Name: Meet Duseja

Batch: T12

Roll-No: 27

Experiment No. 2

Aim: To understand version control system ,install git and create a Github account.

Theory:

Version control system

Version Control is a system that helps developers track and manage changes to a set of files (usually source code) over time. It allows you to keep a historical record of changes, collaborate with others, and revert to earlier versions of a file or a project when necessary.

There are two primary types of version control systems:

1. **Local Version Control**: This is the simplest form, where you keep track of changes locally on your own machine. It might involve manually saving different versions of a file or using a tool that only stores versions locally (without the ability to share changes across teams). This approach is not commonly used today due to its limitations.

2. **Distributed Version Control (DVCS)**: In this system, the version history is stored on every contributor's machine, meaning every developer has a full copy of the project's history. Git is the most widely used distributed version control system today. With DVCS, you can work offline and later synchronize changes with a central server (like GitHub, GitLab, or Bitbucket).

Key Benefits of Version Control

1. **History Tracking**: Every change made to files is recorded, with details such as the change description, who made the change, and when the change was made. This allows you to see the evolution of a project over time.

2. **Collaboration**: Multiple developers can work on the same project simultaneously without overriding each other's changes. Version control tools allow developers to merge changes from different contributors.

3. **Branching and Merging**: Developers can create branches to work on different features or bug fixes without affecting the main codebase. Later, the changes can be merged back into the main branch, allowing for parallel development.
4. **Rollback and Recovery**: If something goes wrong, version control allows you to revert to a previous version of a file or the entire project. This makes it easy to undo mistakes or recover from issues caused by new changes.

5. **Conflict Resolution**: In collaborative environments, version control helps to resolve conflicts when multiple people make changes to the same part of a project. The system will notify you when conflicts occur and usually give you the opportunity to fix them manually.

6. **Backup and Safety**: Version control systems (especially when using a remote repository like GitHub or GitLab) offer a form of backup for your code. Even if your local machine crashes, your code is still safe on the remote server.

7. **Improved Code Quality**: With version control systems, code reviews, and pull requests (in systems like GitHub), teams can review and discuss proposed changes before they are integrated into the main project, leading to better code quality.

Git:

Git is a distributed version control system (VCS) that helps developers track and manage changes to their codebase over time. It's designed to handle everything from small to large projects with speed and efficiency. Git allows you to record the history of your code, revert to previous versions, and collaborate with other developers.

Key Concepts in Git

To understand how Git works, let's explore its core concepts.

1. **Repository (Repo)**: A repository is where your project files are stored, along with the history of all changes. A Git repository can either be local (on your machine) or remote (on a server like GitHub, GitLab, etc.).

2. **Working Directory**: This is the directory on your machine where you make changes to your project. It contains the actual files you are working on (e.g., source code files, images, etc.).

3. **Staging Area (Index)**: This is an intermediate area where you can prepare changes before committing them. When you make changes to files, you add them to the

staging area with the git add command. The staging area allows you to selectively choose which changes to include in your next commit.

4. **Commit**: A commit is like a snapshot of your project at a particular point in time. Each commit contains a reference to the changes made, the author of the changes, and a message describing the change. Commits are stored locally in your Git repository and provide a historical record of all changes made to the project.

5. **Branch**: Git allows you to create multiple branches to work on different features, bug fixes, or experiments without affecting the main codebase. The default branch is usually called main (or master in older versions). Branches enable parallel development, allowing teams to work on features independently.

6. **Merge**: When you've made changes in a branch and want to incorporate those changes into another branch, you use the git merge command. Git will try to automatically merge the changes. If there are conflicting changes (e.g., two different modifications to the same line of code), Git will flag a conflict that must be resolved manually.

7. **Clone**: To make a local copy of an existing Git repository (usually hosted on a remote server), you use the git clone command. This allows you to work on a project locally and push your changes to a remote server later.

8. **Pull and Push**:
    - **git pull**: Fetches and integrates changes from a remote repository into your local repository.
    - **git push**: Sends your local commits to a remote repository, making them available to other developers.

9. **Remote**: A remote is a version of your repository hosted on a server, often on a platform like GitHub, GitLab, or Bitbucket. It’s where you store your code so that others can collaborate, view, and contribute. **Git Workflow**

A typical Git workflow involves the following steps:

1. **Clone** a remote repository to your local machine.

2. **Create a new branch** for the feature or bug fix you're working on.

3. **Make changes** to your working directory.

4. **Stage** the changes you want to commit using git add.
5. **Commit** the staged changes with a message that describes the change (git commit -m "message").
6. **Push** the commits to the remote repository using git push.
7. **Create a Pull Request** (PR) to merge your changes into the main branch. PRs are reviewed and approved by other developers before being merged.

Github:

GitHub is a cloud-based platform that uses Git to host repositories and provides additional features like issue tracking, code reviews, and collaborative workflows. It's the most popular service for hosting Git repositories and is commonly used in open-source and private software projects.

GitHub enables developers to:

- **Host Git repositories**: Store your code in the cloud, accessible from anywhere.
- **Collaborate on projects**: Multiple developers can work on the same project, manage changes, and handle versioning.
- **Track issues**: GitHub provides an issue tracker for bug reports, feature requests, and general tasks.
- **Pull Requests**: GitHub facilitates code review through pull requests, where developers can review, comment on, and approve changes before they are merged.
- **Project Management**: GitHub provides tools for managing projects, such as project boards and milestones.
- **Continuous Integration**: You can automate testing and deployment workflows using GitHub Actions.
- **Social Networking**: GitHub includes social features like followers, stars (to mark favorite repositories), and forks (to copy someone else's repository).

Key Features of GitHub

1. **Repositories**: Each project on GitHub resides in a repository. A repository can be public (open to everyone) or private (restricted access). Repositories contain the code and all of its version history.

2. **Forks**: A fork is a copy of a repository that allows you to make changes without affecting the original project. Forking is commonly used in open-source development, where contributors can propose changes via pull requests.
3. **Pull Requests (PRs)**: A pull request is a way to propose changes to a repository. After creating a pull request, other contributors can review your changes, suggest improvements, and eventually merge them into the main codebase if they are approved.

