MIT WORLD PEACE UNIVERSITY

Full Stack Development
Third Year B.Tech, Semester 1

Version control with Git

ASSIGNMENT 1

Prepared By A1−02. Mayur Behere
Batch A1

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 &#39;version control&#39; 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 (VCS) that is widely used in software development to track changes in source code and collaborate on software projects. It was created by Linus Torvalds in 2005 and has since become one of the most popular version control systems in the world.

Version control, in general, is a system that allows multiple people (usually software developers) to work on the same project simultaneously. It helps track changes to files over time, provides a history of modifications, and enables collaboration among team members. Version control systems are crucial for managing and organizing code in software development projects.

Git specifically offers the following features:

1. **Tracking Changes**: Git keeps a record of every change made to a project's files, allowing you to see what was added, modified, or deleted at any point in time.

2. **Branching**: Git allows you to create branches, which are independent lines of development. Developers can work on different features or bug fixes in separate branches without affecting the main codebase. Branches can later be merged back into the main code when the work is complete.

3. **Collaboration**: Git enables collaboration by allowing multiple developers to work on the same project simultaneously. It provides tools for sharing and merging changes made by different team members.

4. **History and Revert**: Git maintains a complete history of changes, making it easy to go back to a previous state of the code if needed. This is essential for troubleshooting and rolling back changes that introduce problems.

5. **Distributed**: Git is a distributed version control system, meaning that each developer has a complete copy of the project's history on their local machine. This decentralization makes it resilient and suitable for both small and large development teams.

6. **Open Source**: Git is an open-source project, which means it is freely available for anyone to use and modify. It has a vast and active community of users and contributors.

In summary, Git is a powerful tool for managing and tracking changes in software projects, making it easier for teams to collaborate, maintain code, and ensure the integrity of their software over time.

2. How to use Git for version controlling?

Using Git for version control involves several key steps and commands. Here's a step-by-step guide on how to use Git for version control:

1. **Install Git:**
   - If Git is not already installed on your system, you can download and install it from the official Git website (https://git-scm.com/).

2. **Configure Git:**
   - Before you start using Git, you should configure your name and email address to associate with your commits. Use the following commands to set your Git identity:
```
  git config --global user.name "Your Name"
  git config --global user.email "your.email@example.com"
```

3. **Initialize a Git Repository:**
   - To start version controlling a project, navigate to the project's root directory in your terminal and run the following command to initialize a Git repository:
```
  git init
```

4. **Add Files to the Repository:**
   - You can add files to the staging area using the `git add` command. For example, to add all files in the current directory and its subdirectories, use:
```
  git add .
```

5. **Commit Changes:**
   - Once your changes are staged, you can commit them with a descriptive message using the `git commit` command:

```
  git commit -m "Your commit message here"
  ```

6. **Create Branches (Optional):**
  - If you're working on a new feature or bug fix, it's a good practice to create a new branch for your work. Use the following commands to create and switch to a new branch:
  ```
  git checkout -b new-feature-branch
  ```

7. **Work on Your Code:**
  - Make changes to your code within your branch.

8. **Stage and Commit Changes:**
  - As you make changes, continue to use `git add` and `git commit` to save your progress with meaningful commit messages.

9. **Merge Branches (Optional):**
  - If you've created a feature branch and your work is complete, you can merge it back into the main branch (e.g., `main` or `master`) using the following commands:
  ```
  git checkout main
  git merge new-feature-branch
  ```

10. **Push to a Remote Repository (Optional):**
   - If you're collaborating with others or using a remote Git hosting service (e.g., GitHub, GitLab), you can push your changes to the remote repository:
   ```
   git push origin main
   ```

11. **Pull Changes from a Remote Repository (Optional):**
   - To update your local repository with changes made by others, use the `git pull` command:
   ```
   git pull origin main
   ```

12. **Review History:**
   - You can view the commit history and changes using `git log`:
   ```
   git log
   ```

13. **Tagging (Optional):**
   - You can create tags to mark specific points in your project's history, such as release versions:
   ```
   git tag -a v1.0 -m "Version 1.0"
   ```

These are the essential steps and commands for using Git for version control. Git provides powerful tools for tracking changes, collaborating with others, and maintaining a history of your project's development. Depending on your project's complexity and team workflow, you may explore more advanced Git features and workflows.

