



PUCIT
Mobile Application Development

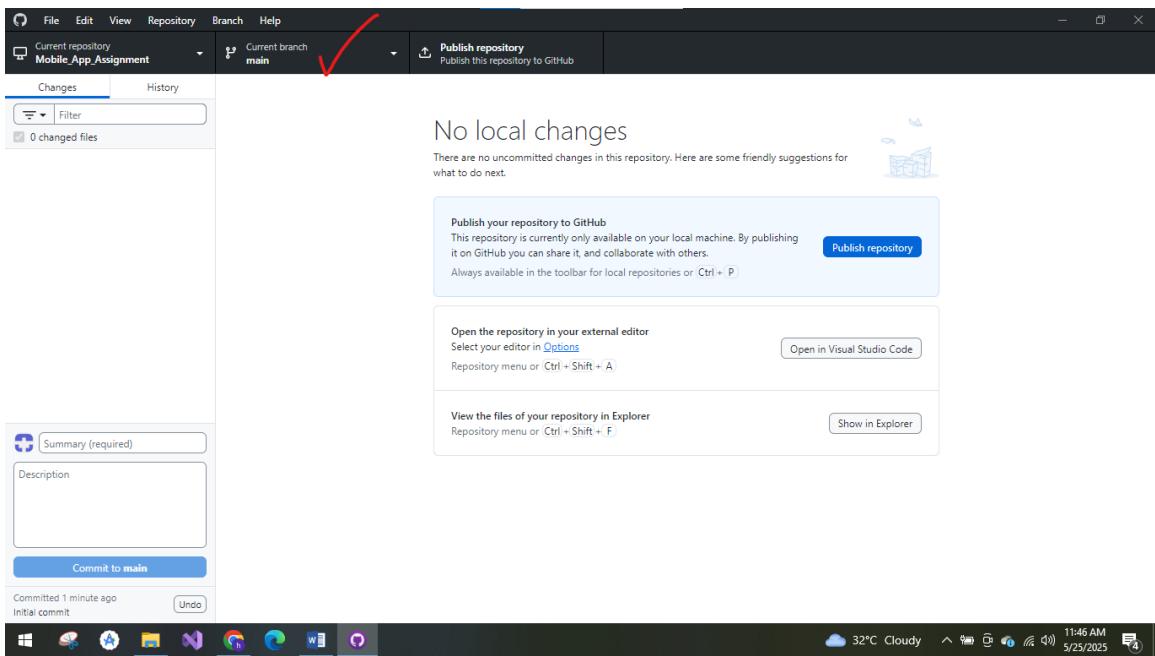
SUBMITTED TO: Professor Burhan Ali

SUBMITTED BY: **Hamza Zahoor** **BSEF22M531**

GIT Assignment:

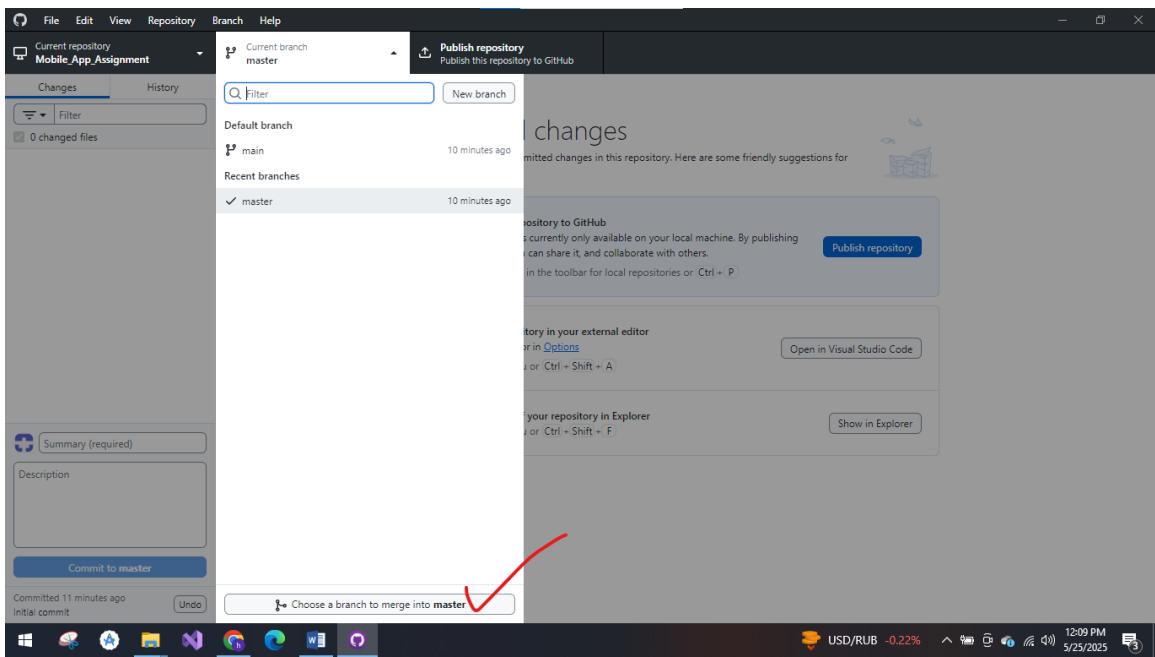
1. Difference Between Main and Master Branch:

- In Git, both main and master are branch names. Traditionally, master was the default branch name, but GitHub now uses main by default to promote more inclusive language. In GitHub Desktop:
- We can see which branch is the default when we clone or open a repository.
- If a project uses master, it will show that in the branch dropdown at the top center.
- If it uses main, that will be shown instead.
- They are conceptually the same thing. The convention just changed: previously the primary branch was called master and for newer repository that defaults to main. You should only ever have one of those in a single repository (nothing breaks if you have both, but it's unlikely to be intentional).
- Our project is using main branch.

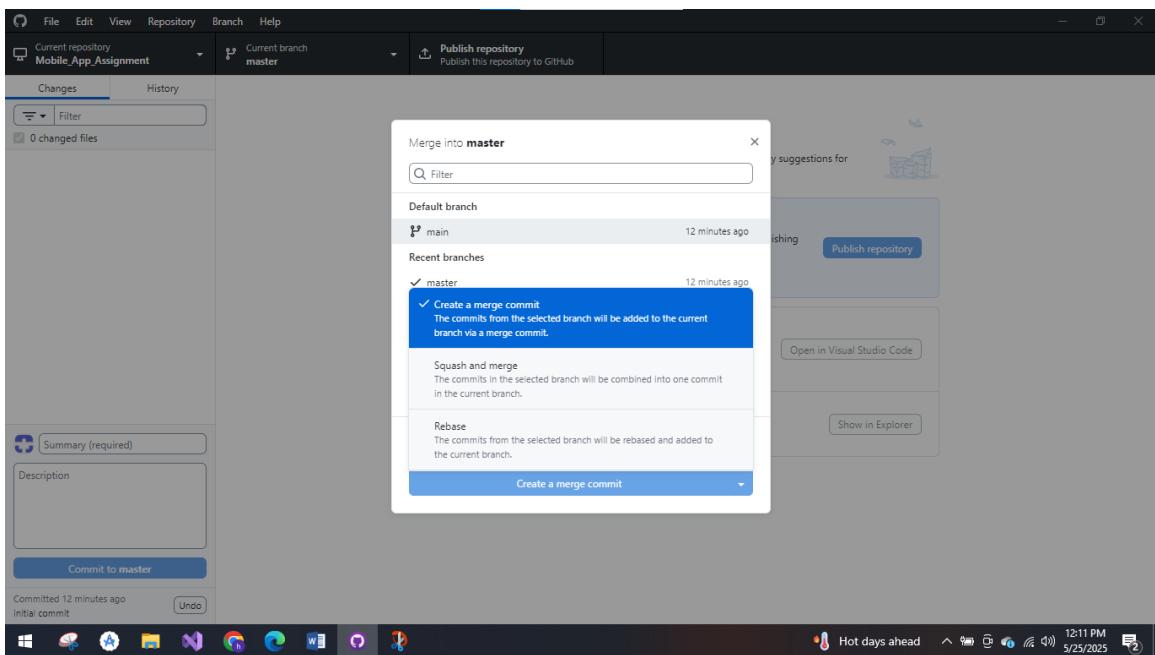


2. How to merge code into main to master:

- First of all we have to create a master branch if not already present. In GitHub desktop. When the repository is created, the default branch created is main. Add a new branch and name it master by simply clicking on branch dropdown and then clicking New Branch and naming it master.
- After doing this now select Master branch and click on Choose a branch to merge into master.



