**FSD Laboratory 01**

**Name: Shabbar Adamjee**

**Roll No: PB57**

**PRN: 1032221508**

**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 stands out as the most widely adopted version control system worldwide. It originated in 2005 through the efforts of Linus Torvalds and continues to be actively maintained as an open-source project. Git facilitates collaborative software development by efficiently tracking changes in code. It supports non-linear development through numerous parallel branches, enabling multiple developers to work concurrently.

Version control is a fundamental practice in software engineering, managing file versions, particularly source code files, but applicable to any file type. Each contributor typically works on their own branch, and changes are integrated into the main source code after thorough review and approval. This approach not only maintains code organization but also enhances productivity by streamlining the development process.

In essence, a version control system monitors changes made to software and captures snapshots of every modification. For example, if new functionalities added by a team of developers cause issues, the version control system allows reverting to a previous version, ensuring project stability and continuity.

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

The core functionality of Git revolves around three primary states for files: modified, staged, and committed. When including a remote repository, Git's functionality can be categorized into four states. Your files can exist in any of these states at a given time.

1. **Modified State**:

Files in this state have been altered but not yet saved in Git. They are considered uncommitted changes, representing ongoing work that is not permanently saved. Modifications can continue to be made to these files until they are staged for commit.

2. **Staged State**:

Files in the staged state are ready to be committed to the ".git" repository. Although not yet committed, Git has been instructed to monitor these specific versions of the files. Even after staging, files can still be modified, and these changes can be added to the staging area again.

3. **Committed State**:

The committed state signifies that files have been successfully saved in the ".git" repository by creating a commit. At this stage, the staged version of the files is permanently recorded in the Git directory. Each commit is accompanied by a commit message detailing the changes made or features added to the code.

**FAQ:**

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

Branching involves creating a separate path of development that diverges from the mainline, allowing work to progress independently without affecting the primary codebase. Virtually all Version Control Systems (VCS) include support for branching. In Git, a branch represents an alternate version of the main repository.

When you commit changes in Git, a commit object is created, which includes a pointer to the snapshot of the staged content. Typically, the default branch in a Git repository is named "main" or "master."

Branches in Git provide a way to abstract the edit/stage/commit process. They function as a mechanism to request a distinct working directory, staging area, and project history. This separation enables developers to keep different versions of their code neatly isolated. Branches are invaluable for collaborative team efforts, allowing multiple contributors to work on different aspects of a project concurrently. They are also beneficial for individual developers when starting work on a new feature or experimenting with changes.

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

Creating a branch in git involves mainly two options:

* + git branch <branch-name> – This simply creates a new branch with name specified in the <branch-name> field.
  + git checkout -b <branch-name>– Whereas this option creates a new branch and checkouts automatically to the new branch in the workspace.

When a feature is complete or a bug fix is ready, you can merge the branch back into the main branch. To merge two branches in Git, following commands are used:

* + git checkout <main-branch>
  + git merge <branch-name>

**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.

**Output**:

