LAB ACTION **PLAN FOR WEEK 2**

Objective:

1. Pushing multi-folder project into private repository (by student).
2. Students must explore all listed git commands on the multi-folder project in local and remote repository.
3. Students must explore all git commands on given scenario-based question.
4. Students need to make note in observation book and also upload the executed exercise on scenario-based question.

Task 1: Pushing multi-folder project into private repository(Note: student take screenshots for upload).

1. Create a Private Repository on GitHub

- Go to https://github.com
- Click **New repository**
- Give it a name
- Check **Private**
- Click **Create repository**

2. Initialize Git in Your Local Project (if not already done)

Open a terminal in the **root** of your multi-folder project:

```
cd /path/to/your/project
git init
```

3. Add the Remote Repository (in bash)

You’ll see a URL on GitHub after creating the repo. It will look as below , run this command in git bash:

```
git remote add origin https://github.com/yourusername/your-private-repo.git
```

4. Add and Commit Your Files(in bash)

```
git add .
git commit -m "Initial commit"
```

5. Push to the Private Repository

If the repo is new and empty, push like this from git bash:

```
git push -u origin master
```

If you're using the main branch (GitHub default):

```
git push -u origin main
```

You may be prompted to log in (via username/password or token).

Task 2: Students must explore all listed git commands on the multi-folder project in local and remote repository.

1. **Git Commands:**

- git version : The command git version is used to check the version of git.

  git --version

- git config: Configures Git settings. Commonly used to set up user information

  git config --global user.name "Your Name"

  git config --global user.email "youremail@example.com"

- **git config --list**: Displays all the Git configurations for the current user.

2. **Repository Management**

- **git init**: Initializes a new Git repository in the current directory

  git init

- **git clone**: Creates a copy of an existing Git repository from a remote source (e.g., GitHub) to your local machine

  git clone https://github.com/username/repository.git

- **git remote –v :** To see the remote repository that is connected to your local repository

  git remote –v

- **git remote add :** To add a new remote repository to your local repository

  git remote add origin https://github.com/username/repository.git

- **Pushing Changes to Remote:** To push your local commits to a remote repository:

  git push <remote_name> <branch_name>

  git push origin main

- **Pulling Updates from Remote:** To pull the latest changes from the remote repository and merge them into your local branch

  git pull <remote_name> <branch_name>

  git pull origin main

- **git remote remove:** To remove a remote repository from your local configuration:

  git remote remove <remote_name>

  git remote remove origin

- **Renaming a Remote:** to rename an existing remote:

```
git remote rename <old_name> <new_name>
```

git remote rename origin upstream

- **Fetching Updates from Remote:** To fetch updates from the remote repository but not merge them into your local branch

  git fetch <remote_name>

- **Changing Remote URL :** To change the URL of a remote (e.g., after changing the remote repository address):

  git remote set-url <remote_name> <new_url>

  git remote set-url origin https://github.com/username/new-repository.git

- Viewing the Remote Repository's Information: to see detailed information about a remote repository

  git remote show <remote_name>

  git remote show origin

3. Staging and Committing

- **git status**: Shows the status of changes in your working directory and staging area. It tells you which files are untracked, modified, or ready to be committed.

  ```
  git status
  ```

- **git add**: Adds changes in the working directory to the staging area.

  ```
  git add filename.txt   # Adds a specific file
  git add .              # Adds all changes in the directory
  ```

- **git commit**: Commits the staged changes to the repository with a descriptive message. The -m option allows you to include a commit message.

  ```
  git commit -m "Commit message describing changes"
  ```

4. Branching and Merging

- **git branch**: Lists all branches or creates a new branch

  ```
  git branch             # Lists all branches or
  git branch -a          # Lists all branches
  git branch branch-name   # Creates a new branch
  git branch -d <branch_name>   # Deletes a branch
  ```

Use -D to force-delete if it hasn't been merged.

