**Implementing Basic Version Control with Git**

Project Overview

For this demonstration, we will create a basic project that contains a simple HTML file and a JavaScript file. The purpose is to set up a Git repository, track changes, and manage versions using Git commands.

Project Structure

my-project/

index.html

script.js

**1. Setting Up a Git Repository**

Initialize the Repository

Navigate to the project directory and initialize a Git repository:

git init

Rationale: Initializing a Git repository sets up the necessary structures and configurations to start tracking changes in the project.

**2. Tracking Changes**

Create Initial Files

Create index.html and script.js files with the following content:

index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>My Project</title>

</head>

<body>

<h1>Welcome to My Project</h1>

<script src="script.js"></script>

</body>

</html>

script.js:

console.log('Hello, World!');

Stage and Commit the Initial Version

Add the files to the staging area and commit them:

git add index.html script.js

git commit -m "Initial commit with basic HTML and JavaScript files"

Rationale: Staging and committing the initial files creates the first version of the project in the repository, allowing us to track changes from this point forward.

**3. Managing Versions**

Make Changes to Files

Modify script.js to add a new function:

script.js (after modification):

console.log('Hello, World!');

function greet(name) {

console.log(`Hello, ${name}!`);

}

greet('Alice');

Stage and Commit the Changes

Add the modified file to the staging area and commit the changes:

git add script.js

git commit -m "Added greet function to script.js"

Rationale: Committing changes ensures that every significant modification is recorded, creating a history that can be reviewed or reverted if necessary.

**4. Viewing the Commit History**

Check the commit history to see all changes made:

git log

Rationale: Viewing the commit history allows developers to understand the progression of the project, see who made what changes, and when.

**5. Creating and Switching Branches**

Create a New Branch

Create a new branch called feature-update:

git branch feature-update

Switch to the New Branch

Switch to the feature-update branch:

git checkout feature-update

Make Changes in the New Branch

Modify index.html to add a paragraph:

index.html (after modification):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>My Project</title>

</head>

<body>

<h1>Welcome to My Project</h1>

<p>This is a new paragraph added in the feature-update branch.</p>

<script src="script.js"></script>

</body>

</html>

Stage and Commit the Changes in the New Branch

Add the modified file to the staging area and commit the changes:

git add index.html

git commit -m "Added a new paragraph in index.html"

Rationale: Branching allows developers to work on new features or bug fixes independently without affecting the main codebase. Changes can be merged back into the main branch once they're complete and tested.

**6. Merging Changes**

Switch Back to the Main Branch

Switch back to the main branch (or master):

git checkout main

Merge the Feature Branch into the Main Branch

Merge changes from feature-update into main:

git merge feature-update

Rationale: Merging incorporates changes from different branches, enabling collaborative development where multiple features are developed in parallel and then integrated.

**7. Collaboration with Remote Repositories**

Add a Remote Repository

Add a remote repository (e.g., GitHub):

git remote add origin https://github.com/username/my-project.git

Push Changes to the Remote Repository

Push the commits to the remote repository:

git push -u origin main

Rationale: Remote repositories facilitate collaboration by allowing multiple developers to push their changes to a central repository, enabling distributed version control.

**Benefits of Version Control in Project Management and Collaboration**

History Tracking: Maintains a detailed record of all changes, making it easy to revert to previous states.

Collaboration: Enables multiple developers to work on the same project simultaneously without conflicts.

Branching and Merging: Supports parallel development and seamless integration of new features.

Backup and Restore: Acts as a backup system, storing the entire project history.

Code Review: Facilitates code reviews and ensures high-quality code through pull requests and merge checks.



