Cloud Computing Lab 3

Submitted To:

Sir Waqas and Sir Shoaib

Submitted By:

Hamail Fatima

Section:

5(A)

Roll Number:

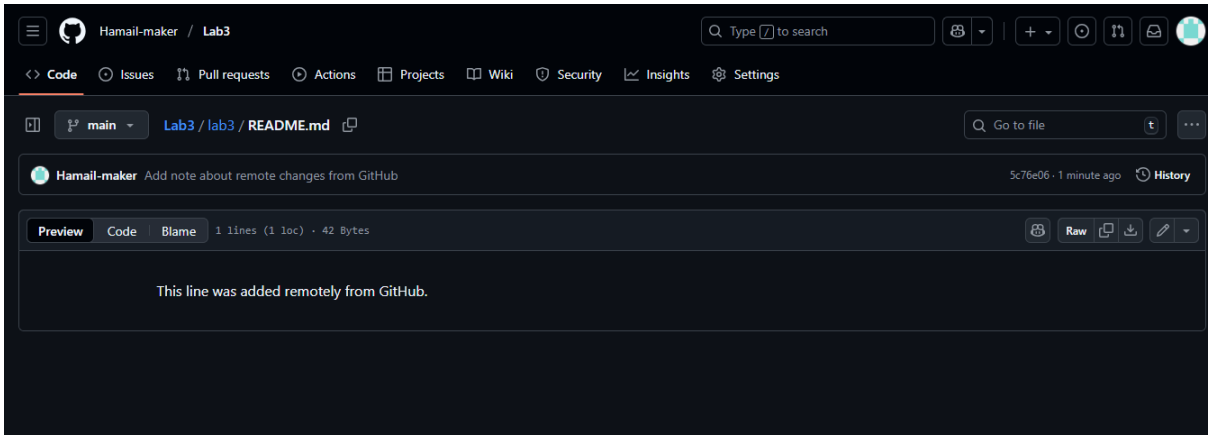
2023-BSE-023

Task 1 – Handling Local and Remote Commit Conflicts (Pull vs Pull --rebase)

This task demonstrates what happens when your local repository and remote repository both have new commits — and how git pull and git pull --rebase behave differently.

Steps

1. Open your repository on **GitHub** and edit the README.md file directly in the browser.
    - Add a new line such as:
    - This line was added remotely from GitHub.
    - Commit your change on GitHub (e.g., message: *Remote update to README*).
    - Save a screenshot of your browser showing the change as remote_edit.png.

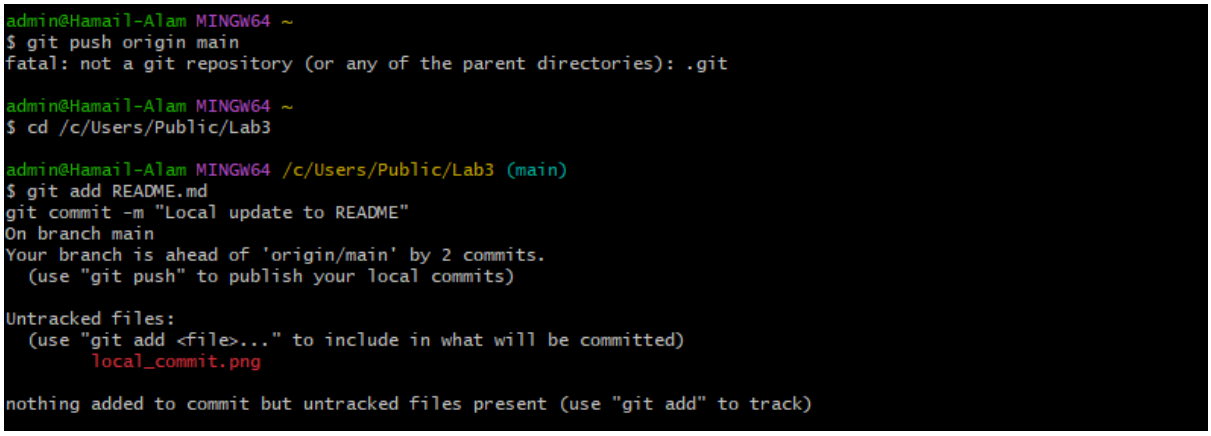


1. On your **local machine**, open the same repository folder.

- Edit the same README.md file locally, for example:
- This line was added locally.
- Stage and commit your change:
- git add README.md

git commit -m "Local update to README"

- Save a screenshot of your terminal as local_commit.png



1. Try to push:

git push origin main

You'll see an error message similar to:

! [rejected] main -> main (fetch first)

error: failed to push some refs to 'github.com:username/repo.git'

hint: Updates were rejected because the remote contains work that you do not have locally.

- Save a screenshot of this error as push_error.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ cd /c/Users/Public/Lab3      # or your Lab3 path
echo "This line was added locally." >> README.md
git add README.md
git commit -m "Local update to README"
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it
[main d7a1892] Local update to README
 1 file changed, 1 insertion(+)
```

To fix this, pull the latest changes from remote:

git pull --no-rebase origin main

Git will merge your local and remote commits, creating a merge commit.

Save a screenshot as merge_commit.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git pull --no-rebase origin main
From https://github.com/Hamail-maker/Lab3
 * branch            main       -> FETCH_HEAD
Already up to date.
```

1. Push again:

   git push -u origin main

   - Save a screenshot as push_after_merge.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git push -u origin main
branch 'main' set up to track 'origin/main'.
Everything up-to-date
```

Now repeat the same process, but this time use **rebase instead of merge**:

- Make another remote edit on GitHub in README.md.
- Then make another local edit and commit it.
- Instead of using git pull, run:

git pull --rebase origin main

- Git will **replay your local commits on top** of the pulled commits, keeping history linear.
- Save a screenshot as rebase_pull.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git pull --rebase origin main
From https://github.com/Hamail-maker/Lab3
 * branch            main       -> FETCH_HEAD
Already up to date.
```

Push again:

git push -u origin main

- Save a screenshot as push_after_rebase.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git push -u origin main
branch 'main' set up to track 'origin/main'.
Everything up-to-date
```

Task 2 – Creating and Resolving Merge Conflicts Manually

This task will help you understand how Git handles file conflicts when two people modify the same line of code differently.

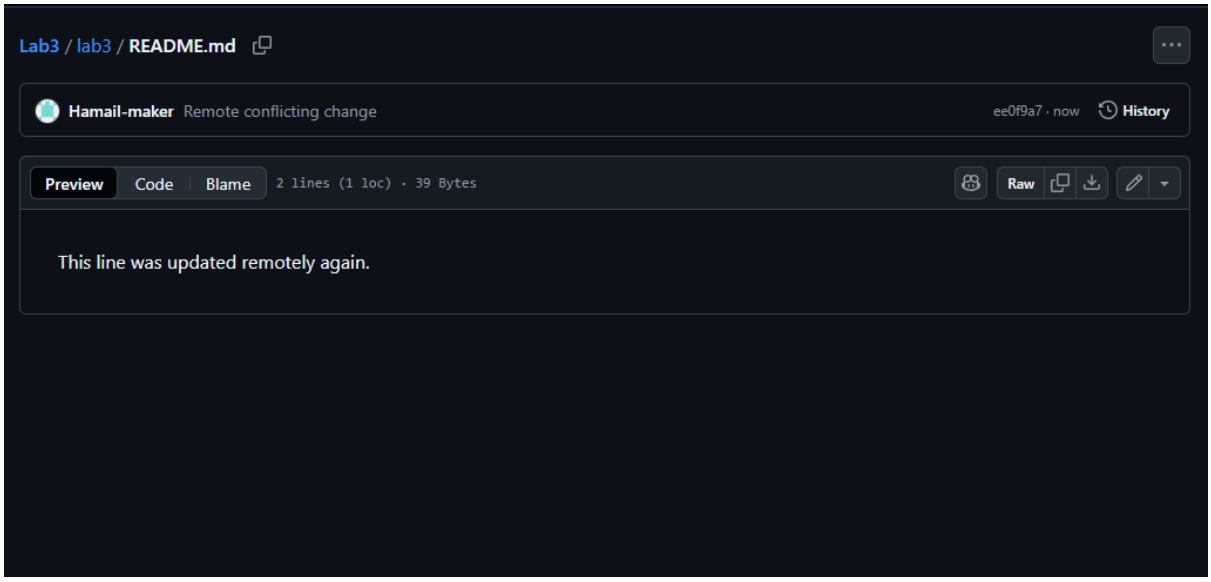
Steps

1. On **GitHub (remote)**, open your README.md file and change an existing line.
   For example, if it currently says:
2. This line was added remotely from GitHub.

Change it to:

This line was updated remotely again.

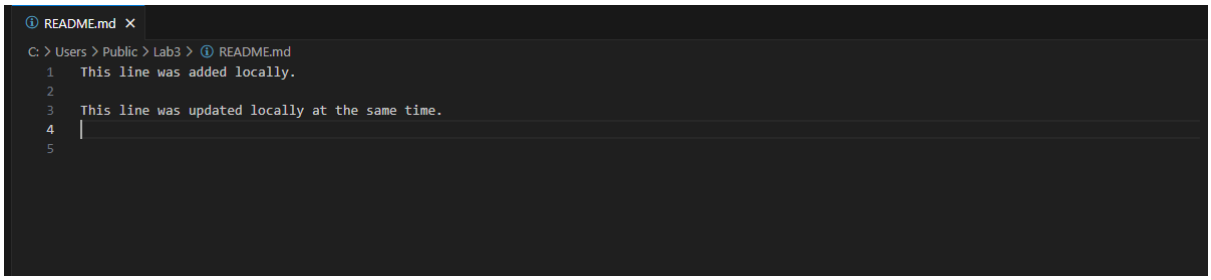
- Commit your change with the message: *Remote conflicting change*
- Save a screenshot as remote_conflict_edit.png.



On your **local machine**, edit the **same line** in the same file but make a **different change**:

This line was updated locally at the same time.

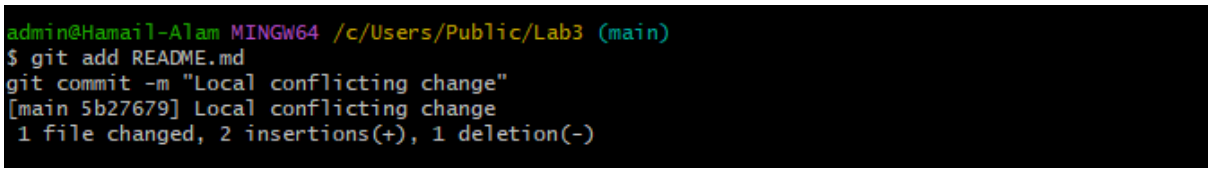
- Save a screenshot of your edit in VS Code as local_conflict_edit.png



1. Stage and commit your local change:
2. git add README.md

git commit -m "Local conflicting change"

- Save a screenshot as local_conflict_commit.png.



Open the README.md file in your editor — you'll see conflict markers like this:

<<<<<<< HEAD

This line was updated locally at the same time.

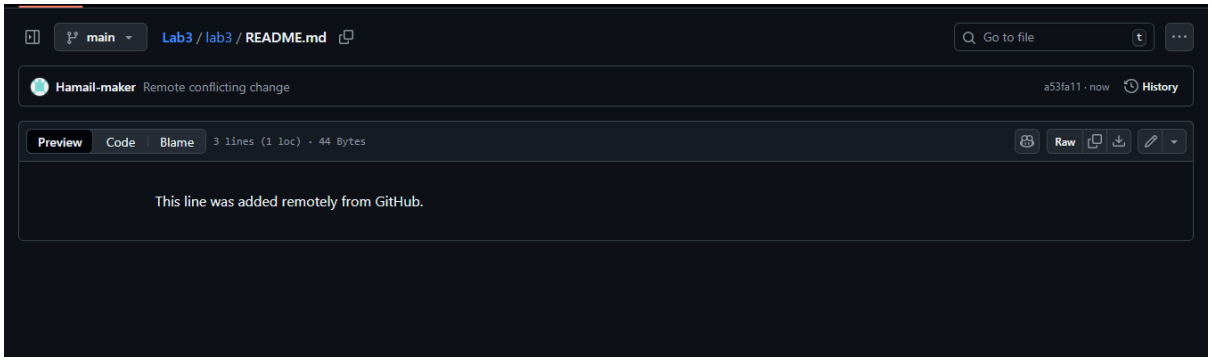
=======

This line was updated remotely again.

>>>>>>> origin/main

Manually edit the file to keep the correct line (for example, merge both ideas).

Save a screenshot of the resolved file as resolved_readme.png.



1. After fixing the file, mark the conflict as resolved and continue the rebase:
2. git add README.md

git rebase --continue

- Save a screenshot of this step as rebase_continue.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$  git add README.md
git rebase --continue
fatal: no rebase in progress
```

