**Objective:**

* To understand and resolve merge conflicts in Git.
* To learn advanced Git operations like rebasing and cherry-picking.
* To explore Git hooks and automation for streamlined workflows.

### **Assessment & Exercises:**

1. Explain the purpose of Git rebase and how it differs from merge.

|  |  |  |
| --- | --- | --- |
| Feature | Git Rebase | Git Merge |
| Definition | Moves the feature branch to the latest commit of the main branch, replaying commits one by one. | Combines two branches by creating a new merge commit. |
| Commit History | Keeps a linear commit history. | Creates a new merge commit, preserving history. |
| Best Used For | Keeping feature branch up to date with the main branch before merging. | Combining branches while maintaining the commit history. |
| Command | git rebase main | git merge feature-branch |

1. How can you resolve a merge conflict manually?

Attempt to merge branches:

git merge feature-branch

If a conflict occurs, Git will indicate the conflicting files. Open the affected file(s).

Locate the conflict markers (<<<<<<<, =======, >>>>>>>) and manually edit the file to keep the correct changes.

Save the file and mark the conflict as resolved:

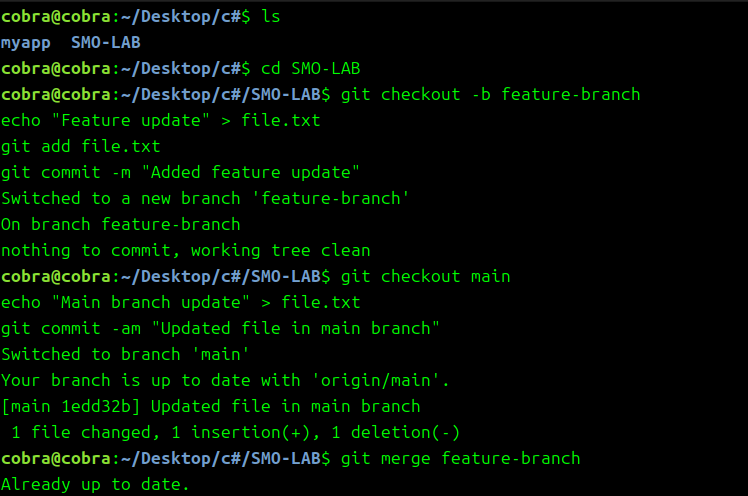
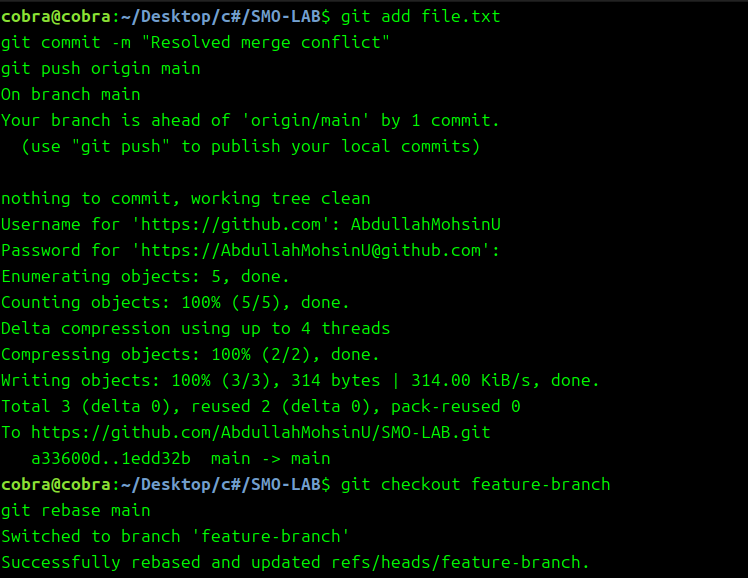
git add <conflicted-file>

Complete the merge with:

git commit -m "Resolved merge conflict"

Push the resolved changes to the remote repository:

git push origin main



1. What is the use of git cherry-pick, and when should it be used?

**Purpose**

* git cherry-pick is used to apply specific commits from one branch to another without merging the entire branch.

**When to Use?**

* When you want to include a single bug fix from a feature branch into the main branch.
* When a feature was mistakenly committed to the wrong branch and needs to be applied elsewhere.