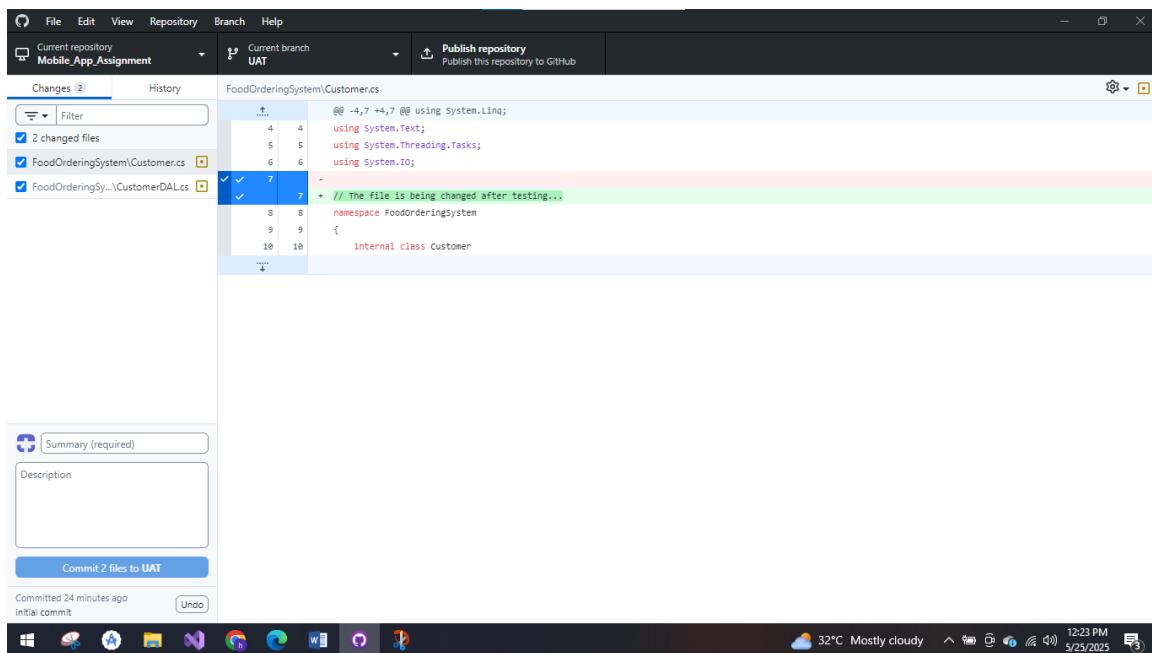
- After clicking it, then select main branch and then select create a merge commit so that master branch contains all changes from the main branch.



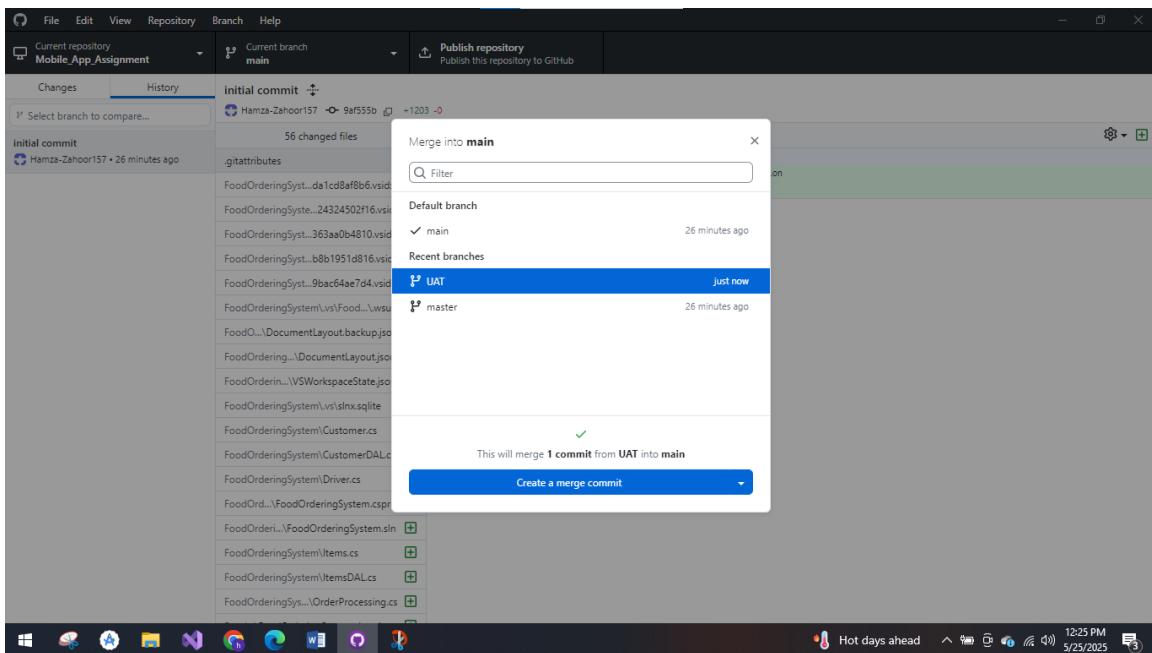
- After performing a merge from the main branch to the master branch using GitHub Desktop, I opened the History tab while on the main branch and compared it with the master branch. GitHub Desktop showed the message:
- “Your branch is up to date with the compared branch (master)”, which confirms that all the changes and commits are now successfully reflected in both branches. This visual confirmation ensures the code is now present in both main and master.