1. Finally, push your changes:

   git push -u origin main

- Save a screenshot as push_after_resolve.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git pull --rebase origin main
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (4/4), 1016 bytes | 16.00 KiB/s, done.
From https://github.com/Hamail-maker/Lab3
 * branch            main       -> FETCH_HEAD
   56ae40d..a53fa11  main       -> origin/main
Updating 56ae40d..a53fa11
Fast-forward
 lab3/README.md | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git rebase --continue
git push --force-with-lease origin main
fatal: no rebase in progress
Everything up-to-date
```

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

In this task, you will learn how to use a .gitignore file to exclude files or directories from version control and how to remove them from Git's tracking while keeping them locally.

Steps

1. Create a new folder named textfiles inside your repository:

mkdir textfiles

1. Inside the textfiles folder, create three text files:
2. echo "File A content" > textfiles/a.txt
3. echo "File B content" > textfiles/b.txt

echo "File C content" > textfiles/c.txt

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ echo "File A content" > textfiles/a.txt
echo "File B content" > textfiles/b.txt
echo "File C content" > textfiles/c.txt
```

1. Add and commit the new directory:
2. git add .
3. git commit -m "Added textfiles directory with three files"

git push origin main

- Save a screenshot as push_textfiles.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git add .
git commit -m "Added textfiles directory with three files"
git push origin main
warning: in the working copy of 'textfiles/a.txt', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'textfiles/b.txt', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'textfiles/c.txt', LF will be replaced by CRLF the next time Git touches it
[main 84bdf68] Added textfiles directory with three files
 16 files changed, 3 insertions(+)
 create mode 100644 local_commit.png
 create mode 100644 local_conflict_commit.png
 create mode 100644 local_conflict_edit.png
 create mode 100644 merge_commit.png
 create mode 100644 push_after_merge.png
 create mode 100644 push_after_rebase.png
 create mode 100644 push_after_resolve.png
 create mode 100644 push_error.png
 create mode 100644 rebase_continue.png
 create mode 100644 rebase_pull.png
 create mode 100644 remote_conflict_edit.png
 create mode 100644 resolved_readme.png
 create mode 100644 textfile_create.png
 create mode 100644 textfiles/a.txt
 create mode 100644 textfiles/b.txt
 create mode 100644 textfiles/c.txt
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 8 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (19/19), 279.16 KiB | 11.63 MiB/s, done.
Total 19 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Hamail-maker/Lab3.git
   a53fa11..84bdf68  main -> main
```

1. Now, create a .gitignore file in the root of your repository:

echo "textfiles/*" > .gitignore

2. Add and commit the .gitignore file:
3. git add .gitignore
4. git commit -m "Added .gitignore to ignore textfiles directory"

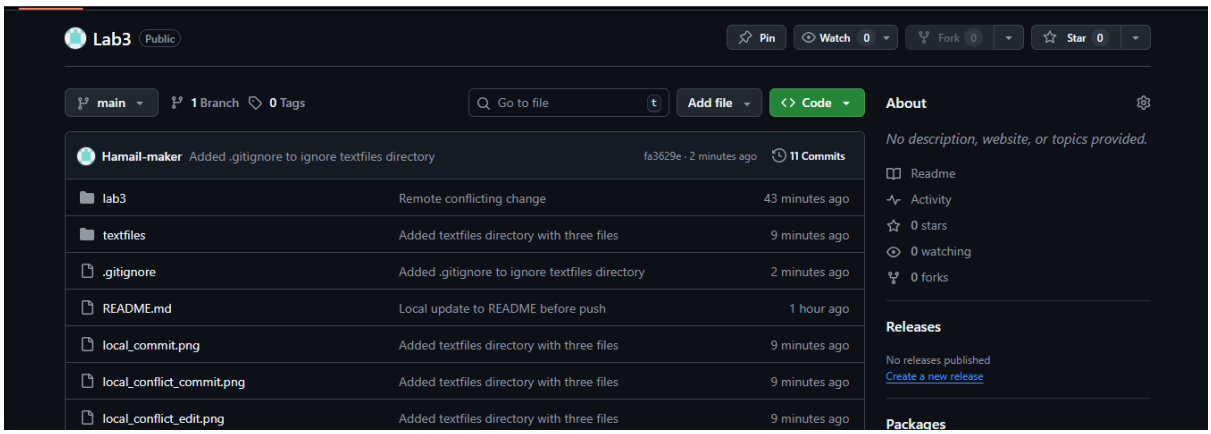
git push origin main

- Save a screenshot as gitignore_push.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ echo "textfiles/*" > .gitignore

admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git add .gitignore
git commit -m "Added .gitignore to ignore textfiles directory"
git push origin main
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the next time Git touches it
[main fa3629e] Added .gitignore to ignore textfiles directory
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 310 bytes | 155.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Hamail-maker/Lab3.git
   84bdf68..fa3629e  main -> main
```

1. Go to **GitHub** and check your repository — notice that the textfiles directory is **still visible** on the remote.
    - Save a screenshot as repo_still_has_textfiles.png.



1. Check your GitHub repository again — the textfiles folder should now be **deleted remotely**.
    - Save a screenshot as repo_textfiles_removed.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git rm -r --cached textfiles
rm 'textfiles/a.txt'
rm 'textfiles/b.txt'
rm 'textfiles/c.txt'
```

ask 4 – Create Temporary Changes and Use git stash

This task demonstrates how git stash allows you to temporarily save uncommitted changes so that you can safely switch branches without losing work.

Steps

1. Create a feature-branch.

git checkout -b feature-branch

Modify any file (for example, README.md) by adding a few test lines.

git add README.md

git commit -m "Changes the README.md file"

- Save a screenshot as modified_readme.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git checkout -b feature-branch
Switched to a new branch 'feature-branch'

admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (feature-branch)
$ git add README.md
git commit -m "Changes the README.md file"
[feature-branch 7cf4d7b] Changes the README.md file
 3 files changed, 3 deletions(-)
 delete mode 100644 textfiles/a.txt
 delete mode 100644 textfiles/b.txt
 delete mode 100644 textfiles/c.txt
```

1. Without committing or stashing the changes, try to switch to another branch:

git checkout main

You'll see an error message similar to:

error: Your local changes to the following files would be overwritten by checkout:

README.md

Please commit your changes or stash them before you switch branches.

- Save a screenshot as checkout_error.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git checkout main
M       README.md
Already on 'main'
Your branch is up to date with 'origin/main'.
```

1. To fix this, **stash your changes**:

git stash

- Save a screenshot as stash_command.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git stash
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it
Saved working directory and index state WIP on main: fa3629e Added .gitignore to ignore textfiles directory
```

Try switching branches again — it should now work:

git checkout main

- Save a screenshot as branch_switched.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git checkout main
Already on 'main'
Your branch is up to date with 'origin/main'.
```

1. Return to your previous branch (for example, feature-branch):

git checkout feature-branch

   - Save a screenshot as back_to_feature.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git checkout feature-branch
Switched to branch 'feature-branch'
```

1. Check your working directory status:

   git status

   It should show:

   nothing to commit, working tree clean

- Save a screenshot as status_clean.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (feature-branch)
$ git status
On branch feature-branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Checkout_error.png
        back_to_feature.png
        branch_switched.png
        gitignore_push.png
        modified_readme.png
        push_textfiles.png
        repo_still_have_textfiles.png
        repo_textfiles_removed.png
        stash_sommand.png

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

1. Now restore your stashed changes:

   git stash pop

   You'll see a message indicating that changes were reapplied:

   On branch feature-branch

   Changes not staged for commit:

   (use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

- Save a screenshot as stash_pop.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (feature-branch)
$ git stash pop
On branch feature-branch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Checkout_error.png
        back_to_feature.png
        branch_switched.png
        gitignore_push.png
        modified_readme.png
        push_textfiles.png
        repo_still_have_textfiles.png
        repo_textfiles_removed.png
        stash_sommand.png
        status_clean.png

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (992578731854d21ed90048022c6bc009913cc965)
```

1. Confirm that your previous edits are back in the file.

Task 5 – Checkout a Specific Commit Using git log

1. View commit history:

   git log --oneline

   - Save a screenshot as log_before_checkout.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (feature-branch)
$ git log --oneline
7cf4d7b (HEAD -> feature-branch) Changes the README.md file
fa3629e (origin/main, origin/HEAD, main) Added .gitignore to ignore textfiles directory
84bdf68 Added textfiles directory with three files
a53fa11 Remote conflicting change
56ae40d Local update to README before push
5b27679 Local conflicting change
ee0f9a7 Remote conflicting change
d7a1892 Local update to README
d6c0ae2 Local update to README
26518ad Local update to README
5c76e06 Add note about remote changes from GitHub
19d7403 Create README.md
```

   Copy any previous commit hash (e.g., a1b2c3d).

