* **What is Git?**

Git is an open-source distributed version control system (DVCS) that allows developers to track and manage changes to their codebase. You can easily manage small as well as large projects with high speed and efficiency by Git. Unlike traditional version control systems, Git allows multiple developers to work on a project simultaneously without interfering with each other's work. We can use Git privately as well as publically.

* **Why Use Git?**

Git offers numerous benefits to developers and development teams:

1. **Version Control**: Git helps in tracking changes, allowing you to go back to previous states if something goes wrong.
2. **Collaboration**: It enables multiple developers to work on a project simultaneously without interfering with each other’s work.
3. **Backup**: Your entire project history is saved in a Git repository, providing a backup of all versions.
4. **Branching and Merging**: Git’s branching model allows you to experiment with new features or bug fixes independently from the main project.
5. **Open Source Projects**: Most open source projects use Git for version control. Learning Git allows you to contribute to these projects.
6. **Industry Standard**: Git is widely used in the software industry, making it an essential skill for developers.

* **Working with Git**

The Git workflow involves the following steps:

1. **Initializing a Repository**: When you initialize a folder with Git, it becomes a repository. Git logs all changes made to a hidden folder within that repository.
2. **Staging Changes**: Git marks modified files as “staged.” Staging prepares changes for a snapshot you want to keep.
3. **Committing Changes**: Once staged changes are satisfactory, commit them. Git maintains a complete record of each commit.
4. **Create and switch branches:**Use "git branch" to create a new branch, and "git checkout" to switch to a different branch.
5. **Merge branches:**Use "git merge" to merge changes from one branch into another. This combines the commits from both branches into a single branch.
6. **Push and pull changes:** Use "git push" to upload your local commits to a remote repository, and "git pull" to fetch and integrate remote changes into your local repository

* **What is Github?**

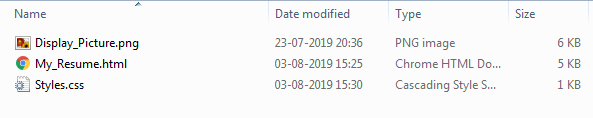
GitHub, a hosting service for Git repositories, allows you to access and download projects from any computer. Here’s what you can do with GitHub:

1. **Store Repositories**: GitHub hosts your repositories.
2. **Collaborate**: Work with other developers from any location.
3. **Version Control**: Manage collaborative workflows using Git and GitHub.

