**Version Control System**

**Version Control:** Version control is a system that helps track changes to files, allowing multiple people to collaborate efficiently, revert to previous versions, and manage different versions of a project. It is commonly used in software development to manage source code.

It is the practice of tracking and managing changes to software code. Essentially, it's a system that records changes to a file or set of files over time so that you can recall specific versions later.

**Types of Version Control Systems:**

1. **Local Version Control**

Stores versions on a local machine.

1. **Centralized Version Control (CVCS)**

Uses a central server where all versions are stored (e.g., SVN (**subversion)**).

1. **Distributed Version Control (DVCS)**

Each user has a complete copy of the repository (e.g., Git, Mercurial, Bit Keeper, Bitbucket, azure repo, GitLab and Bazaar).

**Disadvantages of Centralized Version Control Systems (CVCS) Over Distributed Version Control Systems (DVCS)**

1. **Single Point of Failure**
   * If the central server crashes, all history and version control operations are lost unless there is a backup.
   * In **DVCS**, every user has a complete copy of the repository, reducing data loss risks.
2. **Requires Continuous Internet Connection**
   * Users must be connected to the central server to commit changes, check history, or perform version control operations.
   * In **DVCS**, developers can work offline and sync changes later.
3. **Slower Performance**
   * Every operation (commit, diff, log, etc.) requires communication with the central server, making it slower.
   * In **DVCS**, most operations happen locally, making it much faster.
4. **Limited Collaboration**
   * Developers must always push their changes to the central repository, leading to potential conflicts.
   * **DVCS** allows developers to share code with each other before pushing it to the main repository.
5. **Difficult Branching and Merging**
   * CVCS often makes branching and merging complex, leading to merge conflicts and slower development.
   * **DVCS** (like Git) makes branching easy, lightweight, and fast.
6. **Higher Server Load**
   * Since every operation depends on the central server, heavy usage can slow it down.
   * **DVCS** reduces server load as users perform most operations locally.
7. **Security Risks**
   * If the central server is hacked or corrupted, all data might be compromised.
   * **DVCS** reduces risk since every user has a complete copy of the repository.

### ****Differences between Distributed Version Control Systems (DVCS) and Centralized Version Control Systems (CVCS)****

| **Feature** | **DVCS (Distributed Version Control System)** | **CVCS (Centralized Version Control System)** |
| --- | --- | --- |
| **Repository Structure** | Every user has a full copy of the repository, including history. | A single central server holds the repository, and users get only working copies. |
| **Offline Work** | Can commit, view history, and create branches locally without an internet connection. | Requires an active connection to the central server for most operations. |
| **Performance** | Faster operations (like commits, diffs, and history lookups) since everything is local. | Slower because every operation requires communication with the server. |
| **Collaboration** | Changes are shared by pushing/pulling between distributed repositories. | Users commit changes directly to the central server. |
| **Backup & Recovery** | Multiple copies of the repository exist, reducing the risk of data loss. | If the central server fails, all version history may be lost unless backups exist. |
| **Branching and Merging** | Lightweight, flexible, and efficient branching. | Branching can be complex and slow. |
| **Security & Access Control** | Harder to enforce strict access control since everyone has a full copy. | Easier to manage access control since everything is centralized. |
| **Examples** | Git, Mercurial, Fossil, Bazaar, BitKeeper | SVN (Subversion), Perforce, CVS, TFS |
| **Local server** | Every client (developer) has its own server which is installed locally while initializing it. | No local server is installed, It commits the changes of code directly to the Central server (master repository. |

**Note:** Both the DVCS and the CVCS are works on server and client principle.

**Basics of GitHub**

**Git** is a **Distributed Version Control System (DVCS)** used to track changes in files and collaborate on projects. It is widely used in software development to manage source code efficiently.

**Key Features of Git:**

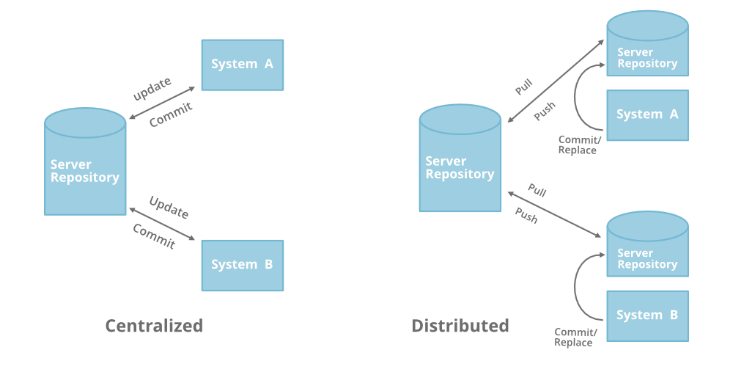
1. **Distributed System** – Every developer has a full copy of the repository, allowing offline work.
2. **Fast and Efficient** – Operations like commits, branching, and merging are quick since they happen locally.
3. **Branching and Merging** – Allows easy and flexible branching, making collaboration smooth.
4. **Data Integrity** – Uses cryptographic hashing (SHA-1) to ensure data integrity.
5. **Collaboration** – Supports multiple workflows with remote repositories (e.g., GitHub, GitLab, Bitbucket).