   Checkout that commit:

   git checkout <commit-hash>

   Save a screenshot as detached_head.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (feature-branch)
$ git checkout 19d7403
error: Your local changes to the following files would be overwritten by checkout:
        README.md
Please commit your changes or stash them before you switch branches.
Aborting
```

1. To return to your main branch:

   git checkout main

   - Save a screenshot as back_to_main.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (feature-branch)
$ git checkout main
M       README.md
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

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

This task demonstrates the difference between a soft and hard reset in Git. In this version, you will **verify the presence of changes in files and confirm the existence of commits after each reset**.

Steps

1. **Add a new line in any file and commit it:**
2. # Edit any file (e.g., README.md), add a line
3. git add .

   git commit -m "Added test line"

   - Save a screenshot as first_commit.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git add README.md

admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git commit -m "Added test line"
[main 1fad9aa] Added test line
 1 file changed, 4 insertions(+)
```

1. **Add another change and commit again:**
2. # Edit the file again, add a different line
3. git add .

   git commit -m "Second test commit"

- Save a screenshot as second_commit.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git add README.md
git commit -m "Second test commit"
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Checkout_error.png
        back_to_feature.png
        back_to_main.png
        branch_switched.png
        detached_head.png
        first_comit.png
        gitignore_push.png
        log_before_checkout.png
        modified_readme.png
        push_textfiles.png
        repo_still_have_textfiles.png
        repo_textfiles_removed.png
        stash_pop.png
        stash_sommand.png
        status_clean.png

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

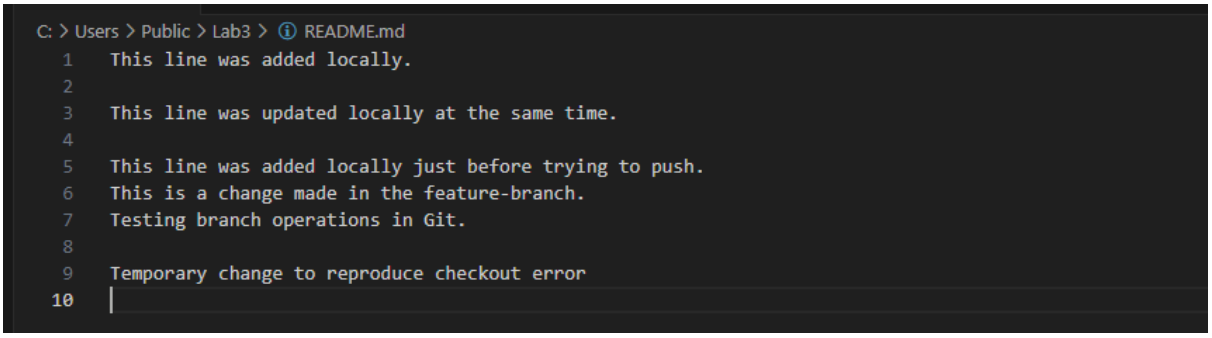
View the history before reset:

git log --oneline

- Save a screenshot as log_before_reset.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git log --oneline
1fad9aa (HEAD -> main) Added test line
fa3629e (origin/main, origin/HEAD) Added .gitignore to ignore textfiles directory
84bdf68 Added textfiles directory with three files
a53fa11 Remote conflicting change
56ae40d Local update to README before push
5b27679 Local conflicting change
ee0f9a7 Remote conflicting change
d7a1892 Local update to README
d6c0ae2 Local update to README
26518ad Local update to README
5c76e06 Add note about remote changes from GitHub
19d7403 Create README.md
```

1. **Check file contents for both changes:**
    - Open the edited file (e.g., README.md) and visually confirm BOTH added lines exist.
    - Save a screenshot as file_before_reset.png.



1. **Perform a soft reset (keeps changes in working directory):**

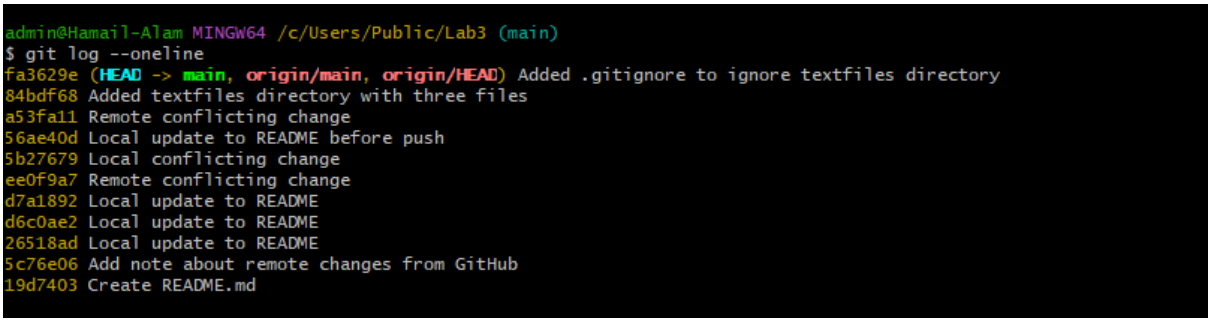
   git reset --soft HEAD~1

   - Save a screenshot as soft_reset.png.

   Check commit history after soft reset:

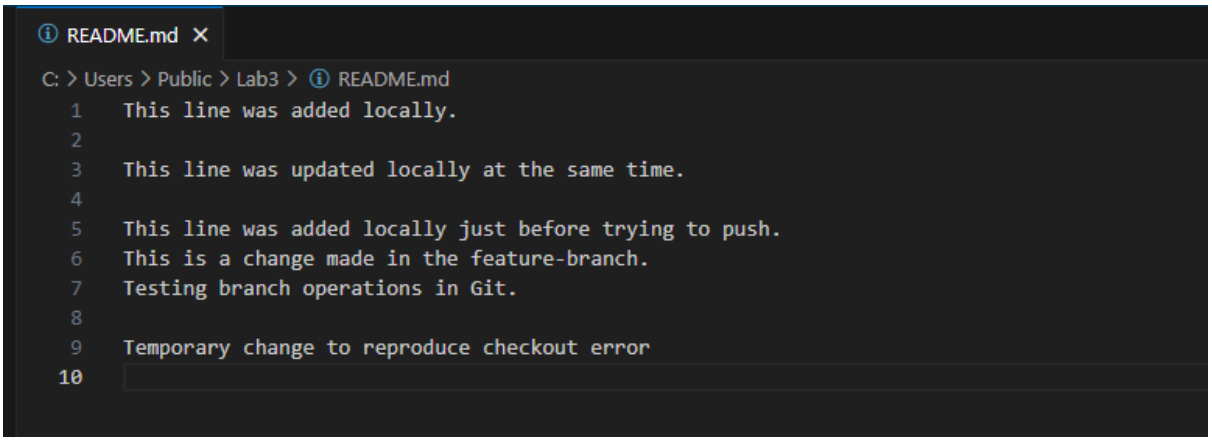
   git log --oneline

- Save a screenshot as log_after_soft_reset.png



1. **Verify changes in the file:**

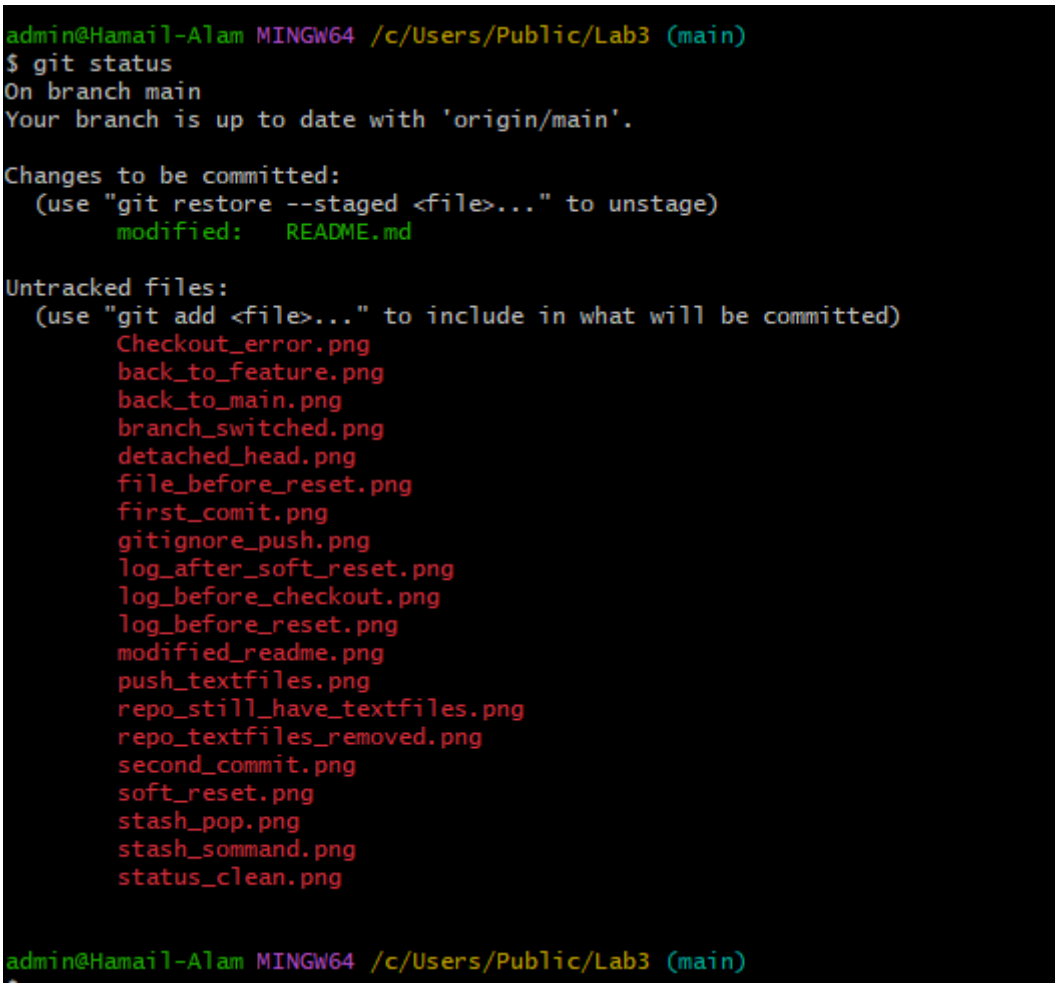
   - Open the file again and confirm both edits are still present (nothing lost).
   - Save a screenshot as file_after_soft_reset.png.



1. **Check git status:**

   git status

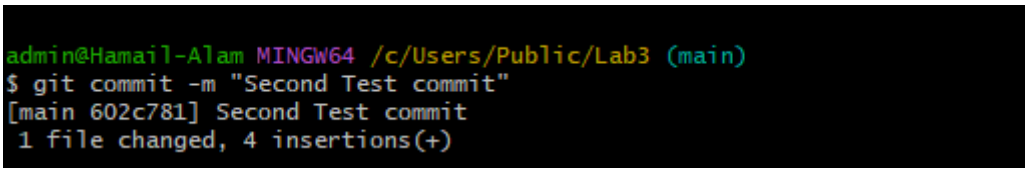
   - You should see your changes are staged (ready to commit again).



1. **Perform commit**

   git commit -m "Second Test commit"

   - Save a screenshot as file_after_hard_reset.png.



1. **Perform a hard reset (discards changes completely):**

   git reset --hard HEAD~1

- Save a screenshot as hard_reset.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git reset --hard HEAD~1
HEAD is now at fa3629e Added .gitignore to ignore textfiles directory
```

