LAB ACTION **PLAN FOR WEEK 3**

Objective:

1. To work on collaborative coding by:
   a. Creating Organization.
   b. Coordinating with others through a shared repository
2. To resolve conflicts when collaborating on same part of code.
3. To create and apply patch
4. Students need to make note in observation book and also upload the screenshots of above executed exercise and scenario-based question.

1. Collaborative coding.

Code collaboration refers to the process of multiple people working together on the same codebase.

There are **two ways** of collaborative coding:

a. Creating Organization
b. Coordinating with others through a shared repository

a. Facilitating Collaborative Work by creating Organization

Organization - Organizations are shared accounts where businesses and open-source projects can collaborate across many projects at once. Owners and administrators can manage member access to the organization's data and projects with sophisticated security and administrative features. Steps to facilitate collaborative work by creating organization are:

Step 1: Click on New organization in Git hub

Click on Create a free organization.

Step 2: Set up your organization by entering the details like name, associated email, account verification, etc.

Click on Next

Step 3: Complete the setup by

- ✓ adding members to the group.
- ✓ Now, the invitation goes to other collaborator and they must accept.
- ✓ Next click on complete setup
- ✓ Goto Github and click on

Settings → Member privileges → Base permissions → Select write → change base permission to "Write"

Next under,

Projects base permissions → Select Write → change base permission to "Write"

Step 4: Create the remote repository for storing the project related files in Organization. This repository is accessible to every member of the team as per the permissions given.

Note: The repository can be made private so that it is accessible only to the group members rather than being in a public domain.

Now all the members of the team can contribute to the development of the project and the different files with all the versions and modification notices will be available in the respective repositories and is accessible to all the members

b. To collaborate effectively on GitHub shared repository by sharing code and coordinating with others through a shared repository.

→**Steps for Collaborator 1 are**: (collaborator-1 has repository that he wants to share with collaborator-2)

- ✓ Go to Settings > collaborators
- ✓ Now in Manage Access click on Add people
- ✓ Now you will be able to see ONE collaborator added
- ✓ After this invitation goes to collaborator 2 and they need to accept it.

→**Steps for Collaborator 2 are:**

✓ 1. Set Up Git and GitHub

In git bash
git config --global user.name "Your Name"
git config --global user.email you@example.com

Note : run `$ git remote -v`

```
If origin  git@github.com:OrgYOG/commonREPO.git (fetch)
   origin  git@github.com:OrgYOG/commonREPO.git (push)
then

$ git remote -v // Note now, not connected to any remote
```

✓ 2. Goto Git hub and copy url of shared repository by collaborator 1

- Copy url (option 1)

✓ 3. Clone the Repository

Get a local copy of the repo:
In git bash
git clone https://github.com/username/repository-name.git
cd repository-name

✓ 4. Create a Branch for Your Work

Always work on a separate branch:
In git bash
git checkout -b feature/your-feature-name

✓ 5. Make Changes and Commit

Edit files, then stage and commit your changes:
In git bash to existing file / new file

```
git add .
git commit -m "Add a clear and descriptive commit message"
```

✓ **6. Push Your Branch to GitHub**

- ➔ In git bash
- ➔ git push origin feature/your-feature-name

✓ **7. Keep Your Branch Updated (has to be done by owner collaborator 1)**

Before making new changes:
In git bash

```
git checkout main
git pull origin main
git checkout feature/your-new-feature
git merge main
```

<u>**Exercise on Fork**</u>

- If you're contributing to an existing project: (in public repository)
  Visit the repository page and click Fork

Note: Following steps needed if you forked from any public repository to contribute

1. Go to: `https://github.com/othername/awesome-project`
2. Click the **"Fork" button at the top-right of the page.**
3. GitHub will create a copy of that repository in your account:

https://github.com/your-username/awesome-project

4. Clone your forked repo: in bash

- ✓ **git clone https://github.com/your-username/awesome-project**
- ✓ **Make changes.**
- ✓ **Commit**
- ✓ **git push origin main**

