**FSD Laboratory 01**

**Aim:** Version control with Git.

**Objectives:**

1. To introduce the concepts and software behind version control, using the example of Git.
2. To understand the use of 'version control' in the context of a coding project.
3. To learn Git version control with Clone, commit to, and push, pull from a git repository.

**Theory:**

**1.** **What is Git? What is Version Control?**

**Git** is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It is free and open-source software created by Linus Torvalds in 2005.

**Key Features of Git:**

1. **Distributed Version Control:** Every developer has a full copy of the repository, including its complete history.
2. **Speed:** Git is fast in terms of handling large projects.
3. **Branching and Merging:** Git supports easy branching and merging, allowing for multiple development workflows.
4. **Data Integrity:** Git ensures the integrity of the source code with checksums.
5. **Staging Area:** Git has a staging area, which allows you to prepare your commits before completing them.

**Version Control** is a system that records changes to a file or set of files over time so that you can recall specific versions later. It helps manage the evolution of your source code or documents, track and manage changes, collaborate with others, and revert back to previous versions if necessary.

**Key Types of Version Control Systems:**

1. **Local Version Control Systems:** Maintain the version history on your local machine. Example: RCS (Revision Control System).
2. **Centralized Version Control Systems (CVCS):** Have a single central server that stores all the versioned files, and a number of clients that check out files from that central place. Example: Subversion (SVN).
3. **Distributed Version Control Systems (DVCS):** Every developer has a complete copy of the entire project history. Examples: Git, Mercurial.

**Benefits of Using Version Control:**

1. **Collaboration:** Multiple developers can work on the same project simultaneously without overwriting each other's work.
2. **History:** You can track and understand changes over time, which helps in debugging and accountability.
3. **Backup:** The version control system serves as a backup mechanism.
4. **Branching and Merging:** Developers can work on features/bug fixes in isolated branches and later merge them into the main codebase.
5. **Revert:** You can revert back to previous versions if something goes wrong in the latest changes.
6. **Tracking Changes:** Easily see what changes were made, who made them, and when they were made.

Git, being a DVCS, combines the benefits of version control with additional features such as local repositories, which allow developers to work offline and commit changes independently before syncing with a remote repository.

**2.** **How to use Git for version controlling?**

Using Git for version control involves a series of steps and commands that help manage your project’s codebase. Here is a basic guide to get you started with Git:

### **1. Install Git**

First, you need to install Git on your computer. You can download it from [Git's official website](https://git-scm.com/downloads" \t "_new).

### **2. Configure Git**

Set up your Git configuration with your name and email address. This information will be associated with your commits.

sh

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

git config --global user.email [your.email@example.com](mailto:your.email@example.com)

### **3. Initialize a Repository**

Navigate to your project directory and initialize a new Git repository.

sh

cd /path/to/your/project

git init

### **4. Add Files**

Add the files you want to track to the repository.

sh

git add filename1 filename2

To add all files in the current directory, use:

sh

git add .

### **5. Commit Changes**

Commit the added files to the repository with a descriptive message.

sh

git commit -m "Initial commit"

### **6. Check the Status**

You can check the status of your repository to see which files are staged, untracked, or modified.

sh

git status

### **7.** **View Commit History**

To view the commit history, use:

sh

git log

### **8. Branching**

Create a new branch to develop a new feature.

sh

git branch new-feature

Switch to the new branch:

sh

git checkout new-feature

### **9. Merging**

Merge changes from the feature branch back into the main branch. First, switch to the main branch:

sh

git checkout main

Then merge the changes:

sh

git merge new-feature

### **10. Pushing to Remote Repository**

If you are working with a remote repository (e.g., on GitHub), add the remote URL.

sh

git remote add origin <https://github.com/yourusername/yourrepository.git>

Push your local commits to the remote repository.

sh

git push origin main

### **11. Pulling Changes**

To update your local repository with changes from the remote repository, use:

sh

git pull origin main

### **Common Git Commands:**

* git clone [url]: Clone a remote repository to your local machine.
* git diff: Show changes between commits, commit and working tree, etc.
* git reset [file]: Unstage a file.
* git stash: Temporarily store changes you’ve made to your working directory.
* git rebase: Apply commits from one branch onto another.