11. **Check commit history after hard reset:**

    git log --oneline

    - Save a screenshot as log_after_hard_reset.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git log --oneline
fa3629e (HEAD -> main, origin/main, origin/HEAD) Added .gitignore to ignore textfiles directory
84bdf68 Added textfiles directory with three files
a53fa11 Remote conflicting change
56ae40d Local update to README before push
5b27679 Local conflicting change
ee0f9a7 Remote conflicting change
d7a1892 Local update to README
d6c0ae2 Local update to README
26518ad Local update to README
5c76e06 Add note about remote changes from GitHub
19d7403 Create README.md
```

11. **Check git status:**

    git status

    - Should report "nothing to commit, working tree clean".

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Checkout_error.png
        back_to_feature.png
        back_to_main.png
        branch_switched.png
        detached_head.png
        file_after_hard_reset.png
        file_before_reset.png
        first_comit.png
        gitignore_push.png
        hard_reset.png
        log_after_hard_reset.png
        log_after_soft_reset.png
        log_before_checkout.png
        log_before_reset.png
        modified_readme.png
        push_textfiles.png
        repo_still_have_textfiles.png
        repo_textfiles_removed.png
        second_commit.png
        soft_reset.png
        stash_pop.png
        stash_sommand.png
        status_clean.png

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

Task 7 – Amending the Last Commit

1. Make a small change in any file.

   Stage it and commit:

   git add .

   git commit -m "Fix log message"

   - Save a screenshot as first_amend_commit.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git add README.md
git commit -m "Fix log message"
[main 89c84a0] Fix log message
 1 file changed, 1 insertion(+), 1 deletion(-)
```

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

1. Make a change and commit it:

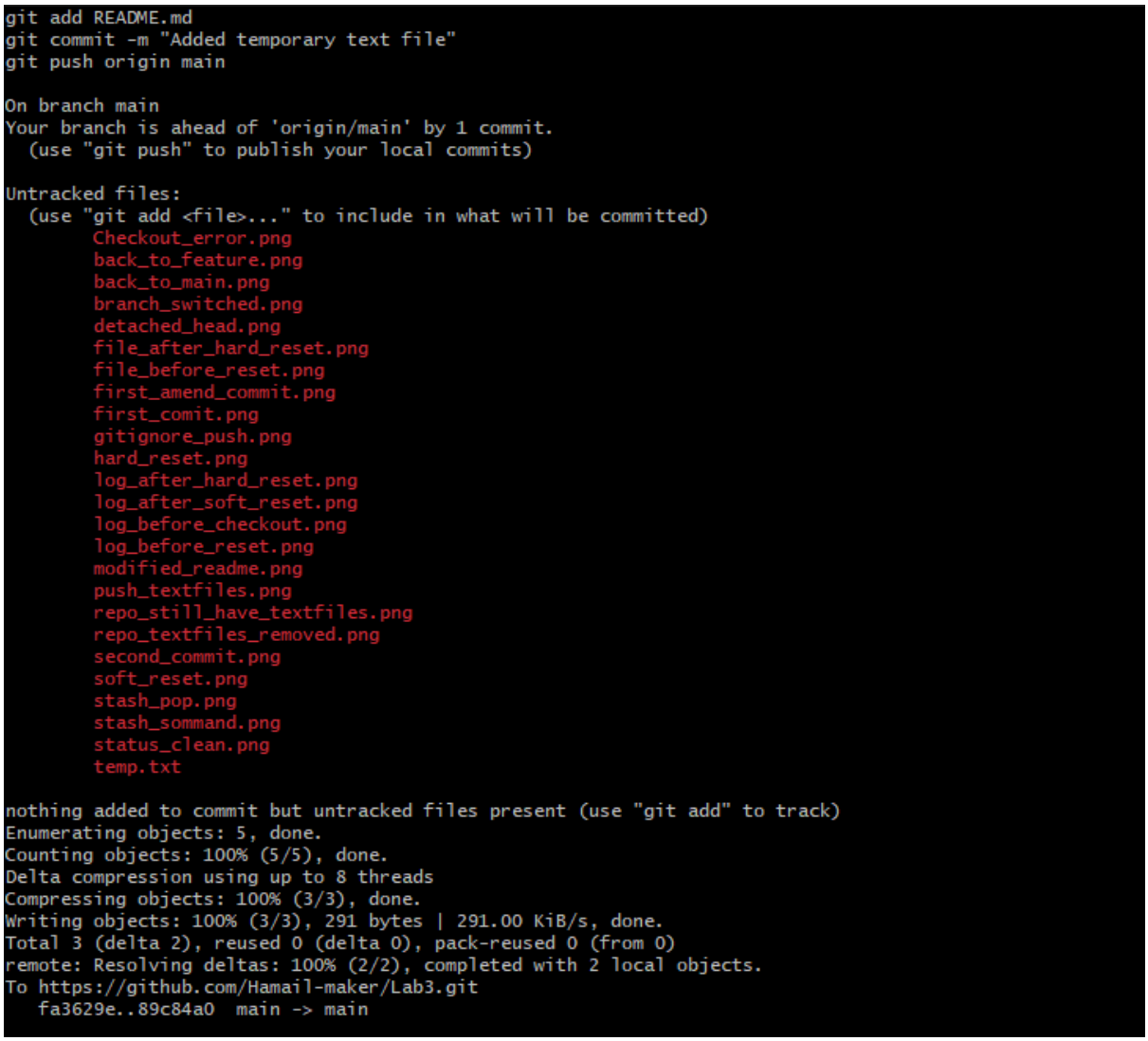
   echo "temporary text" >> temp.txt

   git add .

   git commit -m "Added temporary text file"

   git push origin main

   Save a screenshot as commit_temp_file.png.

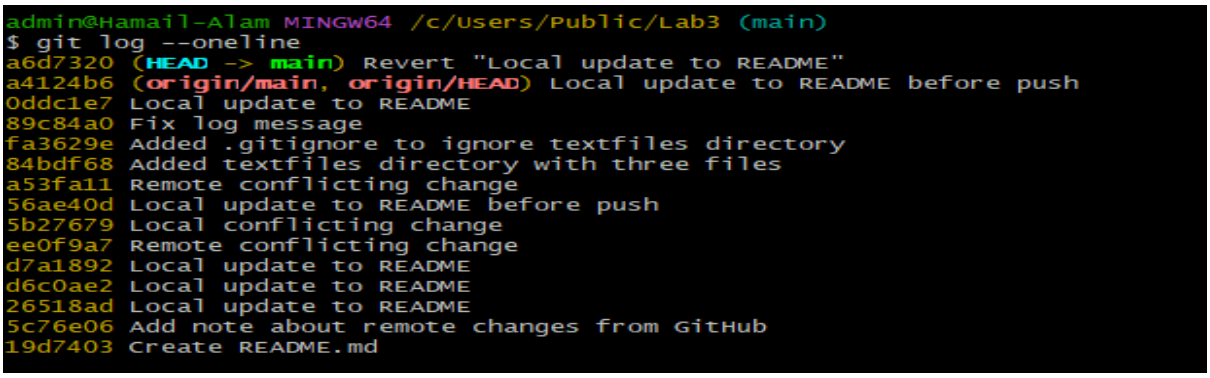


1. Now revert that commit safely (do not delete it):

2. git log --oneline

   git revert <commit-hash>

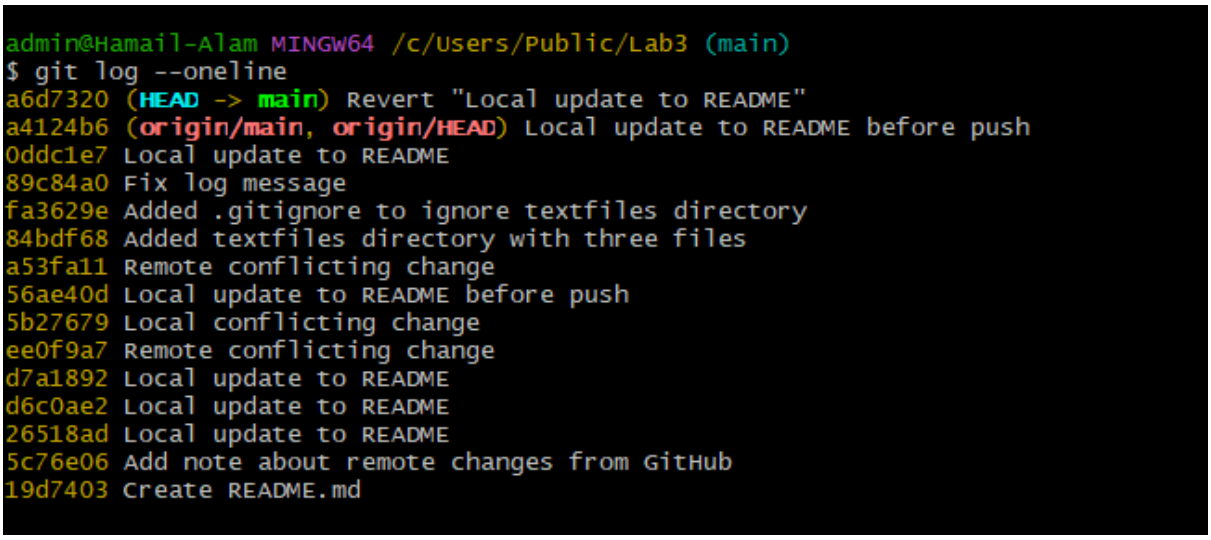
   - Save a screenshot as revert_commit.png.



1. Push the revert commit:

   git push origin main

- Save a screenshot as revert_push.png.



Task 9 – Force Push (With Caution)

⚠️ Use this only in your own branch (never on main or develop).

1. Create a new branch:

   git checkout -b test-force

- Save a screenshot as new_branch.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git checkout -b test-force
Switched to a new branch 'test-force'
```

1. Make and commit a small change.
    - Save a screenshot as force_commit.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (main)
$ git checkout -b test-force
Switched to a new branch 'test-force'

admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (test-force)
$ git branch
  feature-branch
  main
* test-force

admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (test-force)
$ git add README.md
git commit -m "Added test line for force push task"
[test-force 48ecce4] Added test line for force push task
 1 file changed, 1 insertion(+)
```

1. Push it:

git push origin test-force

- Save a screenshot as push_force_branch.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (test-force)
$ git push origin test-force
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 778 bytes | 778.00 KiB/s, done.
Total 6 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
remote:
remote: Create a pull request for 'test-force' on GitHub by visiting:
remote:      https://github.com/Hamail-maker/Lab3/pull/new/test-force
remote:
To https://github.com/Hamail-maker/Lab3.git
 * [new branch]      test-force -> test-force
```

1. Perform a hard reset:

git reset --hard HEAD~1

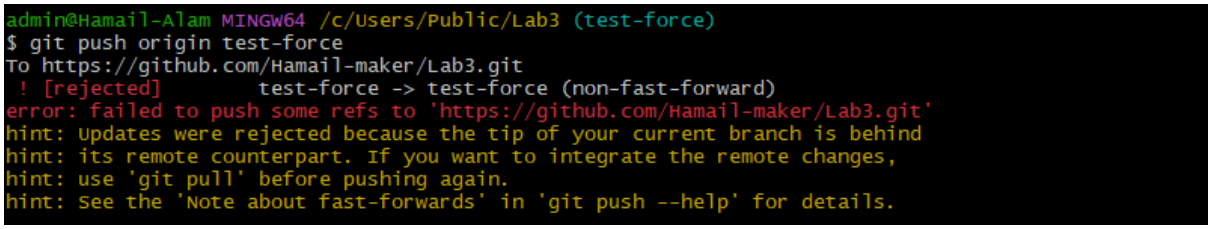
- Save a screenshot as hard_reset_force.png.

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3 (test-force)
$ git reset --hard HEAD~1
HEAD is now at a6d7320 Revert "Local update to README"
```

1. Push again

git push origin test-force

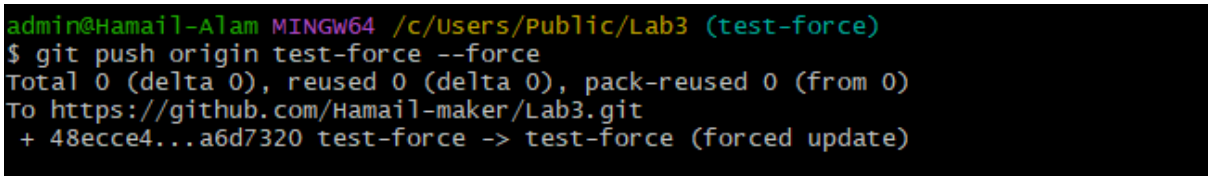
- Rejected by remote repository
- Save a screenshot as normal_push.png.