- **Listing All Remote Branches** : To you want to list all branches on the remote repository

git branch –r

- **git checkout**: Switches to a different branch

  git checkout branch-name # Switches to an existing branch

  git checkout -b new-branch # Creates and switches to a new branch

- Pruning Deleted Remotes : If a branch was deleted on the remote but still shows up locally.

  git remote prune origin

  When someone **deletes a branch on the remote**, your local Git doesn't automatically remove the corresponding remote-tracking branch (like origin/old-feature).
  git remote prune origin removes these outdated references.

- Fetching a Specific Remote Branch: To fetch a specific branch from a remote.
  git fetch <remote_name> <branch_name>

  git fetch origin feature-branch

- **Setting the Upstream Branch for Pushing** : When pushing for the first time and want to set the remote branch you are pushing to:

  git push --set-upstream <remote_name> <branch_name>

git push --set-upstream origin feature-branch

- **git merge**: Merges the specified branch into the current branch. This command integrates the changes from the feature branch into the main branch.

  git checkout main   # Switch to the main branch

  git merge branch-name   # Merge branch-name into main

- Rebasing a Local Branch onto a Remote Branch: If you want to rebase your local branch onto a remote branch (this can be useful to keep your history linear)

  git fetch <remote_name>
  git rebase <remote_name>/<branch_name>

Example:

git fetch origin
git rebase origin/main

5. Undoing Changes

- git reset : Removes the specified file from the staging area but leaves the working directory unchanged. git reset --hard can also reset the working directory and staging area to the last commit.

  git reset <file>:

- git revert**:** Creates a new commit that undoes the changes from a specified commit, leaving the history intact**.**

  git revert <commit>:

6. Viewing History

- git log**:** Shows a history of commits in the repository, including commit hashes, messages, and timestamps. Use git log --oneline for a more concise view.

  git log:

- git diff**:** Displays differences between various commits, the working directory, and the staging area. git diff without arguments shows changes not yet staged.

  git diff:

- git show**:** Shows the details of a specific commit, including the changes made and the commit message.

  git show <commit>

7. To undo the changes made to file before staging it.

- **git restore filename**

8. To correct committed with the wrong message

- **git commit --amend -m "Corrected commit message**"

9. Recovering deleted branches

- **git reflog**
- **git checkout -b feature-ui <commit_hash>**

10. To download the latest changes from the remote without merging

- **git fetch origin**

11. To remove accidentally committed sensitive file from Git history.

- **git filter-branch --force --index-filter \**
- **"git rm --cached --ignore-unmatch secrets.txt" \**
- **--prune-empty --tag-name-filter cat -- --all**

12. To merge changes from another branch

First switch to branch where changes are to applied and then merge another-branch

- **git checkout previous-branch**
- **git merge another-branch**

13. To resolve a merge conflict manually

You tried to merge two branches and Git reported a conflict in x.js. Then for conflict resolution do:

1. **Open x.js and resolve/remove the conflict markers (<<<<<<<, =======, >>>>>>>)**
2. After resolving:

   - ✓ **git add app.js**
   - ✓ **git commit # If Git didn't auto-create a merge commit**

14. The .gitignore

The .gitignore file is used to tell Git which files or directories to ignore in your project. It's a useful way to avoid committing unnecessary files like log files, build outputs, and IDE configurations.

a. Create a .gitignore File

In the root directory of your Git project, create a file called .gitignore

touch .gitignore

b. Add Rules to .gitignore

Inside the .gitignore file, you add patterns for the files and directories you want Git to ignore. Each pattern should be written on a new line.

Here are some common examples:

- Ignore all .log files: ***.log**
- Ignore a specific file: **secret_file.txt**
- Ignore all files in a temp/ directory: **temp/**
- Ignore all files except important_file.txt inside a folder:

  folder/*
  !folder/important_file.txt

- Ignore files with a specific extension: ***.bak**

15. To see who changed a particular line in a file

- **git blame script.py**

16. Git stash

# You modified files A and B, but not committed them.

git stash # saves changes and reverts to clean state

git switch another-branch

# do something else...

git switch main

git stash apply # brings back your changes

17. To check branch is merge

- **git branch –merged**

  If branch not merged then displays its name

18. To Delete multiple local branches at once

- **git branch –d branc1 branch2 branch3**

Task 3: Students must explore all git commands on given scenario-based question. (Note student write command in observation book and paste the answer (git command) below each question for uploading).

