Full Stack Development with AI
Lab 2.1 – A Comprehensive Tutorial on Git

Lab Overview

In this lab, you will learn how to perform version control on your source code using Git and GitHub. You would need two GitHub accounts to simulate a collaborative team development environment. For this purpose, you should register for two GitHub accounts using two different emails. It is not necessary to use a private repository for this lab. The first GitHub account will function as the team lead and the second GitHub account will function as the team member.

This lab assumes that you are using Windows. The Git commands are the same for Windows and non-Windows. However, you will need to replace the native Windows commands with your own operating system's commands (see next section for more information).

In this lab, you would be working with HTML (Hypertext Markup Language) documents for simplicity instead of source files written with a specific programming language. The required HTML source files can be found in "resources.zip".

Commands for Mac and Linux

For Mac and Linux, your home Documents directory should be located at the following location:

```
cd ~/Documents
```

The command for Mac and Linux to create a new directory is as follow:

```
mkdir <DIR_NAME>
```

Software Installation

For **Windows**, follow the instructions below to install Git:

1. Download the installer from https://git-scm.com/downloads/win

2. Run the installer and change the installation folder to C:\Git.

3. Select a default text editor that is already installed on your computer.

   For Windows, the recommended text editor is "Notepad++" (https://notepad-plus-plus.org/downloads).

4. Accept the default settings for all the other configurations and complete the installation.

For **macOS** and **Linux**, Git should be installed by default. Start terminal and run the command `git --version` to check whether Git has already been installed.

Otherwise, follow the instructions here to install Git:

- macOS – https://git-scm.com/download/mac
- Linux – https://git-scm.com/downloads/linux

Exercise 1 – Git's Global Configuration

You need to unset any name and email address configured at the global level because you will be using two different identities on the same laptop. You will, instead, configure your identity at the repository level subsequently.

```
git config --global --list
git config --global --unset user.name
git config --global --unset user.email
git config --global init.defaultBranch main
git config --global --list
```

The current convention is to name the default branch of a new Git repository as "main" instead of "master".

Exercise 2 – Creating a New Local Repository

Let's begin by creating two new directories for our repositories – one for each GitHub account:

```
C:
cd %USERPROFILE%\Documents
mkdir teamlead
mkdir teammember
cd teamlead
```

Next, let's create a new sub-directory in teamlead for our main project directory:

```
mkdir PointOfSaleSystemWeb
cd PointOfSaleSystemWeb
```

Run the following command, and observe and explain the output:

```
git status
```

We are now ready to initialise our main project directory as a Git directory:

```
git init
git status
```

Observe and explain the output.

Set the name and GitHub email address for team lead. Remember to replace with your own name and email address.

```
git config user.name "Team Lead"
git config user.email "teamlead@gmail.com"
git config --list
```

Exercise 3 – Setting Up the New Local Repository

We will now create a README file for our new Git repository. You can refer to this webpage – https://help.github.com/en/articles/basic-writing-and-formatting-syntax – on the GitHub Help website to learn more about the syntax for writing Git README file.

```
git status
echo # PointOfSaleSystemWeb >> README.md
git status
git add README.md
git status
```

Observe and explain the output for each `git status`.

Next, let's perform our first commit:

```
git commit -m "first commit"
git status
```

Observe and explain the output.

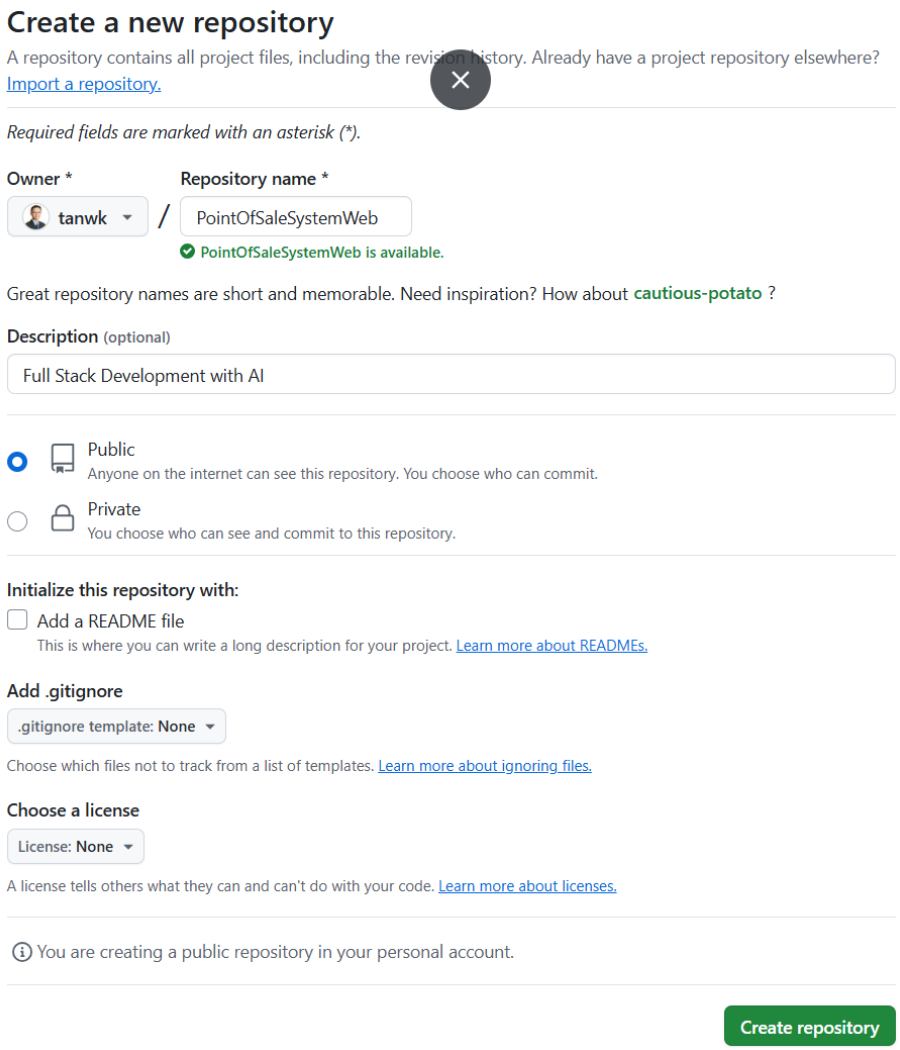
We will now add in the index webpage for PointOfSaleSystemWeb. Use a suitable text editor such as Visual Studio Code to create index.html in the main project directory `%USERPROFILE%\Documents\teamlead\PointOfSaleSystemWeb`. You can use the content from the sample source file `index - initial.html`. When you are done, stage and commit the file:

```
git add index.html
git commit -m "added /index.html"
```

