

SET-222

Software Operations &Maintenance

Experiment # 02

**Experiment Title**

**Advanced Git and GitHub Operations**

**Assessment of CLO(s): 03**

**Performed on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name:** |  | | |
| **Roll No.** |  | **Group** |  |
| **Semester** |  | **Session** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Perf. Level**  **Criteria** | **Excellent**  **(2.5)** | **Good**  **(2)** | **Satisfactory**  **(1.5)** | **Needs Improvement**  **(0 ~ 1)** | **Marks Obtained** |
| **1** | Project Execution & Implementation | Fully functional, optimized, and well-structured. | Minor errors, mostly functional. | Some errors, requires guidance. | Major errors, non-functional, or not Performed. |  |
| **2** | Results & Debugging  Or Troubleshooting | Accurate results with effective debugging  Or Troubleshooting. | Mostly correct, some debugging Or Troubleshooting needed. | Partial results, minimal debugging  Or Troubleshooting. | Incorrect results, no debugging Or Troubleshooting, or not attempted. |  |
| **3** | Problem-Solving & Adaptability  (VIVA) | Creative approach, efficiently solves challenges. | Adapts well, minor struggles. | Some adaptability, needs guidance. | Lacks innovation or no innovation, unable to solve problems. |  |
| **4** | Report Quality & Documentation | Clear, structured, with detailed visuals. | Mostly clear, minor gaps. | Some clarity issues, missing details. | Poorly structured, lacks clarity, or not submitted. |  |
| **Total Marks Obtained Out of 10** | | | | | |  |

**Experiment evaluated by**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructor’s Name** | **Ms. Shagufta Aftab** | | |
| **Date** |  | **Signature** |  |

**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.

**Lab Exercise:**

**Exercise 1: Resolving Merge Conflicts**

1. Create a repository and initialize Git:

git init advanced-git-lab

cd advanced-git-lab

1. Create and switch to a new branch:

git checkout -b feature-branch

1. Modify a file and commit changes:

echo "Feature update" > file.txt

git add file.txt

git commit -m "Added feature update"

1. Switch back to the main branch and make another change:

git checkout main

echo "Main branch update" > file.txt

git commit -am "Updated file in main branch"

1. Merge feature-branch into main:

git merge feature-branch

6. Resolve conflicts manually, then commit the resolved changes.

**Exercise 2: Git Rebase**

1. Switch to the feature branch:

git checkout feature-branch

1. Rebase onto the latest main branch:

git rebase main

1. If conflicts occur, resolve them and continue:

git rebase --continue

1. Push the rebased branch:

git push origin feature-branch --force

**Exercise 3: Cherry-Picking Commits**

1. Identify a commit hash from git log.
2. Apply a specific commit to another branch:

git cherry-pick <commit-hash>

**Exercise 4: Implementing Git Hooks**

1. Navigate to the hooks directory:

cd .git/hooks

1. Create a pre-commit hook:
2. echo "echo 'Checking code style'" > pre-commit

chmod +x pre-commit

1. Test by making a commit.

**Exercise 5: Automating Git Processes**

1. Set up a GitHub Action to run tests on push.
2. Create a .github/workflows/main.yml file and add:

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..."

### **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.