3. How to merge UAT branch into main branch:

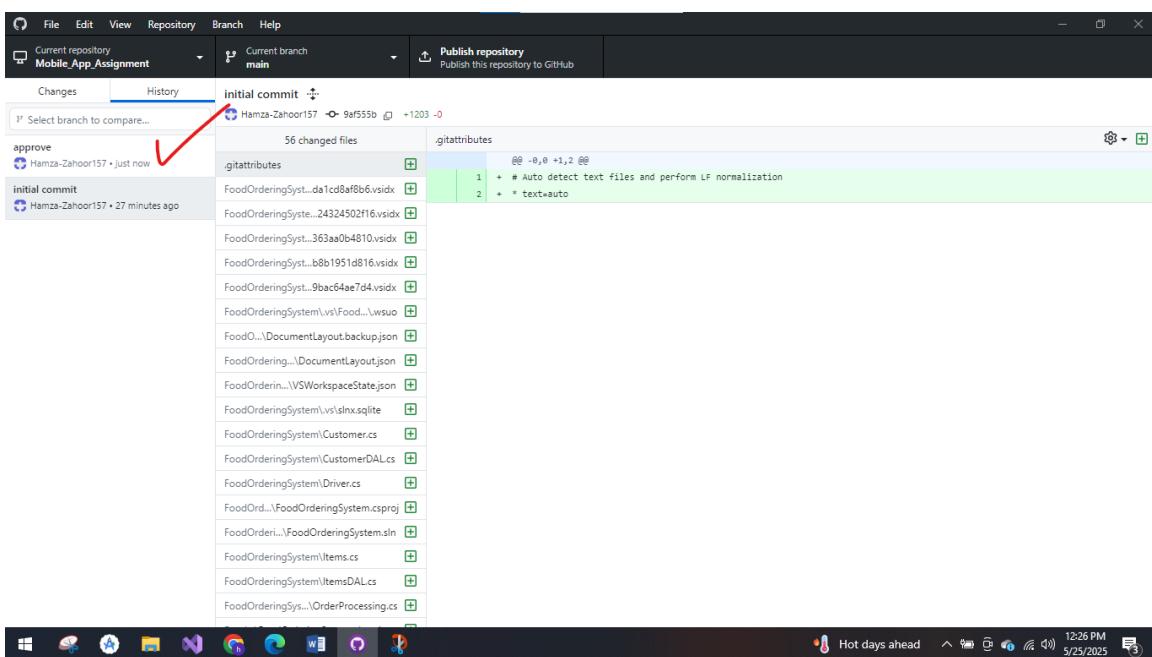
- The UAT branch stands for User Acceptance Testing, where final testing is done before the code goes to production.
- If there are changes made while UAT, let's say that following two files were changed while UAT and now we have to merge UAT into main branch.