**Relation between Git and GitHub**

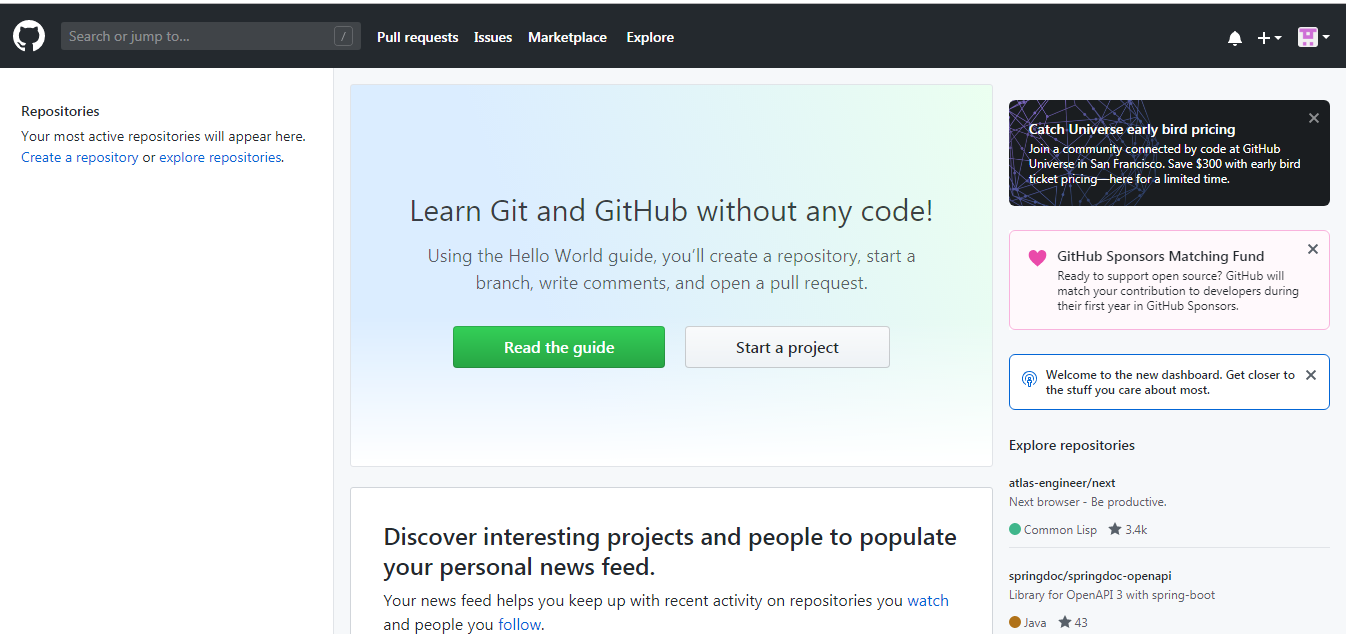
There is a common misconception that Git and GitHub are the same. Git is used to store the source code for a project and track the complete history of all changes to that code, while GitHub is a cloud-based platform built around the Git tool. The major difference is that Git is software that a developer can locally install on a machine to manage source code while GitHub is an online service to which developers who use Git can connect and upload or download resources. GitHub is a **hosting service** for **git repositories**.

