

**Cloud Computing BSE-VB**

**Submitted By**

Tooba Shafique (2023-BSE-065)

**Submitted to**

Sir Shoaib

**LAB-03**

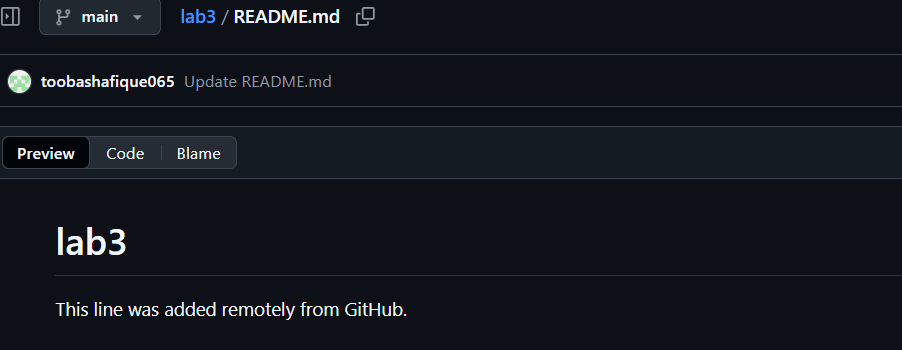
**Task 1 – Handling Local and Remote Commit Conflicts**

(Pull vs Pull --rebase)

**Step 1 – Remote Edit on GitHub**

Open your GitHub repository and edit the README.md file directly in the browser.  
Add a new line “This line was added remotely from GitHub.”  
Commit the change.

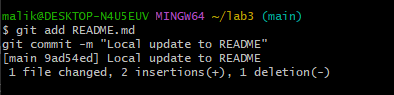
**Screenshot:** remote\_edit.png



**Step 2 – Local Edit in Git Bash**

Open the same repository folder on your local machine.  
Edit the README.md file and add a new line “This line was added locally.”  
Save and commit the change locally.

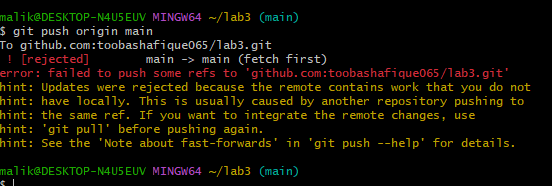
**Screenshot:** local\_commit.png



**Step 3 – Push and Observe Conflict**

Try pushing the local changes to GitHub.  
An error appears because the remote repository already has a new commit.

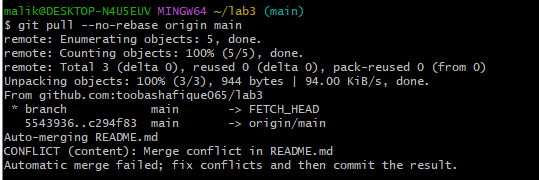
**Screenshot:** push\_error.png

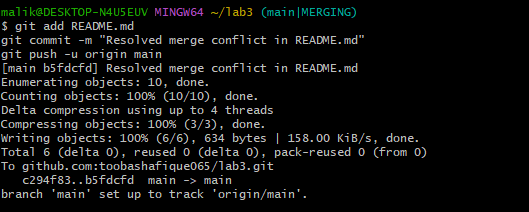


**Step 4 – Fix Using Merge**

Pull the latest changes from GitHub using merge.  
Git merges both changes and creates a merge commit.  
Push the merged update back to GitHub.

**Screenshots:** merge\_commit.png and push\_after\_merge.png

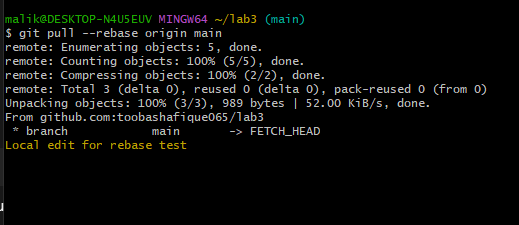


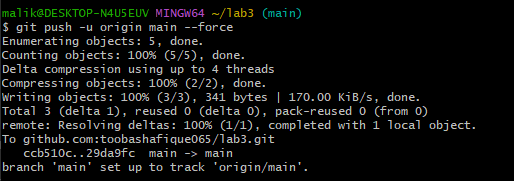


**Step 5 – Repeat Using Rebase**

Make another edit on GitHub and another local edit.  
Pull the latest changes using rebase instead of merge.  
Push the rebased updates to GitHub.

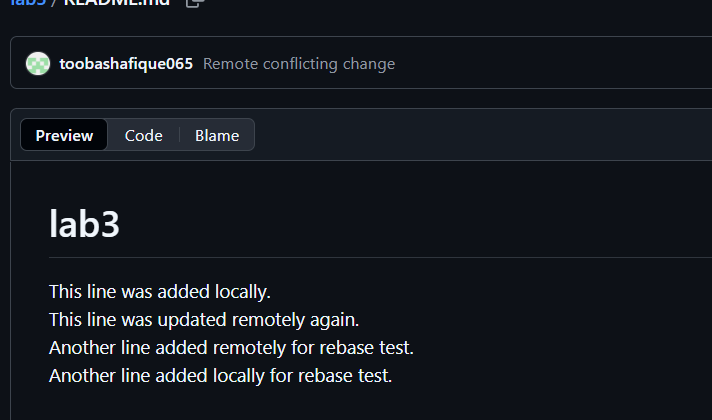
**Screenshots:** rebasepull.png, push\_after\_rebase.png



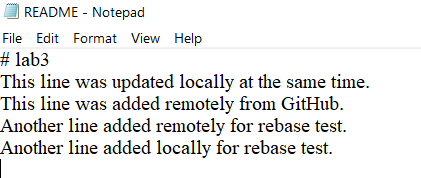


**Task 2 – Creating and Resolving Merge Conflicts Manually**

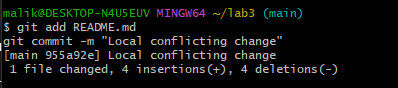
**Step 1:**  
On GitHub, open your README.md file and change an existing line to:  
“This line was updated remotely again.”  
Commit the change with the message “Remote conflicting change.”  
**Screenshot:** remote\_conflict\_edit.png



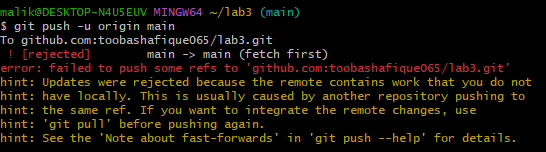
**Step 2:**  
On your local machine, open the same README.md file and edit the same line to:  
“This line was updated locally at the same time.”  
Screenshot: local\_conflict\_edit.png



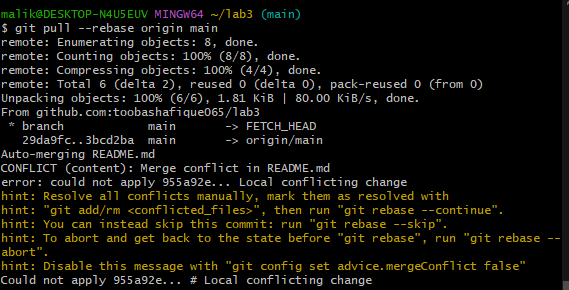
**Step 3:**  
Stage and commit your local change.  
Screenshot: local\_conflict\_commit.png



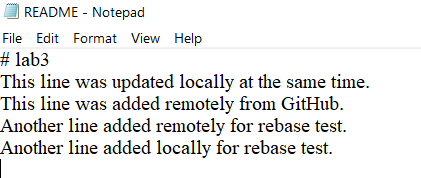
**Step 4:**  
Try to push your local changes. The push will be rejected because the remote has conflicting changes.  
Screenshot: conflict\_push\_error.png



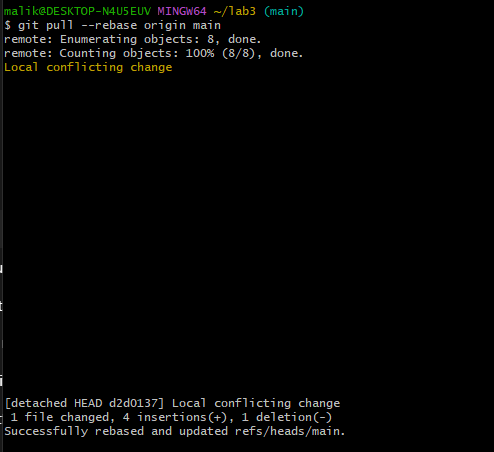
**Step 5:**  
Pull with rebase to bring in remote changes. Git will stop and show a conflict message.  
Screenshot: conflict\_message.png