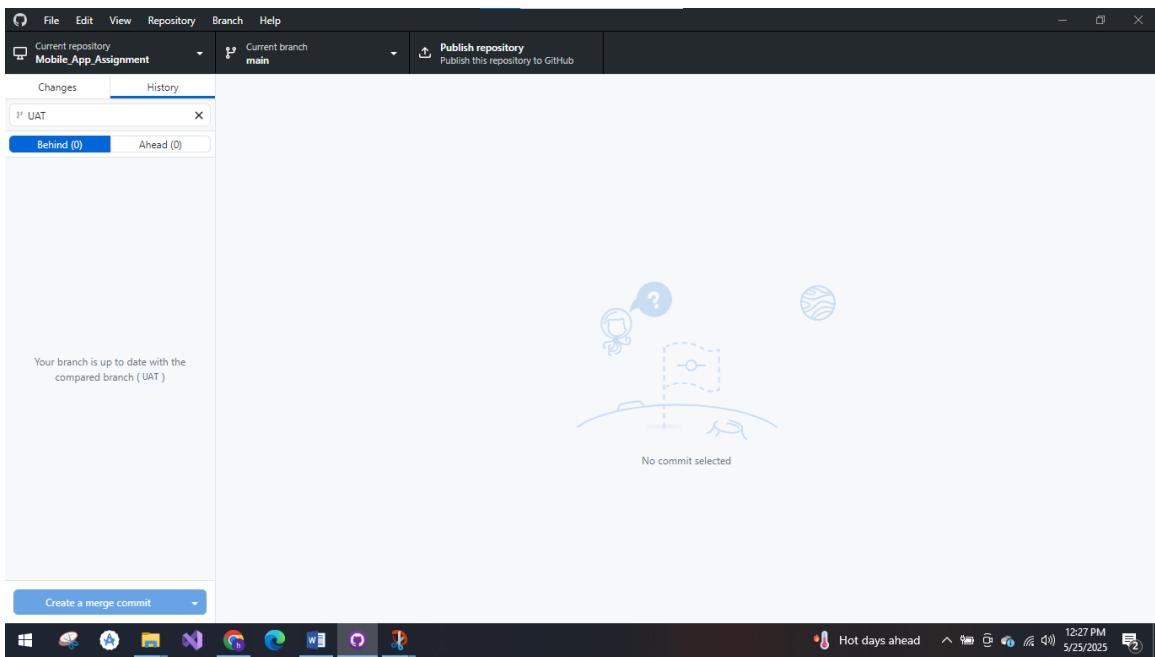
- In GitHub Desktop, I first switched to the main branch. Then I clicked on “Choose a branch to merge into main”, selected the UAT branch from the list, and clicked “Merge UAT into main”.



- This action brought all the approved changes from the UAT branch into the main branch.

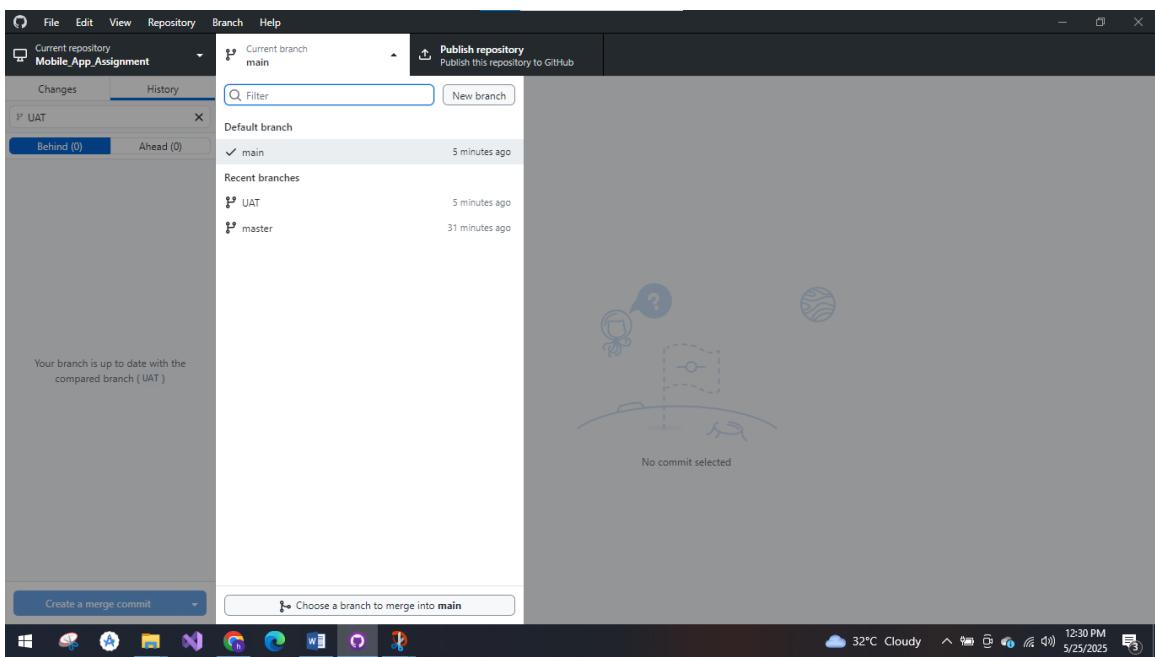


- After the merge, I checked the History tab to confirm the merge was successful and that the latest changes were now part of the main branch.



