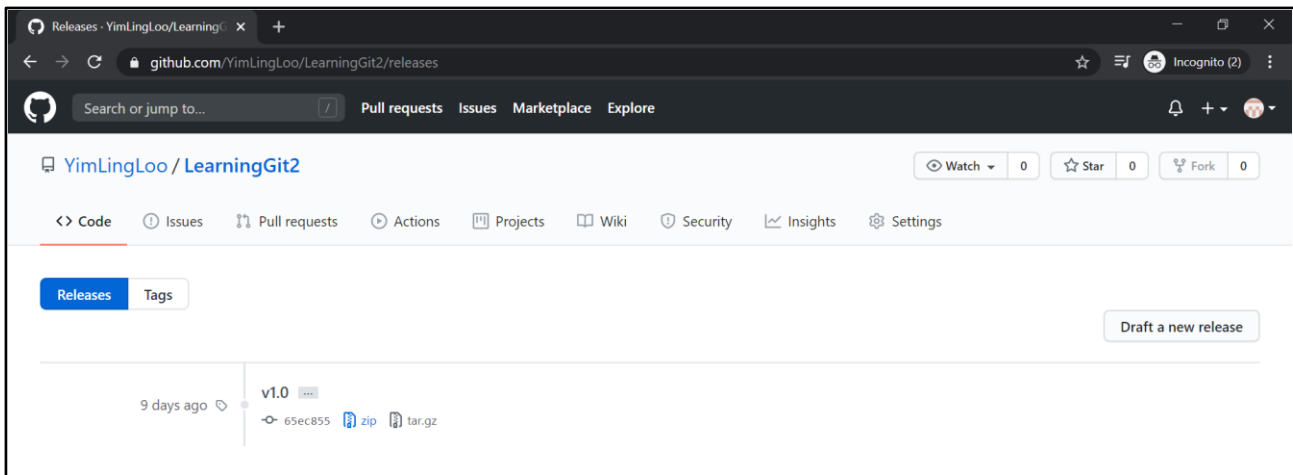


Figure 16: Release of v1.0 for LearningGit2.

Try to download the release of v1.0 (zip/tar.gz) that is made available by Github and see that the files are the files that are intended to be released in this version.

Unlike working on branches, developer cannot “git checkout” on tags. However, developer may have a workaround to create a branch on tags by executing `git checkout -b [branch name] [tag name]` as illustrated in Figure 17.

```
MINGW64/c/Users/looyi/LearningGit2
looyi@DESKTOP-N7EVEB MINGW64 ~/LearningGit2 (master)
$ git push --tag
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/YimLingLoo/LearningGit2.git
 * [new tag]         v1.0 -> v1.0

looyi@DESKTOP-N7EVEB MINGW64 ~/LearningGit2 (master)
$ git checkout -b release1 v1.0
Switched to a new branch 'release1'

looyi@DESKTOP-N7EVEB MINGW64 ~/LearningGit2 (release1)
$ git status
On branch release1
nothing to commit, working tree clean

looyi@DESKTOP-N7EVEB MINGW64 ~/LearningGit2 (release1)
$ git push origin release1
Everything up-to-date

looyi@DESKTOP-N7EVEB MINGW64 ~/LearningGit2 (release1)
$
```

Figure 17: Create a branch on a release tag.

Having done so, a developer may checkout to the branch created for the specific release and work on experimental ideas to the specific release of software. The result of creating a branch on a specific tag may be visualized in Github as shown in Figure 18.