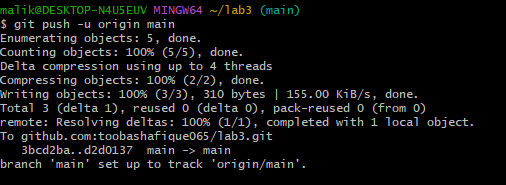
**Step 6:**  
Open the README.md file in your editor. You’ll see conflict markers.  
Edit the file manually to keep the correct version and remove conflict markers.  
Screenshot: resolved\_readme.png



**Step 7:**  
Mark the conflict as resolved and continue the rebase.  
Screenshot: rebase\_continue.png



**Step 8:**  
Push your resolved changes to GitHub.  
Screenshot: push\_after\_resolve.png



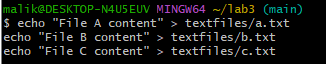
**Task 3 – Managing Ignored Files with .gitignore and Removing Tracked Files**

**Steps**

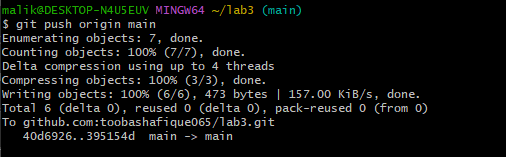
**Step 1:**  
Create a new folder named textfiles inside your repository.  
Screenshot: folder created

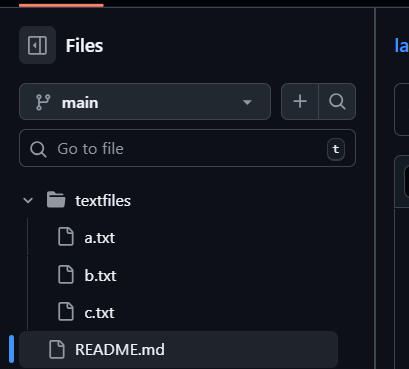


**Step 2:**  
Create three text files inside the textfiles folder named a.txt, b.txt, and c.txt.  
Screenshot: three files created

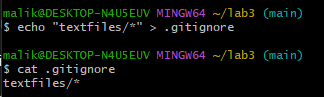


**Step 3:**  
Add and commit the new directory, then push to GitHub.  
Screenshot: push\_textfiles.png

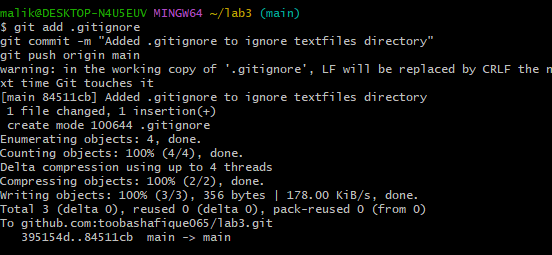




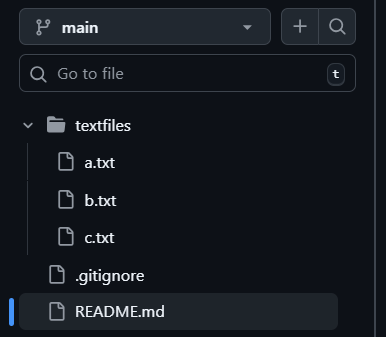
**Step 4:**  
Create a new .gitignore file in the root of your repository and add a rule to ignore the textfiles directory.  
Screenshot: gitignore file created



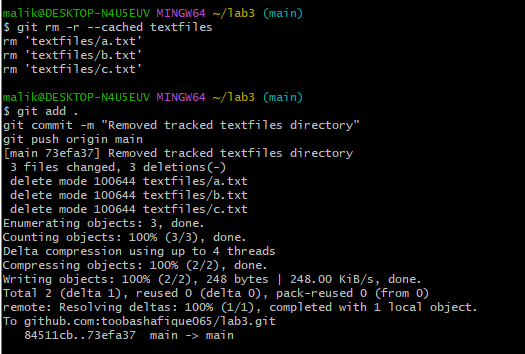
**Step 5:**  
Add, commit, and push the .gitignore file.  
Screenshot: gitignore\_push.png



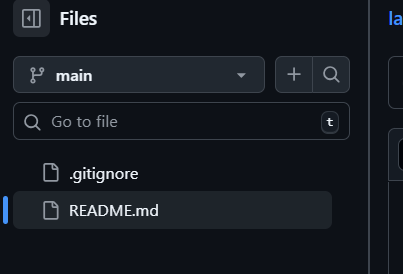
**Step 6:**  
Go to your GitHub repository and notice that the textfiles directory is still visible.  
Screenshot: repo\_still\_has\_textfiles.png



**Step 7:**  
Remove the textfiles folder from Git’s tracking (without deleting it locally).  
Screenshot: rm\_cached\_push.png



**Step 8:**  
Check your GitHub repository again — the textfiles folder should now be removed from the remote repo.  
Screenshot: repo\_textfiles\_removed.png

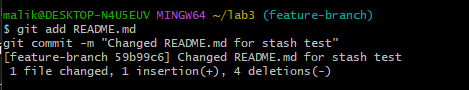


**Task 4 – Create Temporary Changes and Use git stash**

**Step 1**

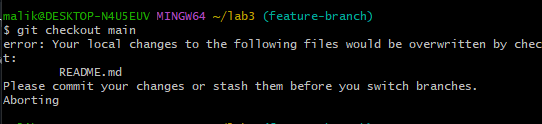
Create a new feature branch and make changes to a file.  
**Screenshot:** modified\_readme.png





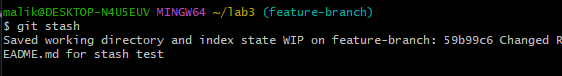
**Step 2**

Try to switch to another branch without committing the changes to see the error message.  
**Screenshot:** checkout\_error.png



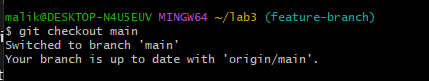
**Step 3**

Temporarily save your uncommitted changes using stash.  
**Screenshot:** stash\_command.png



**Step 4**

Switch branches successfully after stashing your changes.  
**Screenshot:** branch\_switched.png



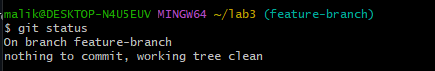
**Step 5**

Return to the previous branch to continue working.  
**Screenshot:** back\_to\_feature.png



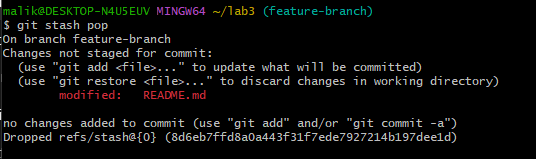
**Step 6**

Check the working directory status to confirm it is clean.  
**Screenshot:** status\_clean.png



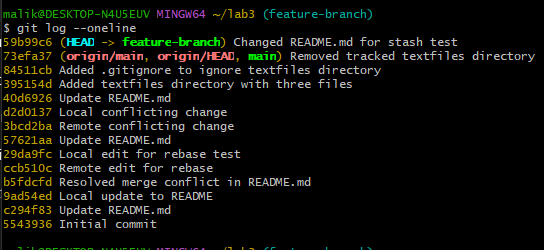
**Step 7**

Restore your stashed changes to bring them back into the working directory.  
**Screenshot:** stash\_pop.png