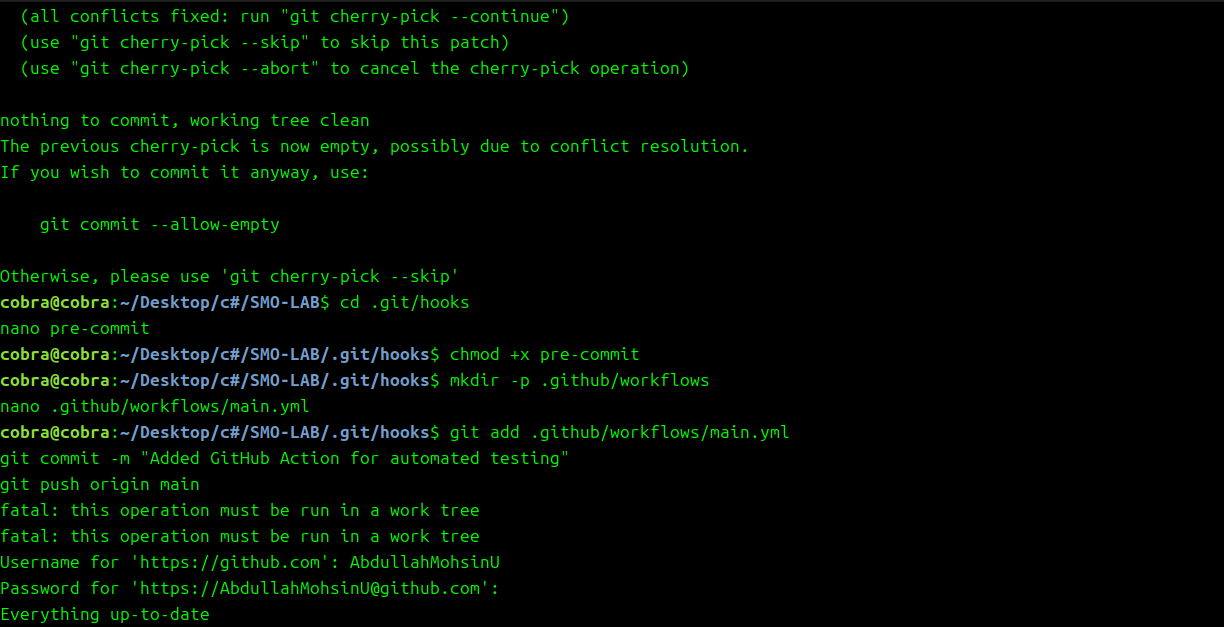
**Command to Cherry-Pick a Commit**

1. Identify the commit hash:

git log --oneline

1. Apply a specific commit to another branch:

git cherry-pick <commit-hash>



1. Implement a Git pre-commit hook that prevents committing large files.

Navigate to the Git hooks directory:

cd .git/hooks

Create a **pre-commit** hook script:

nano pre-commit

Add the following script to prevent committing files larger than 5MB:

#!/bin/sh

maxsize=5000000

for file in $(git diff --cached --name-only); do

if [ -f "$file" ]; then

filesize=$(wc -c <"$file")

if [ $filesize -gt $maxsize ]; then

echo "Error: $file is larger than 5MB. Commit aborted."