UECS2363 SOFTWARE CONSTRUCTION AND CONFIGURATION

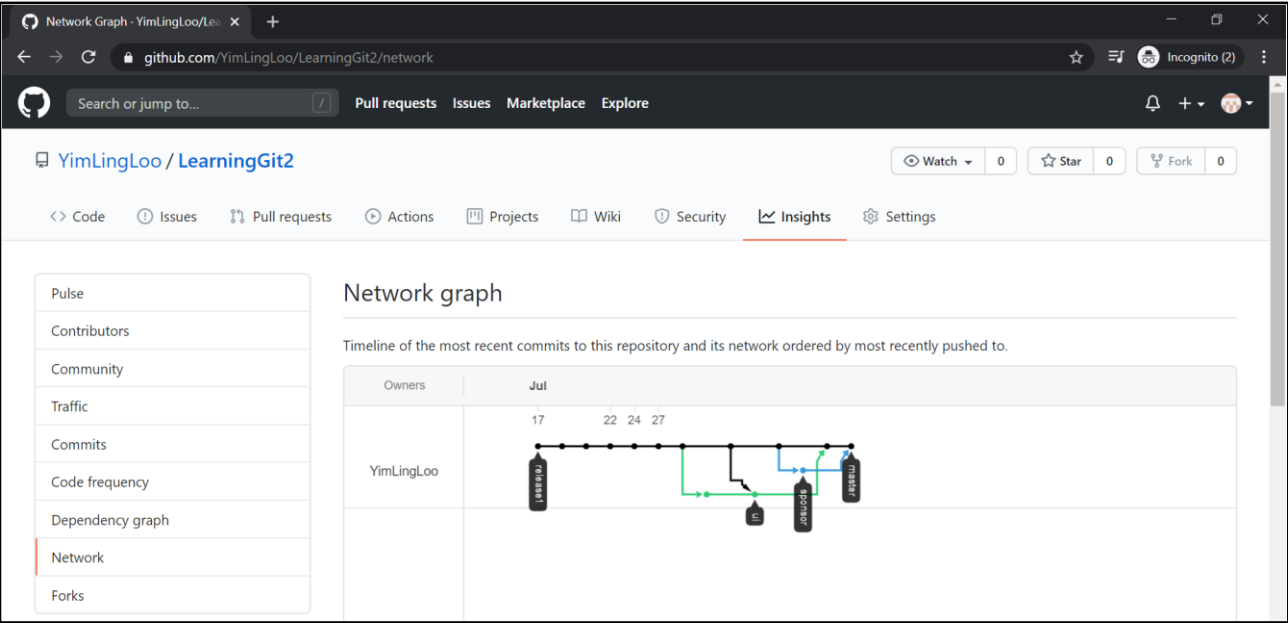


Figure 18: Create branch on a specific software release.

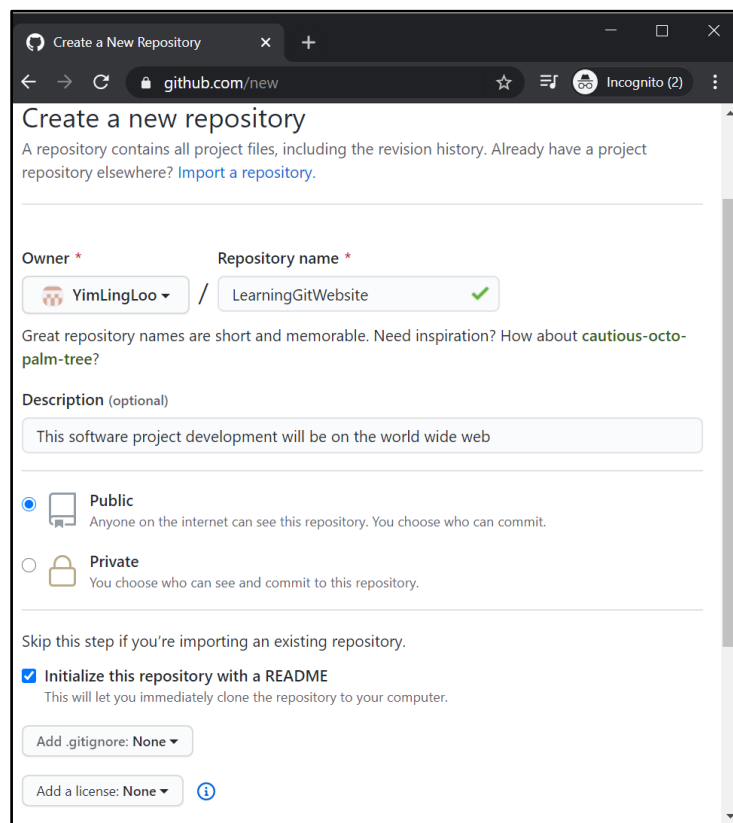
GitHub Pages for Web Hosting.

Previously, we had been working along the line of having local repo which basically allow mostly the main/original developer to work on the software project development and remote repo, for addressing collaboration. The branch of development had always been the “master” branch. In this practical, mode of collaboration other than having a remote repo on Github, is GitHub Pages will be explored.

With Github Pages, developer view a project on a website and even host the project in the world wide web. For Github Pages to work, we need to discard the idea of working with “master” branch but work on “gh-pages”.

Getting Started with GitHub Pages.

In order to comprehend the main concept of using GitHub Pages, firstly, create a new repo in Github as shown in Figure 19.