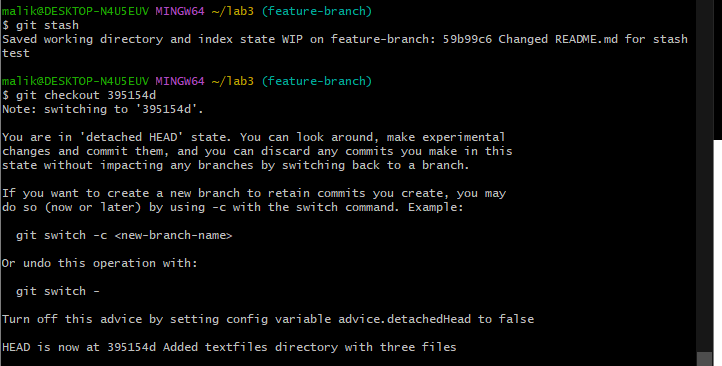


**Task 5 – Checkout a Specific Commit Using git log**

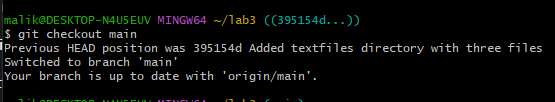
**Step 1:**  
View the commit history in your repository.  
Screenshot: log\_before\_checkout.png



**Step 2:**  
Checkout that specific commit to view your project’s past state.  
Screenshot: detached\_head.png

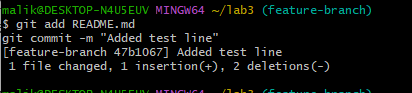


**Step 3:**  
Return to your main branch after checking that commit.  
Screenshot: back\_to\_main.png

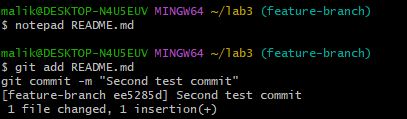


**Task 6 – Resetting Commits (Soft vs Hard Reset) (With Verification Steps)**

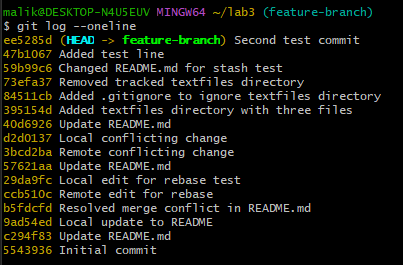
**Step 1:**  
Edit any file (for example, README.md) and add a new line. Commit the change.  
**Screenshot:** first\_commit.png



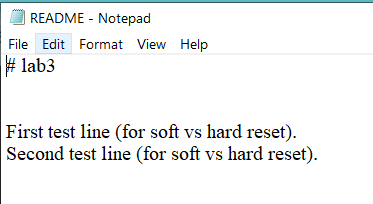
**Step 2:**  
Edit the file again and commit the new change.  
**Screenshot:** second\_commit.png



**Step 3:**  
View the commit history before performing a reset.  
**Screenshot:** log\_before\_reset.png



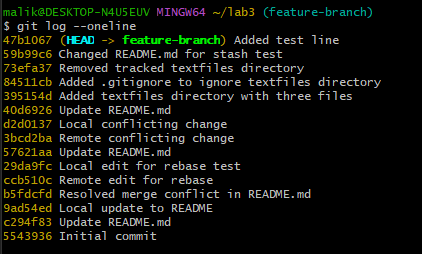
**Step 4:**  
Open the edited file and confirm that both added lines exist.  
**Screenshot:** file\_before\_reset.png



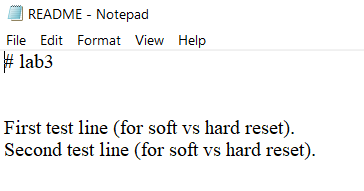
**Step 5:**  
Perform a soft reset to move back one commit while keeping your changes.  
**Screenshot:** soft\_reset.png



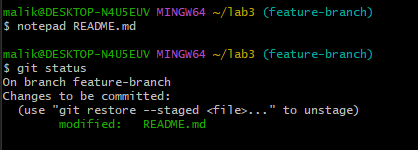
**Step 6:**  
Check the commit history again after the soft reset.  
**Screenshot:** log\_after\_soft\_reset.png



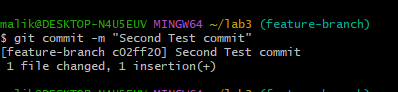
**Step 7:**  
Open the file again and confirm both edits are still present.  
**Screenshot:** file\_after\_soft\_reset.png



**Step 8:**  
Check git status; the changes should be staged and ready to commit again.  
**Screenshot:** status\_after\_soft\_reset.png



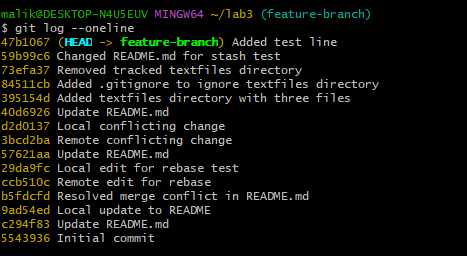
**Step 9:**  
Commit the staged changes again after the soft reset.  
**Screenshot:** commit\_after\_soft\_reset.png



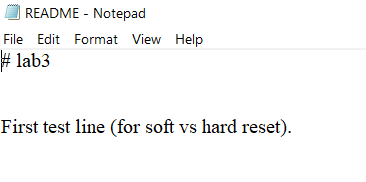
**Step 10:**  
Perform a hard reset to discard all changes and move back one commit.  
**Screenshot:** hard\_reset.png



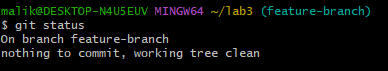
**Step 11:**  
View the commit history after the hard reset.  
**Screenshot:** log\_after\_hard\_reset.png



**Step 12:**  
Open the file again and confirm the latest edit is gone.  
**Screenshot:** file\_after\_hard\_reset.png



**Step 13:**  
Check git status; it should report “nothing to commit, working tree clean.”  
**Screenshot:** status\_after\_hard\_reset.png

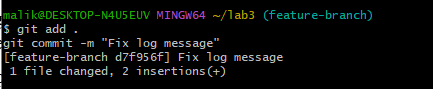


**Task 7 – Amending the Last Commit**

**Step 1:**

Make a small change in any file.

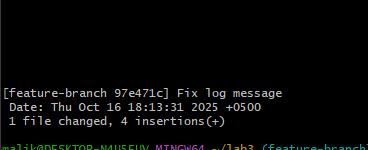
Stage and commit the change.  
**Screenshot:** first\_amend\_commit.png



**Step 3:**

Make another change you forgot to include.

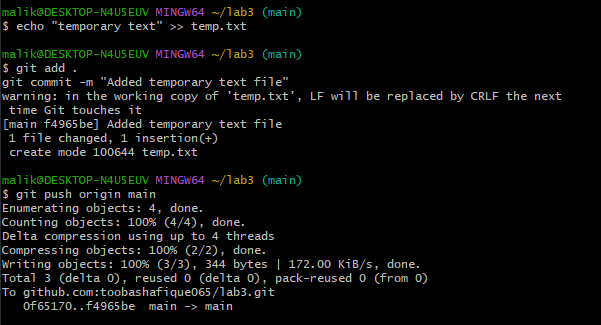
Stage and amend the last commit.  
**Screenshot:** amend\_commit.png



**Task 8 – Reverting a Commit (Safe Undo on Remote Branch)**

**Step 1:**

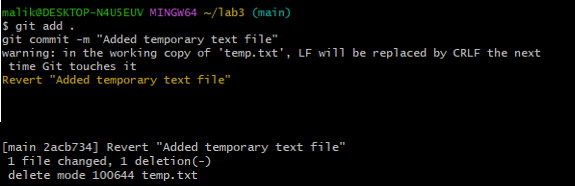
Make a change and commit it.  
**Screenshot:** commit\_temp\_file.png



**Step 2:**

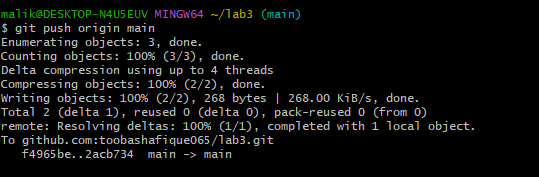
View the commit history.

Revert the specific commit using its hash.  
**Screenshot:** revert\_commit.png



**Step 4:**

Push the revert commit to the remote repository.  
**Screenshot:** revert\_push.png



**Task 9 – Force Push (With Caution)**

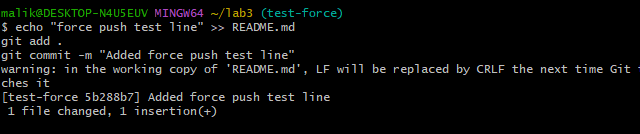
**Step 1:**

Create a new branch.  
**Screenshot:** new\_branch.png



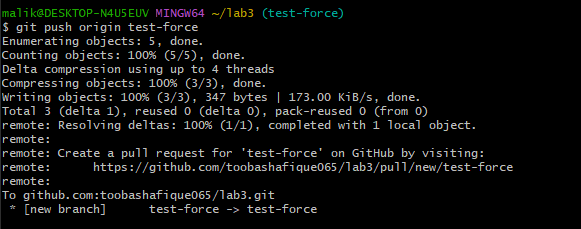
**Step 2:**

Make and commit a small change.  
**Screenshot:** force\_commit.png



**Step 3:**

Push the new branch to the remote repository.  
**Screenshot:** push\_force\_branch.png



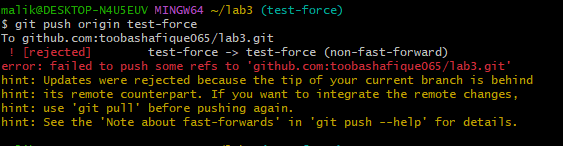
**Step 4:**

Perform a hard reset to remove the last commit.  
**Screenshot:** hard\_reset\_force.png



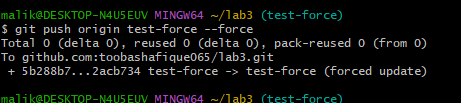
**Step 5:**

Try to push again (it will be rejected).  
**Screenshot:** normal\_push.png



**Step 6:**

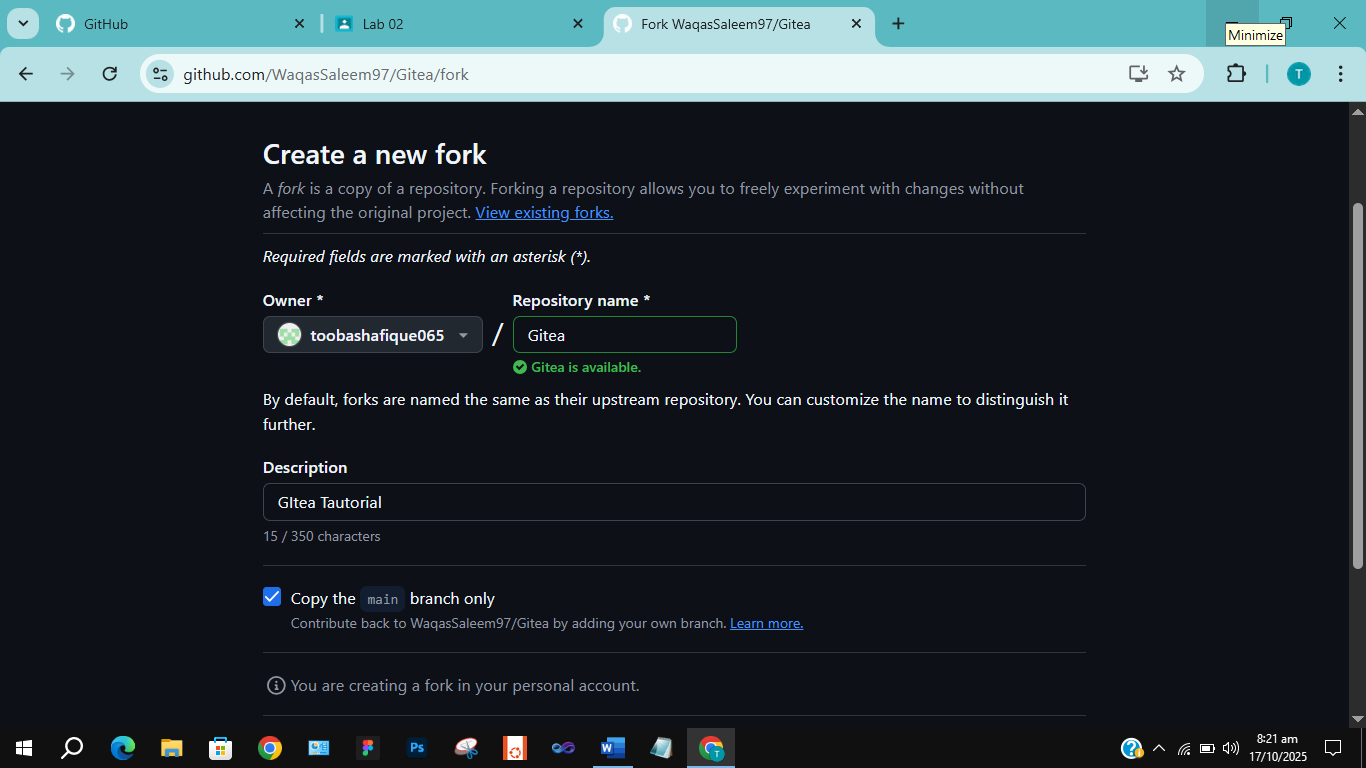
Force push to the remote repository.  
**Screenshot:** force\_push.png