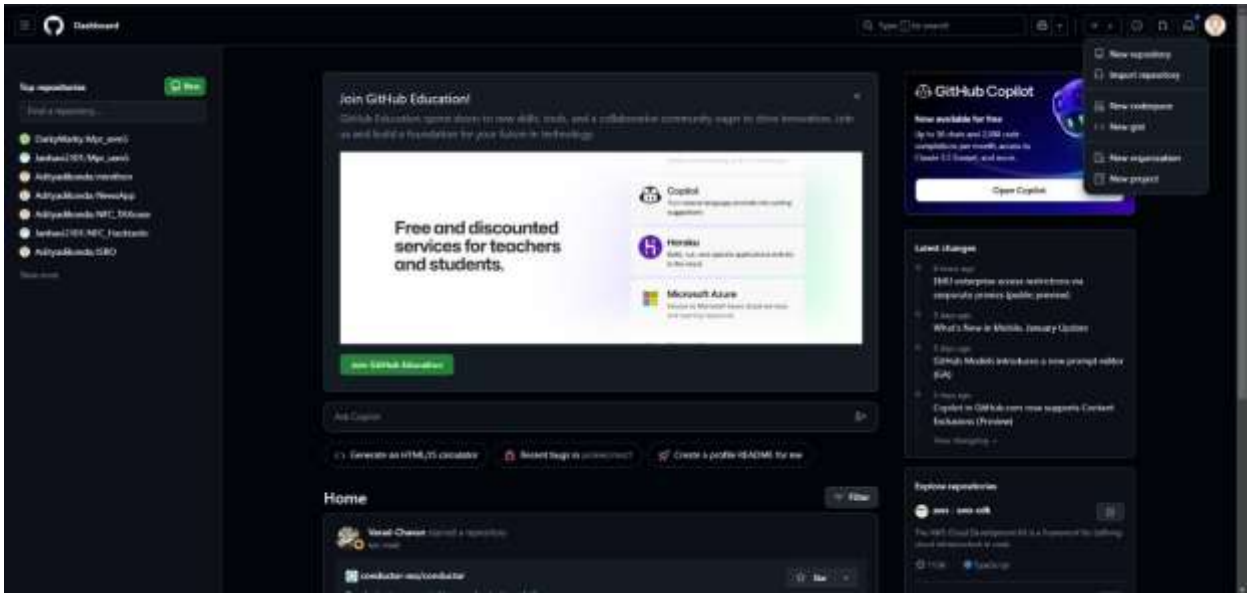
4. **Issues and Projects**: GitHub provides tools for managing tasks and issues, such as bug reports and feature requests. You can organize issues with labels, milestones, and project boards.

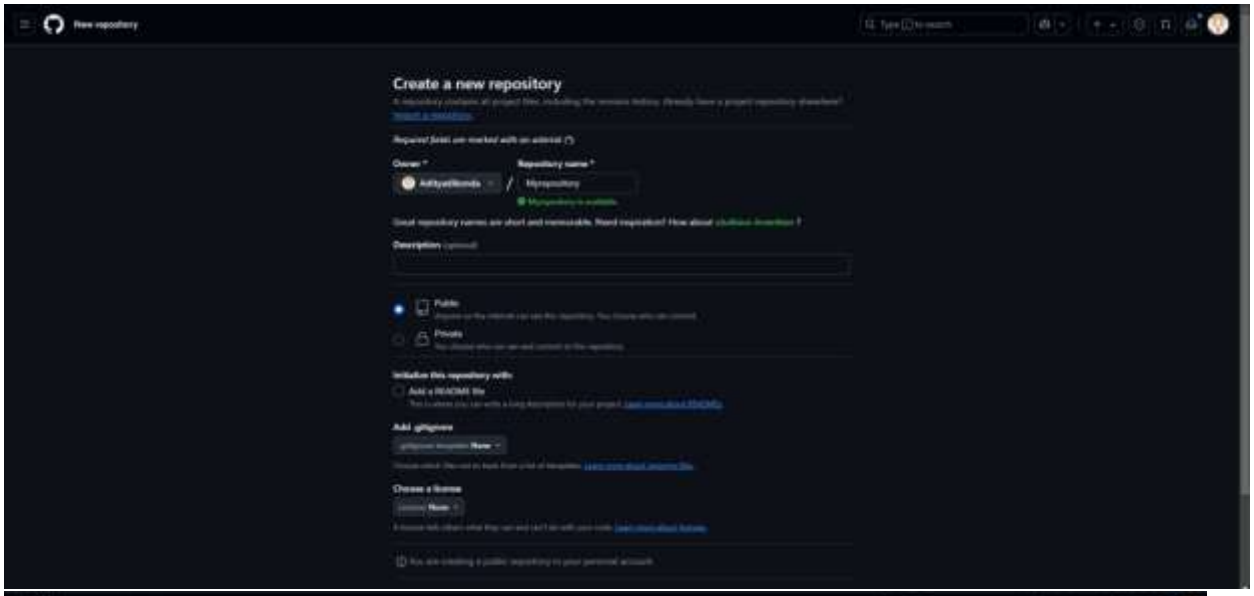
5. **Actions and CI/CD**: GitHub Actions allows you to automate workflows. For example, you can automatically run tests every time you push code, or deploy your application to a cloud provider when changes are merged.