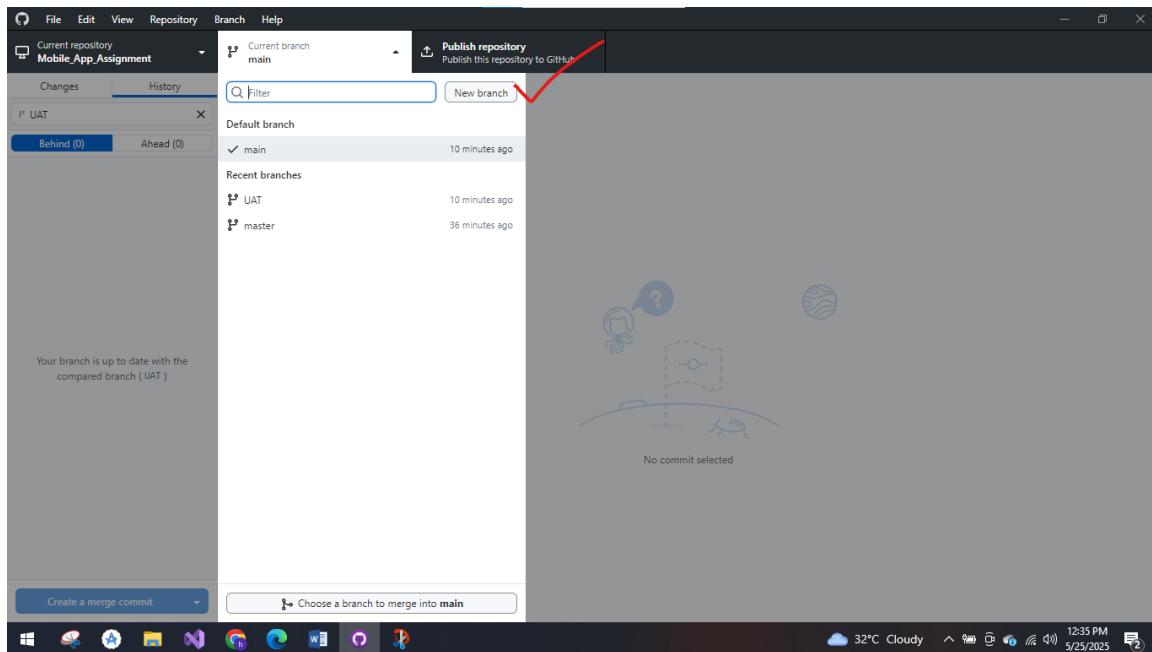
4. How to list all branches in GIT:

- In GitHub Desktop, I can list all branches of a repository by clicking on the "Current branch" dropdown.
- This shows all available branches, including the currently active one and other local branches.
- It also provides a filter box for quickly finding a specific branch and an option to create a new branch.