FAQ’s:

1. What is branching in Git?
Branching in Git is a fundamental concept that allows developers to work on separate lines of development within the same repository. Each branch represents an independent path of changes and is often used to develop new features, fix bugs, or experiment with code changes without affecting the main codebase. Branches provide isolation and allow developers to work on multiple tasks simultaneously. Git's branching system is lightweight and efficient, making it a core feature for collaborative and organized software development.

2. How to create and merge branches in Git? Write the commands used.
Creating and merging branches in Git involves a series of commands. Here are the steps with the corresponding Git commands:
**To create a new branch:**
Use the `git checkout -b` command to create and switch to a new branch simultaneously. Replace `branch-name` with the desired branch name:
```bash
git checkout -b branch-name
```
This command creates a new branch and switches to it, allowing you to start making changes within the new branch.

**To switch between branches:**
Use the `git checkout` command followed by the branch name to switch between branches:
```bash
git checkout branch-name
```
This command allows you to move between different branches in your Git repository.

**To list all branches:**

To see a list of all branches in your repository and see which branch you are currently on, use:

```bash
git branch
```

The branch with an asterisk (`*`) next to it is the currently checked out branch.

To merge a branch into another branch:

To merge changes from one branch into another (e.g., merging a feature branch into the main branch), follow these steps:

1. First, ensure you are on the branch you want to merge changes into. For example, to merge a feature branch into the main branch:

```bash
git checkout main
```

2. Then, use the `git merge` command followed by the branch name you want to merge:

```bash
git merge branch-name
```

This command merges the changes from `branch-name` into the currently checked out branch (in this case, `main`).

3. If there are no conflicts between the branches, Git will perform an automatic merge. If there are conflicts, you will need to manually resolve them and commit the changes.

To delete a branch (after merging):

Once you've merged a branch and no longer need it, you can delete it using the `-d` option:

```bash
git branch -d branch-name
```

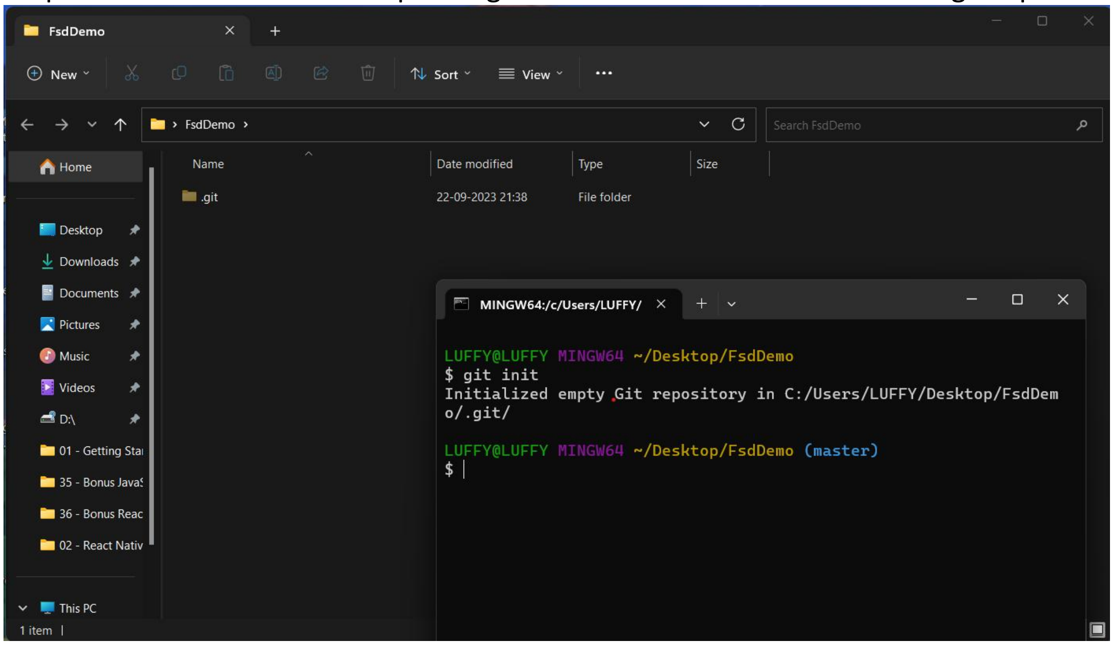
Or, if you want to forcefully delete the branch without checking if changes are unmerged (use with caution):

```bash
git branch -D branch-name
```

These commands cover the basics of creating, switching between, merging, and deleting branches in Git. Branching is a powerful feature that helps organize and manage development workflows effectively.