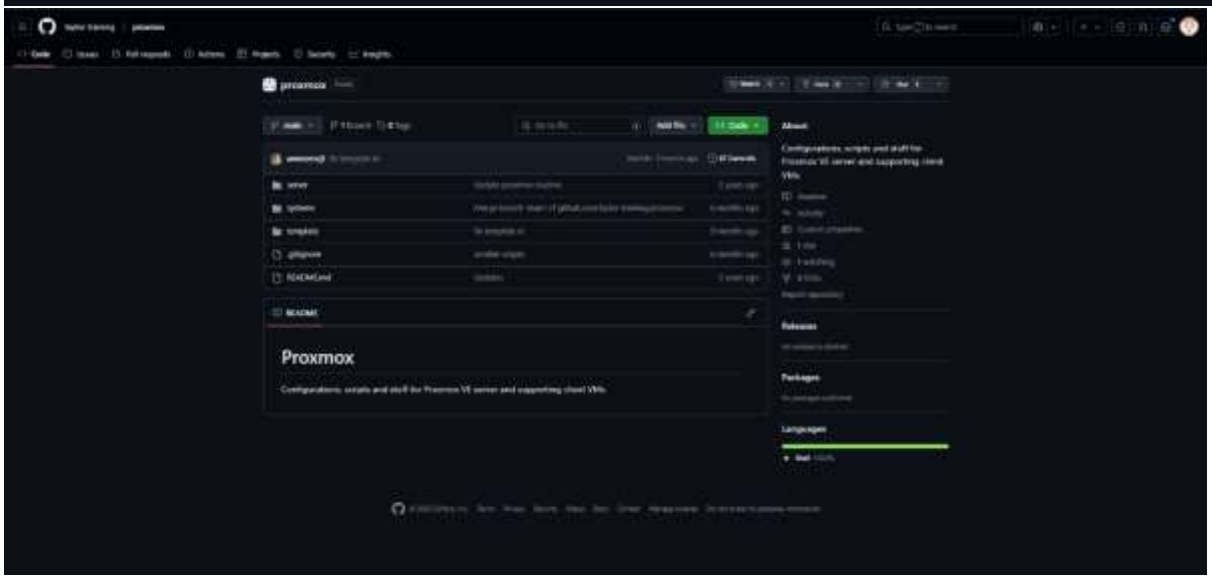
6. **GitHub Pages**: This feature allows you to host static websites directly from a GitHub repository.

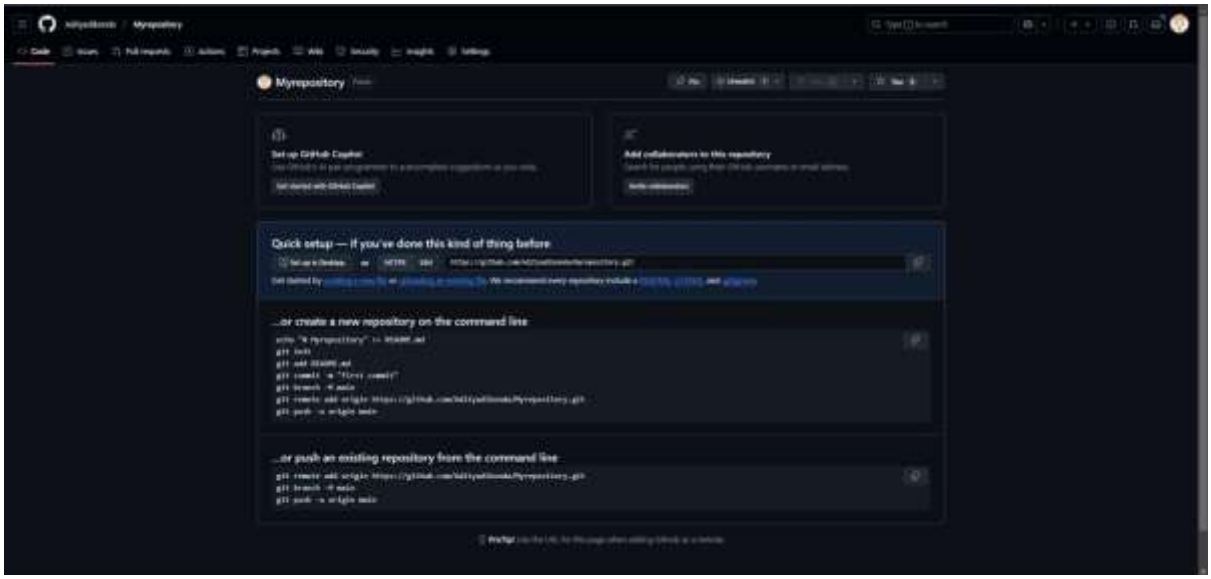
7. **Collaborators and Teams**: In a private repository, you can invite collaborators or set up teams within an organization to manage access control.

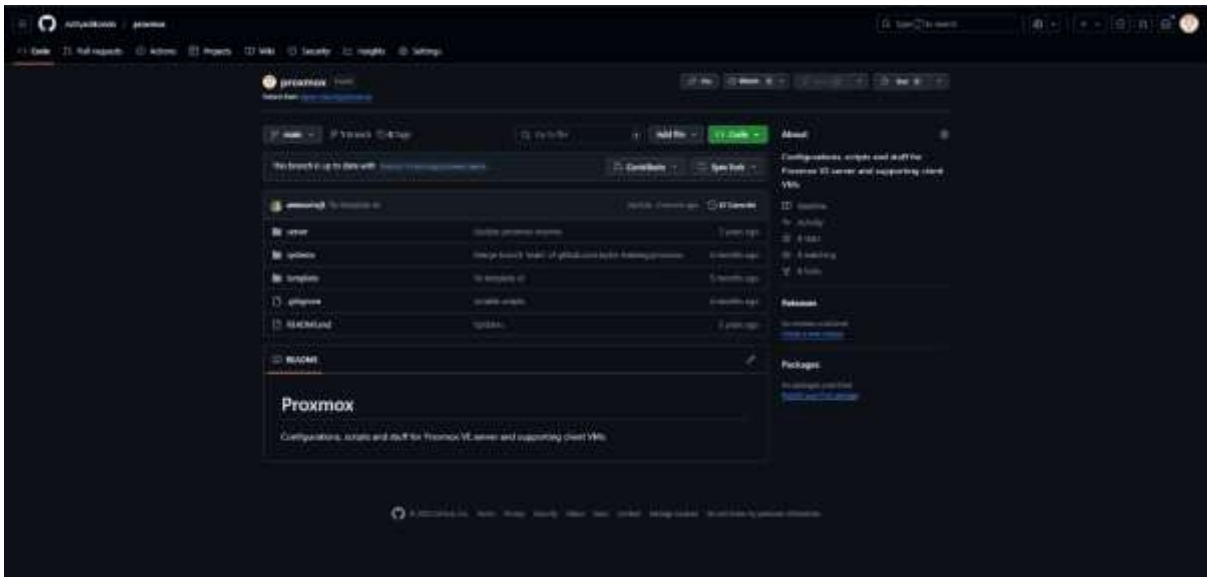
8. **Wiki**: GitHub repositories can have a built-in wiki where you can document your project, provide guides, or share development notes.


