



PUCIT

Mobile Application Development

SUBMITTED TO:

Professor Burhan Ali

SUBMITTED BY:

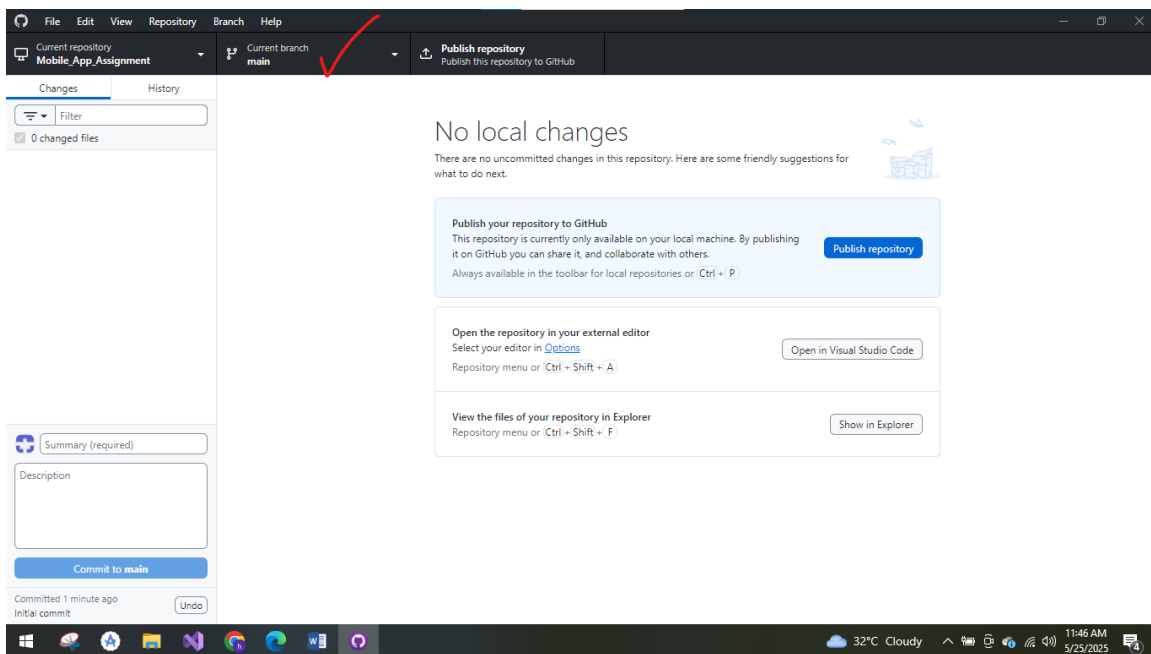
Hamza Zahoor

BSEF22M531

GIT Assignment:

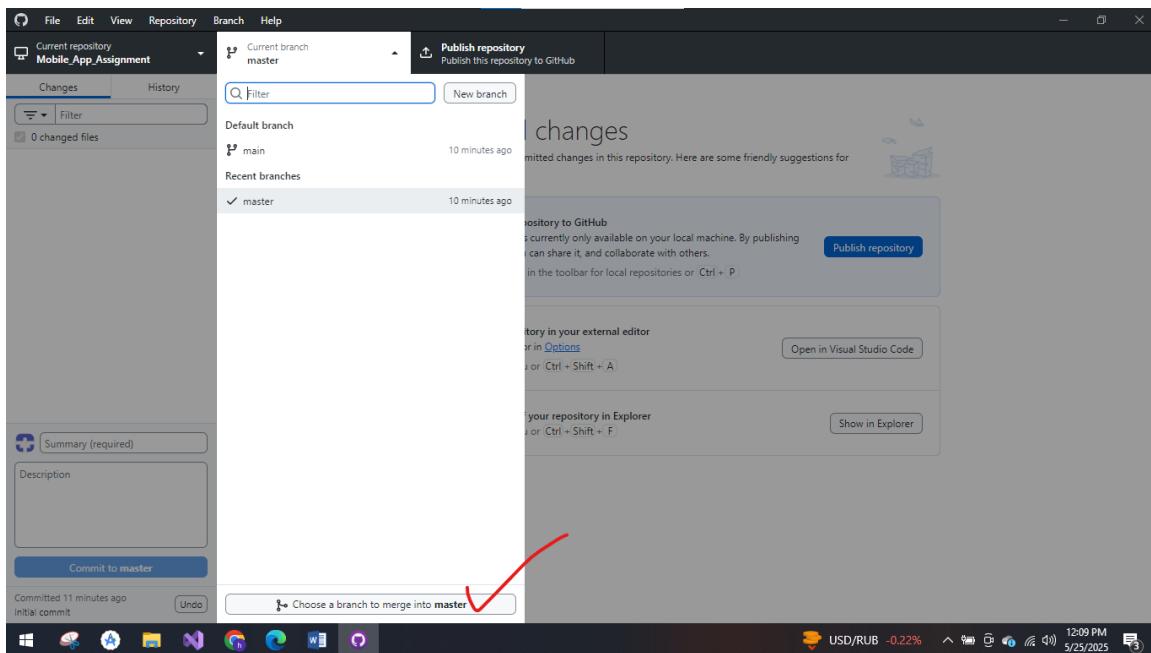
1. Difference Between Main and Master Branch:

- 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.

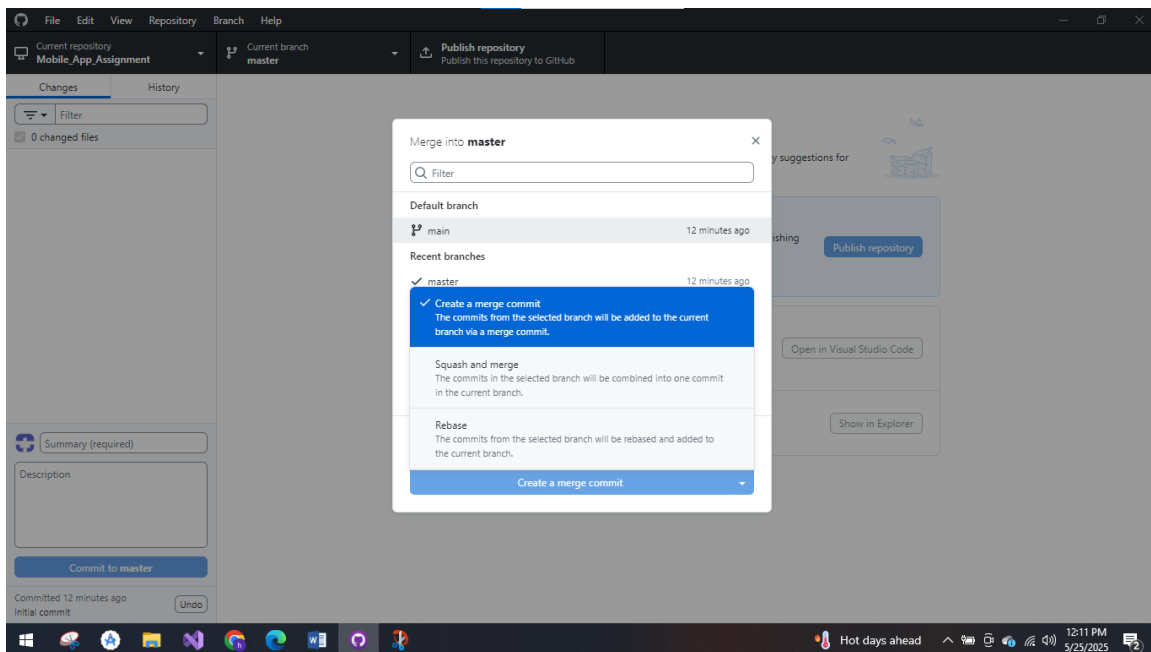


2. How to merge code into main to master:

- 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.