**Below are the sample project files which we want to upload:**



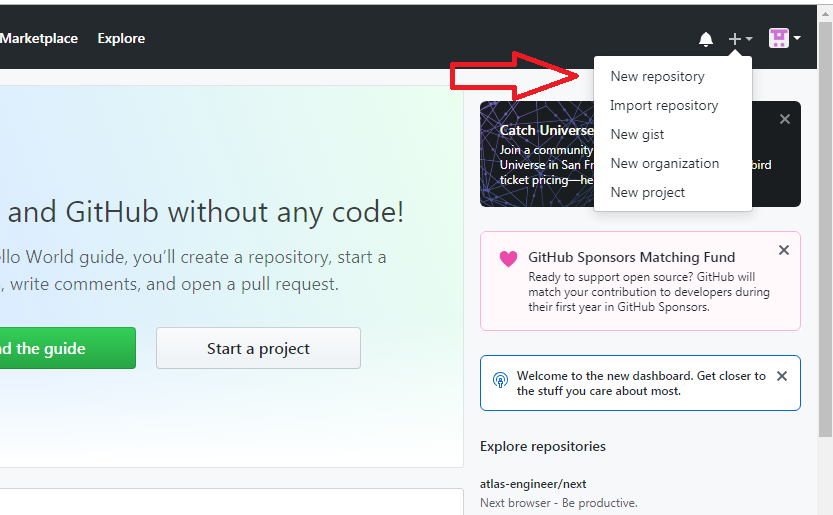
* **Steps to Create the Repository**

**Step 1:** After successfully setting up GitHub account login to your account. You will see the screen as below.

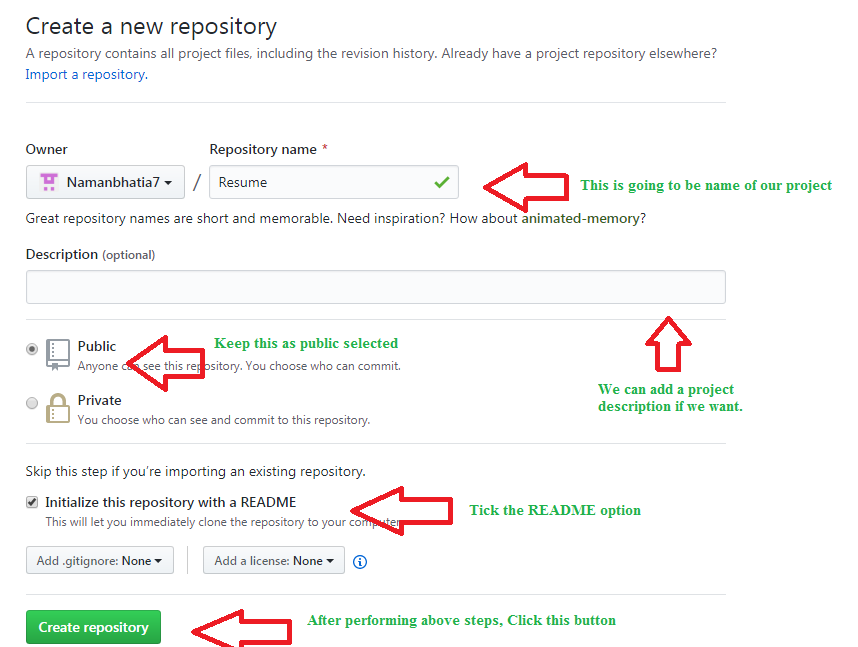


**/**

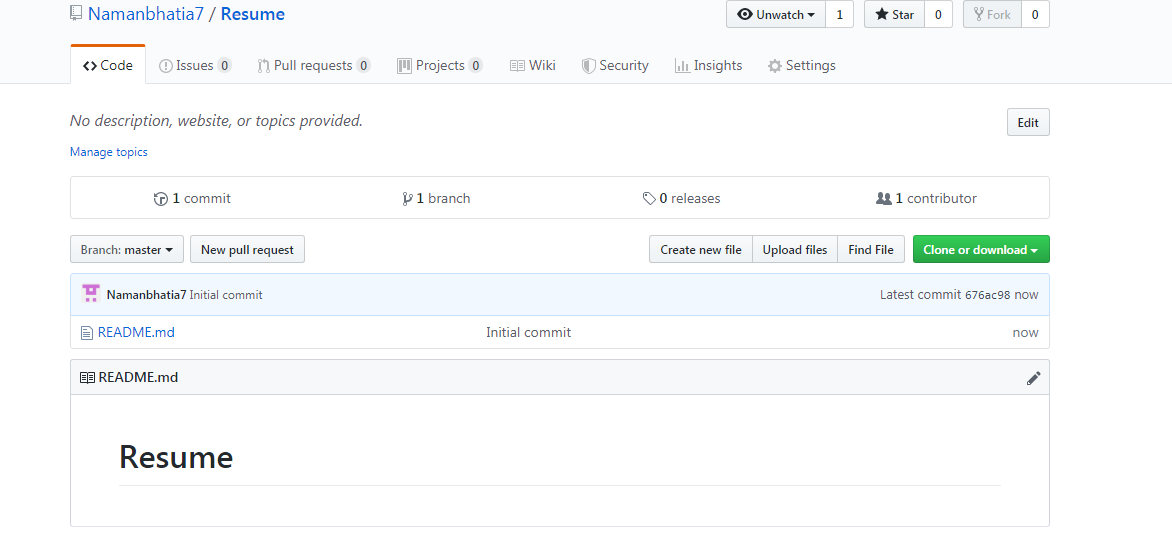
**Step 2:**  Click on the **new repository** option.



**Step 3:** After clicking **new repository** option, we will have to initialize some things like, **naming our project**, choosing the **visibility** etc. After performing these steps click **Create +Repository** button.



**Step 4:** After clicking the button, we will be directed to below page. Right now the only file we have is a readme file.