Dashboard
Join GitHub Education!
Free and discounted services for teachers and students.
GitHub Copilot
Open Copilot
Home


New repository
Create a new repository
Public
Private


Proxmox





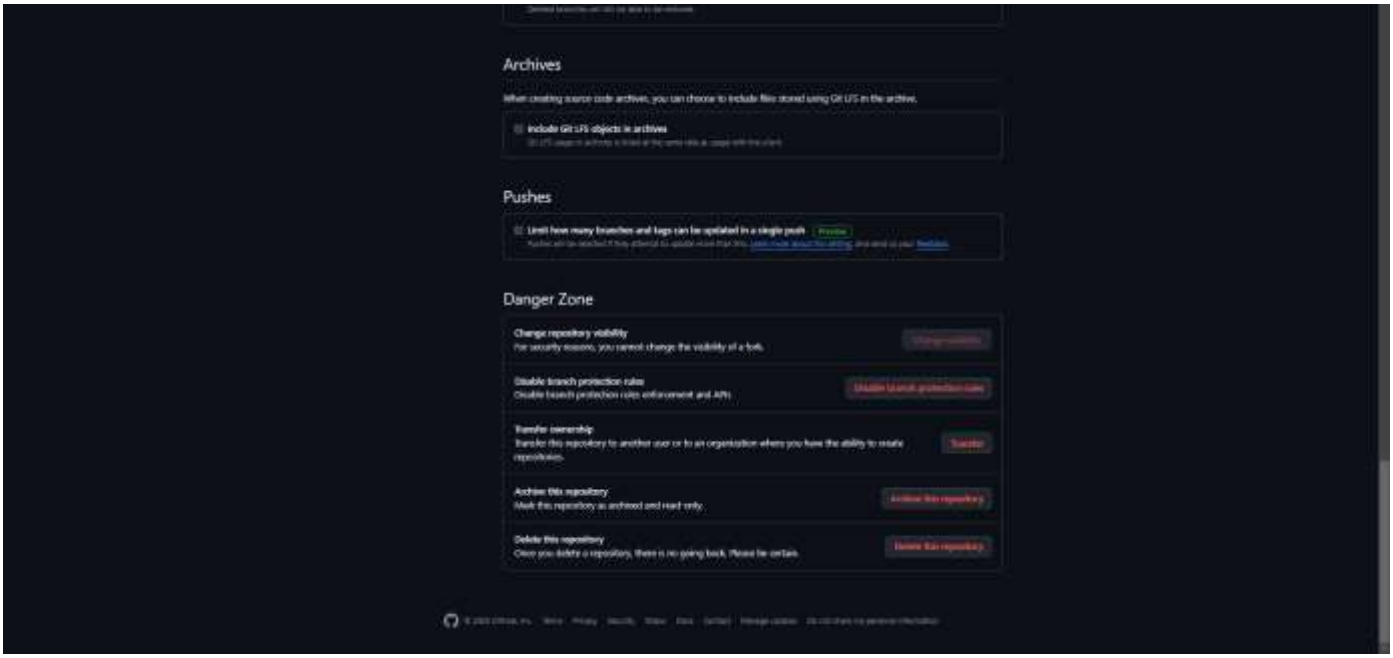
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

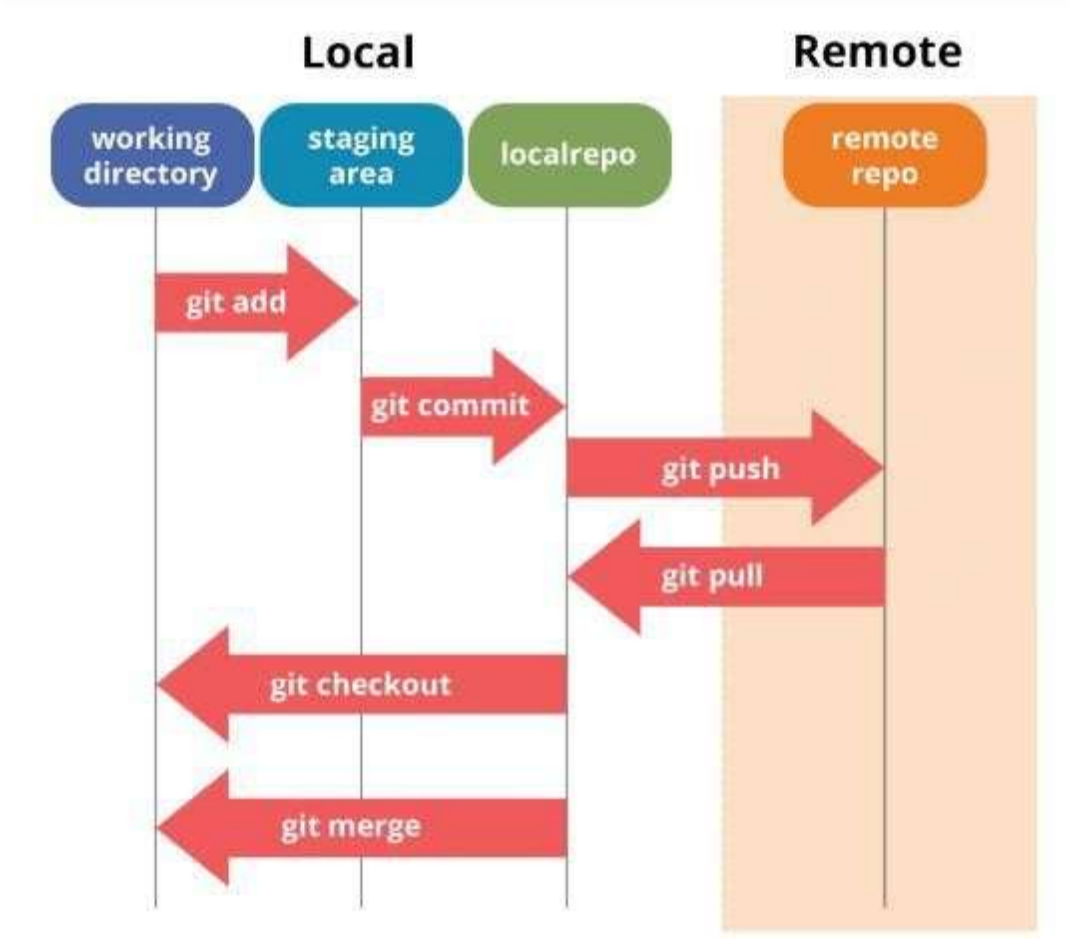
Some of the basic operations in Git are:

1. Initialize
2. Add
3. Commit

4. Pull
5. Push

Some advanced Git operations are:
1. Branching
2. Merging
3. Rebasing

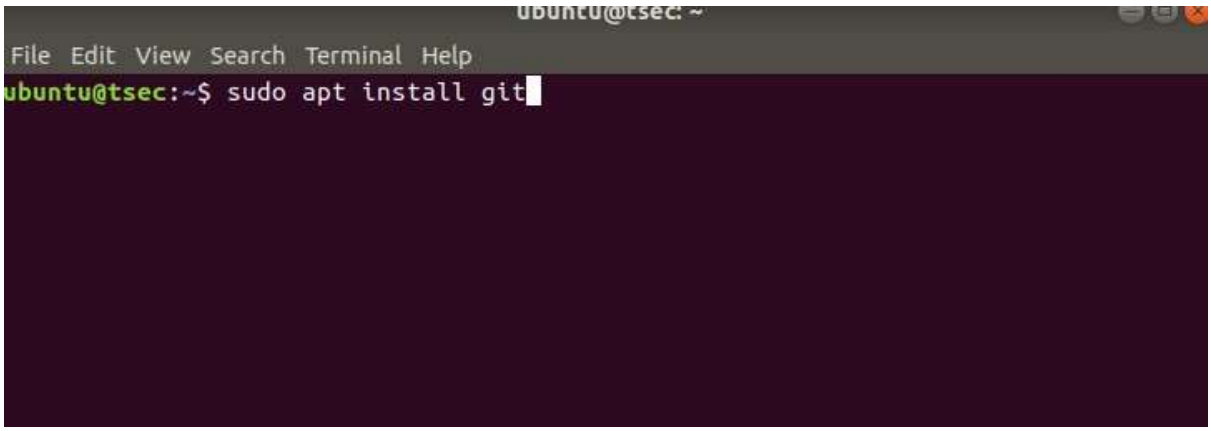


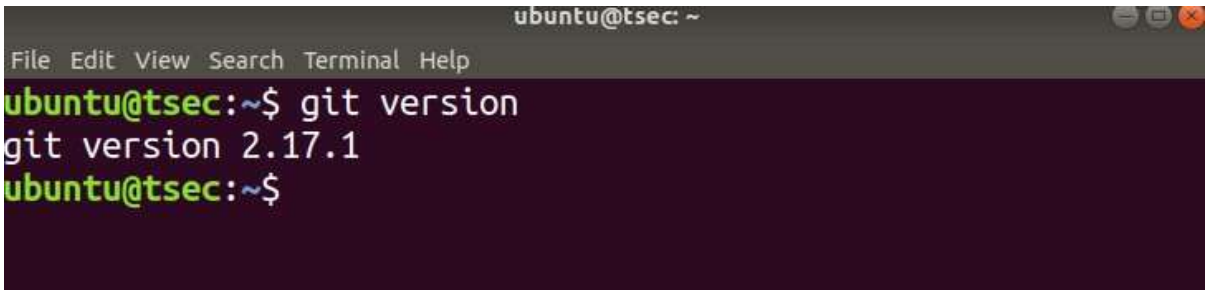


1) In windows, download GIT from https://git-scm.com/ and perform the straightforward installation.

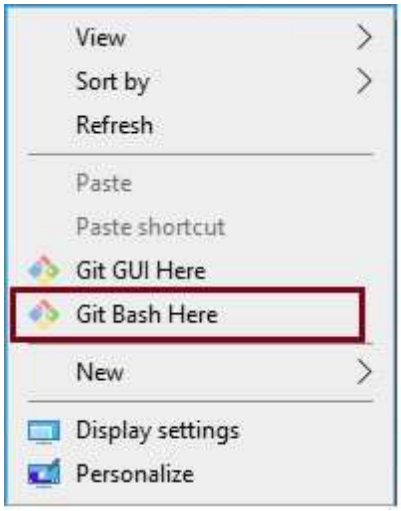
2) In Ubuntu, install GIT using $sudo apt install git, Confirm the version after installation using command $git –version