5. Create a **pull request back to the original repo (`othername/awesome-project) to suggest your changes.**

- ✓ **Click at top Pull request**
- ✓ **Add title**
- ✓ **Create pull request**

Now you see on page message “NO conflicts with base branch”

- ✓ **Add comment any thing**
- ✓ **Click on comment**
- ✓ **Submit the pull request for review**

✅ **6. Review and Discuss Changes**

- Collaborators can comment, request changes, or approve the PR(Pull Request)
- Make edits if needed and push again; they will update the PR automatically

✅ **7. Merge the Pull Request (has to be done by both collaborators)**

Once approved:

- The repo owner can click Merge pull request
- Or you can squash and merge, depending on the team's workflow

✅ **8. Keep Your Branch Updated (has to be done by owner collaborator 1)**

Before making new changes:

In git bash

```
git checkout main
git pull origin main
git checkout feature/your-new-feature
git merge main
cd repository-name
```

2. <u>To resolve conflicts when collaborating on same part of code.</u>

Resolving conflicts when collaborating with GitHub (or Git in general) is a normal part of team development. Conflicts happen when two people make changes to the same part of the code, and Git doesn't know which version to keep.

✅ **Steps to Resolve a Conflict**

1. See Which Files Are in Conflict (Say f1.txt)

In git bash

```
git status
```

Conflicted files will be marked like:

In git bash

```
both modified:  (Say f1.txt)
```

---

3. Open the File and Look for Conflict Markers by clicking on

✓ Click on **Resolve Conflict**

Git will insert conflict markers in the file like this:

```
def greet():
<<<<<<< HEAD
   print("Hello from First collaborator ")
=======
   print("Hello from second collaborator ")
>>>>>>> feature/second collaborator branch
```

This shows the two conflicting changes:

- HEAD is your version
- The section below ======= is the other branch's version or second collaborator branch version

---

3. Edit the File to Fix the Conflict

- ✓ Choose one version or combine them:

```
def greet():
    print("Hello from First collaborator ")
    print("Hello from second collaborator ")
```

- ✓ Then **delete** all the **conflict markers** (<<<<<<<, =======, >>>>>>>).

---

4. Mark the Conflict as Resolved

Once you've edited and saved the file:

- ✓ git add f1.txt

---

5. Complete the Merge or Pull

- ✓ git commit   # Only needed if you're merging

Or if you're rebasing:

- ✓ git rebase --continue

⚠️ If You Want to Abort the Merge

If it gets messy and you want to cancel:
git merge --abort

Here below screen shot shows conflict raised when collaborator 1 wanted to merge changes to same line that already collaborator 2 added the line of code at that line in same file but in their own branch.

Example :

Merge Conflict – Changes made to a file by two different users leads to merge conflict as below.

Say I am collaborator 2, now

- ➔ **Accept invitation from collaborator 1**
- ➔ **Next goes to Shared repository by collaborator 1 (say abcrepo)**
- ➔ **Now open Git bash and clone the shared repo as below steps**

Step 1: first connect to this “abcrepo” to local repository by clone

- ✓ Now change directory to “abcrepo”
- ✓ Switch to branch feature-abc
- ✓ Now add line in README.md file
- ✓ Now git add, commit
- ✓ Now see status and push to remote branch feature-abc
- ✓ Now go to Github and refresh or click on “abcrepo”, you can see README.md

Step 2: Now collaborator 2 adding line in README.md, save, git add, commit, push. No conflicts

Step 3: Also collaborator 1 adding line in README.md at same line collaborator 2 added, then git add, commit, push gives **merge conflict.**

NOTE IMPORTANT :

Now collaborator 2 has to give git pull command then open conflicting file to see conflict markers.

Now Edit the File to Fix the Conflict by keeping either or both collaborator changes and remove conflict markers. Add file, commit, and push.

3. To create and apply patch

A patch in Git is a text file that represents changes between two sets of files, or commits. It's essentially the output of the git diff command, packaged in a format that can be applied to another set of files.