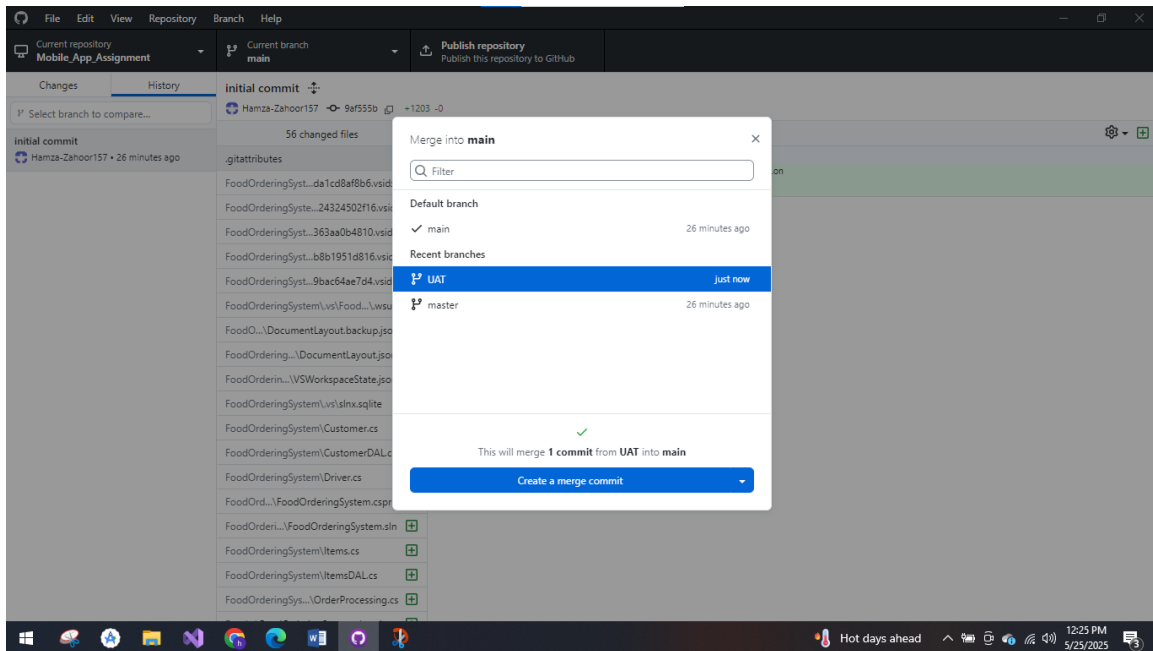
- After clicking it, then select main branch and then select create a merge commit so that master branch contains all changes from the main branch.



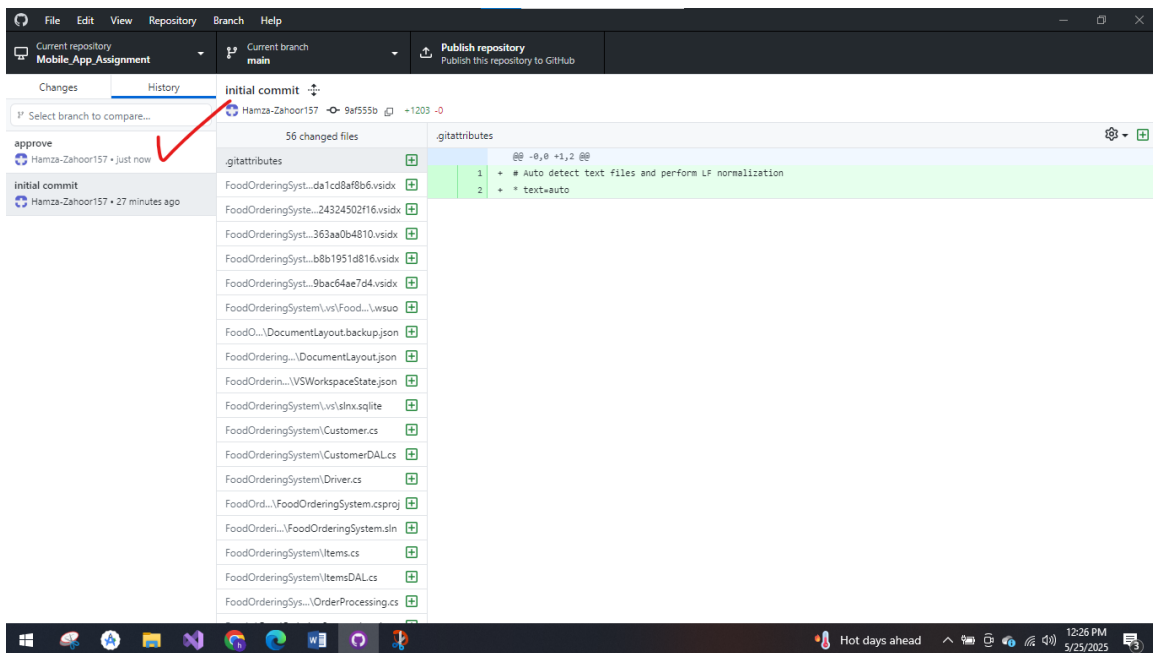
- ### **3. How to merge UAT branch into main branch:**