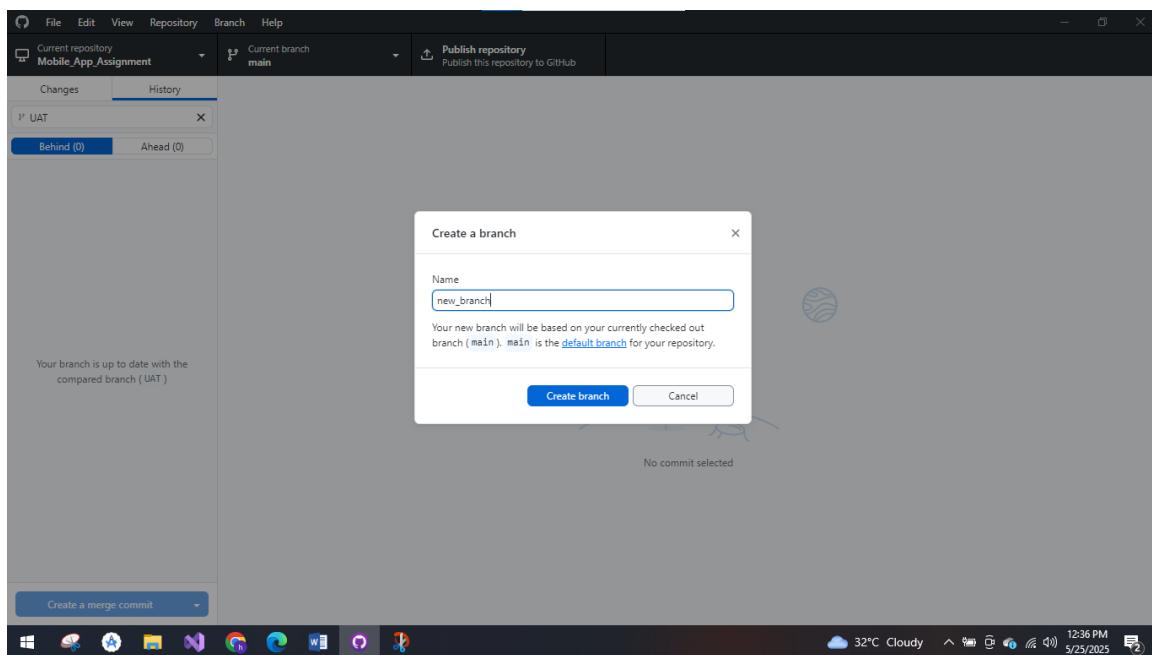


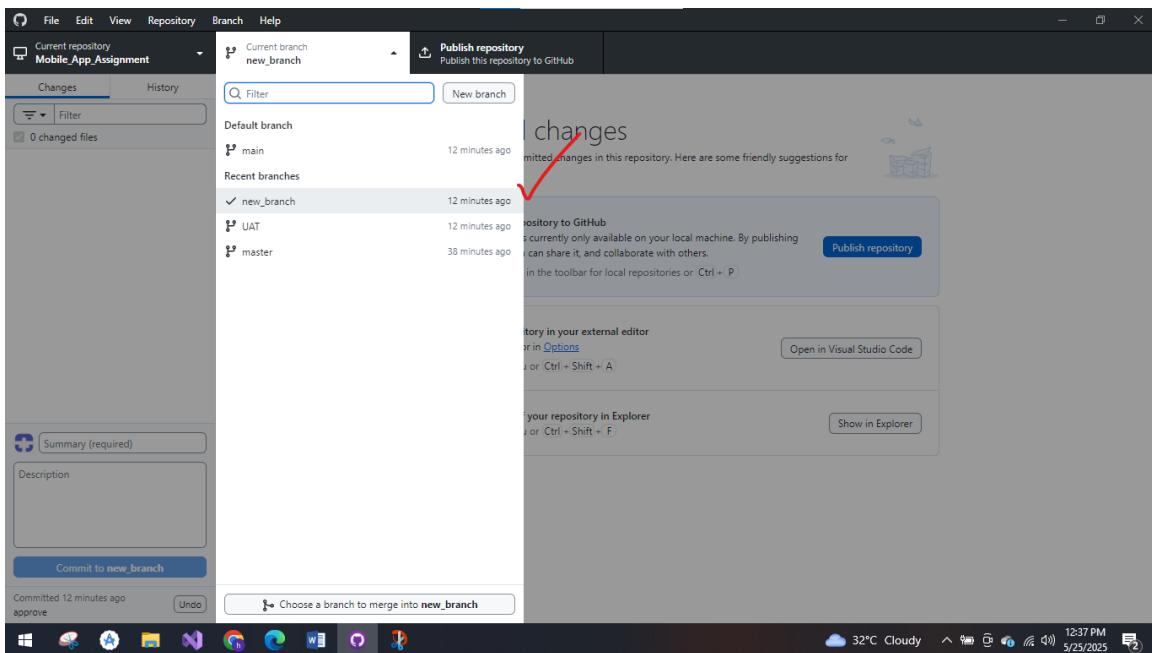
5. How to create a branch in GIT:

- In GitHub Desktop, a new branch can be created by clicking the “**Current Branch**” dropdown and selecting “**New Branch**”.



- Then I enter a branch name and select a base branch (usually main). After clicking **Create Branch**, the new branch is ready for development.