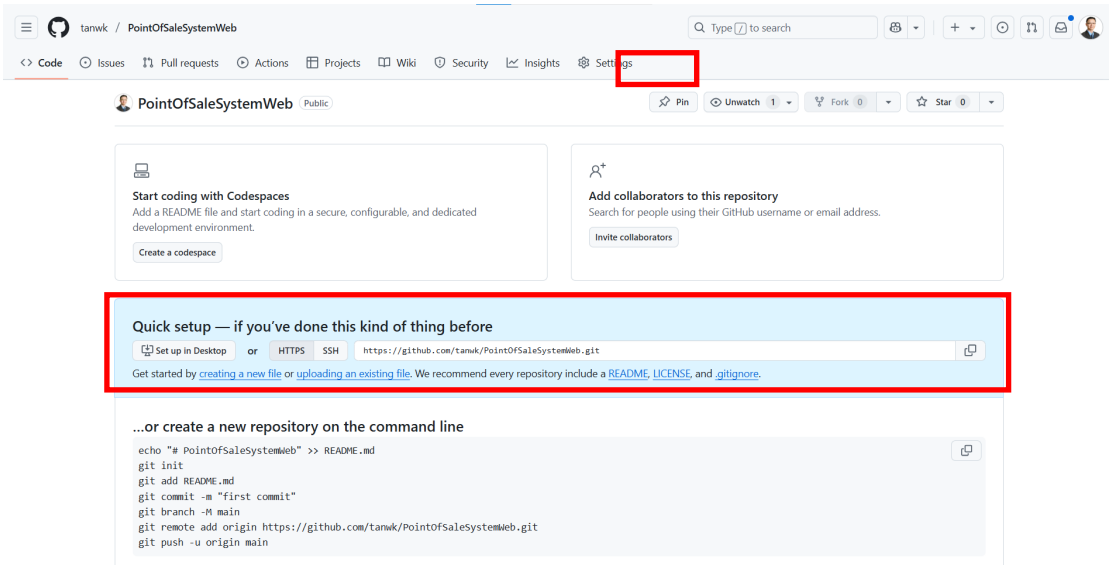
Exercise 4 – Setting Up Your Remote Git Repository on GitHub

Login to GitHub as the team lead with your first email, look for your profile on the top right-hand corner and select "Your repositories". You should not have any repository at this juncture. Follow the instructions below to create a new remote repository on GitHub:

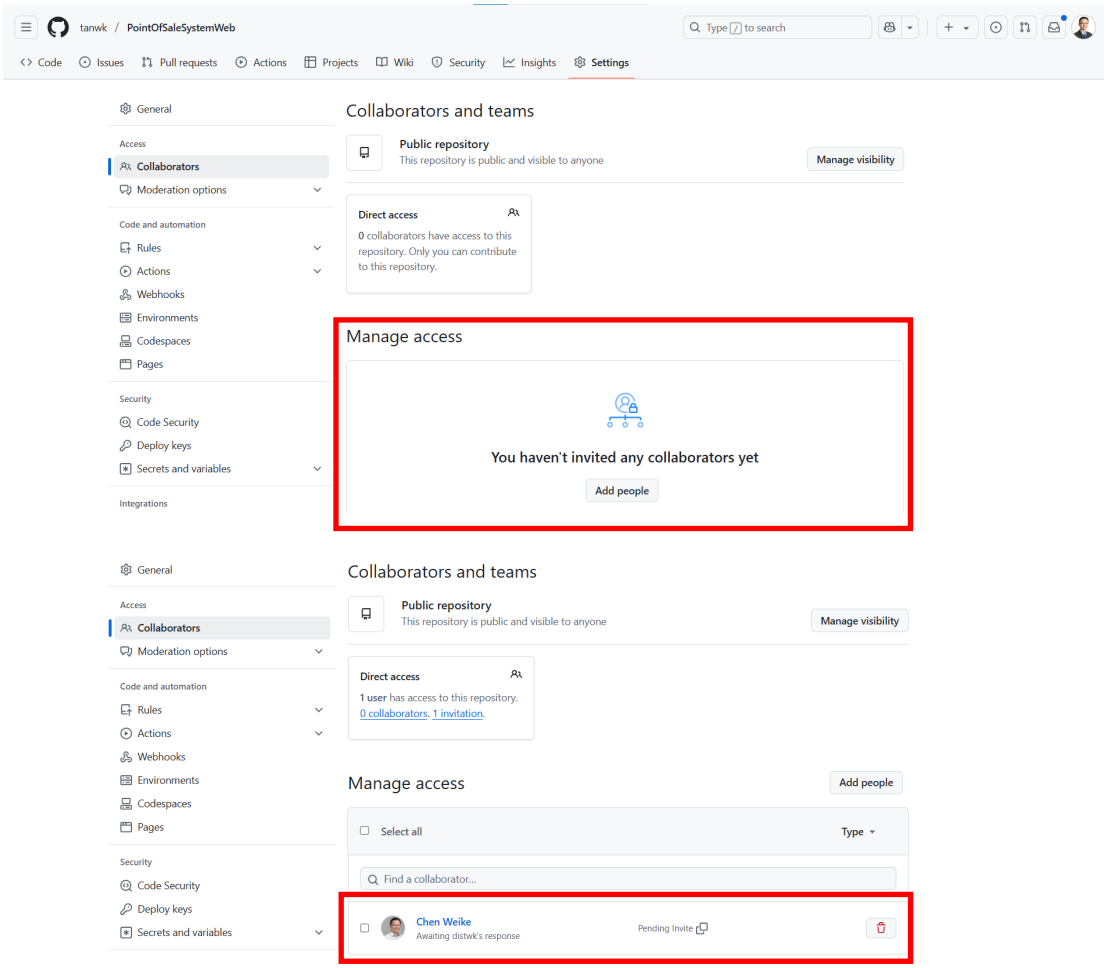
1. Click on the green colour "New" button.
2. Set the repository name as "PointOfSaleSystemWeb" without the double quotes.
3. Set an optional description if desired.
4. Leave the repository's visibility as "Public".
5. Do **NOT** check "Initialize this repository with a README" since we are going to import our local repository into GitHub.
6. Do **NOT** add a .gitignore file, i.e., select None for .gitignore template.
7. Do **NOT** add a license for this practical lab, i.e., select None for license.
8. Click the green colour "Create repository" button to complete the process.
9. You will be redirected to your new repository homepage.