- The image is a screenshot of the Visual Studio IDE. At the top, the menu bar includes File, Edit, View, Repository, Branch, and Help. Below the menu bar is a toolbar with three main sections: 'Current repository' showing 'Mobile_App_Assignment', 'Current branch' showing 'UAT', and a 'Publish repository' button with the text 'Publish this repository to GitHub'. The main workspace is divided into a 'Changes' sidebar on the left and a diff view on the right. The 'Changes' sidebar shows a list of files with checkboxes, indicating '2 changed files': 'FoodOrderingSystem\Customer.cs' and 'FoodOrderingSystem\CustomerDAL.cs'. Below this list is a 'Commit 2 files to UAT' button. The diff view on the right shows the changes for 'FoodOrderingSystem\Customer.cs'. It displays a comparison between the current state and the previous commit. The changes include adding 'using System.Linq;', 'using System.Threading.Tasks;', and 'using System.IO;' at the top. A new line is added: '// The file is being changed after testing...'. The namespace is defined as 'namespace FoodOrderingSystem' and a new internal class 'Customer' is added. The bottom status bar shows 'Committed 24 minutes ago' and 'Initial commit'. The Windows taskbar at the very bottom shows the system clock as 12:23 PM on 5/25/2025, along with various system icons and the temperature 32°C.

