

Software Engineering and Project Management

EXPERIMENT – 2

Name: Nisarg Sampat

Batch: T21

Roll No: 87

AIM:

To understand the 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 (typically source code) over time. It maintains a historical record of modifications, facilitates collaboration, and allows developers to revert to previous versions if needed.

Types of Version Control Systems

1. Local Version Control:

- The simplest form of version control, where changes are tracked only on a local machine.
- Developers manually save different versions or use basic tools for versioning.
- This method is rarely used today due to limited collaboration capabilities.

2. Distributed Version Control (DVCS):

- Stores the complete version history on every contributor's machine.
- Developers can work offline and later synchronize changes with a remote server.
- Git is the most widely used DVCS, with platforms like GitHub, GitLab, and Bitbucket supporting remote collaboration.

Key Benefits of Version Control

1. **History Tracking:** Records every change, including details about the author and timestamp.
2. **Collaboration:** Enables multiple developers to work on the same project without overwriting changes.
3. **Branching and Merging:** Allows developers to create separate branches for new features or bug fixes and merge them into the main codebase later.

4. **Rollback and Recovery:** Provides the ability to revert to previous versions in case of errors.
5. **Conflict Resolution:** Helps manage conflicting changes in collaborative environments.
6. **Backup and Safety:** Ensures code is safely stored in remote repositories, preventing data loss.
7. **Improved Code Quality:** Enables code reviews and discussions before merging changes, leading to better development practices.

Git: A Distributed Version Control System

Git is a fast, scalable, and efficient distributed version control system (VCS) designed to handle small to large projects. It allows developers to track code changes, collaborate, and maintain a complete history of modifications.

Core Concepts in Git

1. **Repository (Repo):** A storage location containing all project files and their version history. Repositories can be local (on a developer's machine) or remote (on a platform like GitHub).
2. **Working Directory:** The folder on your machine where you modify project files.
3. **Staging Area (Index):** A temporary space where changes are prepared before committing them. Developers use `git add` to move changes to the staging area.
4. **Commit:** A snapshot of the project at a particular time. Each commit includes a description, author information, and a unique identifier.
5. **Branch:** A separate line of development. Git allows developers to work on different branches (e.g., feature branches) before merging changes into the main branch.
6. **Merge:** The process of integrating changes from one branch into another using `git merge`.
7. **Clone:** Copies an existing Git repository to a local machine using `git clone`.
8. **Pull and Push:**
 - `git pull`: Fetches and merges updates from a remote repository.
 - `git push`: Uploads local commits to a remote repository.
9. **Remote Repository:** A version of the repository hosted on a platform like GitHub, enabling collaboration and backup.

Git Workflow

A typical Git workflow includes:

1. **Cloning a repository** from a remote server.
2. **Creating a new branch** for feature development or bug fixes.
3. **Making changes** in the working directory.
4. **Staging changes** using git add.
5. **Committing changes** with a descriptive message (git commit -m "message").
6. **Pushing changes** to the remote repository (git push).
7. **Creating a pull request (PR)** to merge changes into the main branch. PRs are reviewed before merging.

GitHub: A Platform for Git-Based Collaboration

GitHub is a cloud-based platform that enhances Git by providing hosting, collaboration tools, and project management features. It is widely used in both open-source and private software development.