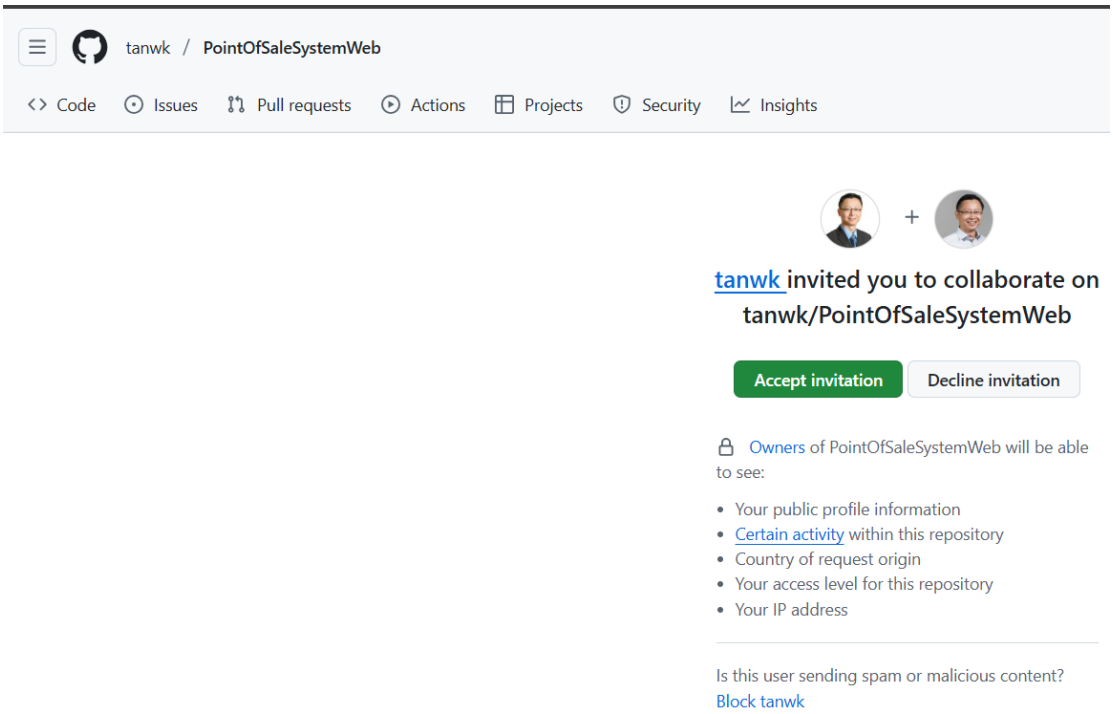
10. Note down your repository path beginning with HTTPS, e.g., https://github.com/tanwk/PointOfSaleSystemWeb.git.



11. Switch to the "Settings" tab and select "Collaborators" from the left side menu. Add your team member GitHub account, i.e., the second email account, as a collaborator.



12. You should receive an invitation email at your second email. Open the invitation link with a different web browser and login to your second email GitHub account to accept the invitation.



Return to your local repository and configure its remote repository setting. Remember to replace with your own GitHub repository path.

```
git remote

git remote add origin
https://github.com/<username>/PointOfSaleSystemWeb.git

git remote

git remote -v
```

Observe and explain the output for each `git remote`.

Before pushing our local repository to GitHub, we will create a staging branch. The default main branch represents our production branch. The staging branch will be the main development and testing branch before merging into main.

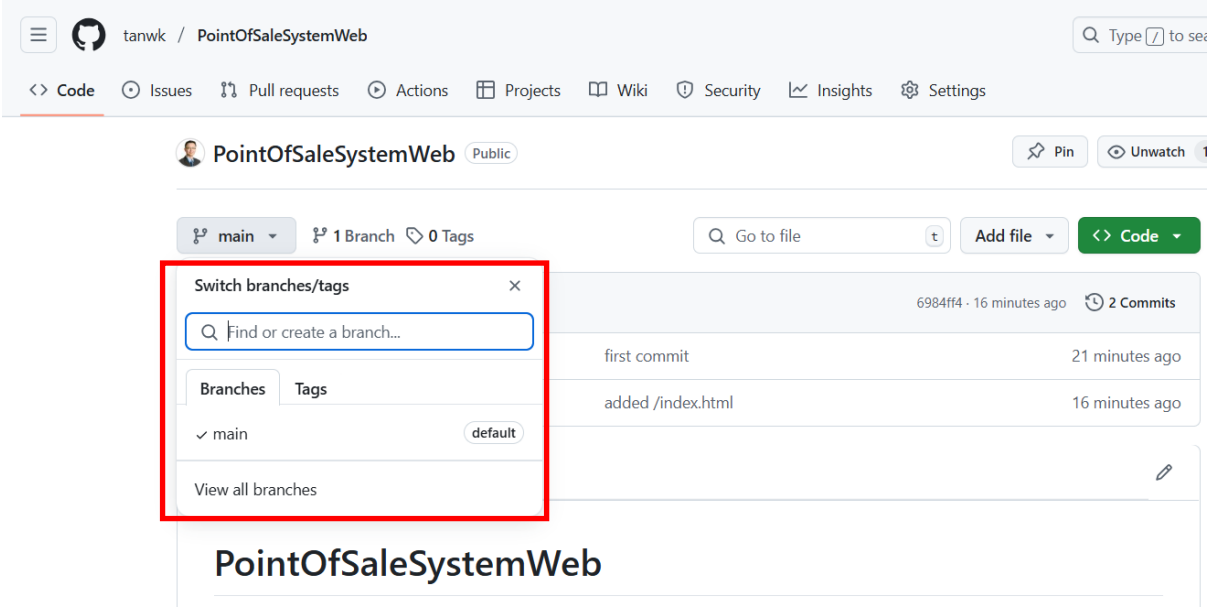
```
git status
git branch staging
git status
git branch
```

Observe and explain the output of `git status` and `git branch`.