Create a New Repository

github.com/new

Create a new repository

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

Owner * Repository name *

YimLingLoo / LearningGitWebsite ✓

Great repository names are short and memorable. Need inspiration? How about [cautious-octo-palm-tree?](#)

Description (optional)

This software project development will be on the world wide web

☒ 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.

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

☒ Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None ▼

Add a license: None ▼ ⓘ

Figure 19: Initialize LearningGitWebsite Repo in Github.

After initializing the LearningGitWebsite repo in Github, initialize a new branch called “gh-pages” apart from existing master branch as illustrated in Figure 20.

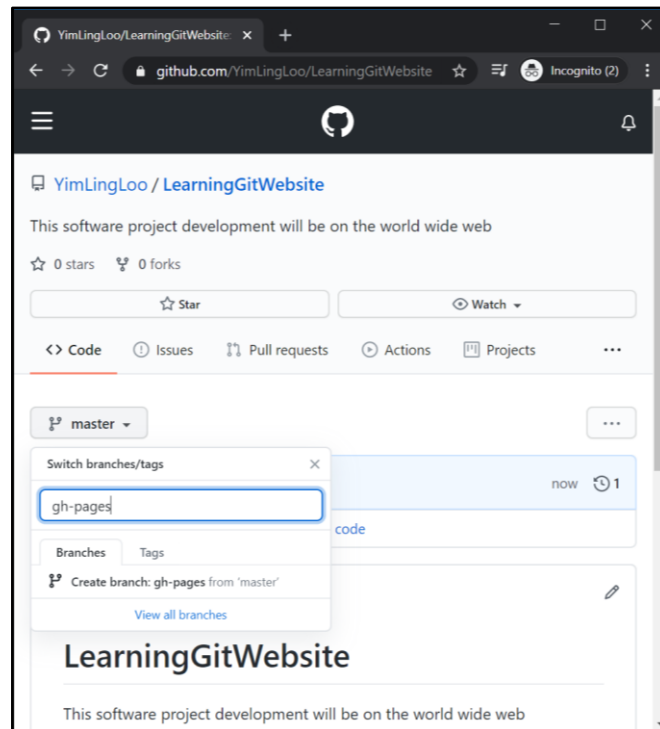


Figure 20: Create “gh-pages” Branch in LeaningGitWebsite Repo.

According to our previous practical on Github, we know that initializing a branch other than “master” branch, is having experimental ideas tested on branches before considering to merge into the main development.

In this practical, in order to comprehend the new concept of hosting the software project, the idea of always working on “master” branch have to be temporarily discarded. Thus, “gh-pages” need to be set as the “default” branch, instead of “master”.

Changing Default Branch.

In the repo page, select “Settings” menu, then select “Branches” on the left navigation pane. Choose “gh-pages” in the drop-down button under “Default branch” and click “Update” button. After doing so, a confirmation pop-up notice will be shown as illustrated in Figure 21.