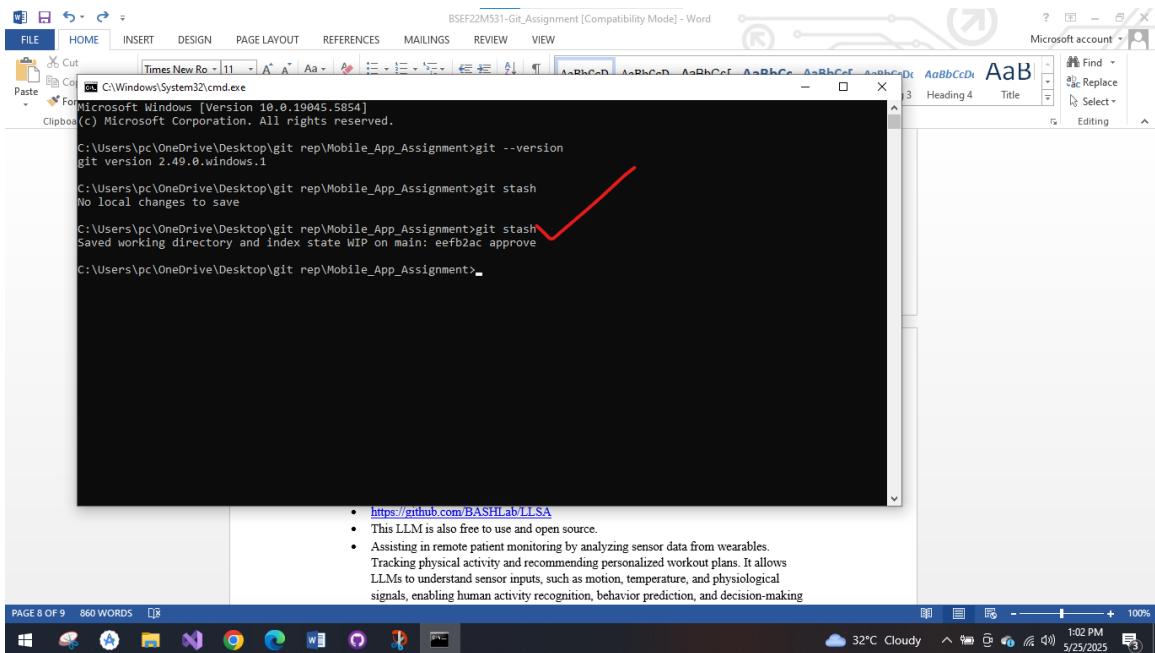




- Optionally, I can publish it to GitHub by clicking "**Publish branch**" so others can access it. This helps in isolating features or fixes before merging them into the main codebase.

6. Git Stash:

- Git stash is a Git feature that allows you to temporarily save your uncommitted changes (like edits to files) without committing them. This is helpful when
- You're in the middle of some work. But need to quickly switch to another branch. Without committing incomplete or messy code. It hides your changes, letting you work cleanly, and later you can reapply (restore) those changes.
- In GitHub Desktop, stash is not directly available, but we can use the **terminal** to run git stash to save changes and git stash pop to restore them later. This helps keep the working directory clean and supports smooth branch switching without losing progress. Steps are as follows:
 - Make sure the repo is open in GitHub Desktop where you have uncommitted changes.
 - Open the path of your project and open command line and enter command: **git stash**.
 - This will **save your current uncommitted changes** and clean your working directory.
 - Output will say: Saved working directory and index state WIP on main: <commit message>:



7. Git Tagging:

- Git tagging is used to label specific points in your project's history most commonly to mark release versions like v1.0, v2.1, etc. Tags are like bookmarks that help you find and refer to specific commits easily.
- Go to your project and open command line and enter **git tag v1.0**
- In order to create annotated tags based on the current commit enter **git tag -a v1.0 -m "First release of Food Ordering System"**
- Following in the screenshot, first tag v1.1 is created and then v1.2 which is annotated one and then all tags are listed using command: **git tag**

The screenshot shows a Windows File Explorer window with a command prompt session. The title bar says "Mobile_App_Assignment". The address bar shows "C:\Windows\System32\cmd.exe". The command history includes:

- Microsoft Windows [Version 10.0.19045.5854]
- © Microsoft Corporation. All rights reserved.
- git tag v1.2
- git tag -a v1.2 -m "First Release of Food Ordering System"

The file tree on the left shows a folder structure with files v1.0, v1.1, and v1.2. The v1.2 folder contains a file named "First Release of Food Ordering System". The taskbar at the bottom shows various pinned icons and the system tray with a battery icon, network status, and date/time.