exit 1

fi

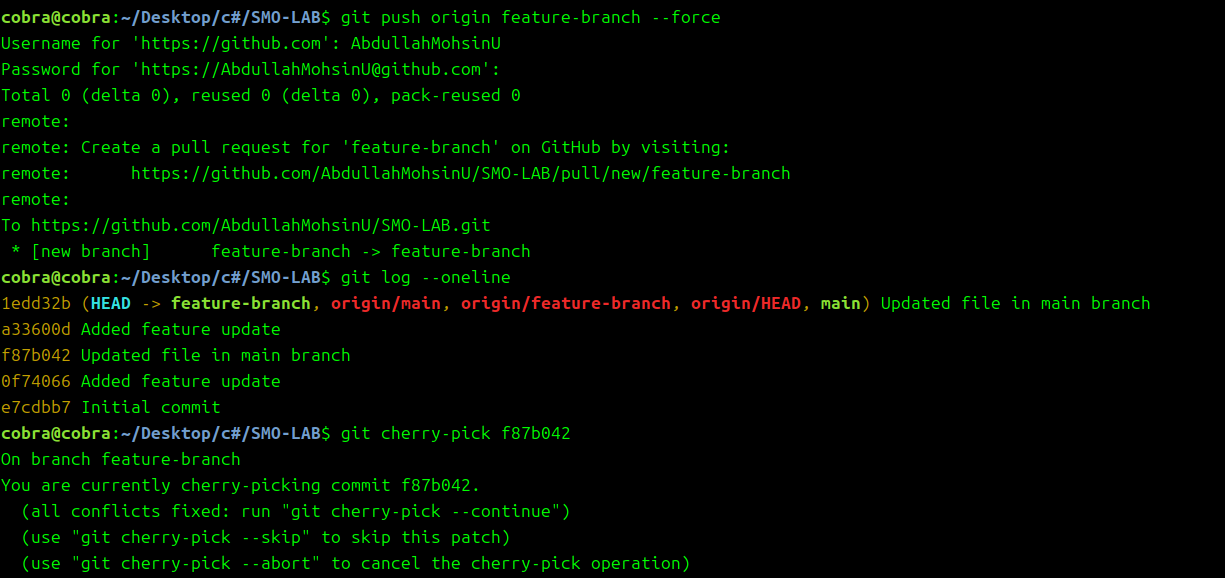
fi

done

Save and exit, then make the script executable:

chmod +x pre-commit

Test by attempting to commit a large file.



1. Configure a GitHub Action workflow for automated code testing.

Navigate to your project folder and create a **GitHub Actions workflow** directory:

mkdir -p .github/workflows

Create a new workflow file:

nano .github/workflows/main.yml

Add the following configuration:

name: CI Pipeline

on: push

jobs:

test:

runs-on: ubuntu-latest

steps:

name: Checkout code

uses: actions/checkout@v2

name: Run Tests

run: echo "Running tests..."

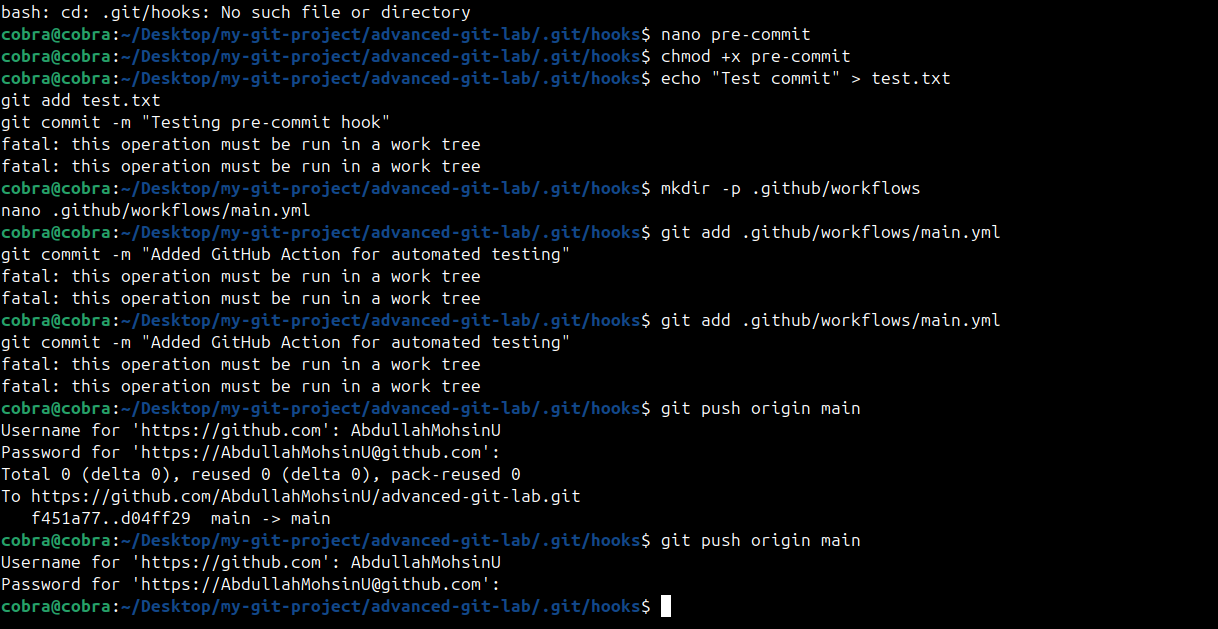
Save the file and commit it:

git add .github/workflows/main.yml

git commit -m "Added GitHub Action for automated testing"

git push origin main

Verify on **GitHub → Actions** that the workflow runs automatically

After running this command, take a **screenshot** of the terminal output showing successful cloning.