We are now ready to push our local repository to GitHub:

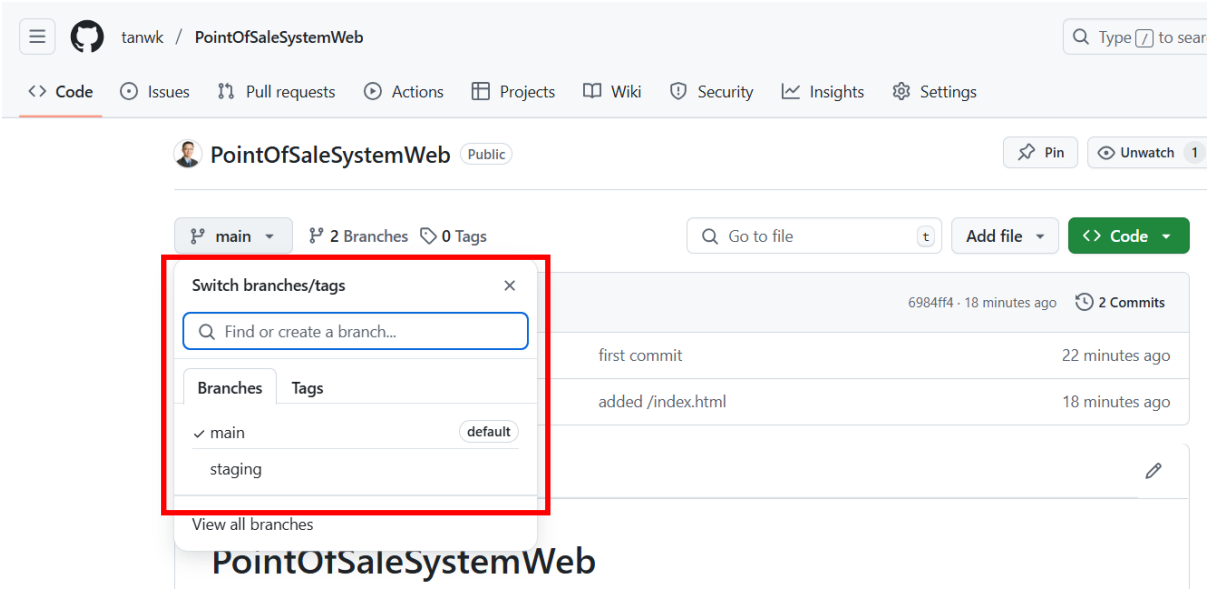
```
git push origin main
git remote show origin
```

Refresh your GitHub repository homepage using the team lead account and observe the changes.




```
git push origin staging
git remote show origin
```

Refresh your GitHub repository homepage and observe the changes.



Exercise 5 – Cloning a Remote Git Repository

We will now assume the role of team member and clone the remote GitHub repository that has been created by the team lead. Remember to replace with your own GitHub repository path, own name, and email address.

```
cd %USERPROFILE%\Documents\teammember
git config --list
git clone https://github.com/<username>/PointOfSaleSystemWeb.git
cd PointOfSaleSystemWeb
git status
git remote
git remote show origin
git config user.name "Team Member"
git config user.email "teammember@gmail.com"
git config --list
```

In the above screenshot, the left terminal represents the team lead, and the right terminal represents team member

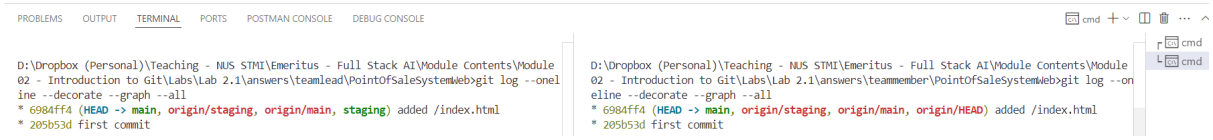
Observe the changes that have been made to your file system.

Do a `git fetch` and observe and explain the output.

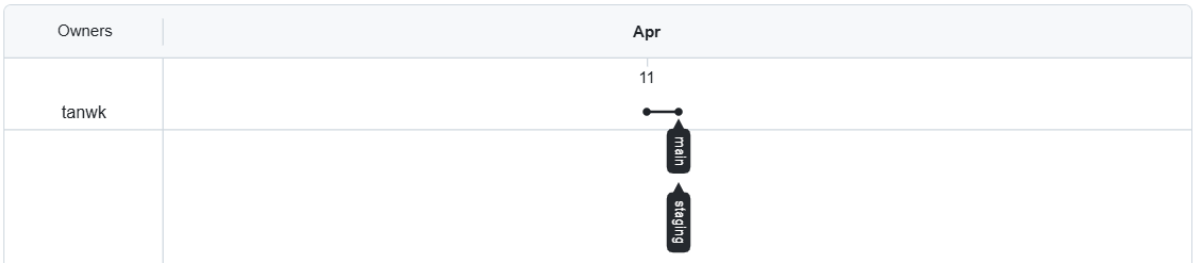
Exercise 6 – Developing Your Project and Recording Changes to the Repository

Before we start development, i.e., making changes to the source files, you can view the graph of your current local repository with the following command (in both team lead and team member):

```
git log --oneline --decorate --graph --all
```



To view the graph of your remote repository, go to the GitHub repository homepage and switch to the "Insights" tab, and then select "Network". At this juncture, you should observe that both the main and staging pointers are pointing to the same location, i.e., the "added /index.html" commit.



In the subsequent exercises, you will be using the team lead account to develop the "System Administration" module of PointOfSaleSystemWeb. We will create a new development branch for the team lead, and a new feature branch for the "System Administration" module. We will then switch to the feature branch for the actual development work.

```
cd %USERPROFILE%\Documents\teamlead\PointOfSaleSystemWeb
git branch d_teamlead
git branch f_sysadmin
git checkout f_sysadmin
git status
git log --oneline --decorate --graph --all
```

Observe and explain the output.

Let's begin by modifying `index.html` to add a list of hyperlinks as the navigation menu. Open `index.html` for editing in the text editor and copy line 13-21 from the sample source file `index.html` to your own `index.html` at the matching location below after 6. Save the change and run the following git commands. Observe and explain the output.

```
git status
git add index.html
git status
git reset HEAD index.html
git status
```

```
git checkout -- index.html
git status
```

Observe the copy of index.html in your text editor and explain the "horrific" implication. Are you able to get back the change that you have made, i.e., the navigation menu for the "System Administration" module?

Repeat the change to recreate the navigation menu for the "System Administration" module. Fortunately, we have the sample source file to fall back on. What happen if we don't? After making the change, save the change and run the following git commands. Observe and explain the output.

```
git status
git add index.html
git status
git commit -m "added nav menu for sysadmin to /index.html"
git status
```

Create a new subdirectory in the main project directory with the following commands. Observe and explain the output.

```
mkdir systemAdministration
cd systemAdministration
git status
```

Erm, wait a moment… Git doesn't seem to realise that we have added a new folder. What has happened?

Let's create a new file `createNewStaff.html` in the newly created subdirectory `systemAdministration`. You can get the content from the sample source file of the same name.