1. Now force-push to remote:

git push origin test-force --force

- Save a screenshot as force_push.png.

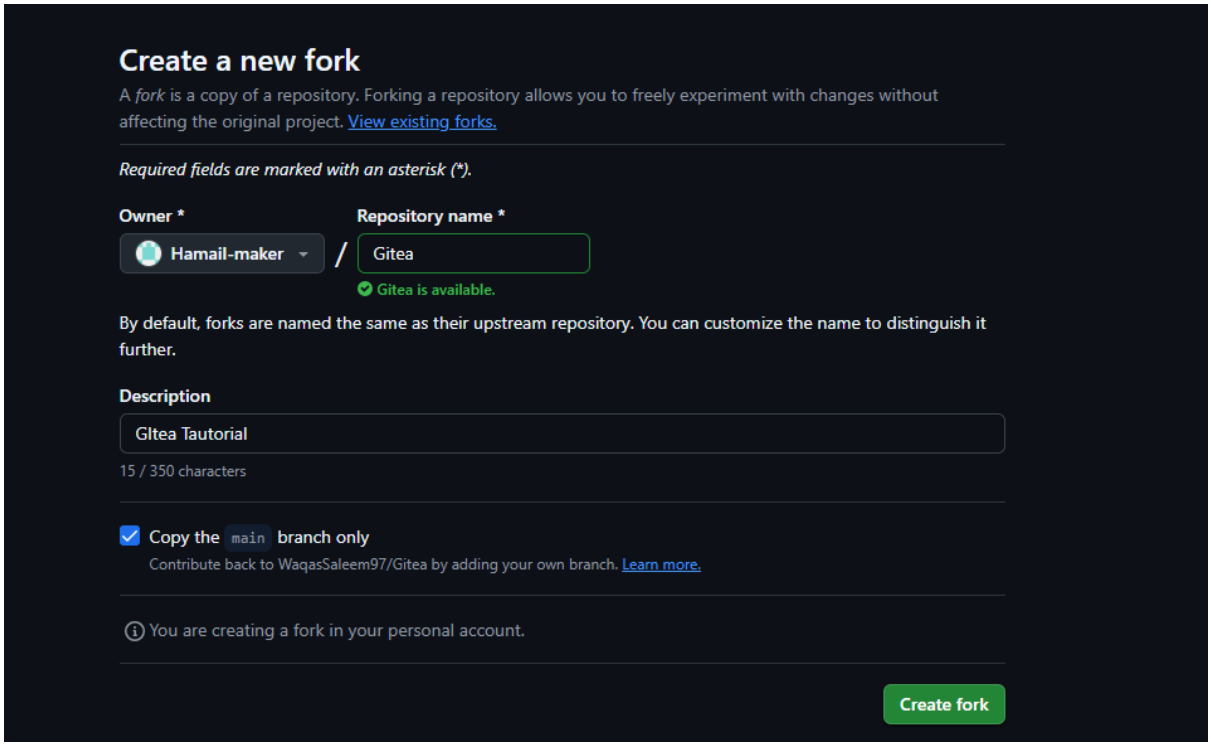


Task 10 – Running Gitea in GitHub Codespaces via Docker Compose

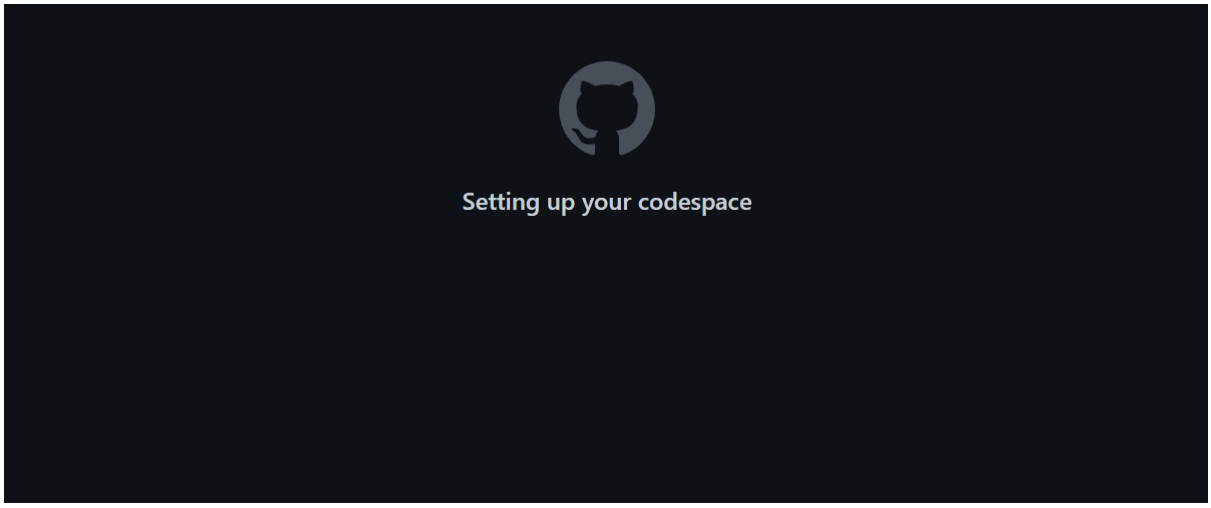
This task will guide you through forking a repository, running a full web application in a GitHub Codespace, and interacting with Gitea (a self-hosted Git service) inside Codespaces.

Steps

1. **Fork the Gitea Repository**
   - Go to Gitea.
   - Click the **Fork** button at the top right and fork it into your own GitHub account.
   - Save a screenshot of your forked repository page as forked_gitea.png.



1. **Open the Forked Repo in GitHub Codespaces**
    - On your forked repo, click **Code** > **Codespaces** > **Create codespace on main**.
    - Save a screenshot of the Codespace loading as codespace_loading.png.

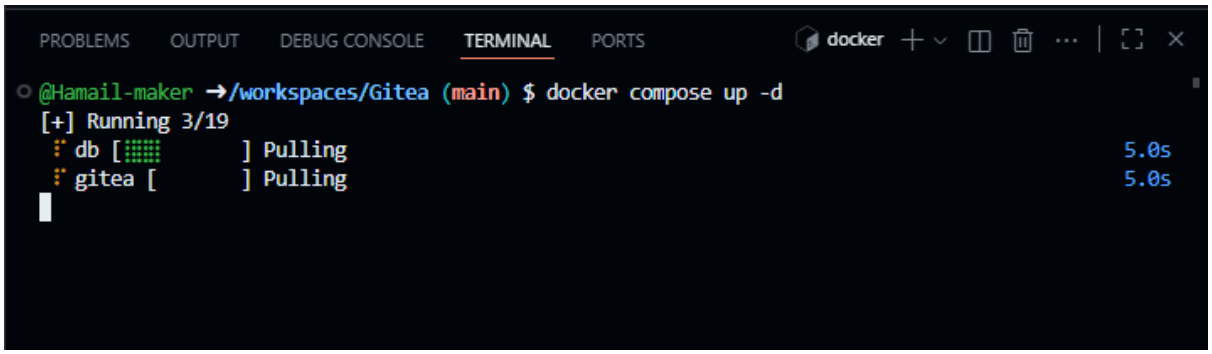


1. **Start Gitea with Docker Compose**
    - In the Codespace terminal, run:

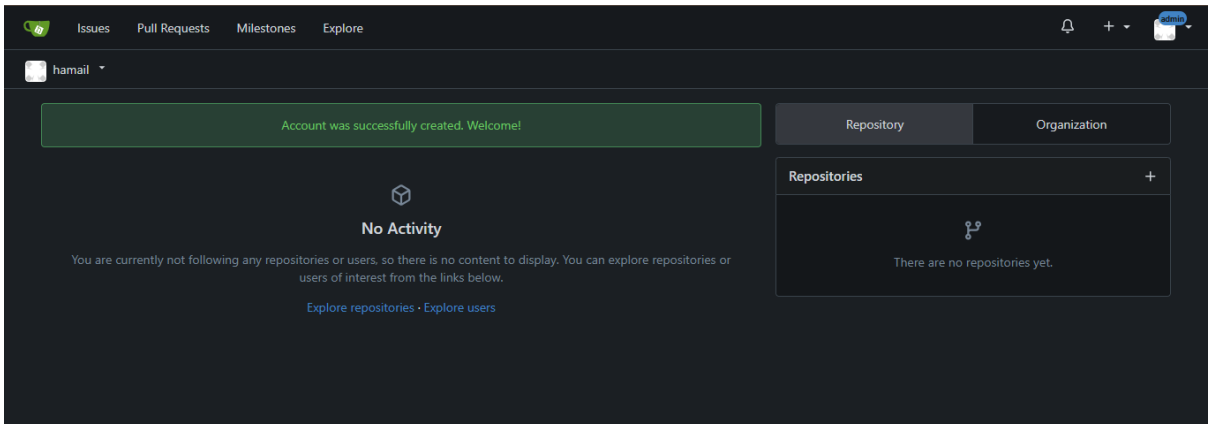
docker compose up -d

- Wait for the containers to start.

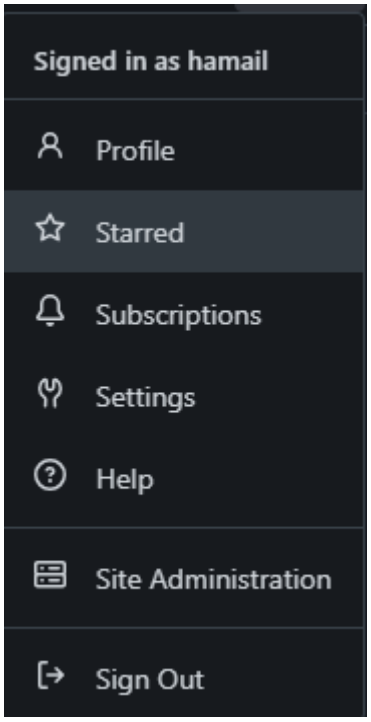
- Save a screenshot of the terminal showing the containers running as docker_up.png.