UECS2363 SOFTWARE CONSTRUCTION AND CONFIGURATION

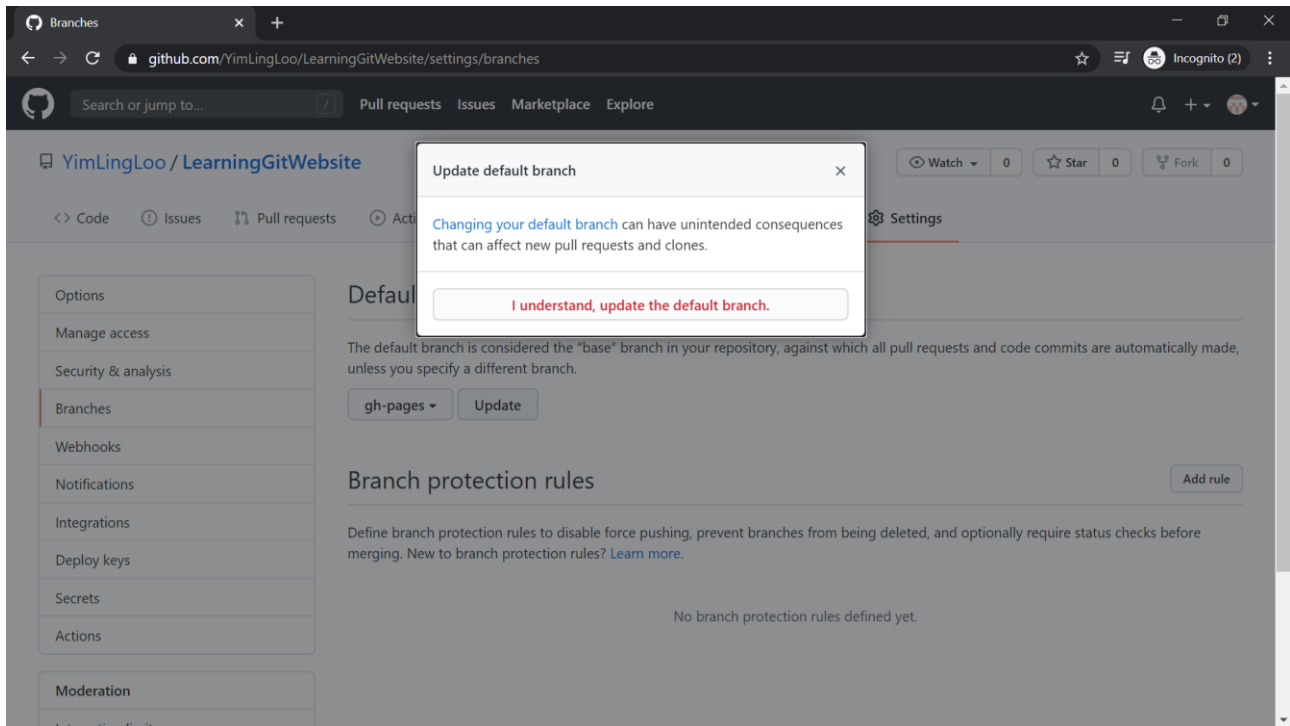


Figure 21: Updating Default Branch of LearningGitWebsite.

Click on the “I understand...” confirmation button to confirm switching default branch from “master” to “gh-pages”. While going back to the homepage of LearningGitWebsite repo, select the “2 branches” beside branches dropdown button as shown in Figure 22.

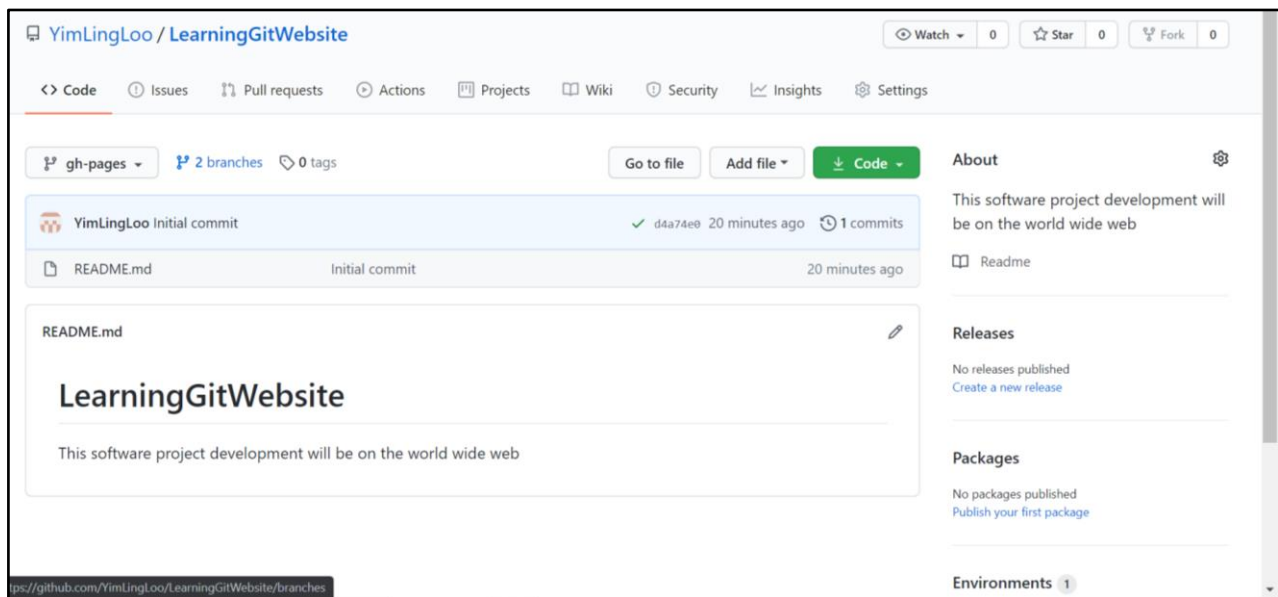


Figure 22: Select to view all branches of LearningGitWebsite.