### **Example Workflow**

1. **Initialize and Add Files**:

sh

git init

git add .

git commit -m "Initial commit"

1. **Create and Switch to a New Branch**:

sh

git branch feature-branch

git checkout feature-branch

1. **Make Changes and Commit**:

sh

# make some changes to your files

git add modified\_file

git commit -m "Added new feature"

1. **Merge Changes**:

sh

git checkout main

git merge feature-branch

1. **Push to Remote Repository**:

sh

git remote add origin https://github.com/yourusername/yourrepository.git

git push origin main

**FAQ:**

**1.** **What is branching in Git?**

**Branching** in Git is a way to create separate lines of development within a repository. It allows you to diverge from the main line of development and continue to work independently without affecting the main codebase. Branches are essential for managing new features, bug fixes, and experiments in a clean and organized manner.

**Key Concepts of Branching**

1. **Branch**: A branch is simply a pointer to a specific commit in the repository. The default branch in a Git repository is typically named main or master.
2. **HEAD**: This is a pointer that points to the current branch reference. It indicates the commit you are currently working on.
3. **Branching Workflow**:
   * Create a new branch for a specific feature or bug fix.
   * Work on the new branch, committing changes as needed.
   * Merge the branch back into the main branch once the work is complete and tested.

**Benefits of Branching**

1. **Isolation**: Each branch can isolate its changes, preventing conflicts with the main codebase or other branches.
2. **Parallel Development**: Multiple developers can work on different branches simultaneously without interfering with each other’s work.
3. **Version Control**: You can easily switch between different versions of your project.
4. **Code Review and Testing**: Changes can be reviewed and tested in isolation before being merged into the main branch.

**Basic Branching Commands**

Here are some common commands used for branching in Git:

**1. Create a New Branch**

bash

git branch new-branch

This command creates a new branch named new-branch but does not switch to it.

**2. Switch to a Branch**

bash

git checkout new-branch

This command switches to the branch named new-branch.

**3. Create and Switch to a New Branch in One Command**

bash

git checkout -b new-branch

This command creates a new branch named new-branch and switches to it immediately.

**4. List All Branches**

bash

git branch

This command lists all branches in the repository, with an asterisk (\*) next to the currently checked-out branch.

**5. Merge a Branch into the Current Branch**

bash

git merge new-branch

This command merges the changes from new-branch into the branch you are currently on.

**6. Delete a Branch**

bash

Copy code

git branch -d new-branch

This command deletes the branch named new-branch. Use -D instead of -d if you want to force delete a branch that hasn’t been merged.

**Example Workflow**

1. **Create a New Branch for a Feature**

bash

git checkout -b feature-branch

1. **Make Changes and Commit**

bash

git add .

git commit -m "Add new feature"

1. **Switch Back to the Main Branch**

bash

git checkout main

1. **Merge the Feature Branch into Main**

bash

git merge feature-branch

1. **Delete the Feature Branch**

bash

git branch -d feature-branch

By using branches, you can keep your development process organized and ensure that new features, bug fixes, and experiments can be developed and tested in isolation before being integrated into the main codebase.

**2.** **How to create and merge branches in Git? Write the commands used.**

Creating and merging branches in Git is a fundamental aspect of managing different lines of development. Here are the steps and commands you need to follow:

**Creating a Branch**

1. **Create a New Branch**

bash

git branch new-branch

This command creates a new branch called new-branch but does not switch to it.

1. **Create and Switch to a New Branch**

bash

git checkout -b new-branch

This command creates a new branch named new-branch and immediately switches to it.

**Switching Between Branches**

1. **Switch to an Existing Branch**

bash

git checkout branch-name

This command switches to the branch named branch-name.

**Viewing Branches**

1. **List All Branches**

bash

git branch

This command lists all the branches in the repository, with the current branch indicated by an asterisk (\*) next to its name.

**Merging Branches**

1. **Merge a Branch into the Current Branch** First, ensure you are on the branch you want to merge into (e.g., main).

bash

git checkout main

Then, merge the changes from new-branch into main.

bash

git merge new-branch

**Deleting Branches**

1. **Delete a Branch**

bash

git branch -d new-branch

This command deletes the branch named new-branch. Use -D instead of -d if you want to force delete a branch that hasn’t been merged.