Run the following commands, and observe and explain the output:

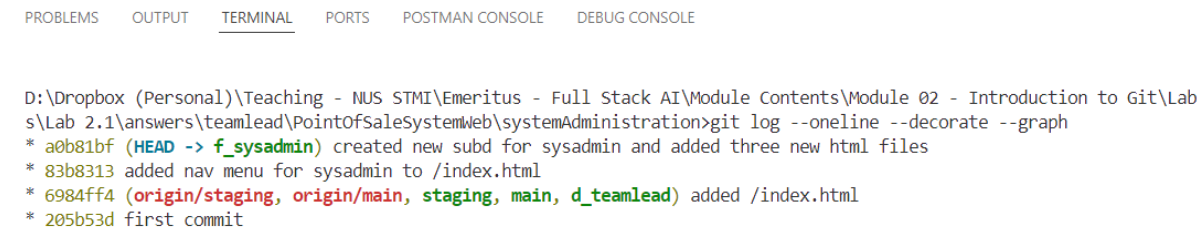
```
git status
git add createNewStaff.html
git status
```

Create two new files `viewStaffDetails.html` and `viewAllStaffs.html` in the newly created subdirectory `systemAdministration`. You can get the content from the sample source file of the same name.

Run the following commands, and observe and explain the output:

```
git status

git add .

git status
```

```
git commit -m "created new subd for sysadmin and added three new html files"

git status

git log --oneline --decorate --graph
```



Suppose you have finished developing and testing the "System Administration" module. You can now merge the changes to your main development branch, i.e., d_teamlead. Execute the following commands and inspect your working directory.

```
cd ..
git checkout d_teamlead
```

Did you see the new folder and files added for the "System Administration" module? How about the changes made to `index.html`?

Now, try merging the f_sysadmin branch into d_teamlead with the following command:

```
git merge f_sysadmin
git status
```

Observe that this would be a fastforward merge and after the merge, you should be able to see the changes made for the "System Administration" module.

Since you are also the team lead, you can merge your changes straight to the staging branch. Run the following commands, and observe and explain the output:

```
git checkout staging
git merge d_teamlead
git status
```

Exercise 7 – Pushing Your Changes to the Remote Repository

You are now ready to share your changes made for the "System Administration" module with your teammate. Inspect your remote GitHub repository using the following command as well as manually browsing the GitHub repository homepage:

```
git remote show origin
```

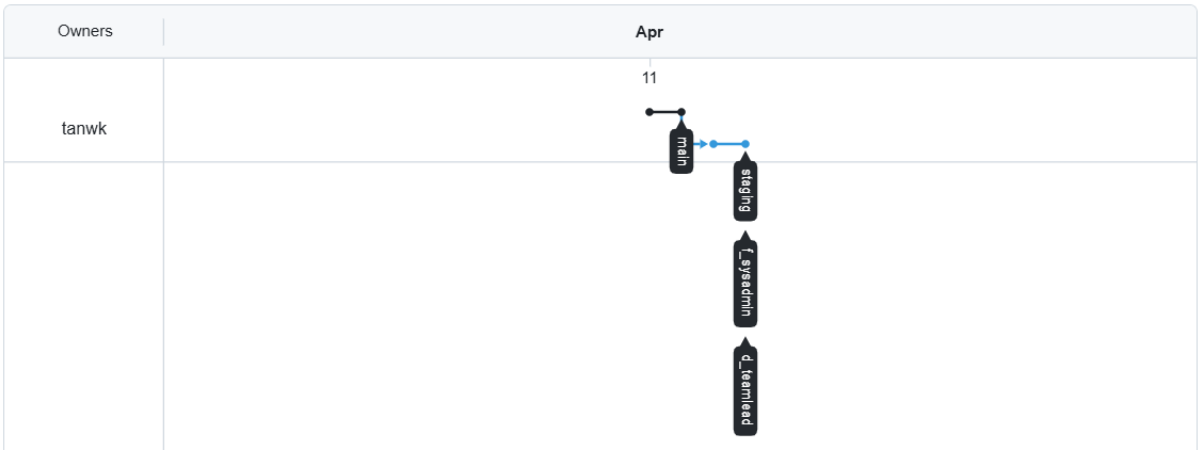
We will use the following command to push all your local branches to the remote GitHub repository:

```
git push origin --all
```

Inspect your remote GitHub repository again and ensure that your changes are now safely stored on the remote repository.

```
git remote show origin
```

The graph of your GitHub remote repository should resemble the one shown below:



Exercise 8 – Fetching and Pulling Changes from the Remote Repository

You will now assume the role of your teammate, i.e., team member, using the second email GitHub account. Inspect your local repository and remote repository:

```
cd %USERPATH%\Documents\teammember\PointOfSaleSystemWeb
git status
git log --oneline --decorate --graph
git remote show origin
```

Observe and explain the output.

Run the following command to synchronise your work with the remote repository and observe and explain the output. Thereafter, inspect your local repository and remote repository again.

```
git fetch origin
git status
```

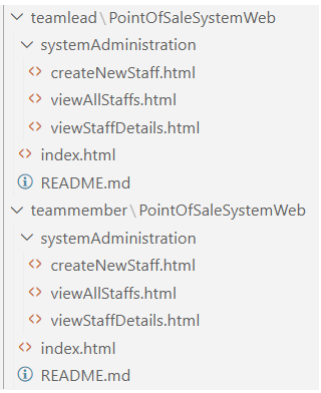
Suppose you want to know what are the changes that the team lead had made. You can the checkout the staging branch and inspect the changes:

```
git checkout staging
git status
git diff
git diff main staging
```

In the `diff` command:

- Press Enter key to scroll 1 line
- Press Space key to scroll 1 page
- Press Q key to quit

Observe and explain the output. You can also manually inspect the source files, if desired. Your local file system should resemble the screenshot below:



Exercise 9 – Collaboratively Developing Your Project and Recording Changes to the Repository

Suppose the team member has understood the changes that had been made by the team lead and is now ready to contribute to the project. You will create a new development branch for the team member, and also a new feature branch for the "Cashier Operation" module. We will then switch to the feature branch for the actual development work.

```
git branch d_teammember
git branch f_cashierop
git checkout f_cashierop
git status
git log --oneline --decorate --graph --all
```

Perform the following changes to develop the "Cashier Operation" module:

1. From sample source file `index.html`, copy the navigation code for "Cashier Operation" module to your `index.html`.
2. Create a new subdirectory `cashierOperation`.
3. Add a new HTML file `checkout.html` using the content from the matching sample source file.
4. Add a new HTML file `voidRefund.html` using the content from the matching sample source file.
5. Add a new HTML file `viewMySaleTransactions.html` using the content from the matching sample source file.