In order to make sure there is no future accidental workings on “master” branch or incidents of mistakenly taking “master” branch as default branch because of its name, the “master” branch need to be deleted. To do so, click on the trash can icon on the master branch as shown in Figure 23.

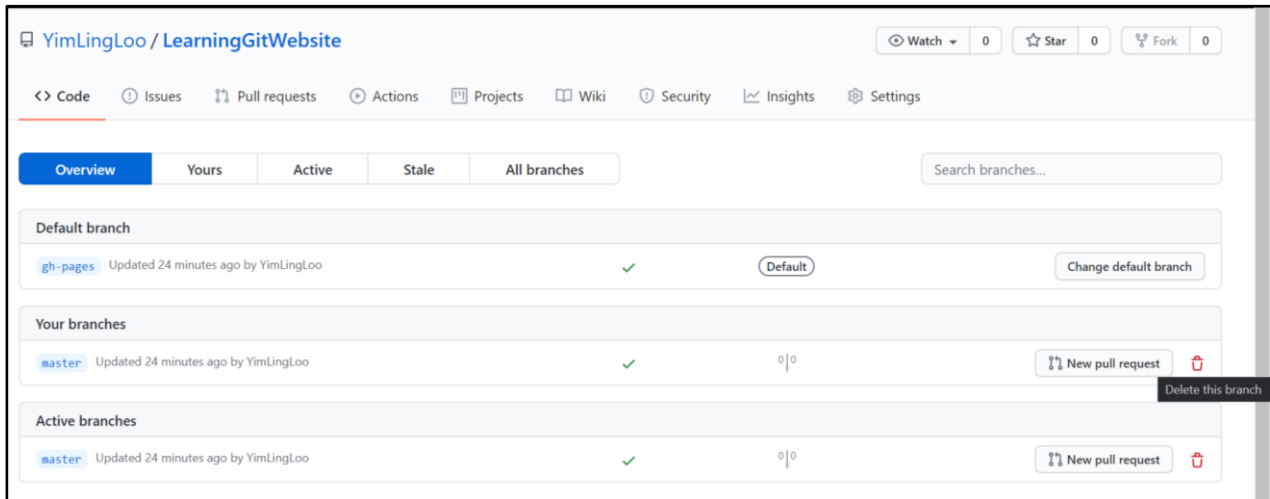


Figure 23: Click on “master” branch trash can icon to delete “master” branch.

After deletion of “master” branch (in which, you may choose to restore, as offered by Github after trash can icon is clicked), see that LearningGitWebsite is posted through GitHub Pages by selecting “Settings” again, scroll down until “GitHub Pages” and see the site of the LearningGitWebsite repo as shown in Figure 24.

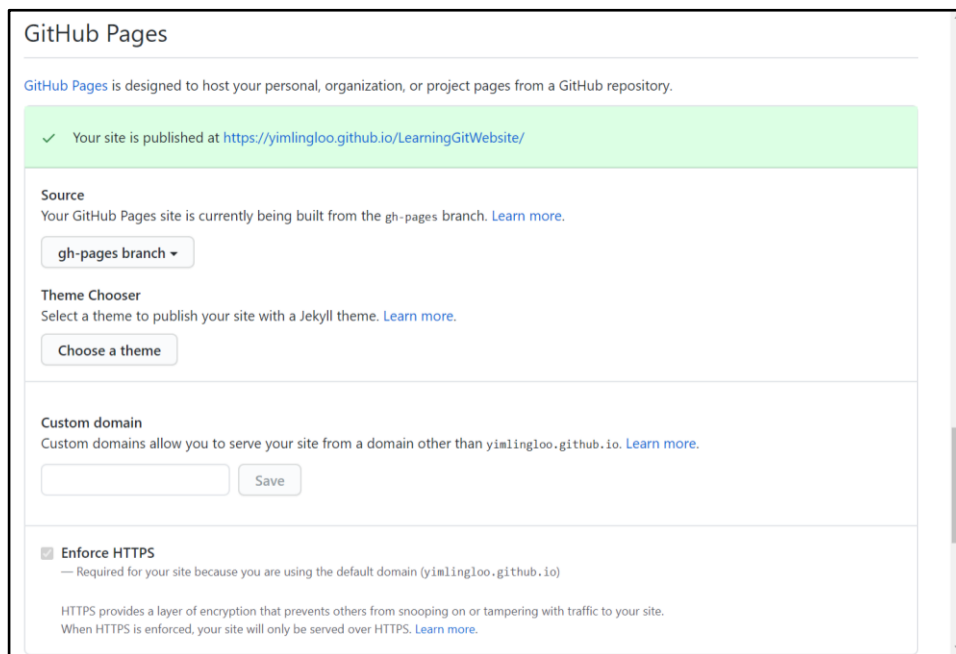


Figure 24: The repo is published on site by GitHub Pages, built from “gh-pages” branch.

