Lab Manual: Hands-on Practice with Git Basics

Lab Assignment

INSTRUCTIONS:

1. Once your assignment is done, you are supposed to submit one pdf file with the following:

a. Name and Roll number at the top of the report

b. Assignment question. Answer/program code

c. Comment the lines of code wherever required.

d. For each question related to codes describe the process of your approach in words/diagrams. Screenshot of the output which you get after executing the program. Multiple screenshots may be uploaded to clarify the execution of the program.

2. While Submission, name the file to be submitted in the following format: <your roll number_assignment no.>

3. DO NOT COPY from others. ***All reports*** *which are found to be copied will be given 0 marks for the assignment.*

Objective:

Install Git, configure your identity, and initialize a Git repository.

Prerequisites:

- Basic familiarity with the command-line interface (CLI).

Lab Setup:

1. Install Git on your laptop following the official instructions for your operating system.

Lab Exercises:

Name:

Roll No:

Group Members with Roll Numbers:

Exercise 1: Getting Started with Git

Objective: Install Git, configure your identity, and initialize a Git repository.

1. **Install Git:**

- Download and install Git from the official website: Git Downloads
- Follow the installation instructions for your operating system.

2. **Configure Identity:**

- Set your username and email globally using the following commands:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

- Verify your Git configuration:

```
git config --global --list
```

3. **Initialize Repository:**

- Create a new directory for your project:

```
mkdir my_project cd my_project
```

- Initialize a new Git repository in the directory:

```
git init
```

Exercise 2: Working with Files and Staging Area

Objective: Add files to the staging area, check status, and unstage files.

1. **Add Files to Staging:**

- Add specific files to the staging area:

```
git add <file_name>
```

- Add all files to the staging area:

```
git add
```

2. **Check Repository Status:**

- Check the status of the Git repository:

```
git status
```

3. **Unstage Files:**

- Unstage specific files from the staging area:

```
git reset <file_name>
```

Exercise 3: Branching and Merging

Objective: Create and switch branches, merge changes, and resolve conflicts.

1. **Create and Switch Branches:**

- Create a new branch for a new feature or bug fix:

```
git checkout -b feature_branch
```

- **Switch to a different branch:**

```
git checkout main
```

2. **Merge Changes:**

- Merge changes from one branch into another:

```
git merge feature_branch
```

3. **Resolve Merge Conflicts:**

- Resolve merge conflicts in files with conflicting changes.
- Add resolved files to the staging area and commit changes.

Exercise 4: Collaborating with Remote Repositories

Objective: Clone a repository, push changes to remote, and collaborate with others.

1. **Clone Repository:**

- Clone a remote repository from GitHub:

```
        git clone <repository_url>
```

2. **Push Changes to Remote:**

- Push committed changes to the remote repository:

```
        git push origin main
```

3. **Pull Changes from Remote:**

- Pull changes from the remote repository to your local repository:

```
        git pull origin main
```

Remote refers to the repository at the github profile of an user whereas the origin is the alias for the repo link that is hosted at GitHub. The command to change the branch name

```
       git branch -m old_name new_name
```

Exercise 5: Advanced Git Operations

Objective: Perform advanced Git operations such as reverting changes, viewing commit details, and stashing changes.

A merge conflict is a case where when 2 branches are created from a branch say main branchand then both the branches edit the same file and they are merged to the main branch then git is not able to decide which content it should keep and so this is the case of a merge conflict.

1. **Revert Changes:**

- Revert changes introduced by a specific commit:

```
    git revert <commit_hash>
```

2. **View Commit Details:**

- View details of a specific commit:

```
    git show <commit_hash>
```

3. **Stash Changes:**

- Git stash temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on. Stashing is handy if you need to quickly switch context and work on something else, but you're mid-way through a code change and aren't quite ready to commit.
- Temporarily shelve changes to the working directory:

```
    git stash
```

4. Apply Stashed Changes:

- Apply stashed changes to the working directory:

```
    git stash apply
```

- Pull requests are a mechanism for a developer to notify team members that they havecompleted a feature. Once their feature branch is ready, the developer files a pull request via their GitHub account. This lets everybody involved know that they need to review the code and merge it into the main branch.

5. **Show last commit details**:

- show last commit details

```
        git log -1 --stat
```

6. **List of commits in one line**:

- List of commits in one line

```
        git log –oneline
```