Scenario based questions on basic git commands

1. You made changes to one file in the above project but haven't staged them yet. You realize they were a mistake. What Git command will you use to discard the local changes?

2. You accidentally ran git add file1.txt (note instead of file.txt consider one file from above project), but you're not ready to commit yet. How do you remove it from the staging area without losing the changes?

3. You made a commit but typed the wrong commit message. You haven't pushed it yet. How do you fix it?

4. You want to view the commit history of the current branch in a readable format. What Git command should you use?

5. After cloning a repo, how can you set your name and email globally for all Git repositories?

6. You've made some edits to your files but haven't staged them. How can you view the changes?

7. You're on the main branch but need to switch to feature/login. What command do you

8. You deleted the feature-ui branch by mistake. You haven’t pushed the deletion. How

9. You’ve made some commits locally and now want to upload them to the remote repository. What do you run?

10. How can you fetch the latest changes from the remote repository without merging them automatically?

11. You’re on the main branch and want to start working on a new feature called search-filter. What command do you use?

12. You committed a file containing an API key and want to completely remove it from the repo's history. What should you do?

13. You want to see all the branches that exist both locally and on the remote. What

14. You’re on main and want to merge the completed feature/signup branch into it. What command do you use?

15. You tried to merge two branches and Git reported a conflict in app.js. What are the general steps to resolve it?

16. You don’t want Git to track changes to .log files or node_modules/. How do you achieve this?

17. You're investigating a bug and want to know who last changed line 25 in script.py. What command do you use?

18. You’re in the middle of working on a file but need to switch branches quickly. What do you do to save your work?

19. You previously ran git stash and now want to restore those changes. What command do you use?

20. The feature/test branch is no longer needed. How do you delete it locally?

21. You just merged the feature-ui branch into main. You want to clean up your local branches. How do you safely delete feature-ui?

22. You created a branch feature-experiment, made some changes, but now realize the code is no longer needed. You want to delete it, even though it hasn’t been merged. What command will you use?

23. You're currently on the feature-login branch and want to delete feature-ui. What must you ensure before running the delete command?

24. You want to delete bugfix-footer, but you're unsure if it's been merged. What should you do before deleting it?

25. You finished working on feature-a, feature-b, and feature-c. All have been merged into main. How do you delete them in one command?

Scenario based questions on working with remote repository using Git commands

1. Specify the git command when you're starting work on a project and need to clone a remote repository to your local machine.

2. Specify the git command if you want to see the remotes that are connected to your local repository.

3. Specify the git command if you want to add a new remote repository to your local repository.

4. Specify the git command to remove a remote repository from your local configuration:

5. Specify the git command if you want to rename an existing remote:

6. Specify the git command to fetch updates from the remote repository but not merge them into your local branch.

7. Specify the git command to pull the latest changes from the remote repository and merge them into your local branch.

8. Specify the git command to push your local commits to a remote repository.

9. Specify the git command to when pushing for the first time and want to set the remote branch you are pushing to.

10. Specify the git command to change the URL of a remote (e.g., after changing the remote repository address).

11. Specify the git command if you want to list all branches on the remote repository.

12. Specify the git command if a branch was deleted on the remote but still shows up locally.

13. Specify the git command to fetch a specific branch from a remote.

14. Specify the git command if you want to see detailed information about a remote repository.

15. Specify the git command if you want to rebase your local branch onto a remote branch (this can be useful to keep your history linear)

Conclusion:

Successfully uploading a multi-folder project to both public and private GitHub repositories demonstrates student's practical understanding of **core Git commands** and **remote repository management**. By handling diverse file types (e.g., images, Java files, web pages, and pom.xml) and pushing them to different remote repositories, students show proficiency in:

- Initializing and configuring Git repositories
- Staging, committing, and managing version history
- Connecting to and working with multiple remote repositories
- Handling Gitignore and tracking large files appropriately
- Applying best practices for public vs. private repository usage

Completing these tasks equips students with essential version control skills and the ability to collaborate securely and professionally. It also reflects their readiness to contribute effectively to real-world software projects where source code management, team coordination, and secure code sharing are vital.