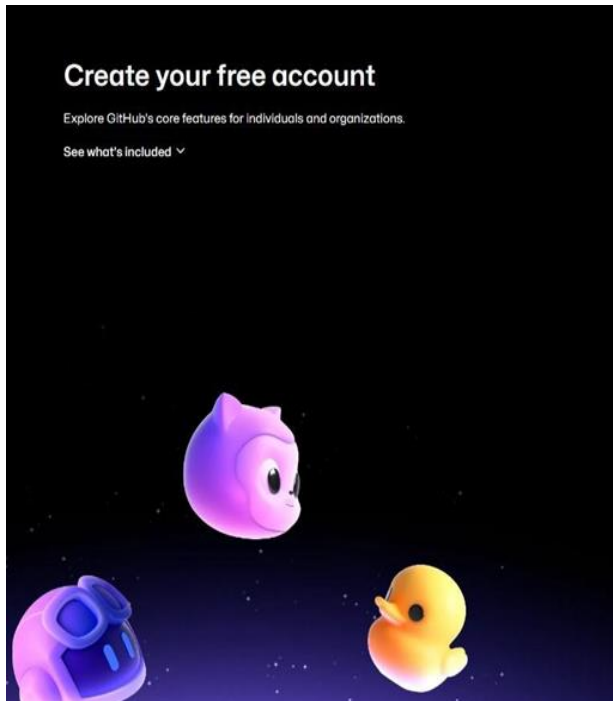
Key Features of GitHub

1. **Repositories:** Centralized storage for code, available as public or private repositories.
2. **Forks:** Personal copies of repositories, used to propose changes without modifying the original project.
3. **Pull Requests (PRs):** A way to propose changes. Contributors submit PRs for review before merging updates.
4. **Issue Tracking:** Tools for reporting bugs, requesting features, and managing development tasks.
5. **GitHub Actions & CI/CD:** Automates testing and deployment workflows.
6. **GitHub Pages:** Hosts static websites directly from repositories.
7. **Collaboration & Teams:** Developers can manage permissions, assign roles, and coordinate contributions.
8. **Wikis:** Built-in documentation for repositories.

How GitHub Enhances Development

- **Code Hosting:** Enables remote storage of Git repositories.
- **Collaborative Workflows:** Supports real-time collaboration among teams.
- **Project Management:** Organizes development tasks using milestones and project boards.

- **Continuous Integration:** Automates testing and deployment through GitHub Actions.
- **Community Engagement:** Encourages open-source contributions through forks, PRs, and discussions.



Already have an account?

Sign up to GitHub

Email*

Password*

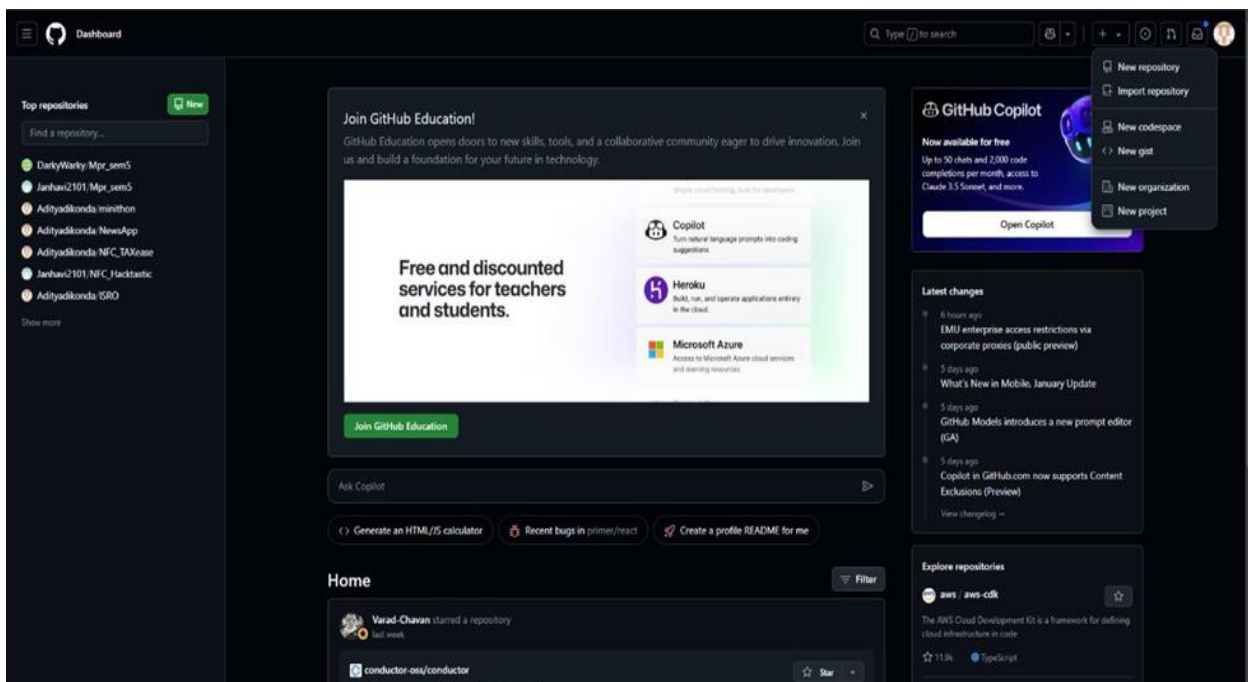
Password should be at least 15 characters OR at least 8 characters including a number and a lowercase letter.

