**Exercise 4 Git-HOL**

**1. Introduction**

A merge conflict occurs when Git is unable to automatically reconcile changes between two branches. This typically happens when the same lines of code in a file are modified differently in both branches, or when one branch deletes a file that another branch has modified.

This lab will guide you through the process of intentionally creating a merge conflict and then resolving it manually.

**Objectives**

* Understand how to create a merge conflict.
* Manually resolve a merge conflict by editing a file.
* Commit the resolved changes and clean up the repository.

**Prerequisites**

* **Git** is installed and configured.
* You have a working Git repository.
* A basic understanding of Git branching and committing.
* We will be using manual text editing to resolve conflicts, which works universally and doesn't require a specific merge tool.

**Part 1: Creating a Merge Conflict**

**Step 1: Create a branch and make a change**

First, ensure you are on a clean main branch.

git status

Now, create a new branch named feature-branch and switch to it.

git branch feature-branch

git switch feature-branch

In the feature-branch, create a file named message.txt with some content.

echo "Hello from the feature-branch!" > message.txt

Stage and commit this new file.

git add message.txt

git commit -m "feat: Add message.txt in feature-branch"

**Step 2: Create a competing change on the main branch**

Switch back to the main branch.

git switch main

Now, create the *same* file, message.txt, but with *different* content.

echo "Hello from the main branch!" > message.txt

Stage and commit this change on the main branch.

git add message.txt

git commit -m "feat: Add message.txt in main branch"

**Step 3: Observe the diverged history**

At this point, the main and feature-branch have diverged, each with a different version of the same file. To visualize this, use the git log command.

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

**Part 2: Resolving the Conflict**

**Step 1: Attempt to merge**

While you are on the main branch, attempt to merge the feature-branch.

git merge feature-branch

Git will stop and report a merge conflict because it cannot automatically decide which version of message.txt to keep.

**Step 2: Inspect the conflict**

Check the status of your repository to see which files are in an "unmerged" state.

git status

Now, open message.txt in a text editor. You will see Git's conflict markers:

<<<<<<< HEAD

Hello from the main branch!

=======

Hello from the feature-branch!

>>>>>>> feature-branch

**Explanation of markers:**

* <<<<<<< HEAD: Marks the beginning of the conflicting change from your current branch (main).
* =======: Separates the changes from the two branches.
* >>>>>>> feature-branch: Marks the end of the conflicting change from the branch you are merging (feature-branch).

**Step 3: Manually resolve the conflict**

Edit the message.txt file to remove the conflict markers and combine the content as you see fit. For example, you might decide to keep both messages.

Hello from the main branch!

Hello from the feature-branch!

Save the file and close your editor.

**Step 4: Complete the merge**

After resolving the file, you need to inform Git that the conflict is fixed.

Stage the resolved file.

git add message.txt

Now, commit the merge. Git will automatically create a default merge commit message for you.

git commit

**Part 3: Cleanup After Merging**

**Step 1: Delete the merged branch**

Once the feature-branch has been successfully merged, it is good practice to delete it.

git branch -d feature-branch

**Step 2: View the final history**

Use git log one last time to see the merge commit that connects the two branches, confirming a successful resolution.

git log --oneline --graph --decorate