Installation of GIT

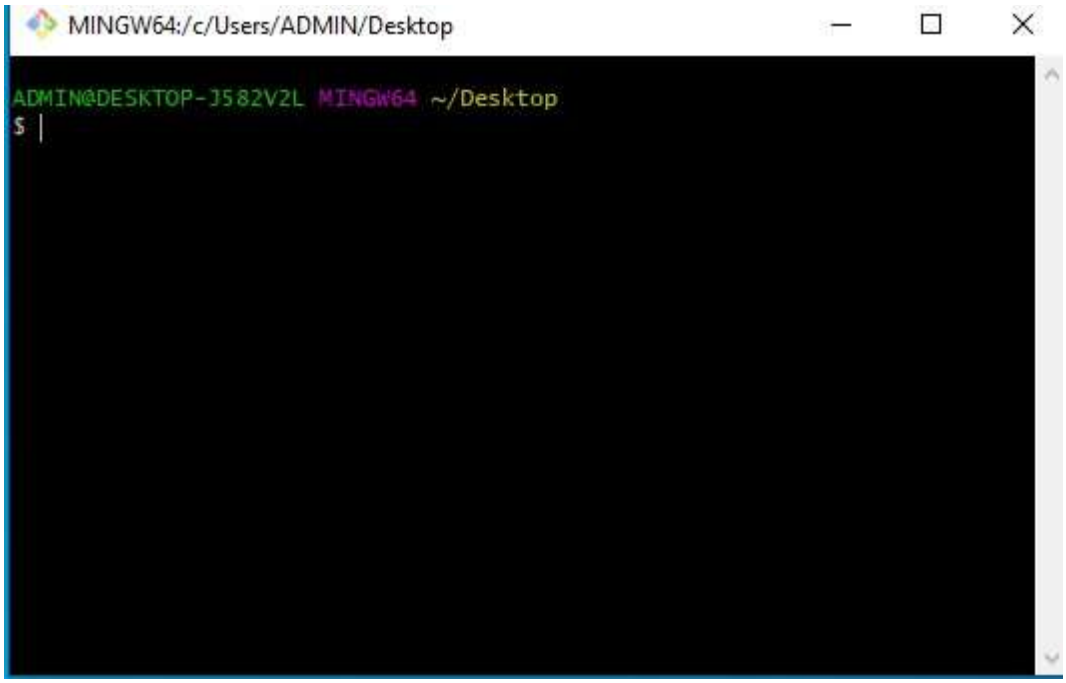


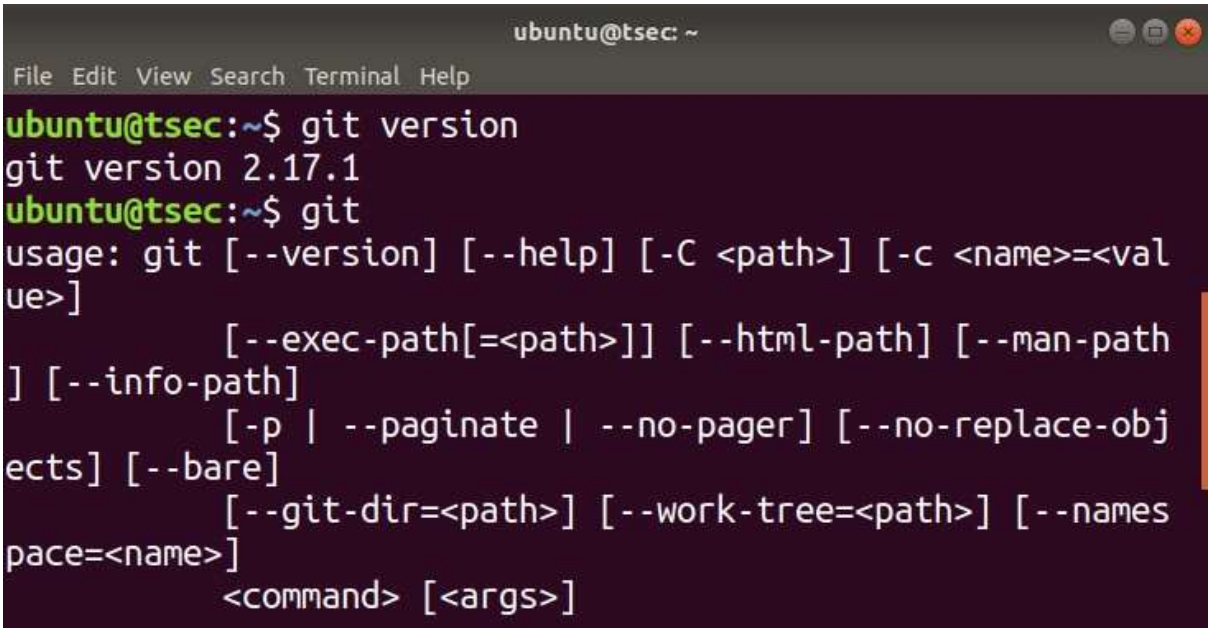


Once installation is done, open the terminal in Ubuntu and perform the following steps or in windows Right click and select Git bash here.



The output of GIT Bash in windows and GIT shell in Ubuntu is shown below





To perform version control, let us create a directory dvcs (Distributed version control system) and change directory to dvcs.

$ mkdir git-dvcs

$ cd git-dvcs/

Now check the user information using

$ git config –global

As there are no users defined, let us define it using following two commands

$ git config --global user.name "bhushan"

$ git config --global user.email "bhushan,jadhav1@gmail.com"

Now, check the list of users

$ git config --global --list user.name=zwar-alt
user.email=vinitkadam@gmail.com

Let us create a repository for version control named ”git-demo-project”

$ mkdir git-demo-project

$ cd git-demo-project/

Now, initialize the repository using following command

$ git init

```
C:\Users\203\Desktop\git-dvcs>mkdir git-demo-project

C:\Users\203\Desktop\git-dvcs>cd git-demo-project/

C:\Users\203\Desktop\git-dvcs\git-demo-project>git init
Initialized empty Git repository in C:/Users/203/Desktop/git-dvcs/git-demo-project/.git/

C:\Users\203\Desktop\git-dvcs\git-demo-project>ls -a
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\203\Desktop\git-dvcs\git-demo-project>git add .

C:\Users\203\Desktop\git-dvcs\git-demo-project>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Sample Text Document.txt


C:\Users\203\Desktop\git-dvcs\git-demo-project>git commit -m "First commit"
[master (root-commit) fc69bf4] First commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Sample Text Document.txt
```

The output of above command shown below which adds .git hidden directory in current repository.

If you have existing repository, then simply delete .git file and reinitialize it.

rm -rf .git/ $

git init

Initialized empty Git repository in C:/Users/ADMIN/Desktop/git-dvcs/git-demo-project/.git/

let us add some files inside our repository “git-demo-project”

To add files in the repository by create or copy some doc,html,image files inside current directory to see index and staging area. The add command is used along with dot (. Dot means current directory) for adding files in current repository i.e. making them in staging mode. They are untracked until we commit them.

$ git add .

Index and staging area

To check the status of repository, use $

git status

Which will show you some untrack files, so untracks files can be tracked using commit command.