When you are done, commit the changes

```
git status
git add .
git status
git commit -m "created new subd and files for cashier op module"
git status
```

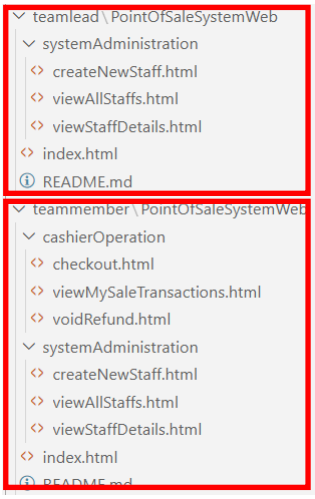
Then merge the changes for the "Cashier Operation" module to your main development branch:

```
git checkout d_teammember
git merge f_cashierop
git status
```

Observe and explain the output. Notice that this is also a fastforward merge.

Since you are not the team lead, you will not directly merge your changes to the staging branch. Instead, you will push all of your changes to the remote repository for the team lead to handle it.

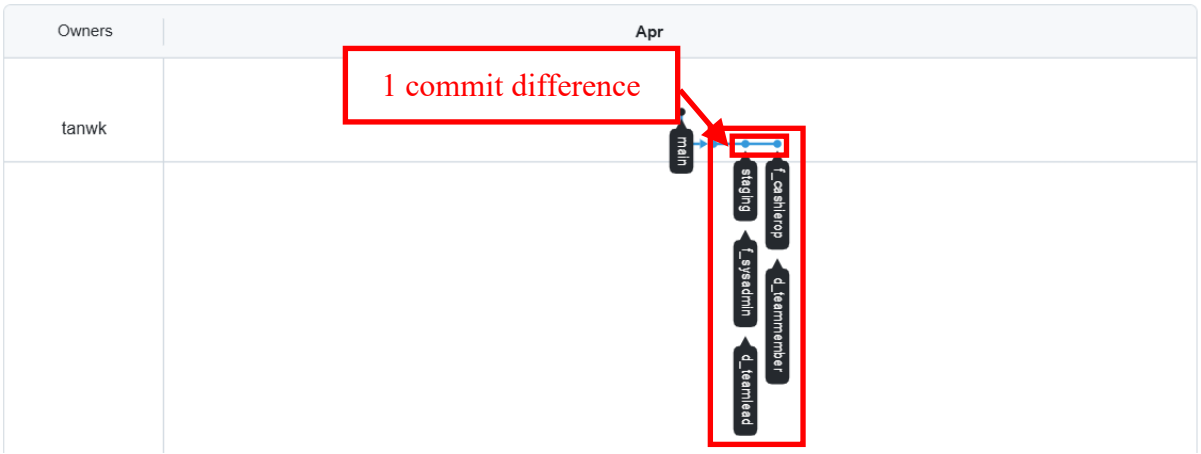
But before that, check your local file system, which should resemble the screenshot below. Observe that the `teamlead` directory does not contain the `cashierOperation` subdirectory at this juncture.



Now run the following commands to push your changes to the remote repository:

```
git remote show origin
git push origin --all
git remote show origin
```

Inspect your remote GitHub repository again and ensure that your changes are now safely stored on the remote repository. The graph of your GitHub remote repository should resemble the one below:



Observe that the team lead's branches and the staging branch are one commit behind that of the team member.

Exercise 10 – Integrating Changes Made by Collaborator

Assume back the role of team lead and fetch the changes from the remote repository. Inspect the changes that have been made by your team mate.

```
cd %USERPROFILE%\Documents\teamlead\PointOfSaleSystemWeb
git remote show origin
git fetch origin
git diff staging origin/d_teammember
```

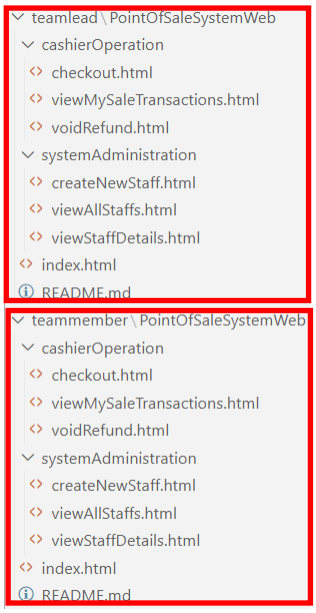
Observe and explain the output.

When you are satisfied that the changes are in order, merge your teammate's changes into the staging branch, the one in your local repository:

```
git status
git checkout staging
git merge origin/d_teammember
git status
```

Manually inspect the source files to ensure that the staging branch now incorporate the latest changes for both the "System Administration" module and the "Cashier Operation" module.

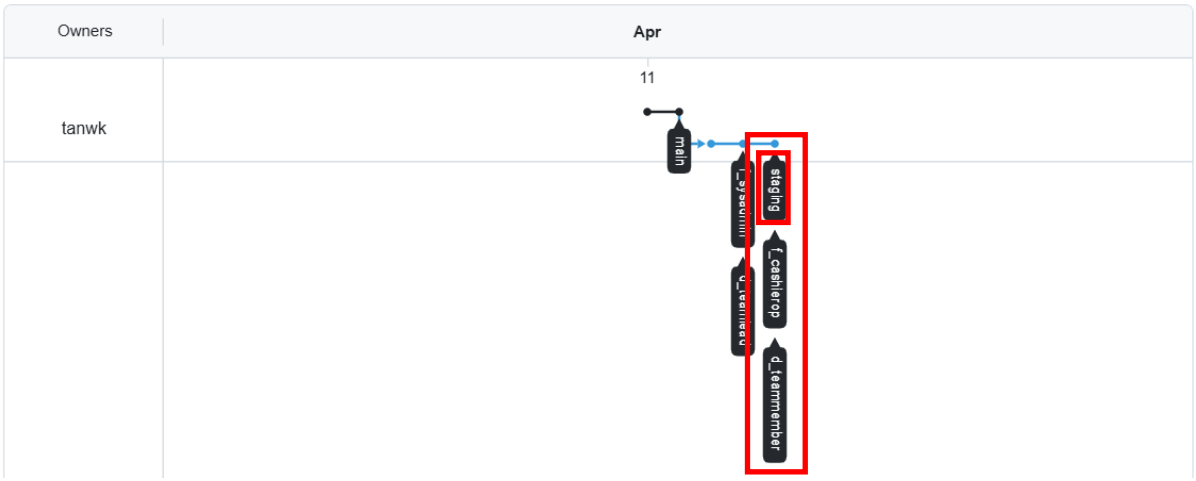
Check your local file system, which should resemble the screenshot below. Observe that the `teamlead` directory will now contain the `cashierOperation` subdirectory.



Finally, share the changes back with your teammate. This will ensure that the staging branch of your teammate is synchronised with yours after you have incorporated his/her changes.

```
git push origin --all
```

The graph of your GitHub remote repository should now resemble the one below. Observe that the



Observe that the team lead's branches are still one commit behind that of the team member. But the staging branch has now moved forward to point to the same commit as the team member's branches.