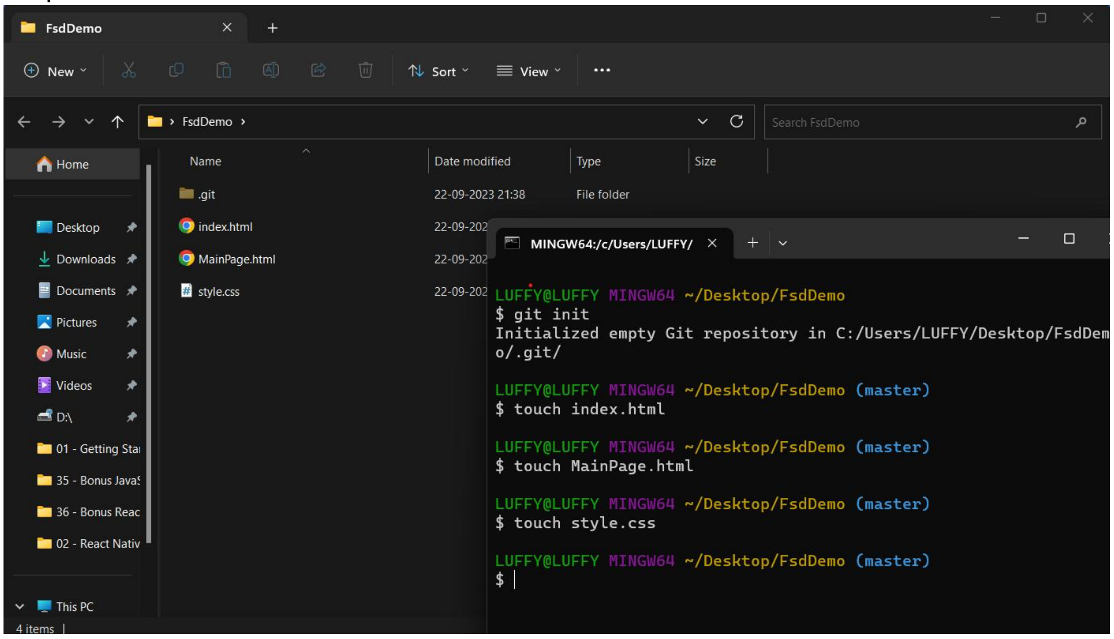
Output:

Step 1: Created a Folder and opened git bash in the folder and initialized git repo init.



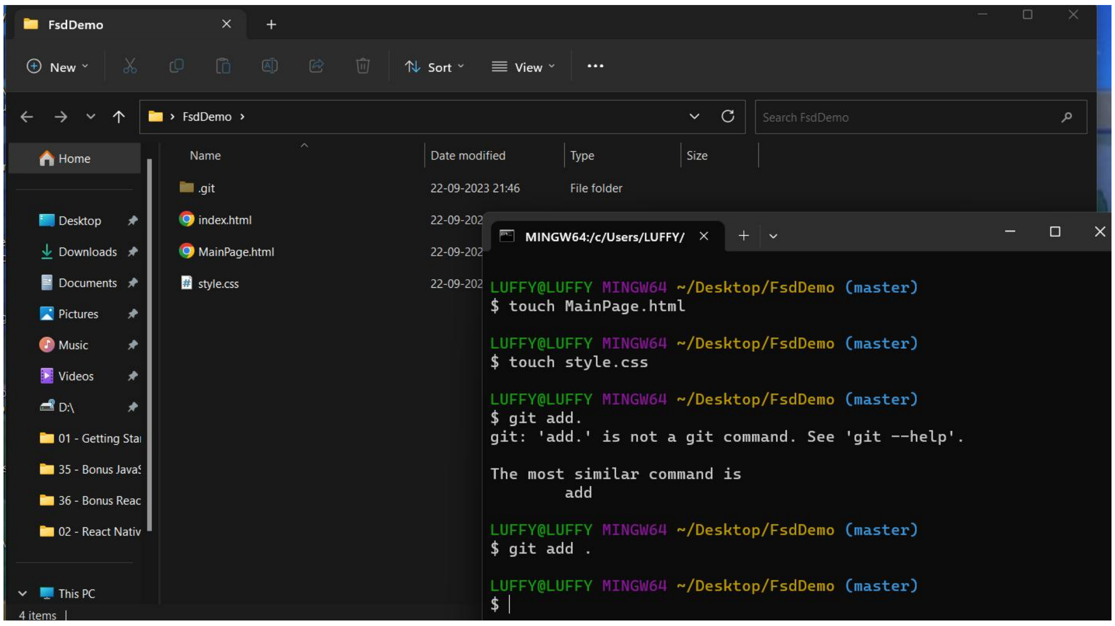
```
LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo
$ git init
Initialized empty Git repository in C:/Users/LUFFY/Desktop/FsdDem
o/.git/

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ |
```

Step 2 : Created 3 files into the folder



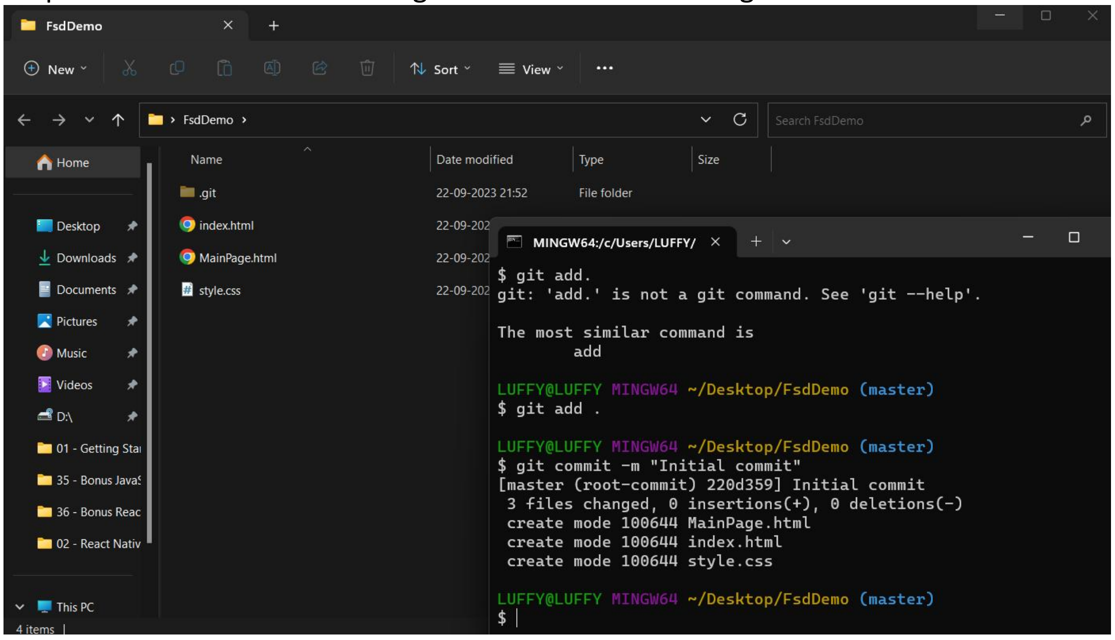
```
LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo
$ git init
Initialized empty Git repository in C:/Users/LUFFY/Desktop/FsdDem
o/.git/

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ touch index.html

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ touch MainPage.html

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ touch style.css

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ |
```

Step 3 : Added all the files to the Staging State.



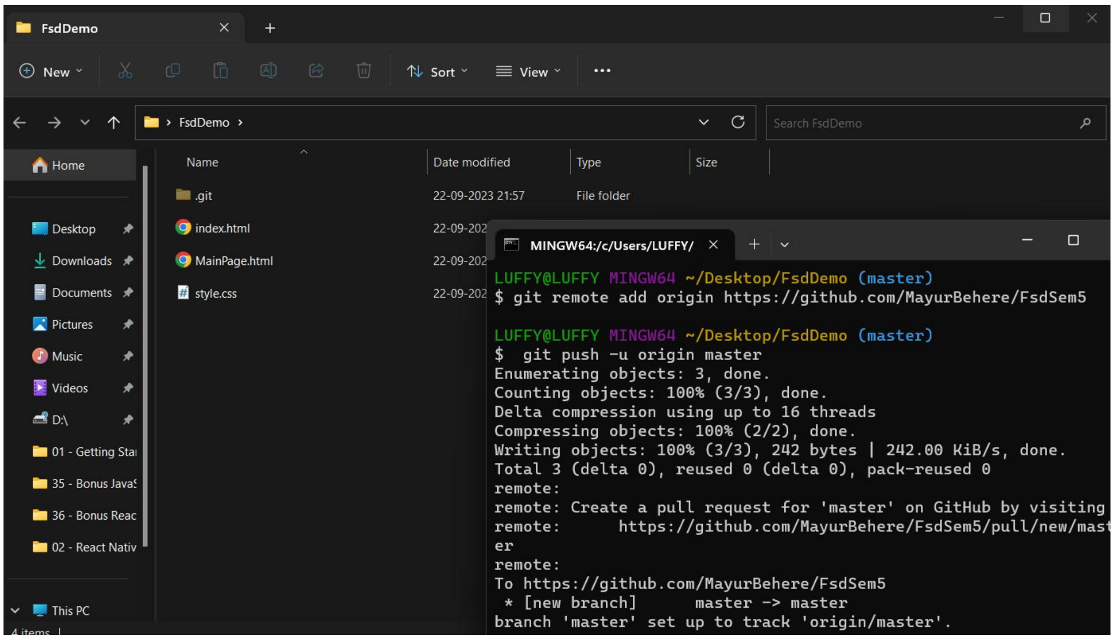
```
LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ touch MainPage.html

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ touch style.css

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
        add

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ git add .

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ |
```