**Task 10 – Running Gitea in GitHub Codespaces via Docker Compose**

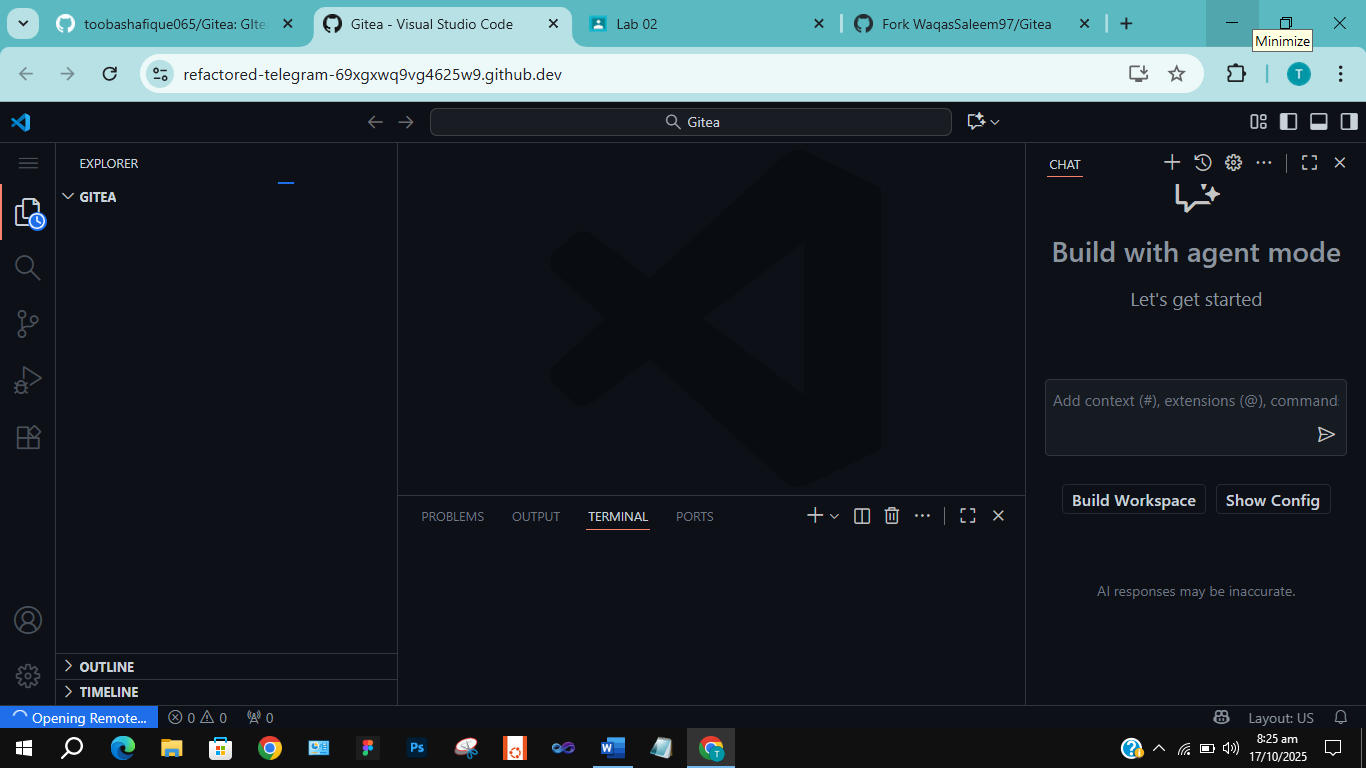
**Step 1:**

Fork the Gitea Repository.  
**Screenshot:** forked\_gitea.png



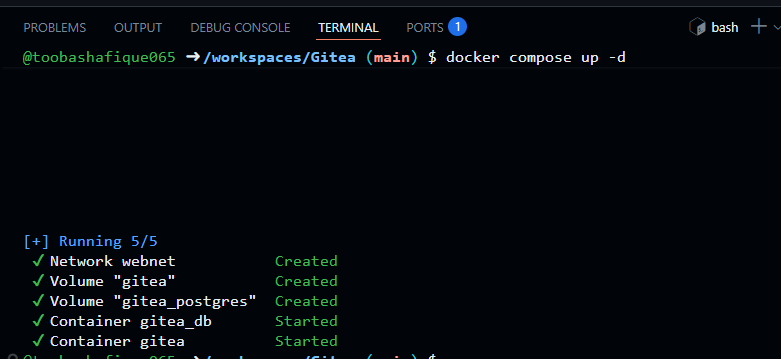
**Step 2:**

Open the Forked Repo in GitHub Codespaces.  
**Screenshot:** codespace\_loading.png



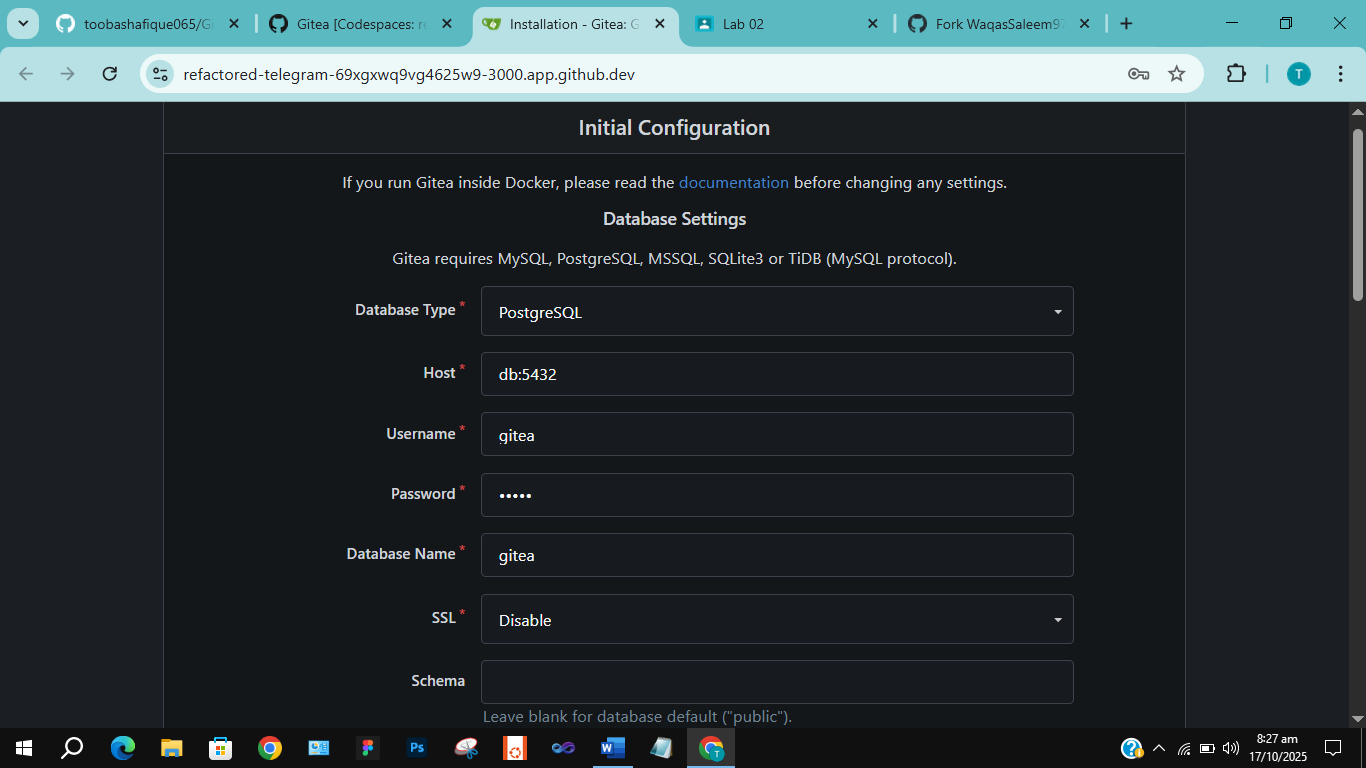
**Step 3:**

Start Gitea with Docker Compose using the command docker compose up -d.  
**Screenshot:** docker\_up.png



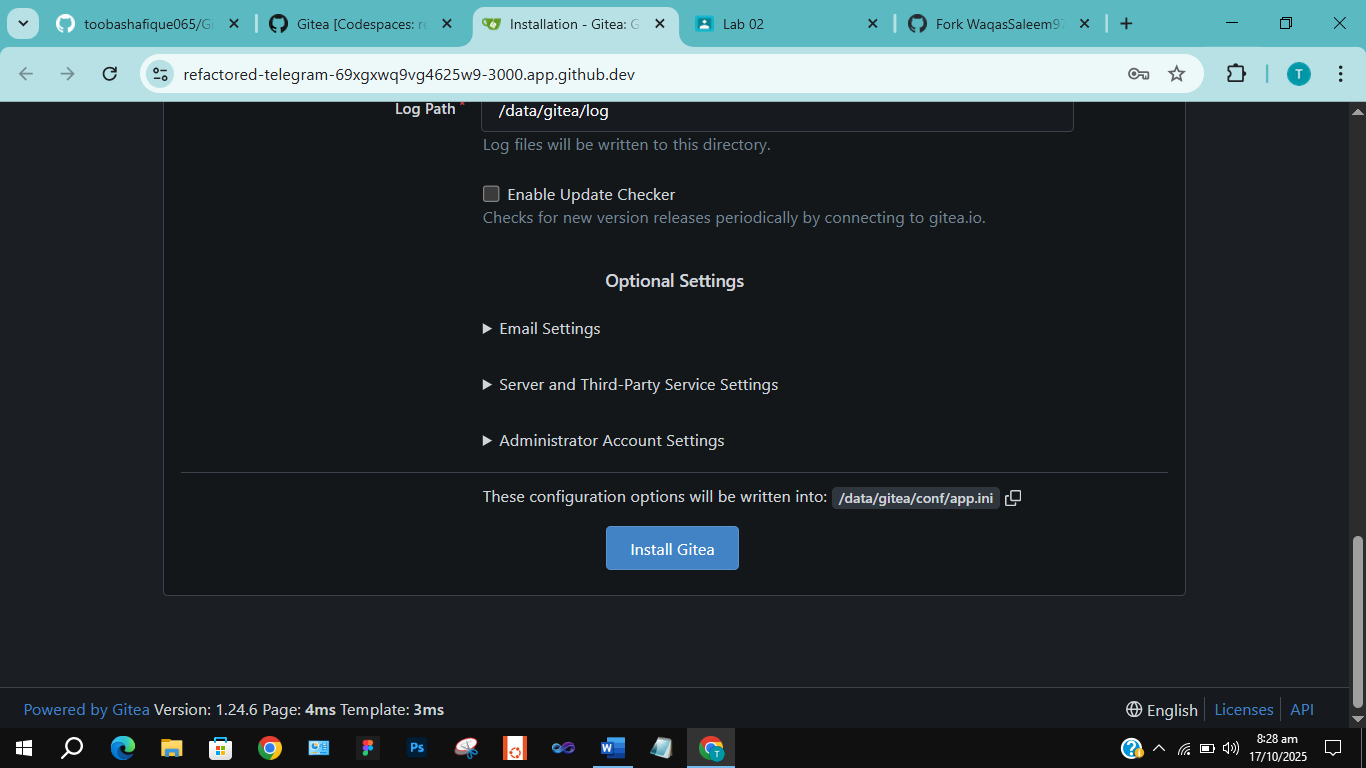
**Step 4:**

Access Gitea Web Interface by forwarding port 3000 in Codespaces and opening it in your browser.  
**Screenshot:** gitea\_install\_page.png