Username*

Username may only contain alphanumeric characters or single hyphens, and cannot begin or end with a hyphen.

[Continue](#)

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.



New repository

Q Type [] to search

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *

Repository name *

AdityaDikonda

/ Myrepository

Myrepository is available.

Great repository names are short and memorable. Need inspiration? How about [studious-invention](#)?

Description (optional)

☒ Public
Anyone on the internet can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

Licenses: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

You are creating a public repository in your personal account.

taylor training / proxmox

CodeIssuesPull requestsActionsProjectsSecurityInsights

proxmoxPublic

WatchForkStar

main1 BranchTagsGo to fileAdd fileCode

awsomegitfor template id26kfile · 3 months ago47 Commits

server	Update proxmox readme	3 years ago
systems	Merge branch 'main' of github.com:taylor-training/proxmox	6 months ago
template	fix template id	5 months ago
.gitignore	Addable scripts	6 months ago
README.md	Updates	3 years ago

README

Proxmox

Configurations, scripts and stuff for Proxmox VE server and supporting client VMs

About

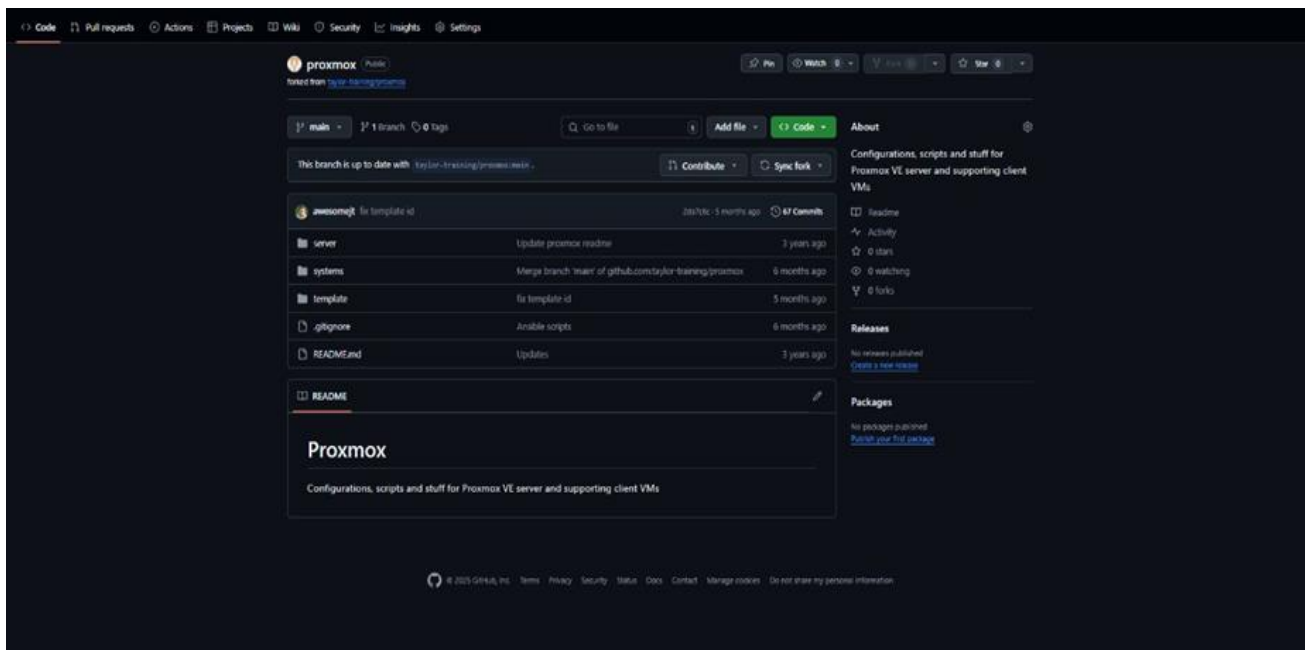
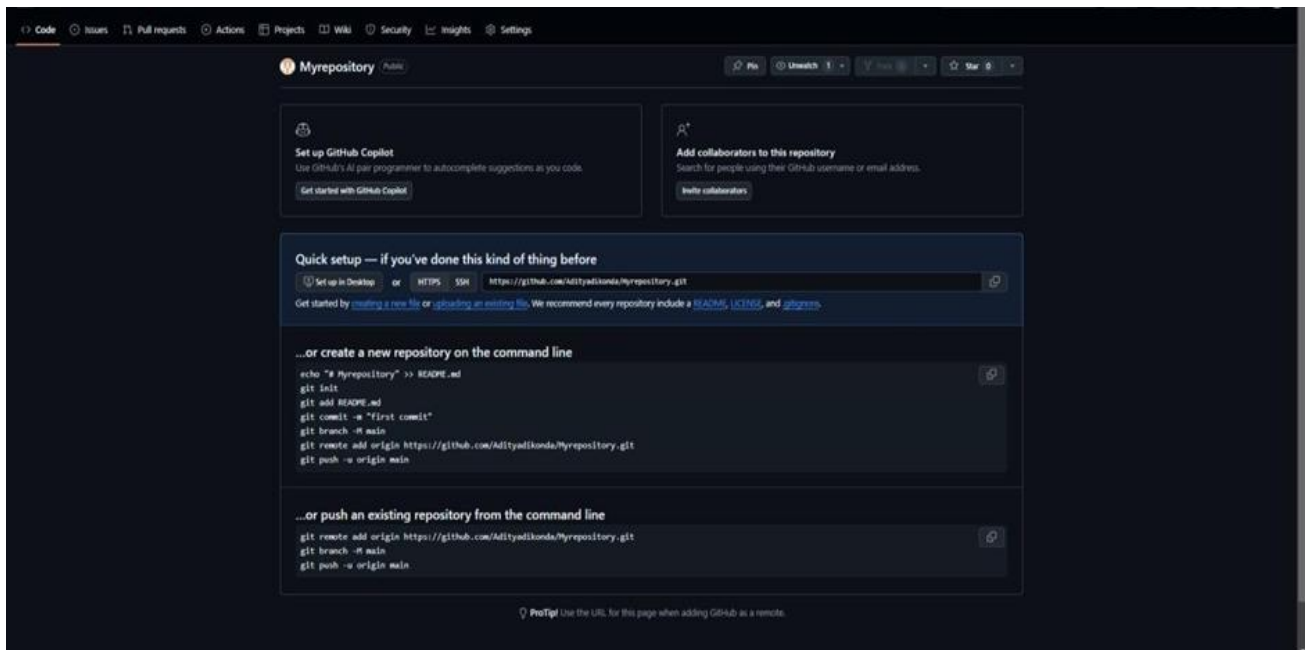
Configurations, scripts and stuff for Proxmox VE server and supporting client VMs