**Example Workflow**

1. **Create and Switch to a New Branch**

bash

git checkout -b feature-branch

1. **Make Changes and Commit**

bash

git add .

git commit -m "Add new feature"

1. **Switch Back to the Main Branch**

bash

git checkout main

1. **Merge the Feature Branch into Main**

bash

git merge feature-branch

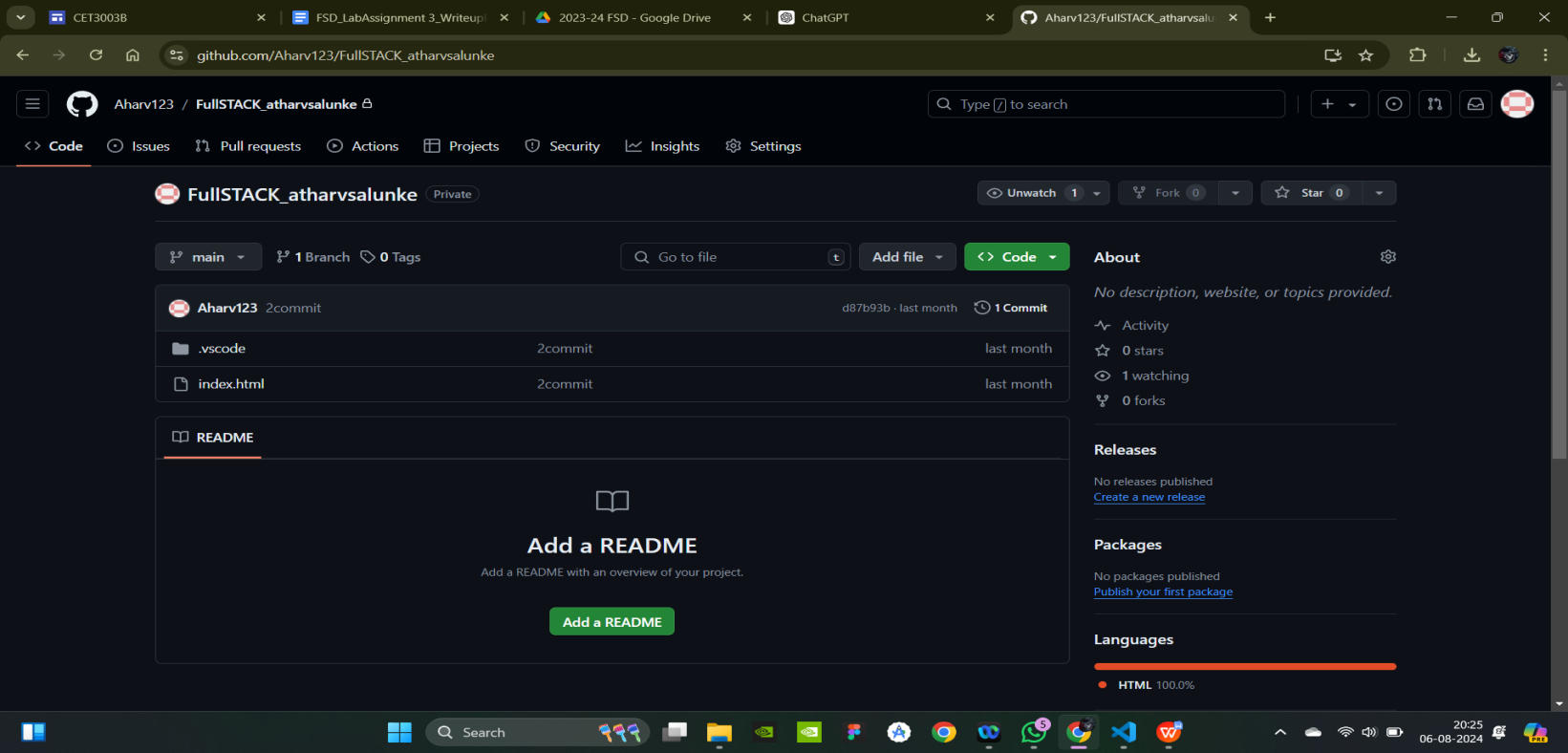
1. **Delete the Feature Branch**

bash

git branch -d feature-branch

By following these steps and using these commands, you can effectively manage branches in your Git repository, allowing for isolated development and easier integration of changes.

**Output:** Screenshots of the output to be attached.



**Problem Statement:**

Create a public git repository for your team and submit the repo URL as a solution to this assignment, Learn Git concept of Local and Remote Repository, Push, Pull, Merge and Branch.