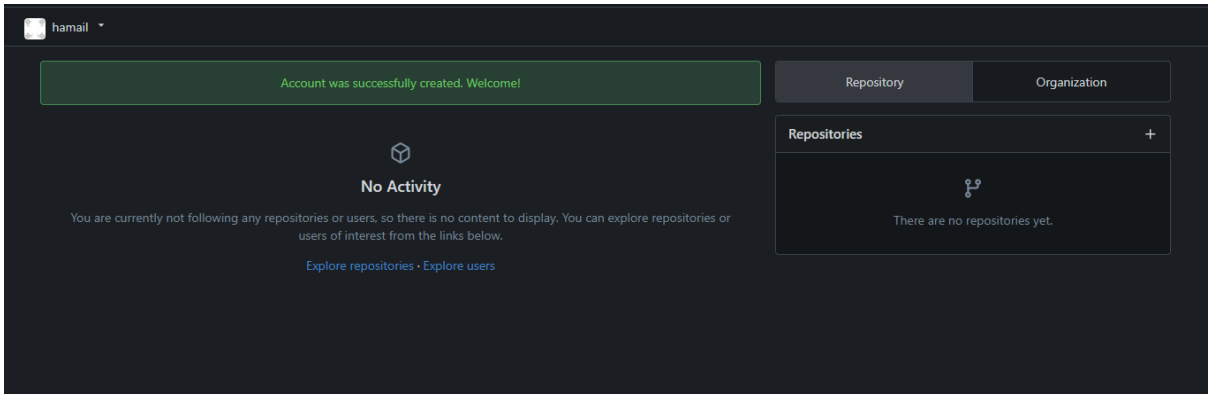
1. **Access Gitea Web Interface**
   - In Codespaces, forward **port 3000** (see Codespaces port forwarding UI).
   - Change the Visibility of **port 3000** from **private** to **public**.
   - Click the forwarded port 3000 link to open Gitea in your browser tab.
   - Save a screenshot of the Gitea install page as gitea_install_page.png.



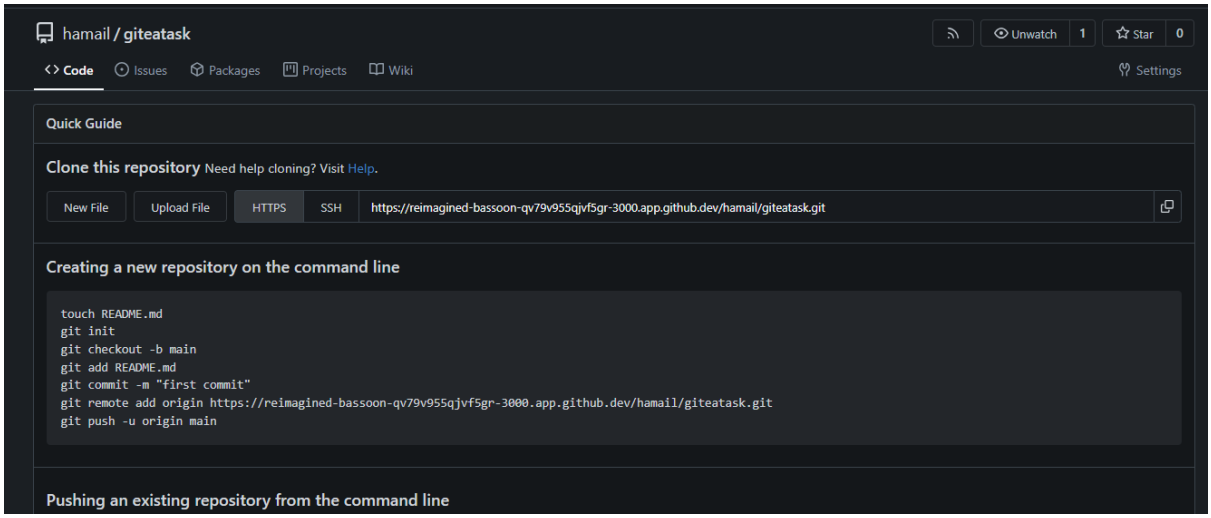
1. **Install Gitea**
   - Fill out the installation form, providing an **admin username** and **password** (e.g., admin / yourpassword).
   - Save a screenshot of the completed setup (show the admin account setup) as admin_setup.png.



1. **Log In to Gitea**
    - Use your admin credentials to log in.
    - Save a screenshot of the Gitea dashboard after login as gitea_dashboard.png.



1. **Create a New Repository in Gitea**
    - Click **New Repository** or **+** > **New Repository** in Gitea.
    - Fill out the repository details and create it.
    - Save a screenshot of the new repository page as gitea_new_repo.png.



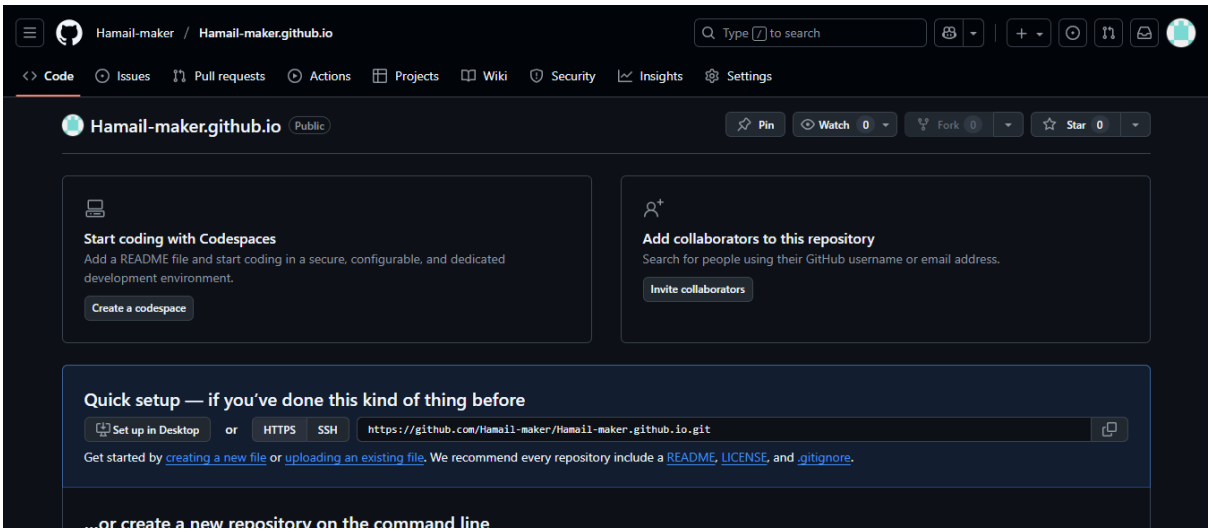
Task 11 – Creating a GitHub Pages Portfolio Site

This task will guide you through creating a personal portfolio/static site using GitHub Pages.

Steps

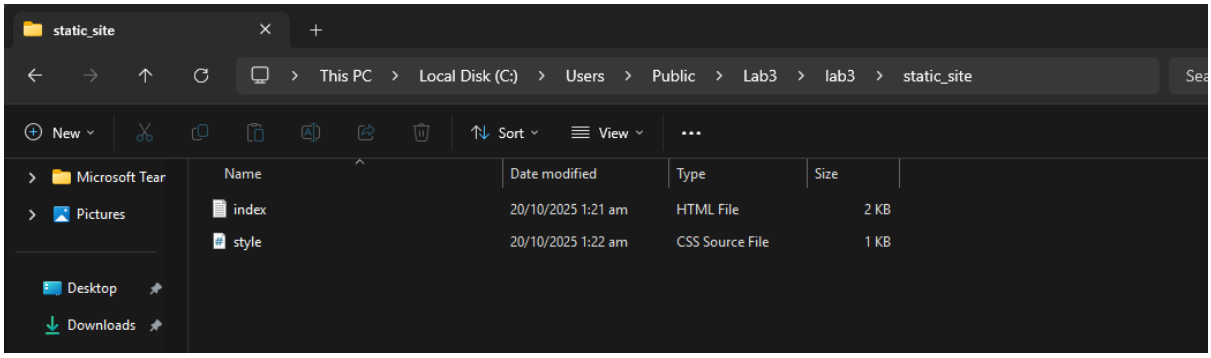
1. **Create a GitHub Pages Repository**

   - Go to GitHub and create a new repository named <your-username>.github.io (e.g., if your username is WaqasSaleem97, the repo should be WaqasSaleem97.github.io).
   - Make sure the repository is **public**.
   - Save a screenshot of the newly created repository as github_pages_repo.png.



Add Static Website Code

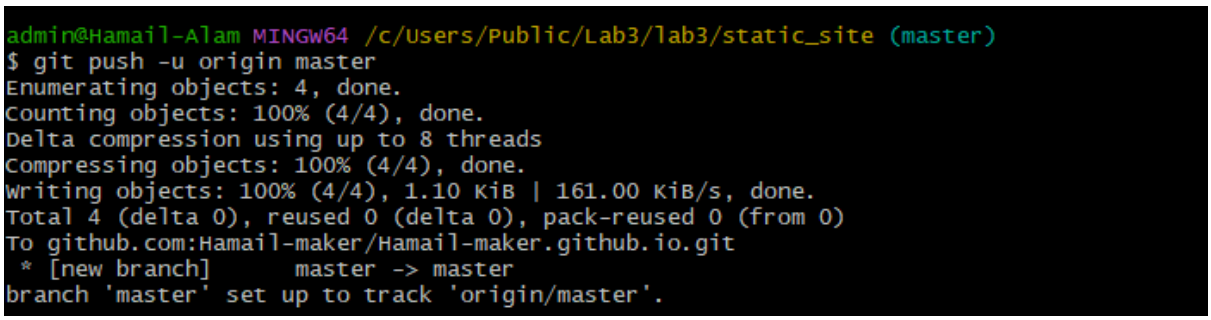
- Prepare a simple static website (e.g., your CV or portfolio) in HTML/CSS/JS by using the sample code provided in the repository Iqra Bashir CV.
- Put your files in a local folder.
- Save a screenshot of your local files as local_static_site.png.



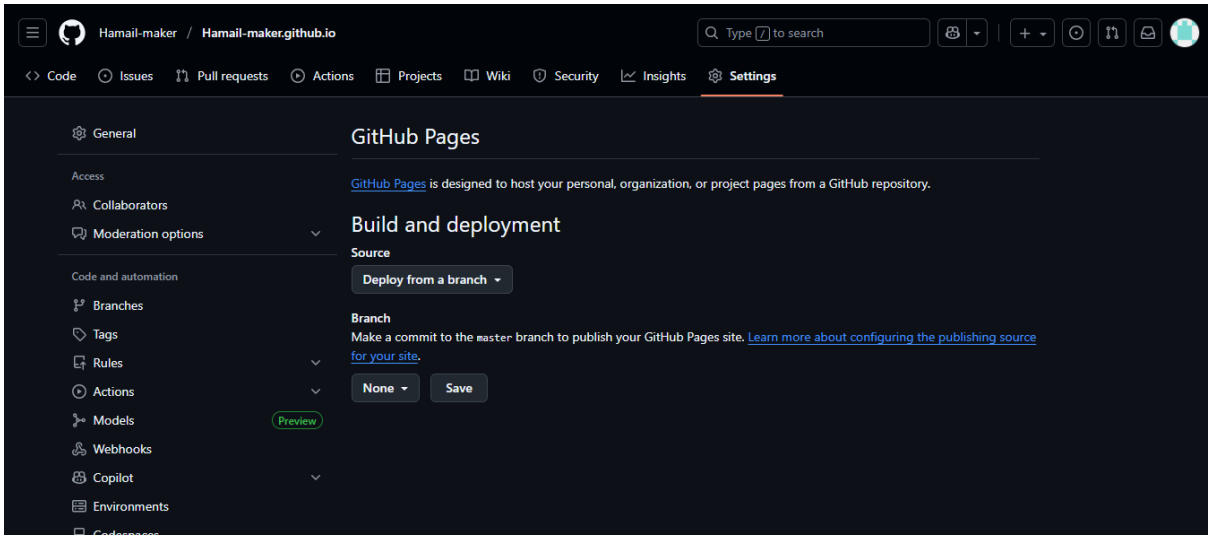
- Add and commit your files:
- git add .
- git commit -m "Add portfolio site for GitHub Pages"

git push -u origin main

- Save a screenshot of your terminal showing a successful push as push_static_site.png

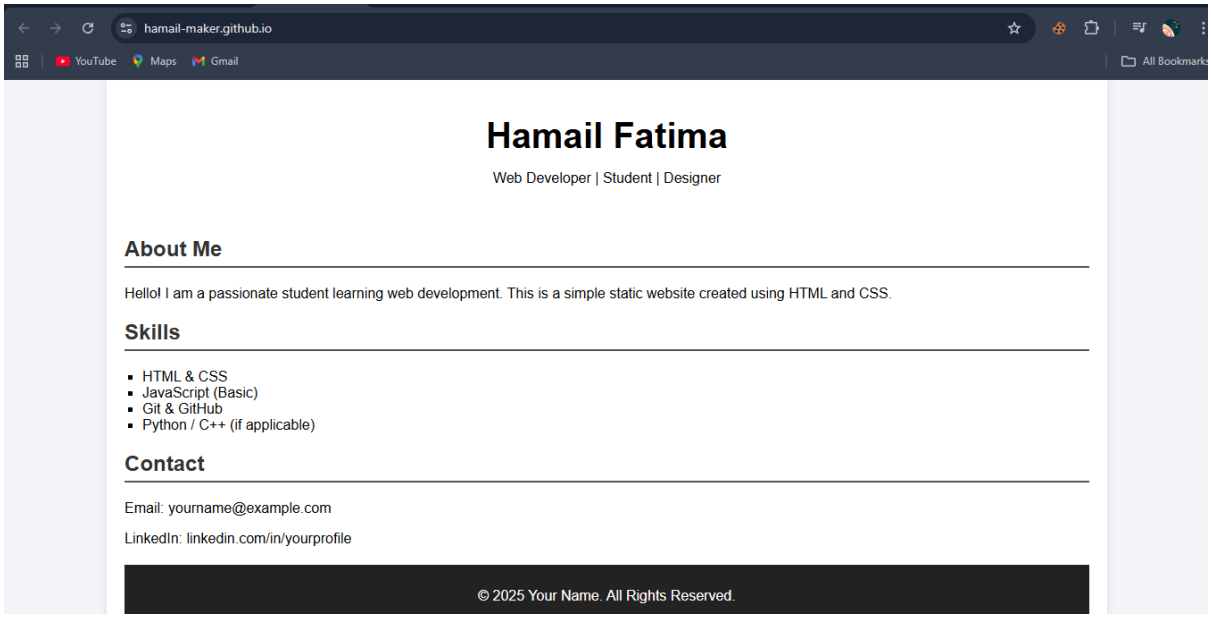


1. **Check GitHub Pages Settings**
    - Go to your repository on GitHub.
    - Click **Settings** > **Pages**.
    - Confirm that your site is published and see the link to your live site (e.g., https://your-username.github.io).
    - Save a screenshot of the Pages settings as github_pages_settings.png.



Visit Your Live Site

- Open your GitHub Pages site in the browser.
- Save a screenshot of your live site as live_site.png



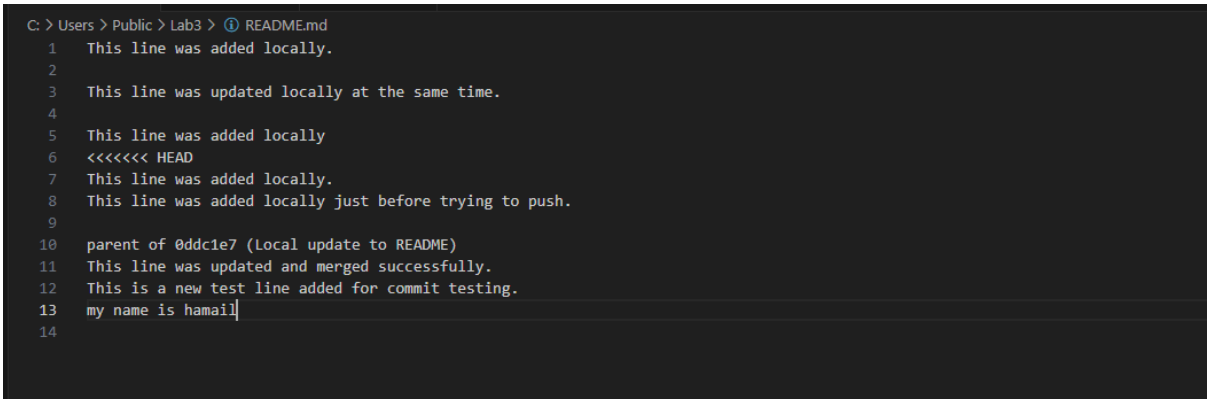
Exam Evaluation Questions

1. Local vs Remote Conflict Resolution

Steps:

1. On GitHub, edit a file (e.g., README.md) and commit the change.

   Screenshot as Q1_remote_edit.png



On your local machine, edit the same file differently (Avoid Conflict) and commit.

- Screenshot as Q1_local_edit.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ cd /path/to/repo
git checkout main
git pull origin main          # ensure you're up to date before editing
# edit README.md locally (change a different line or same file differently)
git add README.md
git commit -m "Local edit: change README"
bash: cd: /path/to/repo: No such file or directory
error: pathspec 'main' did not match any file(s) known to git
fatal: couldn't find remote ref main
fatal: pathspec 'README.md' did not match any files
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

1. Try to push your local commit and observe the error.
    - Screenshot as Q1_push_error.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git push origin master
Everything up-to-date
```

1. Resolve the conflict by running git pull (merge), then push.
    - Screenshot as Q1_merge_resolution.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git push origin master
Everything up-to-date
```

1. Repeat with another remote/local change, but resolve using git pull --rebase, then push.
    - Screenshot as Q1_rebase_resolution.png

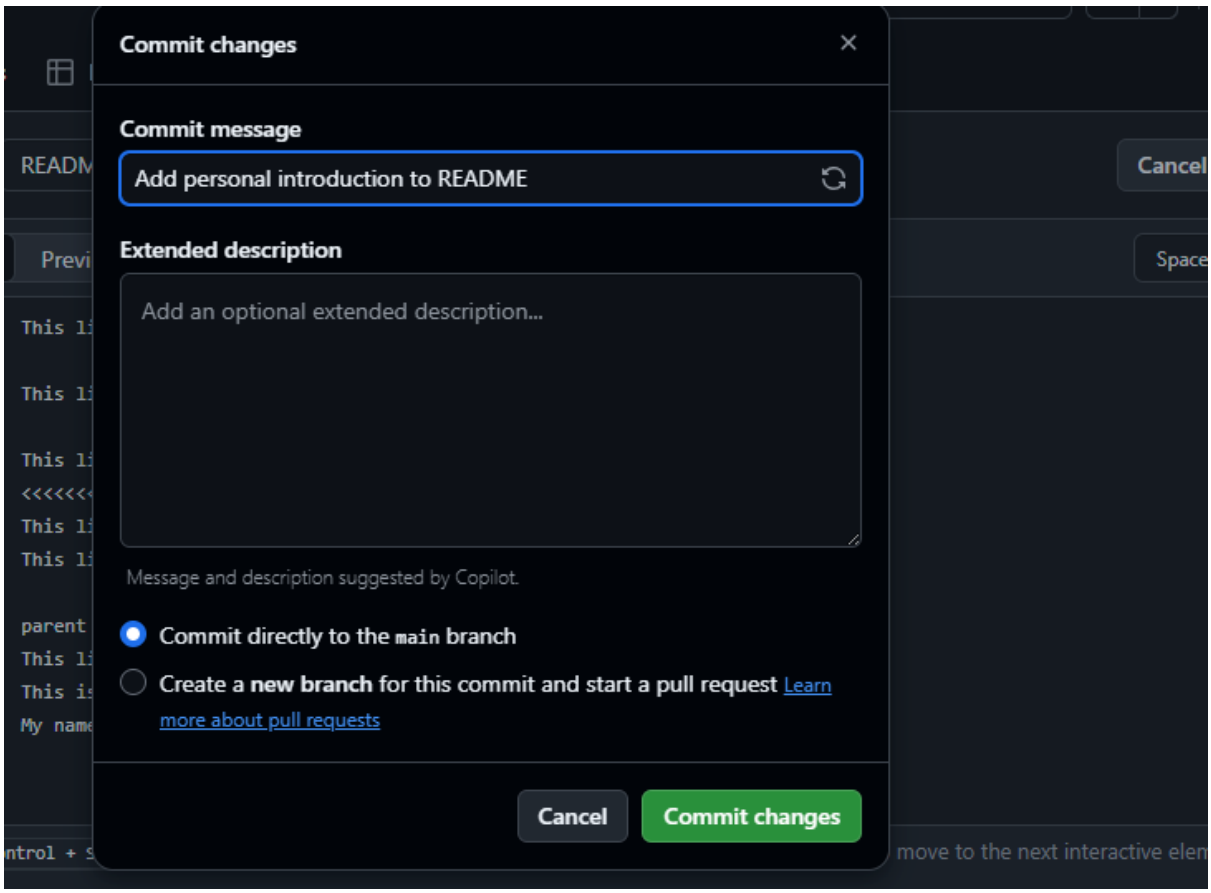
```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git pull --rebase origin master
# if no conflicts, your local commits will be replayed after remote commits
git push origin master
From github.com:Hamail-maker/Hamail-maker.github.io
 * branch            master     -> FETCH_HEAD
Already up to date.
Everything up-to-date
```

2. Manual Merge Conflict Handling

Steps:

1. On GitHub, change a specific line in a file and commit.

- Screenshot as Q2_remote_conflict_edit.png



2. Locally, change the same line differently and commit.

- Screenshot as Q2_local_conflict_edit.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git checkout master
git pull origin master  # to be safe
# edit same line in file differently
git add README.md
git commit -m "Local conflicting change"
Already on 'master'
Your branch is up to date with 'origin/master'.
From github.com:Hamail-maker/Hamail-maker.github.io
 * branch            master     -> FETCH_HEAD
Already up to date.
fatal: pathspec 'README.md' did not match any files
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