Exercise 11 – Handling Three-Way Merge with Merge Conflict

You will now simulate a divergent in the change history that would necessitate a three-way merge with merge conflict. You will then attempt to resolve the merge conflict.

Suppose the team member accidentally changed `index.html` in his staging branch and push the change. Further suppose the team lead also wanted to make a change to `index.html` before pushing the final set of changes to production, i.e., the main branch.

Within the team member workspace: execute the following command:

```
cd %USERPROFILE%\Documents\teammember\PointOfSaleSystemWeb
git fetch origin
git checkout staging
git merge origin/staging
```

Then modify line 24 of `index.html` to change the copyright year from "2025" to "April 2025". Then commit locally and push the change to the remote repository:

```
git add .
git commit -m "modified index.html copyright"
git push origin --all
```

Within the team lead workspace, modify line 24 of `index.html` to change the copyright year from "2025" to "May 2025". Then commit and push the change locally:

```
cd %USERPROFILE%\Documents\teamlead\PointOfSaleSystemWeb
git add .
git commit -m "modified index.html copyright"
git status
```

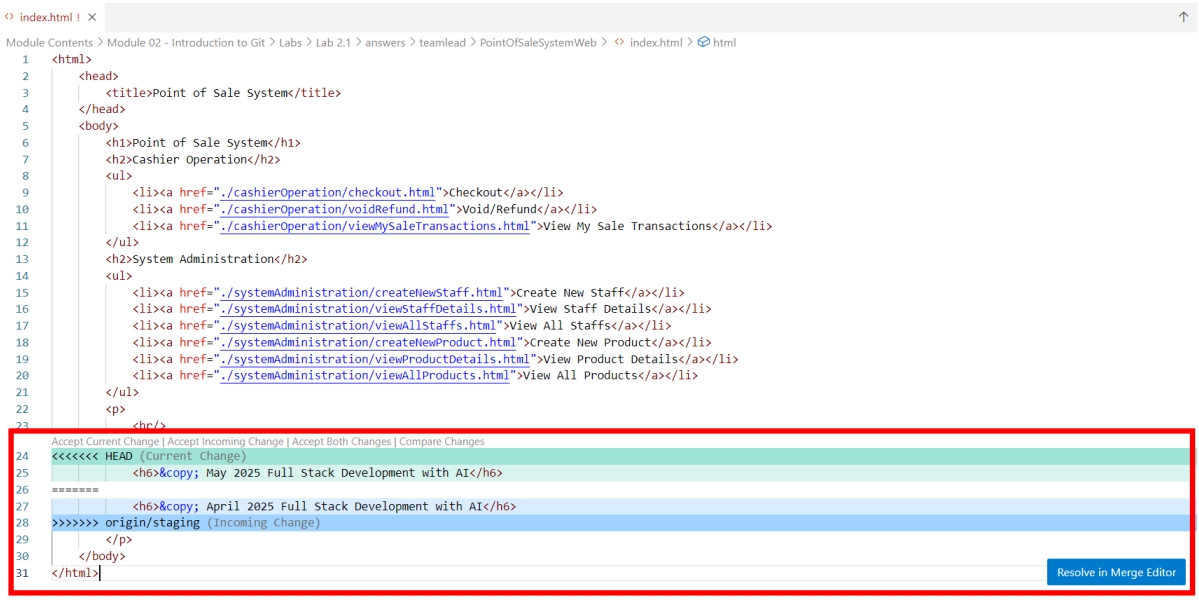
Now, let's try to merge the change to index.html made by the team member into the team lead's `index.html`, all on the staging branch:

```
git fetch origin
git merge origin/staging
```

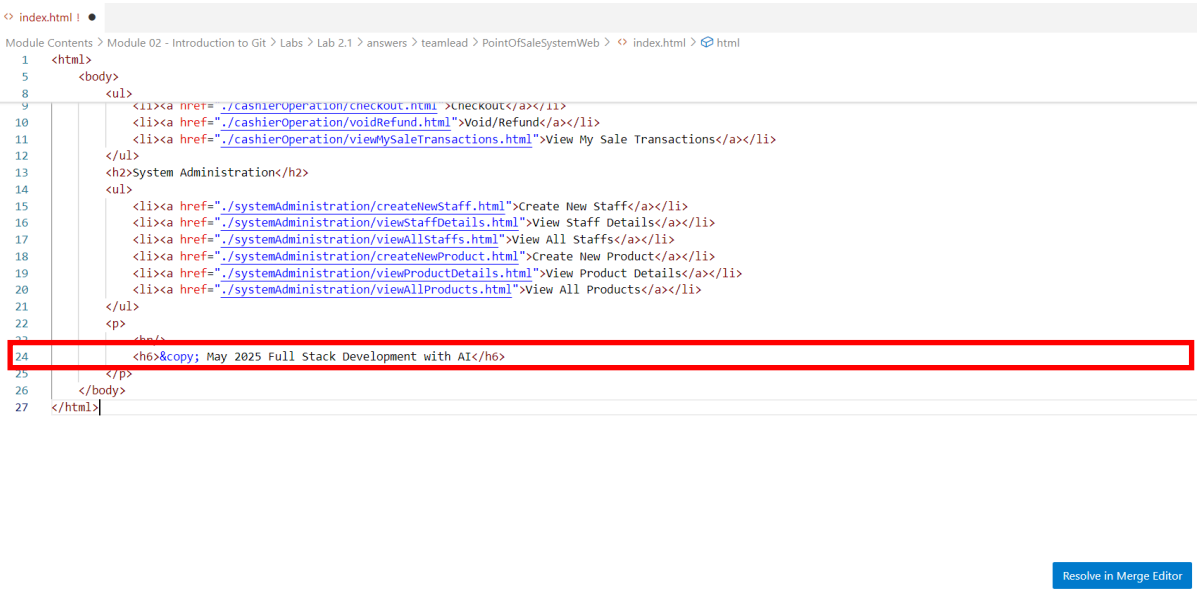
Observe and explain the output:

```
PROBLEMS   OUTPUT   TERMINAL   PORTS   POSTMAN CONSOLE   DEBUG CONSOLE

D:\Dropbox (Personal)\Teaching - NUS STMI\Emeritus - Full Stack AI\Module Contents\Module 02 - Introduc
tion to Git\Labs\Lab 2.1\answers\teamlead\PointOfSaleSystemWeb>git merge origin/staging
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Within the team lead workspace, open index.html in the text editor for editing. You should see something like the following. Notice the conflict resolution markers from line 24 to 28. The following screenshot is based on VS Code.



Resolve the merge conflict by accepting the change made by the team lead. This can be done by deleting the code below "=======", i.e., line 26 to line 28. Don't forget to remove the conflict resolution marker on line 24 as well.



Mark the conflict in `index.html` as resolve and finalise the merge conflict using the following commands:

```
git status

git add index.html

git status

git commit -m "merge commit, resolve conflict in index.html, accepted team lead staging"

git status

git push origin --all
```

Within the team member workspace, pull the resolved changes:

```
cd %USERPROFILE%\Documents\teammember\PointOfSaleSystemWeb
git fetch origin
git merge origin/staging
```

Open `index.html` in the team member workspace and verify that the copyright information is “May 2025” and not “April 2025”.

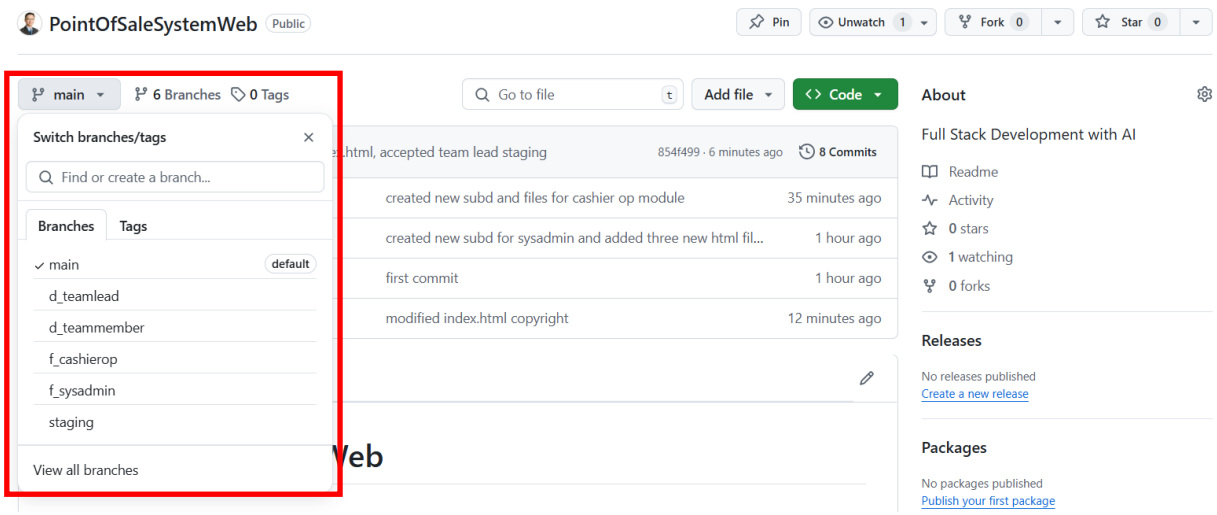
Exercise 12 – Pushing to Production

Finally, after the team lead has completed final testing, the source files from the staging branch can now be merged into the main branch for deployment to production by an automation server.

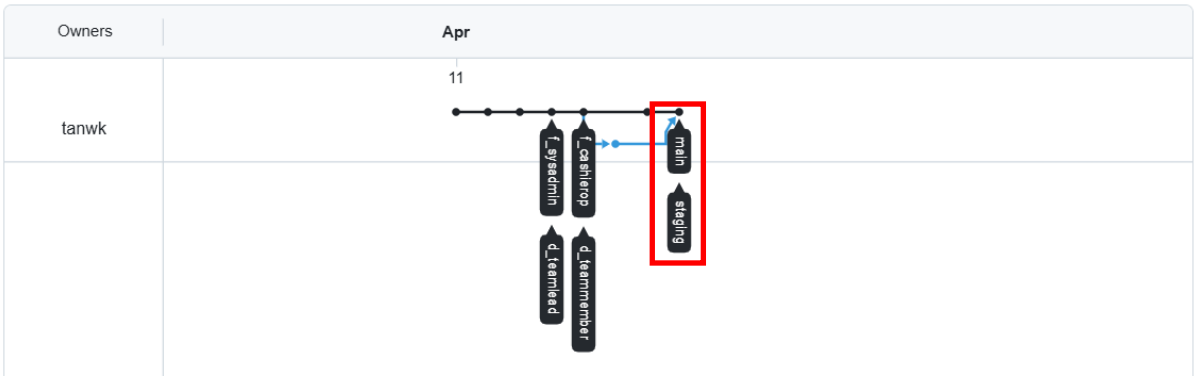
Run the following commands from within the team lead workspace:

```
cd %USERPROFILE%\Documents\teamlead\PointOfSaleSystemWeb
git checkout main
git merge staging
git push origin --all
```

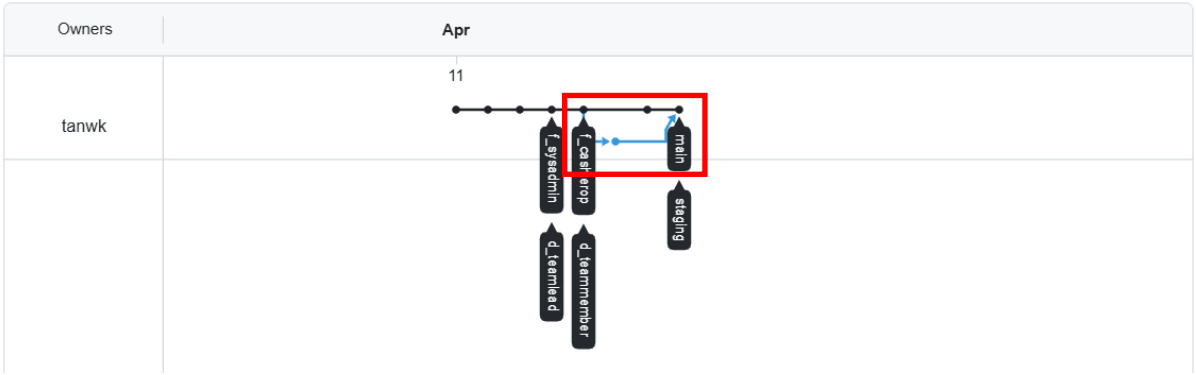
Inspect your remote GitHub repository again and ensure that all changes are now safely stored on the remote repository within the main branch.



The final graph of your GitHub remote repository should resemble the one below:



Observe that the main branch is now on the same commit as staging, pointing to the latest changes across all files.



More importantly, observe the divergence in the development history that caused the three-way merge and merge conflict earlier in Exercise 11. This divergence eventually merges back with the main development after the merge conflict has been resolved.

-- End of Lab --