Step 4 : Committed all the changes with a commit message.



```
$ git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
        add

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ git add .

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ git commit -m "Initial commit"
[master (root-commit) 220d359] Initial commit
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 MainPage.html
 create mode 100644 index.html
 create mode 100644 style.css

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ |
```

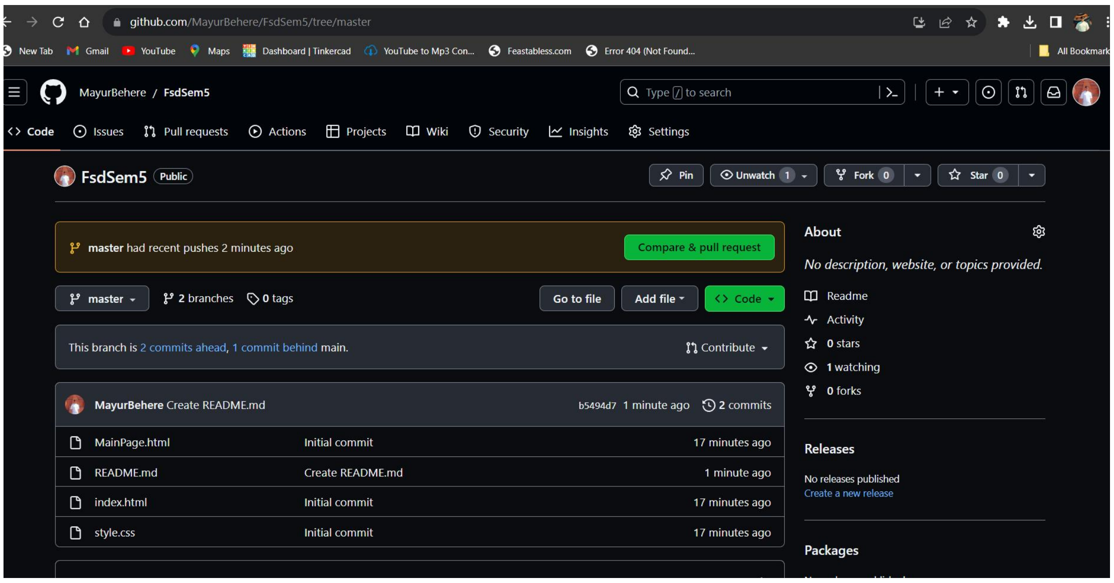
Step 5 : Linked local Repo to the Github Repo and Pushed the committed items into master branch of Github Repo.



Name	Date modified	Type	Size
.git	22-09-2023 21:57	File folder	
index.html	22-09-202		
MainPage.html	22-09-202		
style.css	22-09-202		

MINGW64:/c/Users/LUFFY/

```
LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$ git remote add origin https://github.com/MayurBehere/FsdSem5

LUFFY@LUFFY MINGW64 ~/Desktop/FsdDemo (master)
$  git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 242 bytes | 242.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'master' on GitHub by visiting
remote:      https://github.com/MayurBehere/FsdSem5/pull/new/mast
er
remote:
To https://github.com/MayurBehere/FsdSem5
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```



master had recent pushes 2 minutes ago

This branch is 2 commits ahead, 1 commit behind main.

MayurBehere Create README.md b5494d7 1 minute ago 2 commits

MainPage.html	Initial commit	17 minutes ago
README.md	Create README.md	1 minute ago
index.html	Initial commit	17 minutes ago
style.css	Initial commit	17 minutes ago

No description, website, or topics provided.

No releases published
Create a new release

Link For the Repo Created : https://github.com/MayurBehere/FsdSem5/tree/master