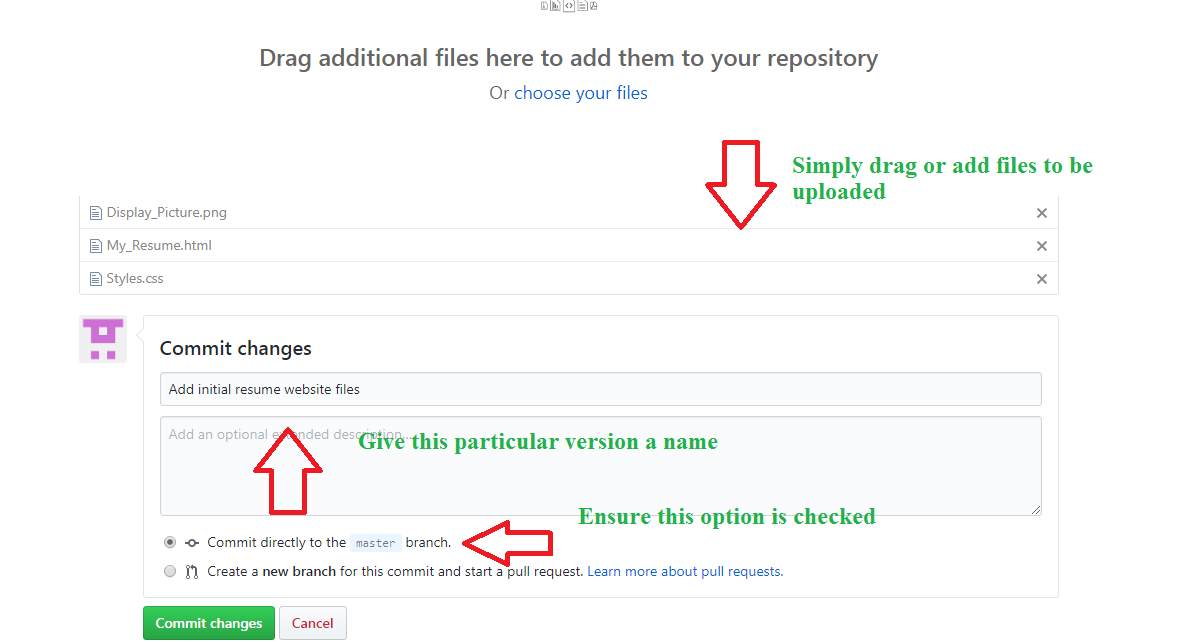


**Step 5:** Now click on the "Upload files" button.

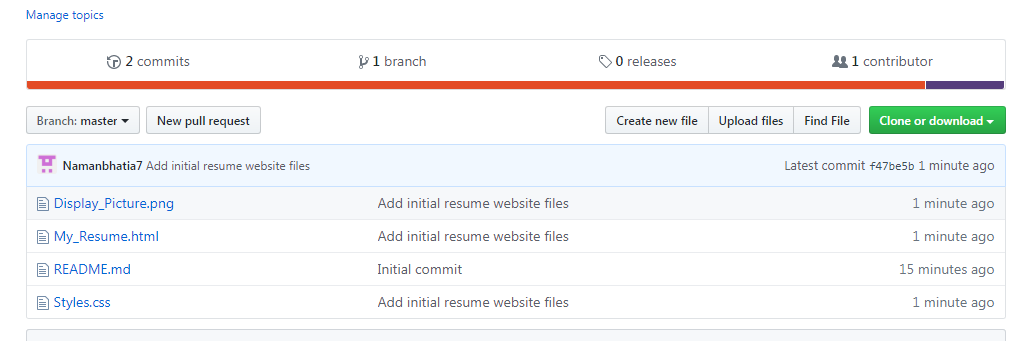
A screenshot of a computer

AI-generated content may be incorrect.

**Step 6:** Follow the steps mentioned in the Picture below and click "commit changes"



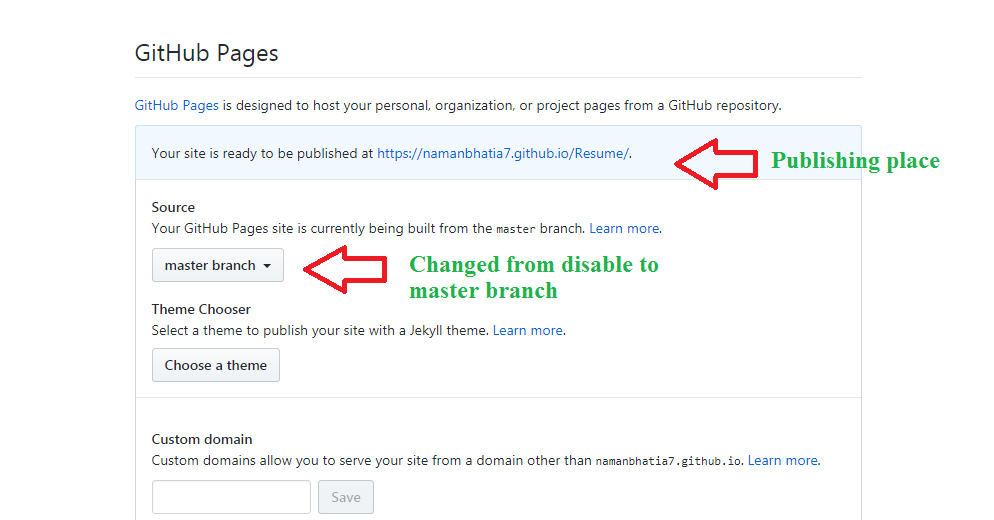
**Step 7:** Now you will see that all of our files uploaded in our github



**Hosting GitHub repository:**

GitHub Pages are designed to host your personal, organization, or project pages from a GitHub repository. As we already have the repository, we only have to activate our pages.

**Step 1:**Go to **settings** and scroll down to github pages section. Change disable option to master branch option. Now github will do some behind the scenes work and going to publish the repository.



**Step 2:** Now we are done and our project can be accessed worldwide.

A screenshot of a computer

AI-generated content may be incorrect.

**Useful Git Commands and Basic Concepts**

**1. Repository (Repo)**

**A repository is a storage space where your project's files and their revision history are kept. It can be local (on your computer) or remote (on a server).**

**2. Commit**

**A commit represents a snapshot of your repository at a specific point in time. Each commit has a unique SHA-1 hash identifier and includes a message describing the changes.**

**3. Branch**

**A branch is a parallel version of your repository. It allows you to work on different features or fixes without affecting the main codebase. The default branch in a new repository is usually called master or main.**

**4. Merge**

**Merging is the process of combining changes from one branch into another. It helps integrate the work done in different branches.**

**5. Clone**

**Cloning is the process of creating a copy of a remote repository on your local machine.**

**6. Remote**

**A remote is a common repository that all team members use to exchange their changes.**

**7. Staging Area / Index**

**The staging area (or index) is where you prepare changes to be committed. It allows you to review and organize changes before making a commit.**

**8. Working Directory**

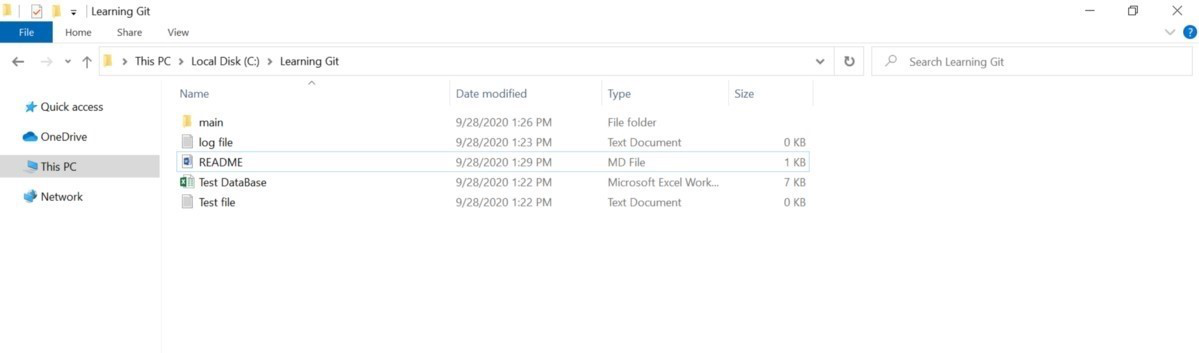
**The working directory is where you modify your files. Changes in the working directory are not tracked by Git until you stage them.**