ReadmeActivityCustom properties1 star1 watching0 forksReport repository

ReleasesNo releases published

PackagesNo packages published

LanguagesShell 100.0%



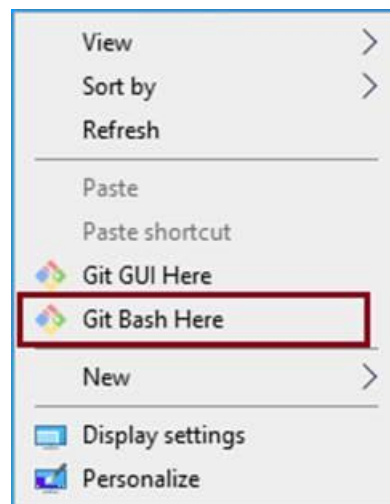
Git:

Installation of GIT

```
ubuntu@tsec: ~  
File Edit View Search Terminal Help  
ubuntu@tsec:~$ sudo apt install git
```

```
ubuntu@tsec: ~  
File Edit View Search Terminal Help  
ubuntu@tsec:~$ git version  
git version 2.17.1  
ubuntu@tsec:~$
```

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

```
ubuntu@tsec: ~  
File Edit View Search Terminal Help  
ubuntu@tsec:~$ git version  
git version 2.17.1  
ubuntu@tsec:~$ git  
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]  
        [--exec-path[=<path>]] [--html-path] [--man-path  
] [--info-path]  
        [-p | --paginate | --no-pager] [--no-replace-obj  
ects] [--bare]  
        [--git-dir=<path>] [--work-tree=<path>] [--names  
pace=<name>]  
        <command> [<args>]
```

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
```



```

C:\Users\203>cd Desktop

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

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

C:\Users\203\Desktop\git-dvcs>git config -global
error: key does not contain a section: -global

C:\Users\203\Desktop\git-dvcs>git config --global user.name "zwar_alt"

C:\Users\203\Desktop\git-dvcs>git config --global user.email "mhatreswar@gmail.com"

C:\Users\203\Desktop\git-dvcs>git config --global --list
user.name=zwar_alt

```

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)

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 b4efde6b8fb8fbedfaf508c9550c0dc2a39bad62902 (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>

```

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

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk (*).

Owner *

 zwar-alt

Repository name *

/ time-tracker

time-tracker is available.


By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

Maven training - time tracker project

☒ Copy the `main` branch only

Contribute back to taylor-training/time-tracker by adding your own branch. [Learn more.](#)

 You are creating a fork in your personal account.

Create fork

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

```

C:\Users\203\Desktop\git-dvcs\git-demo-project>git remote add origin2 https://ghp_sd0N6XSvmr0y9S1LK7nw
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>|

```

```

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

```

ADMIN@DESKTOP-J582V2L MINGW64 ~/Desktop/git-dvcs/git-demo-project (master)
$ cat index.html
<!doctype html>
<html>
  <head>
    <title>FETCH....HAPPY NEW YEAR 2020</title>
  </head>
  <body>
    <p>This is an example paragraph. Anything in the <strong>body</strong> tag will appear on the page, just like this <strong>p</strong> tag and its contents.</p>
  </body>
</html>

ADMIN@DESKTOP-J582V2L MINGW64 ~/Desktop/git-dvcs/git-demo-project (master)
$ git merge
Merge made by the 'recursive' strategy.
 index.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

ADMIN@DESKTOP-J582V2L MINGW64 ~/Desktop/git-dvcs/git-demo-project (master)
$ cat index.html
<!doctype html>
<html>
  <head>
    <title>HAPPY NEW YEAR 2020</title>
  </head>
  <body>
    <p>This is an example paragraph. Anything in the <strong>body</strong> tag will appear on the page, just like this <strong>p</strong> tag and its contents.</p>
  </body>
</html>

```

CONCLUSION:

Thus, we have successfully understood the version control system ,install git and created a Github account.