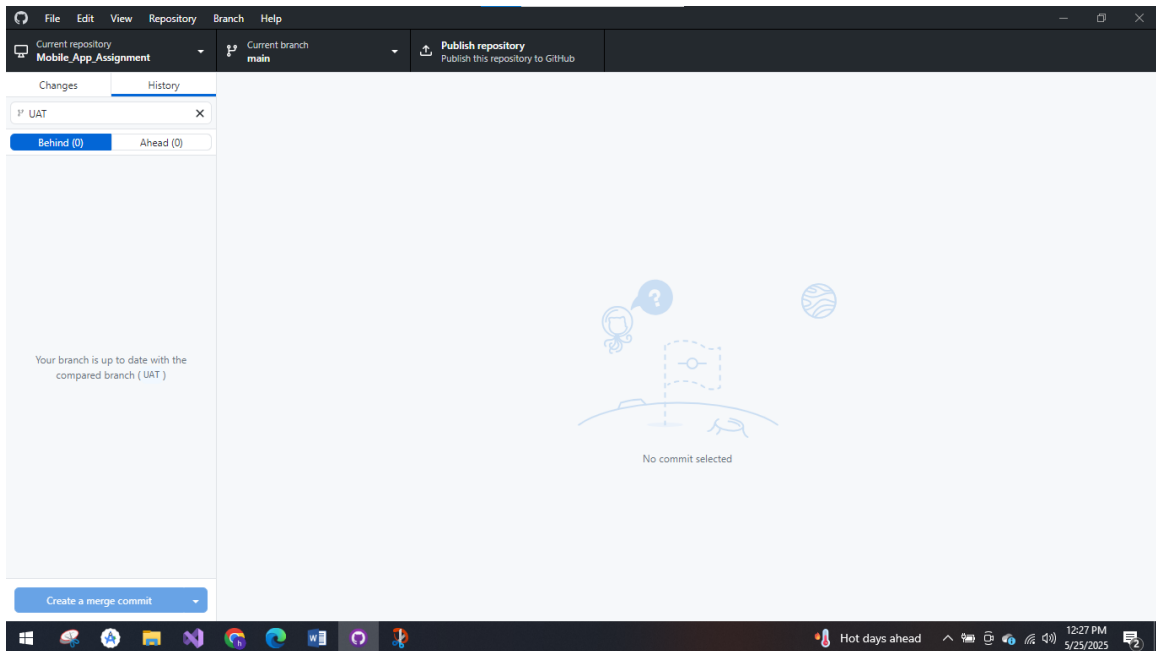
- 4



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

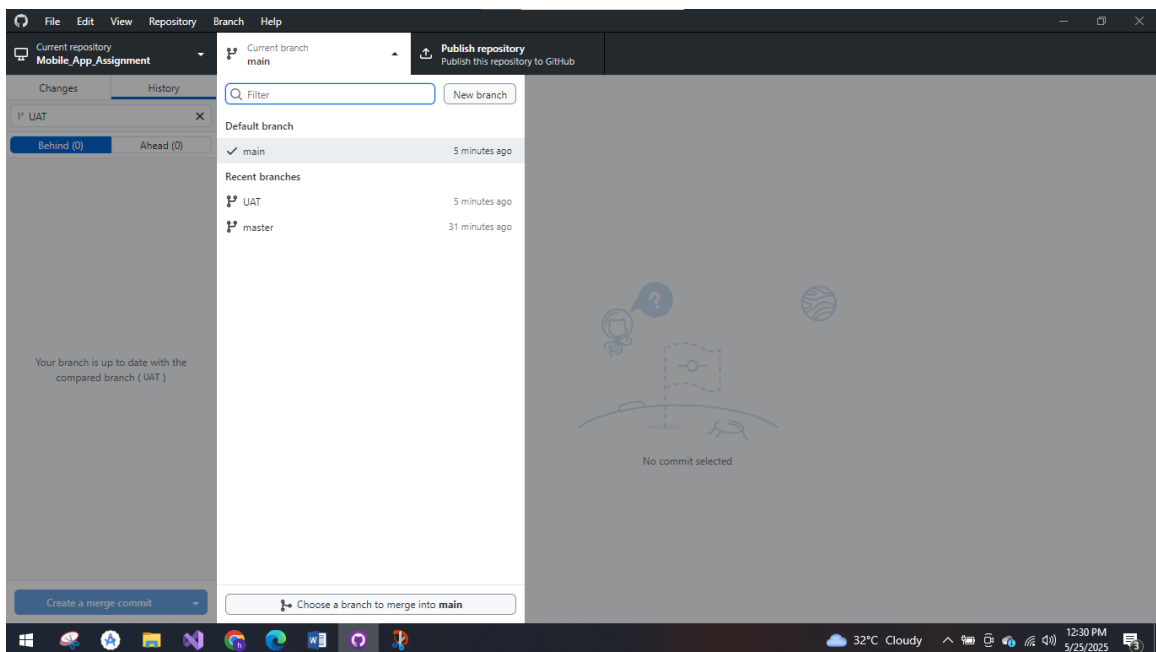


- After the merge, I checked the History tab to confirm the merge was successful and that the latest changes were now part of the main branch.



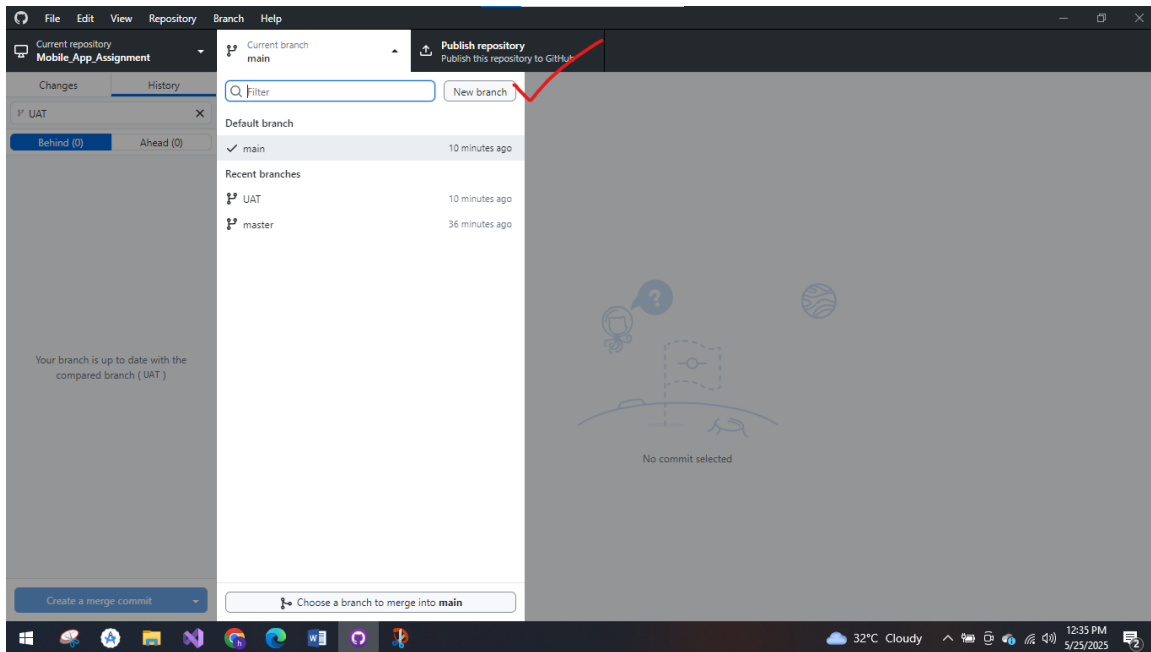
4. How to list all branches in GIT:

- 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.