* **Setting up our Git Repository**

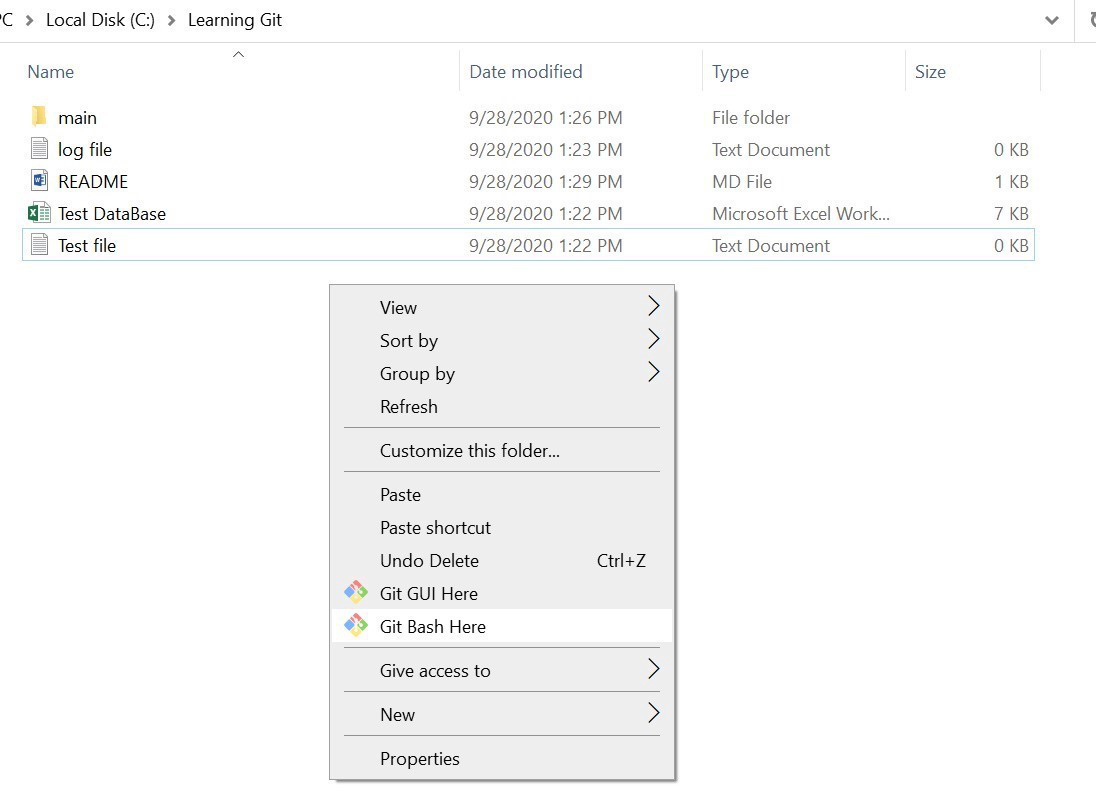
**Let’s make a Git Repository first of all and learn about all these Git commands.**

**Let’s start by creating a folder where we can have different files like a text file, an Excel Database file, a Markdown File, a Folder consisting of Source Code files (An actual Project may have many different files).**