Now, let us commit the changes
$ git commit -m "First Commit" (#here -m for message) git
add .
$ git commit -am "express Commit" (#Here -a used for express commit)
$ nano index.html
$ touch teststatus
Changes are Discarded by checkout
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)

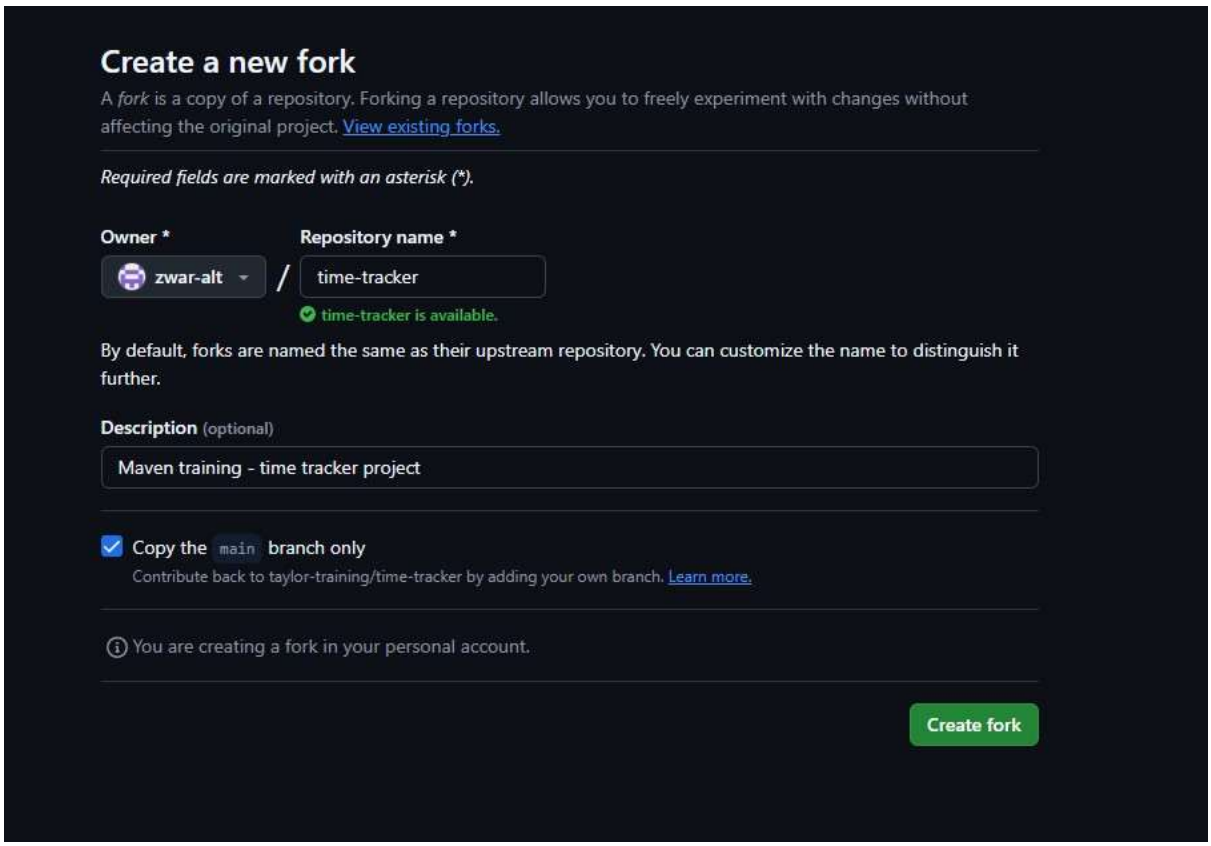
```
C:\Users\203\Desktop\git-dvcs\git-demo-project>git add "Text 2.txt"

C:\Users\203\Desktop\git-dvcs\git-demo-project>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Text 2.txt


C:\Users\203\Desktop\git-dvcs\git-demo-project>git restore "Text 2.txt"

C:\Users\203\Desktop\git-dvcs\git-demo-project>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Text 2.txt
```

Now let us see history of commits. The log command is used for seeing the commit history.
$ git log