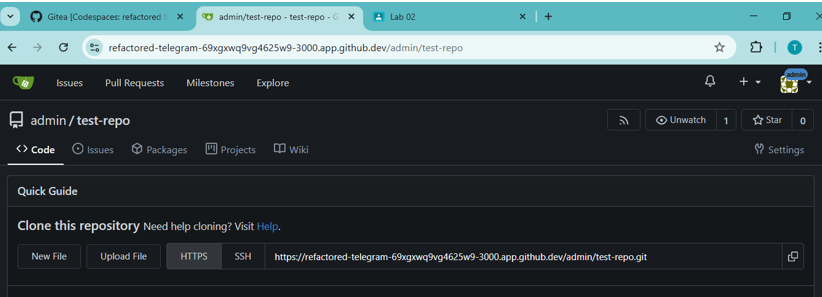
**Step 5:**

Install Gitea by completing the setup form and providing admin credentials.  
**Screenshot:** admin\_setup.png



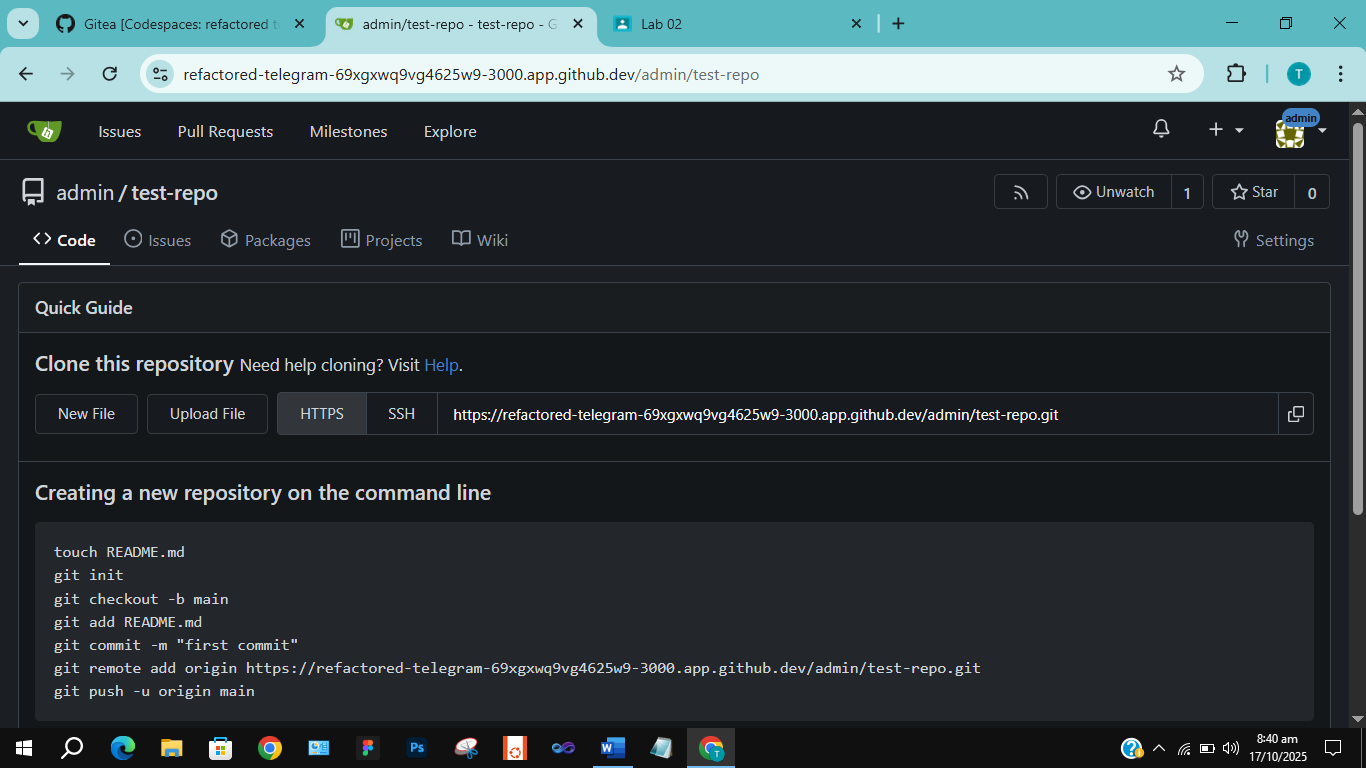
**Step 6:**

Log In to Gitea using your admin account.  
**Screenshot:** gitea\_dashboard.png



**Step 7:**

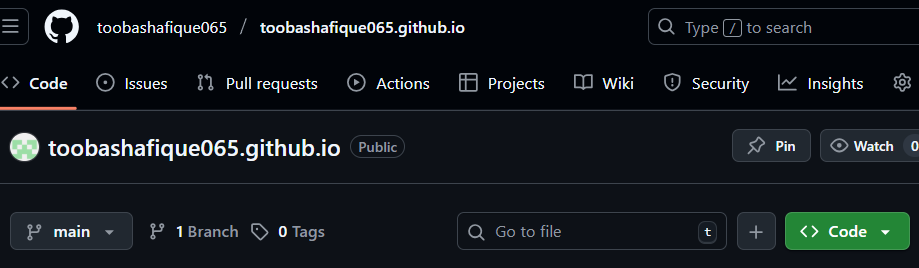
Create a New Repository in Gitea.  
**Screenshot:** gitea\_new\_repo.png



**Task 11 – Creating a GitHub Pages Portfolio Site**

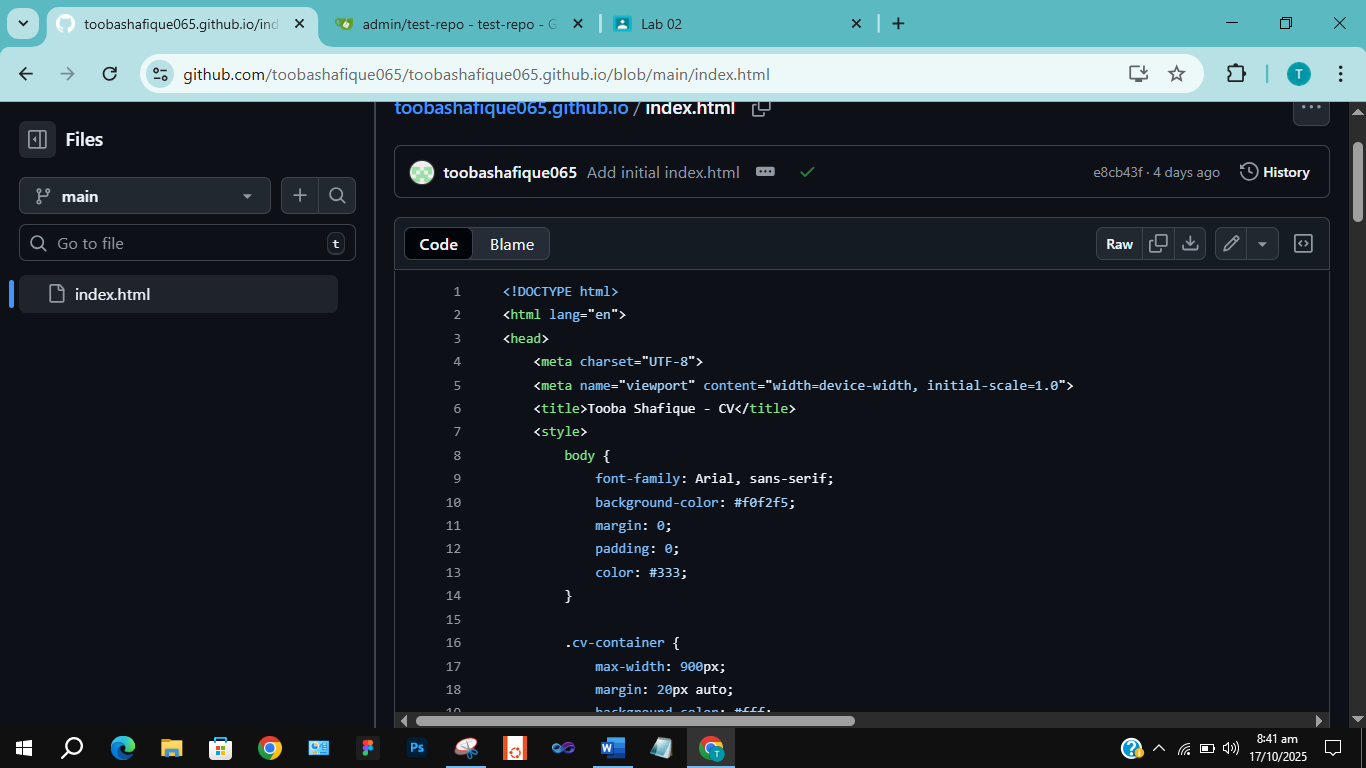
**Step 1:**

Create a GitHub Pages Repository named <your-username>.github.io.  
**Screenshot:** github\_pages\_repo.png