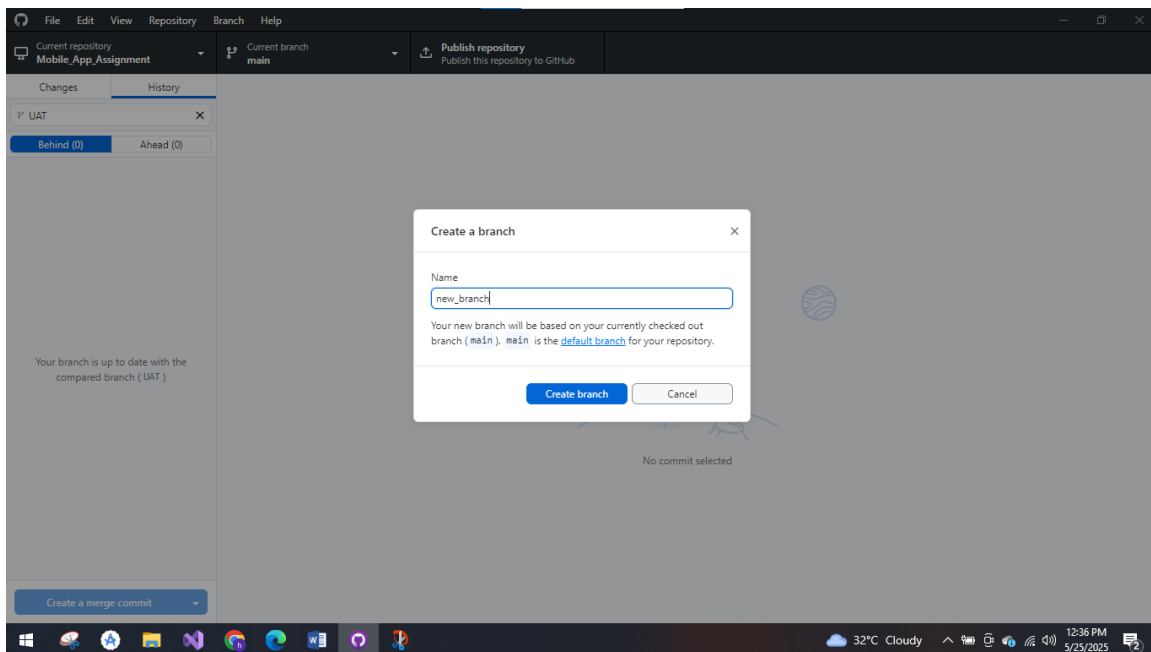


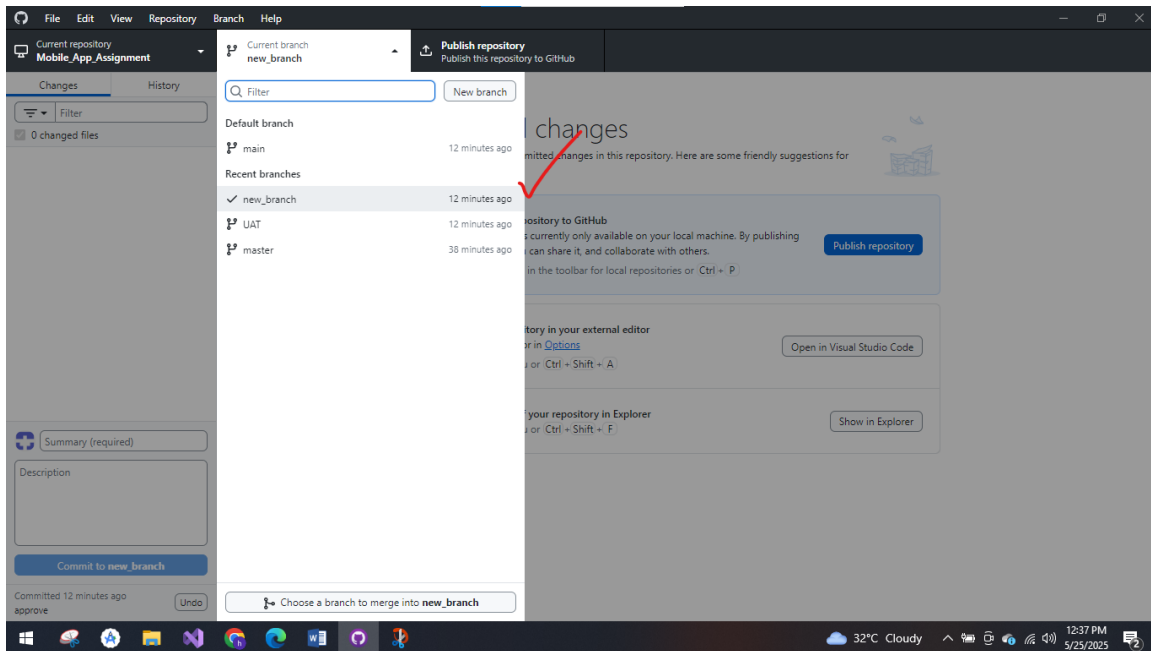
5. How to create a branch in GIT:

- 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.

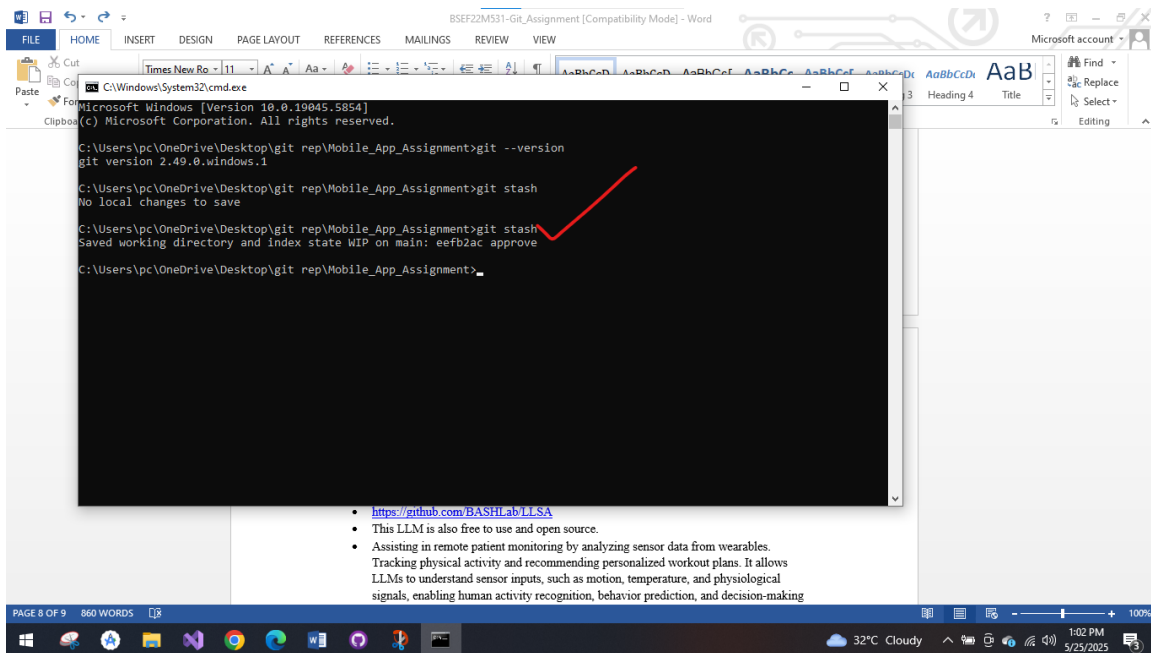




- Optionally, I can publish it to GitHub by clicking "**Publish branch**" so others can access it. This helps in isolating features or fixes before merging them into the main codebase.

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>:



- Now we can Switch to another branch in GitHub Desktop and do other commits/work safely
- Once ready, open the terminal again and type: **git stash pop**
- This will restore the changes you saved earlier.

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**