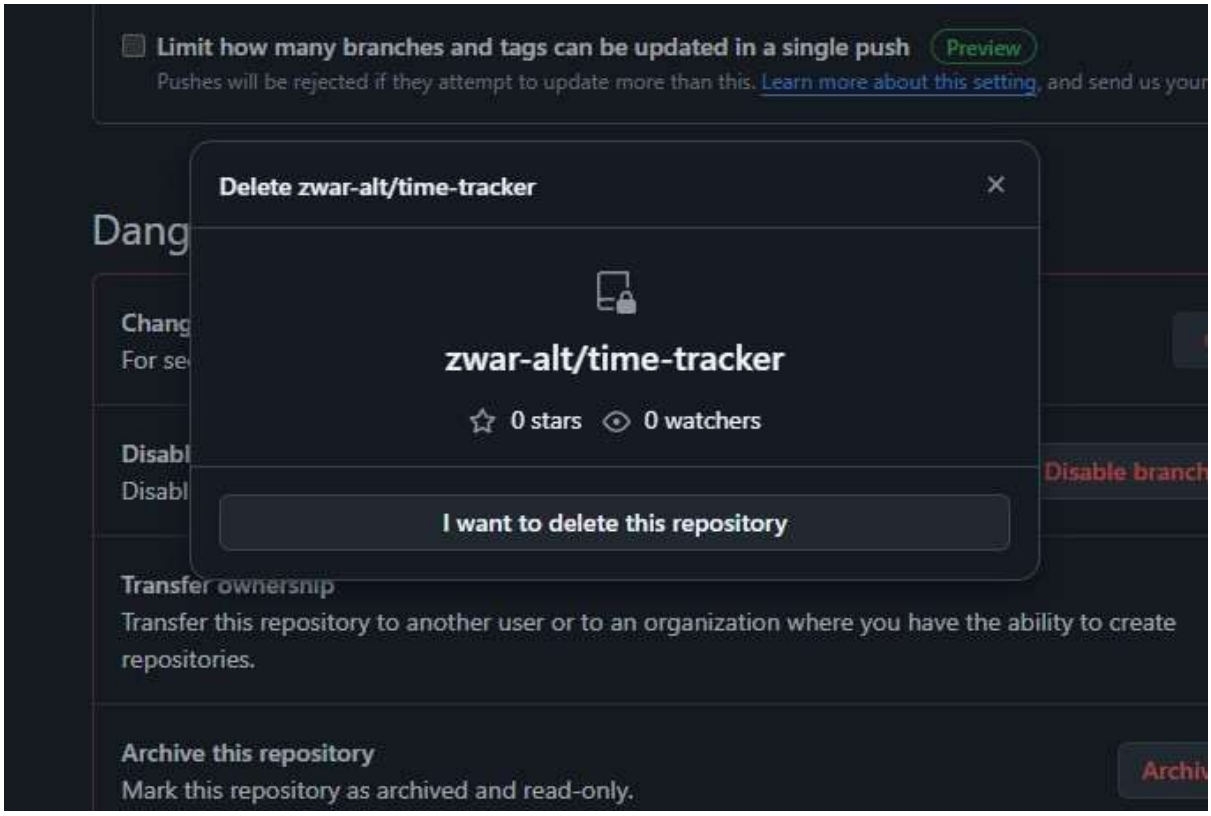
```
C:\Users\203\Desktop\git-dvcs\git-demo-project>git add "Text 2.txt"

C:\Users\203\Desktop\git-dvcs\git-demo-project>git commit -am "Express commit"
[master b4efde6] Express commit
 1 file changed, 1 insertion(+)
 create mode 100644 Text 2.txt

C:\Users\203\Desktop\git-dvcs\git-demo-project>git log
commit b4efde6b8fbedfaf508c9550c0dc2a39bad62902 (HEAD -> master)
Author: zwar_alt <mhatreswar@gmail.com>
Date:   Mon Feb 3 12:57:17 2025 +0530

    Express commit

commit 6d7628a9ebda6eb330224500a5a03dfa2785e7dc
Author: zwar_alt <mhatreswar@gmail.com>
Date:   Mon Feb 3 12:50:32 2025 +0530

    express Commit

commit fc69bf4c3acdea3917771f25f3d30b4e6d904670
Author: zwar_alt <mhatreswar@gmail.com>
Date:   Mon Feb 3 12:48:56 2025 +0530

    First commit

C:\Users\203\Desktop\git-dvcs\git-demo-project>git log --oneline "Text 2.txt"
b4efde6 (HEAD -> master) Express commit

C:\Users\203\Desktop\git-dvcs\git-demo-project>
```

To see all the operation in oneline use the –-oneline option in log command
–-oneline option for particular file in log command

By default, we can create public repository in Github. So we can copy the entire public repository of any other users in to own account using “FORK” Operation. Now fork the repository (Sharing with other users who wants to contribute).
Login with another account→Copy and Paste URL of repository→then just click on fork to clone to others account. Suppose we want to fork public repository “timetracker”. So search for “timetracker” github repository on google and once its opened clicked on “Fork button” from the top of the github web page as shown below. After fork it will be added in your local repository



To delete the repository, open the desired repository you want to delete and go to the settings option. There you will see delete repository button to delete it.



Now, if you want to download a repository in local machine, then git clone command is used followed by path to repository. In GitHub the path of repository can be known through clone or download button and it can be downloaded using git clone command as shown below.

```
C:\Users\203>cd Desktop

C:\Users\203\Desktop>cd git-dvcs

C:\Users\203\Desktop\git-dvcs>git clone https://github.com/TeenageMutantCoder/Alarm-Clock
Cloning into 'Alarm-Clock'...
remote: Enumerating objects: 157, done.
remote: Counting objects: 100% (157/157), done.
remote: Compressing objects: 100% (125/125), done.
remote: Total 157 (delta 71), reused 94 (delta 26), pack-reused 0 (from 0)Receiving objects:  93% (147/157), 420.00 KiB
Receiving objects:  99% (156/157), 420.00 KiB | 775.00 KiB/s
Receiving objects: 100% (157/157), 572.54 KiB | 953.00 KiB/s, done.
Resolving deltas: 100% (71/71), done.
```

Pull and Push Processes

The pull command used to fetch the repository from github to local while push is used to commit files from local repository to Github.

Push → Push changes to Web repository

Pull → Pull changes to Local repository

The following commands are used for pull and push repositories

A) Push command

$ git remote add origin https://github.com/bhushanjadhav1/siesworkshop.git $ git remote show origin

```
C:\Users\203\Desktop\git-dvcs\git-demo-project>git remote add origin2 https://ghp_sd0N6XSvmrOy9SlLK7nw
14sZQqpkwa2x0YyF@github.com/zwar-alt/Sepm_exp2.git

C:\Users\203\Desktop\git-dvcs\git-demo-project>git push -u origin2 main
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 20 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (11/11), 915 bytes | 915.00 KiB/s, done.
Total 11 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/zwar-alt/Sepm_exp2.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin2/main'.

C:\Users\203\Desktop\git-dvcs\git-demo-project>
```

If you add remote again then will show you fatal error.

$ git remote add origin https://github.com/bhushanjadhav1/Myrepository.git fatal: remote origin already exists.

So, to delete origin rm origin command is used

$ git remote rm origin

Now, to push the local repository to remote github following command is used $ git push -u origin master

B) Pull Changes

Pull command is used to download the remote updated repository into local one. The command for download is

$ git pull

```
C:\Users\203\Desktop\git-dvcs\git-demo-project>git pull
Updating 3fa12e9..288ff46
Fast-forward
 text 2 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 text 2
```

Now you can see the changes in local repository using git log.

C) Fetch

Suppose you have a file in github and you have changes that.

```
C:\Users\203\Desktop\git-dvcs\git-demo-project>git fetch
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 922 bytes | 230.00 KiB/s, done.
From https://github.com/zwar-alt/Sepm_exp2
   3fa12e9..288ff46  main       -> origin2/main

C:\Users\203\Desktop\git-dvcs\git-demo-project>git pull
Updating 3fa12e9..288ff46
Fast-forward
 text 2 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 text 2
```

Here fetch will not show you like updated changes file as like push. So use merge command to merge the changes so use following command for merge.

$ git merge origin/master