3. Try to push your local change and observe the conflict error.

- Screenshot as Q2_conflict_push_error.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git push origin master
Everything up-to-date
```

4. Use git pull --rebase to fetch changes and trigger the conflict.

- Screenshot as Q2_rebase_conflict.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git pull --rebase origin master
From github.com:Hamail-maker/Hamail-maker.github.io
 * branch            master     -> FETCH_HEAD
Already up to date.
```

5. Edit the conflicted file to resolve the conflict manually.

- Screenshot as Q2_resolved_file.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git merge README.md
git rebase --continue
merge: README.md - not something we can merge
fatal: no rebase in progress
```

6. Mark the conflict resolved (git add <file>, git rebase --continue) and push.

- Screenshot as Q2_resolution_complete.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git push origin master
Everything up-to-date
```

3. Managing Ignored and Tracked Files

Steps:

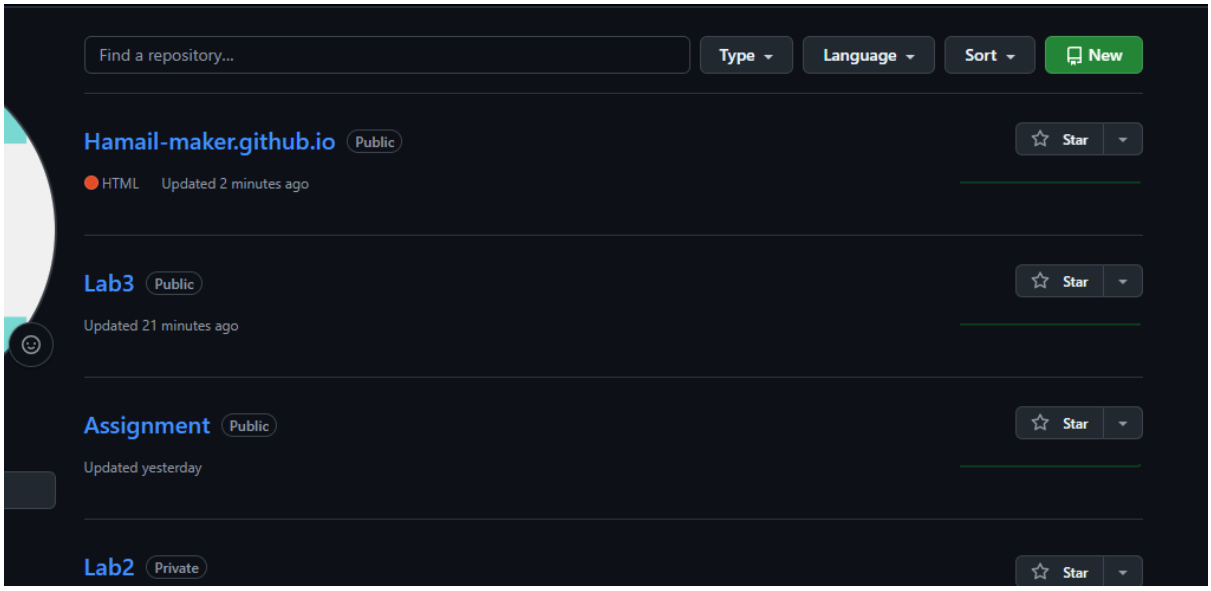
1. Create a new folder (e.g., DocFiles) and add several files inside.

- Screenshot as Q3_folder_created.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ mkdir DocFiles
echo "Doc 1" > DocFiles/doc1.txt
echo "Doc 2" > DocFiles/doc2.txt
git add DocFiles
git commit -m "Add DocFiles with 2 docs"
git push origin master
warning: in the working copy of 'DocFiles/doc1.txt', LF will be replaced by CRLF the next time Git touche
warning: in the working copy of 'DocFiles/doc2.txt', LF will be replaced by CRLF the next time Git touche
[master 057a092] Add DocFiles with 2 docs
 2 files changed, 2 insertions(+)
 create mode 100644 DocFiles/doc1.txt
 create mode 100644 DocFiles/doc2.txt
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 419 bytes | 419.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Hamail-maker/Hamail-maker.github.io.git
   f7a88e9..057a092  master -> master
```

2. Commit and push the folder/files to GitHub.

- Screenshot as Q3_files_pushed.png



3. Add the folder to your .gitignore file.

- Screenshot as Q3_gitignore_added.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git add .gitignore
git commit -m "Ignore DocFiles"
git push origin master
fatal: pathspec '.gitignore' did not match any files
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

4. Commit and push the .gitignore update.

- Screenshot as Q3_gitignore_pushed.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git rm -r --cached DocFiles
git commit -m "Stop tracking DocFiles"
git push origin master
rm 'DocFiles/doc1.txt'
rm 'DocFiles/doc2.txt'
[master c88bf9f] Stop tracking DocFiles
 2 files changed, 2 deletions(-)
 delete mode 100644 DocFiles/doc1.txt
 delete mode 100644 DocFiles/doc2.txt
```

5. Remove the folder from tracking using git rm -r --cached <folder>.

- Screenshot as Q3_folder_untracked.png

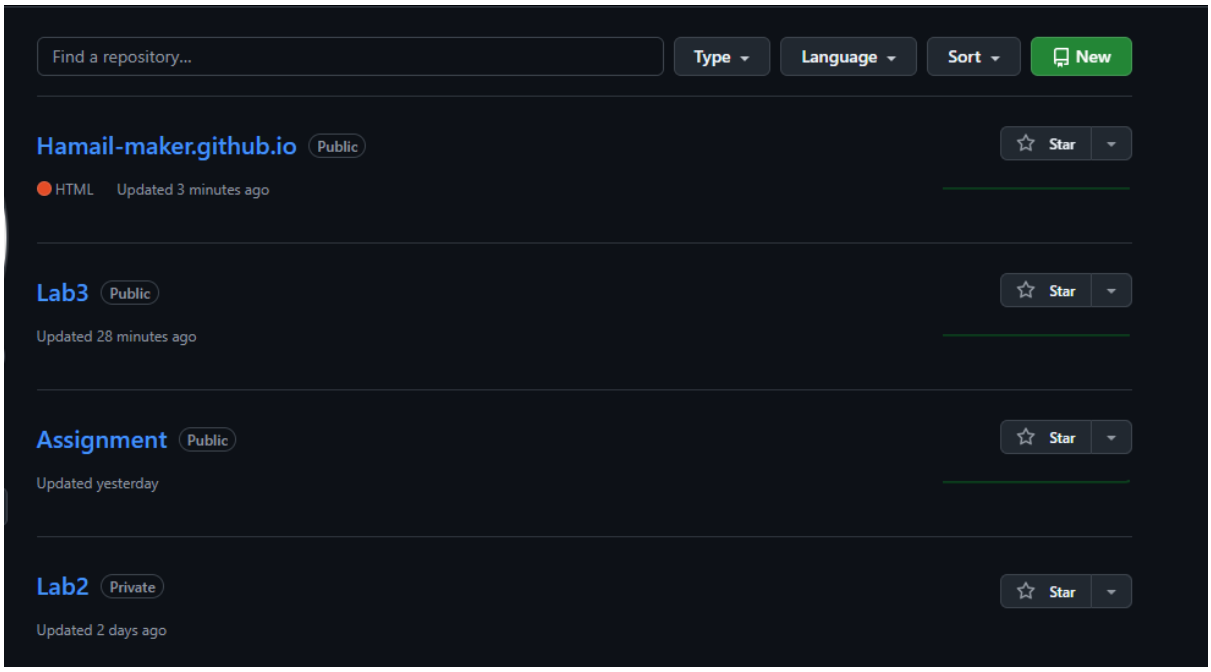
```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git rm -r --cached DocFiles
git commit -m "Stop tracking DocFiles"
git push origin master
fatal: pathspec 'DocFiles' did not match any files
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DocFiles/

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

6. Commit and push the change, then verify the folder is no longer tracked on GitHub.

- Screenshot as Q3_folder_removed_github.png



Commit History Manipulation and Recovery

Goal: show soft reset and hard reset effects and how to recover.

Important: Before resets, make sure these are safe in your repo. Do not run --hard on commits you need unless you understand they will be lost locally (can be recovered via reflog but be careful).

A — Make changes and commits

1. First change:

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ # change file A
git add A
git commit -m "First commit"
fatal: pathspec 'A' did not match any files
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DocFiles/

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

2.Make another change and commit again.

Screenshot as Q4_second_commit.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ # change file B
git add B
git commit -m "Second commit"
fatal: pathspec 'B' did not match any files
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DocFiles/

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

1. View your commit history.

- Screenshot as Q4_commit_history.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git log --oneline --graph --decorate --all
* c88bf9f (HEAD -> master, origin/master) Stop tracking DocFiles
* 057a092 Add DocFiles with 2 docs
* f7a88e9 Add portfolio site for GitHub Pages
```

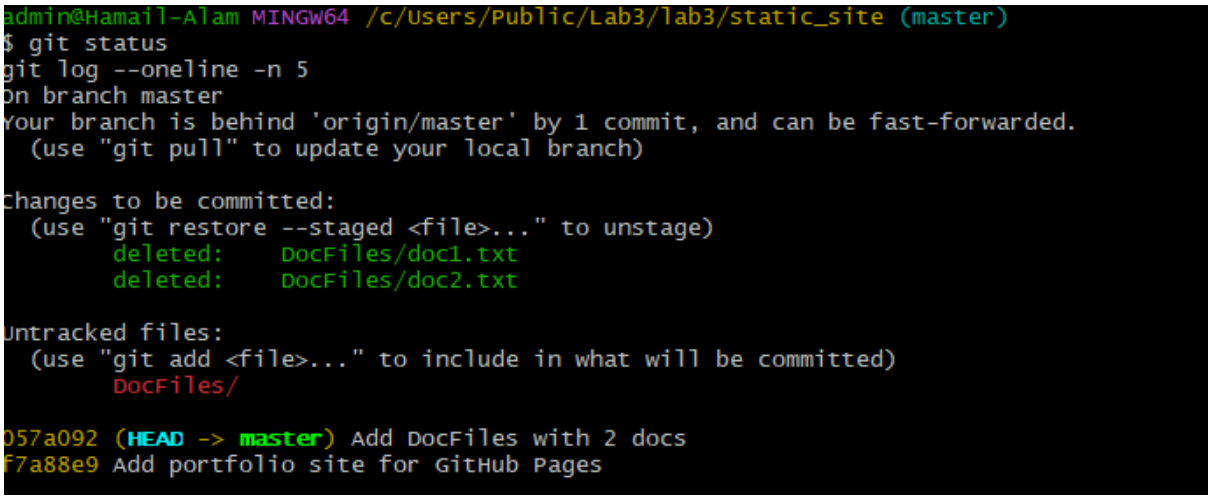
2. Perform a soft reset (git reset --soft HEAD~1) and observe your file and history.

- Screenshot as Q4_soft_reset.png

```
admin@Hamail-Alam MINGW64 /c/Users/Public/Lab3/lab3/static_site (master)
$ git reset --soft HEAD~1
```

3. Make commit again.

- Screenshot as Q4_third_commit.png



4. Perform a hard reset (git reset --hard HEAD~1) and observe the changes.

- Screenshot as Q4_hard_reset.png