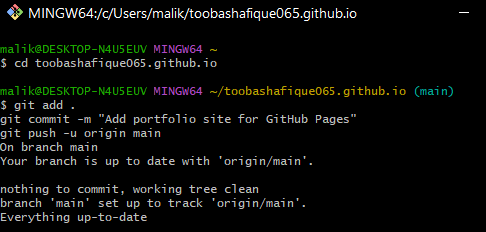
**Step 2:**

Add your static website files (HTML, CSS, JS) to a local folder.  
**Screenshot:** local\_static\_site.png



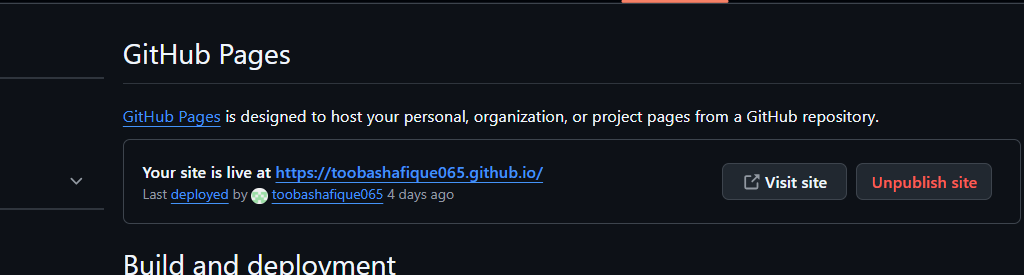
**Step 3:**

Push the Files to GitHub using git commands (git add, git commit, git push).  
**Screenshot:** push\_static\_site.png

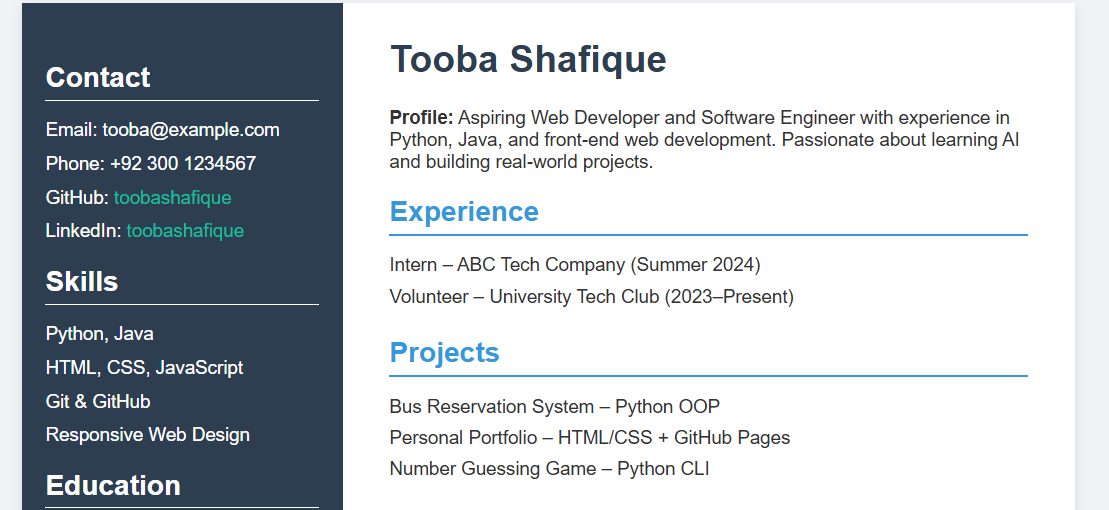


**Step 4:**

Check GitHub Pages Settings in your repository (Settings > Pages) to confirm the site is published.  
**Screenshot:** github\_pages\_settings.png

**Step 5:**

Visit your live GitHub Pages site in the browser.  
**Screenshot:** live\_site.png

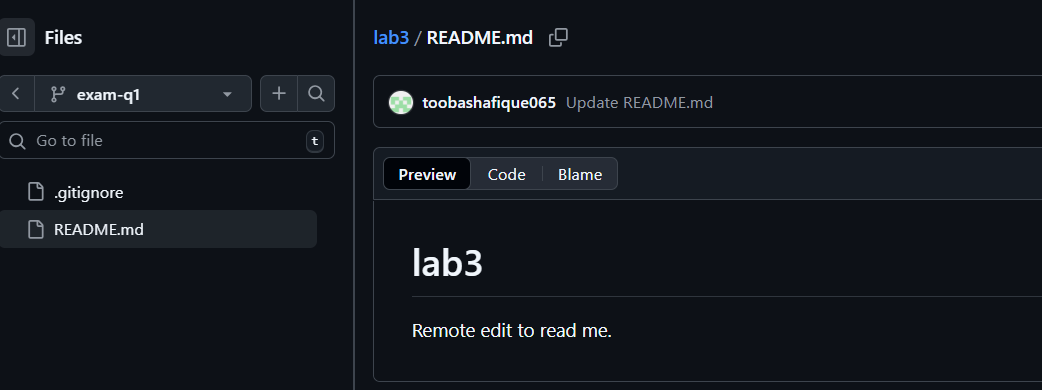


**Exam Evaluation Questions**

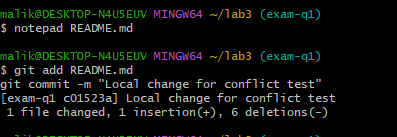
**Question 1 – Local vs Remote Conflict Resolution**

**Step 1:**

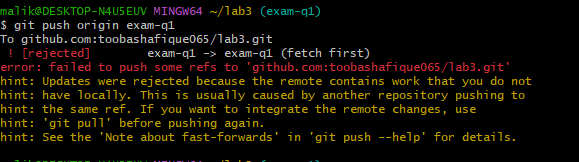
On GitHub, edit a file (e.g., README.md) and commit the change.  
**Screenshot:** Q1\_remote\_edit.png



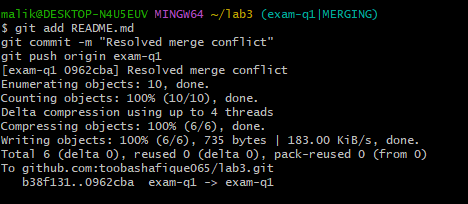
**Step 2:** On your local machine, edit the same file differently (avoid conflict) and commit.  
**Screenshot:** Q1\_local\_edit.png



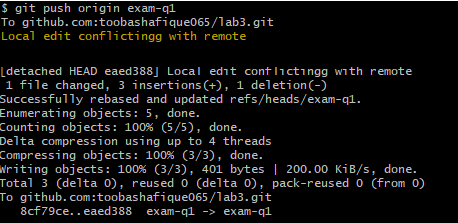
**Step 3:** Try to push your local commit and observe the error.  
**Screenshot:** Q1\_push\_error.png