Steps:

1. Goto github and take public project url

2. Clone into git bash

3. Now open file and edit it, add, commit

4. Run git log

5. Create patch with commit hash code as:

   ```
   git format-patch -1 commit-hash-code
   ```

   This creates 0001-commit-from-cloned-person.patch file.

Note: git format-patch -1 <commit-hash>

- -1 means "create a patch from 1 commit"
- This creates a .patch file for a single commit.
- Example:

  ```
  git format-patch -1 abc1234
  ```
- This creates a file like 0001-Your-commit-message.patch

To create patches for the **last 3 commits**: git format-patch -3

or

Create patches from multiple commits

```
git format-patch <base_commit>..HEAD
```

6. Get the path of above patch file and email or what's up to the remote owner of file.

7. Once the remote owner gets the patch file 0001-commit-from-cloned-person.patch next the owner need to apply patch file in his git bash using git command below:

Syntax:

```
git apply my-changes.patch
```

Or, if it's a patch made by git format-patch, use:

```
git am 0001-Your-commit-message.patch
```

- ✓ **git apply 0001-Your-commit-message.patch**

4. <u>Scenario based questions on working with remote repository</u>

1. You've cloned a repository and made some changes to a local branch. Now you want to push these changes to the remote repository, but you're getting an error saying "rejected - non-fast-forward." How would you resolve this?

2. You’ve been working on a feature branch, and now you need to push it to the remote repository. However, the remote repository already has a main branch. How do you push your feature branch without affecting the main branch?

3. You cloned a remote repository, but after a while, the repository’s structure changed and new branches were added. How would you keep your local repository updated with the latest changes from the remote repository?

4. A colleague has pushed some changes to the main branch, but you have local changes in the same branch. You want to pull their changes, but you want to avoid merge conflicts. What steps would you take?

5. You accidentally pushed a sensitive file (e.g., API keys) to the remote repository. How would you fix this situation?

6. You’re working on a feature branch, and your manager requests that you integrate the latest changes from main into your feature branch. What steps would you take?

7. You cloned a remote repository, but later you find that you need to push your changes to a different remote repository. How do you configure your local repository to push to this new remote?

8. After running git pull, you notice that your local branch is behind the remote branch. How would you proceed to bring your local branch up to date without losing your local changes?

9. You’re working on a project with multiple collaborators, and you notice that your local changes conflict with changes that have been pushed by others. How would you resolve the conflicts?

10. You’ve pushed a feature branch to a remote repository, but now you need to delete the branch from the remote. How would you do that?

Scenario:

You are working on a collaborative project hosted on GitHub with a team of four developers. The main branch is main, and each developer works on their own feature branches.

You are assigned to implement a new UI component in a branch called feature/ui-update. Meanwhile, another team member has made a critical bug fix and pushed it directly to the main branch. When you try to push your changes, your push fails due to upstream changes.

Additionally, a teammate has submitted a patch file with a small CSS fix and shared it via email. You need to apply the patch, test it, and merge everything into main without breaking any functionality.

11. What command will you use to bring your local main branch up to date with the remote repository before merging it into your feature branch?

12. How will you update your feature/ui-update branch to reflect the latest changes from main?

13. Which command should you use to attempt pushing your local feature branch again, and what should you do if it fails due to conflicts?

14. How do you apply **a .patch file** provided by your teammate and include it in your commit history?

15. After successful testing, describe the steps (with commands) to merge your feature branch into main and push it to GitHub.

Conclusion:

The effective use of Git in student collaborative projects promotes **teamwork, organization, and clear communication**, allowing students to manage code efficiently and track contributions accurately. By following structured Git workflows and best practices—such as regular commits, feature branching, and resolving conflicts early—students gain valuable experience in real-world development processes. This not only enhances their technical skills but also prepares them for **professional teamwork in software engineering**, where version control and collaboration are essential. Ultimately, mastering Git empowers students to deliver cleaner, more reliable code and succeed in collaborative environments.