**But right now for simplicity's sake let’s keep these files empty.**

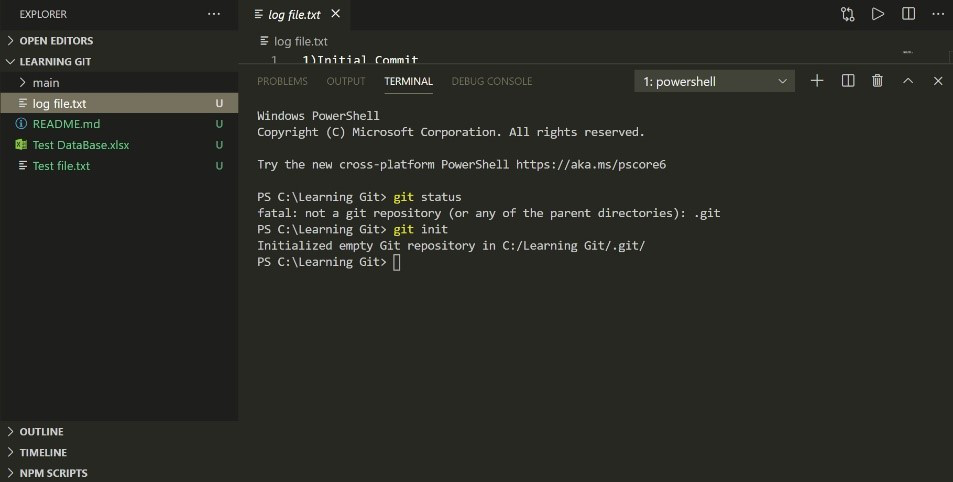
****

**Now let’s open Git Bash by Right-clicking and selecting “Git Bash Here”.**

****

**Initializing our Git Repository**

**Here we are initializing our Git Repository or you can say we are making our directory a Git Repository.**

****

***Note: If you already have made a Git Repository then don’t use this command again else all the changes that you have made to this Repository will get re-initialized.***

**Staging Files Initially**

**We are staging these files initially so that now we can track them using Git.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Creating Initial Commit**

**Let’s make our Initial Commit to our Repository.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Now let’s take a look at some of the useful Git commands for merging branches, forking a Repository, Renaming, and Deleting files using the command line, and much more.**

* **Useful Git Commands**

**1. git init**

**Initializes a new Git repository in the current directory.**

**git init**

**2. git clone**

**Creates a copy of an existing remote repository.**

**git clone <repository-url>**

**3. git status**

**Displays the state of the working directory and the staging area**

**4. git add**

**Adds changes from the working directory to the staging area.**

**git add <file>**

**To add all changes:**

**git add .**

**5. git commit**

**Records the changes in the staging area in the repository with a descriptive message.**

**git commit -m "Your commit message"**

**6. git log**

**Shows the commit history for the current branch.**

**7. git branch**

**Lists all branches in the repository. The \* indicates the current branch.**

**Creates a new branch:**

**git branch <branch-name>**

**8. git checkout**

**Switches to a different branch or restores files in the working directory.**

**git checkout <branch-name>**

**9. git merge**

**Merges changes from one branch into the current branch.**

**git merge <branch-name>**

**10. git pull**

**Fetches changes from a remote repository and merges them into the current branch.**

**git pull origin <branch-name>**

**11. git push**

**Uploads local commits to a remote repository.**

**git push origin <branch-name>**

**12. git remote**

**Manages the set of tracked repositories.**

**To add a new remote:**

**git remote add <name> <url>**

**13. git fetch**

**Downloads objects and refs from another repository.**

**14. git reset**

**Resets the current HEAD to a specified state. It can be used to unstage changes or move the branch pointer.**

**To unstage changes:**

**git reset <file>**

**15. git diff**

**Shows changes between commits, commit and working tree, etc.**

**16. Ignoring files**

**To ignore files, create a .gitignore file in your repository with a line for each pattern. File ignoring will work for the current and sub directories where .git ignore file is placed. In this example, all files are ignored in the logs directory (excluding the .git keep file), whole tmp directory and all files\*.swp.**

**1. Create ignore file:**

**we need a gitignore file if we don’t have one then you have create one.you can create one in your repository root.**

**touch .gitignore**

**2. Edit file:**

**Open the .gitignore file with your preferred text editor and add the patterns for the files and directories you want to ignore.**

**nano .gitignore**

**\*.*the underneath lines need to be added to gitignore file:***

**gitignore file# Ignore all files in the logs directory, but not the .gitkeep file**

**logs/\***

**!logs/.gitkeep**

**# Ignore the entire tmp directory**

**tmp/**

**# Ignore all files with .swp extension**

**\*.swp**