**Step 4:** Resolve the conflict using git pull (merge) and then push.  
**Screenshot:** Q1\_merge\_resolution.png

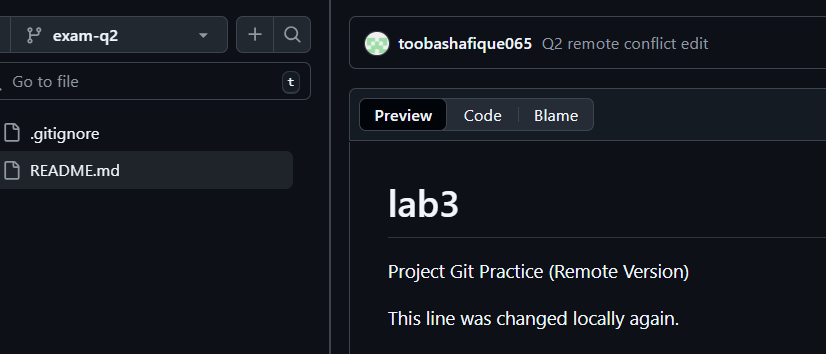


**Step 5:** Repeat with another remote/local change, but resolve using git pull --rebase and then push.  
**Screenshot:** Q1\_rebase\_resolution.png

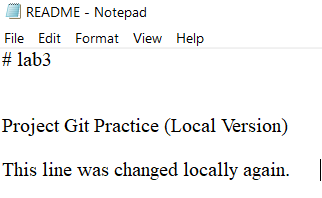


**Question 2 – Manual Merge Conflict Handling**

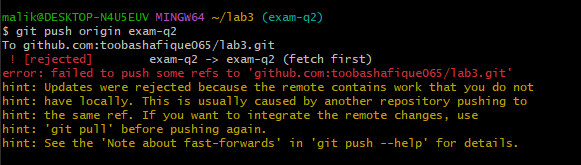
**Step 1:** On GitHub, change a specific line in a file and commit.  
**Screenshot:** Q2\_remote\_conflict\_edit.png



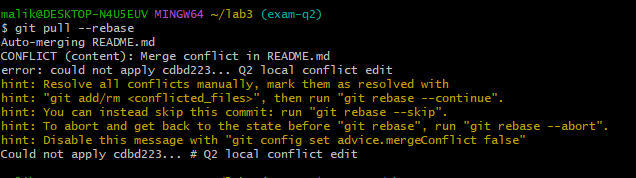
**Step 2:** Locally, change the same line differently and commit.  
**Screenshot:** Q2\_local\_conflict\_edit.png



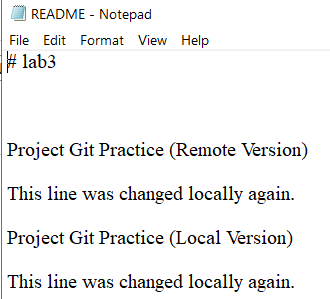
**Step 3:** Try to push your local change and observe the conflict error.  
**Screenshot:** Q2\_conflict\_push\_error.png



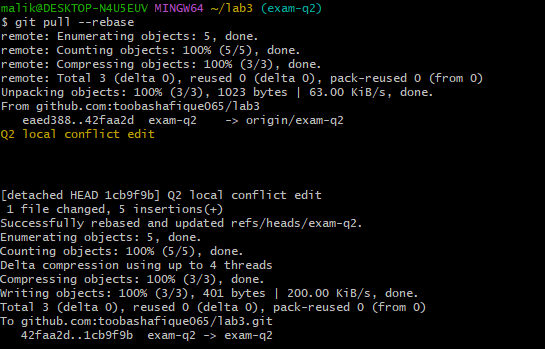
**Step 4:** Use git pull --rebase to fetch changes and trigger the conflict.  
**Screenshot:** Q2\_rebase\_conflict.png



**Step 5:** Edit the conflicted file to resolve the conflict manually.  
**Screenshot:** Q2\_resolved\_file.png

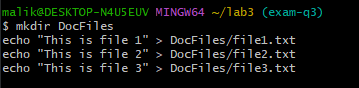


**Step 6:** Mark the conflict as resolved (git add <file> and git rebase --continue), then push.  
**Screenshot:** Q2\_resolution\_complete.png

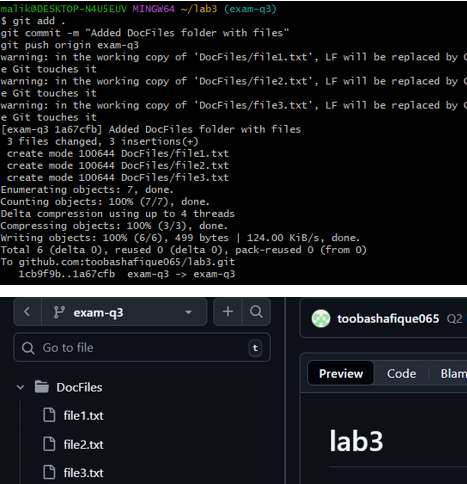


**Question 3 – Managing Ignored and Tracked Files**

**Step 1:** Create a new folder (e.g., DocFiles) and add several files inside.  
**Screenshot:** Q3\_folder\_created.png



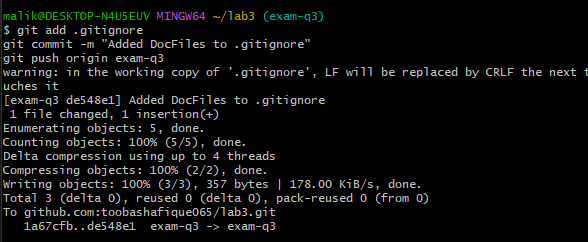
**Step 2:** Commit and push the folder/files to GitHub.  
**Screenshot:** Q3\_files\_pushed.png



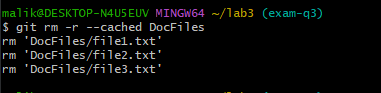
**Step 3:** Add the folder to your .gitignore file.  
**Screenshot:** Q3\_gitignore\_added.png



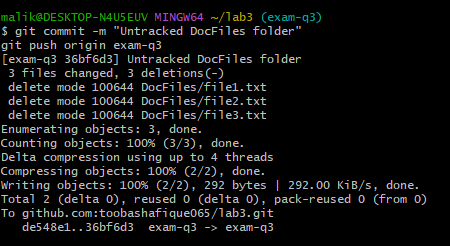
**Step 4:** Commit and push the .gitignore update.  
**Screenshot:** Q3\_gitignore\_pushed.png



**Step 5:** Remove the folder from tracking using git rm -r --cached <folder>.  
**Screenshot:** Q3\_folder\_untracked.png

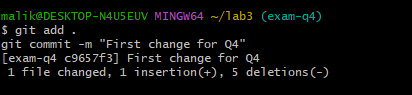


**Step 6:** Commit and push the change, then verify the folder is no longer tracked on GitHub.  
**Screenshot:** Q3\_folder\_removed\_github.png

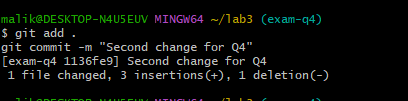


**Question 4 – Commit History Manipulation and Recovery**

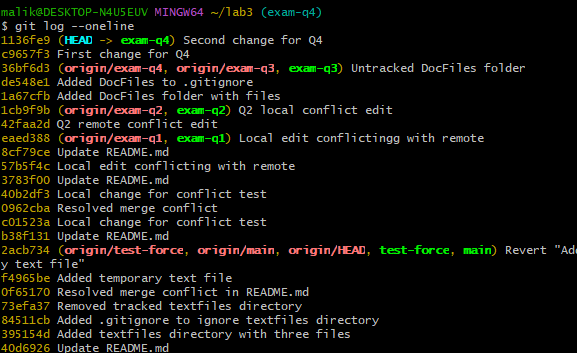
**Step 1:** Make a change and commit it.  
**Screenshot:** Q4\_first\_commit.png



**Step 2:** Make another change and commit again.  
**Screenshot:** Q4\_second\_commit.png

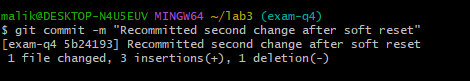


**Step 3:** View your commit history using git log --oneline.  
**Screenshot:** Q4\_commit\_history.png

 **Step 4:** Perform a soft reset (git reset --soft HEAD~1) and observe your file and history.  
**Screenshot:** Q4\_soft\_reset.png



**Step 5:** Make a new commit again.  
**Screenshot:** Q4\_third\_commit.png



**Step 6:** Perform a hard reset (git reset --hard HEAD~1) and observe the changes.  
**Screenshot:** Q4\_hard\_reset.png