There are ways where we can customize the URL, but for now let’s work on getting to see the site from the predefined URL. Commit a new file, into LearningGitWebsite repo and access the predefined URL continued with the new file name and extension to access the file.

Q: What do you see? What can you derive from here?

Recalling from knowledge of building a website, an “index.html” file is an “auto-access” for a URL. Having said so, whenever a URL is accessed, “index.html” is the automatically loaded page/html file. Likewise, create an “index.html” file for hosting our own software project on GitHub Pages as shown in Figure 25.

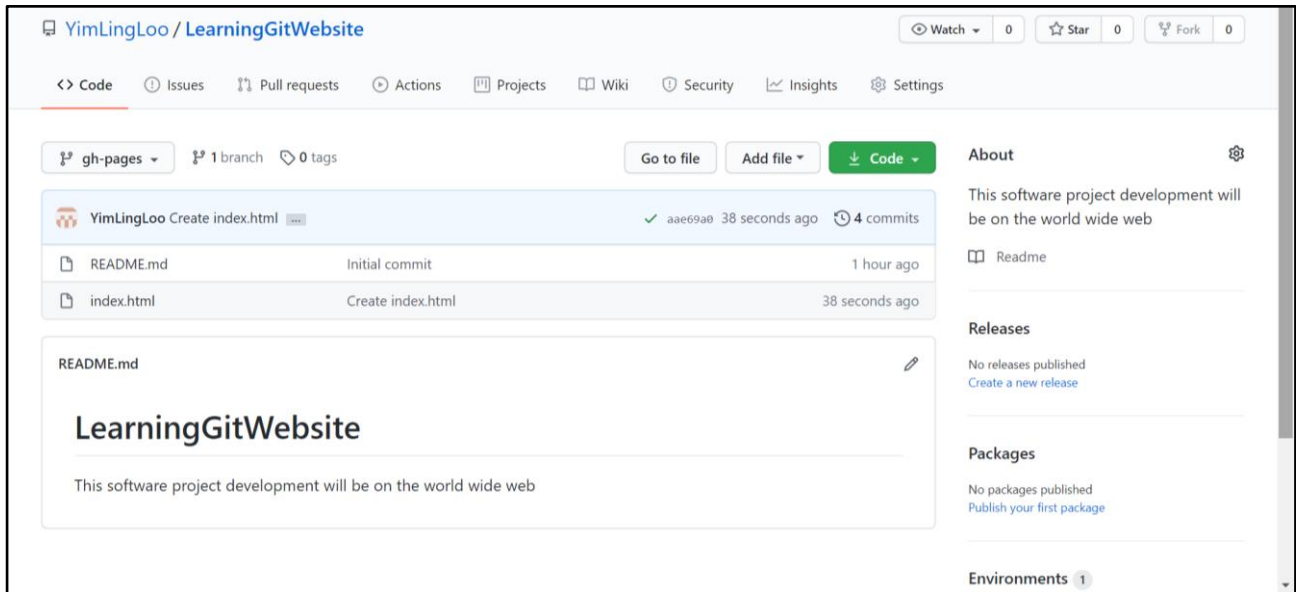


Figure 25: Create “index.html” file for the repo.

After creating an “index.html”, access the predefined URL without any filename and extension continued after the URL.

Q: What do you see? What can you derive from here?

Looking into the website that is hosted, collaborators can contribute to the website by creating “pull request”, making changes to the website, developers can clone the repo, commit branches and changes. All workflow will be the same. However, if one is to work locally and clone this website hosted repo, there is one thing that will be different in one of the commits.

Q: What is the difference?

Summary

1. Git is a **version control software** that helps developer to keep track of the different versions of software project files created and edited. Git enables collaboration of many developers on a software project development.
2. Github is a web server with a website that runs Git software on the platform to enable developers to work on a software project and keep track of the versions.
3. Branch and Merge could be done easily in Git with Git commands and there are two mechanisms to merge; **Merge** and **Rebase**.
4. Developer may **tag** a head or HEAD to mark a software release point, or put historical point of reference.
5. Tag may not be having the mechanism of “checkout” but branching on a tag will enable such concept to be done.
6. **Github Pages** is used for hosting a software project development on website.