**Basic Git Workflow:**

1. **Initialize a Repository:** git init
2. **Clone a Repository:** git clone <repo\_url>
3. **Check Repository Status:** git status
4. **Add Changes to Staging:** git add <file>
5. **Commit Changes:** git commit -m "Message"
6. **Push Changes to Remote:** git push origin <branch>
7. **Pull Latest Changes:** git pull origin <branch>
8. **Create a Branch:** git branch <branch\_name>
9. **Switch Branches:** git checkout <branch\_name>
10. **Merge Branches:** git merge <branch\_name>

**Why Use Git?**

* Helps in team collaboration.
* Provides a complete history of project changes.
* Allows working on different features using branches.
* Reduces risk of data loss with distributed copies.

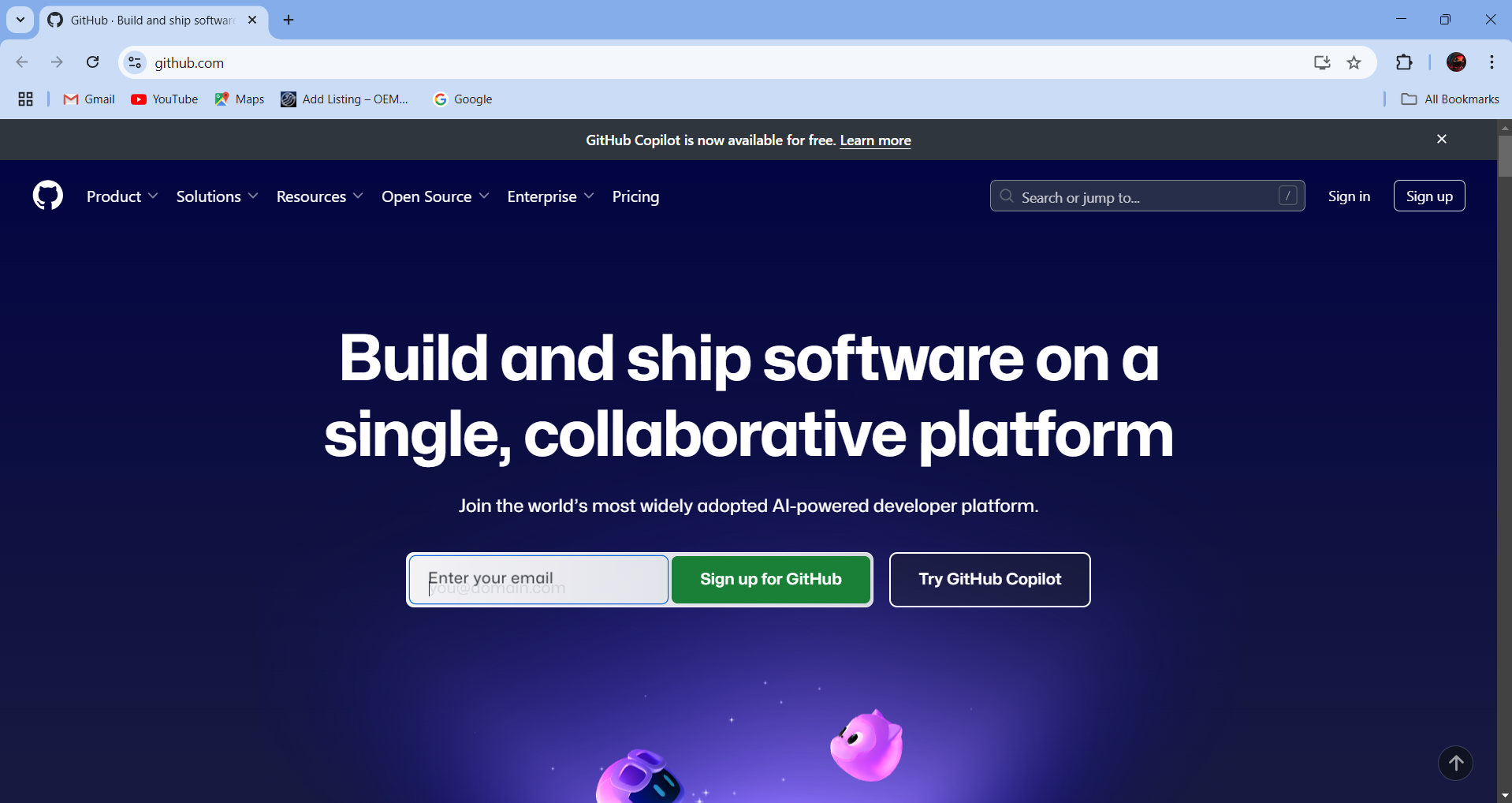


**Note1:** By usin Git (DVCS) the local repositories (local servers) are created automatically in our local machine while performing Git init command.

**Note2:** Whereas SVN (CVCS) as no local repository installation in our local machine it directly commits the changes or communicate with main repository (centralized repository).

**Creation of GitHub account**

step1: Open any Browser and search “github.com”.

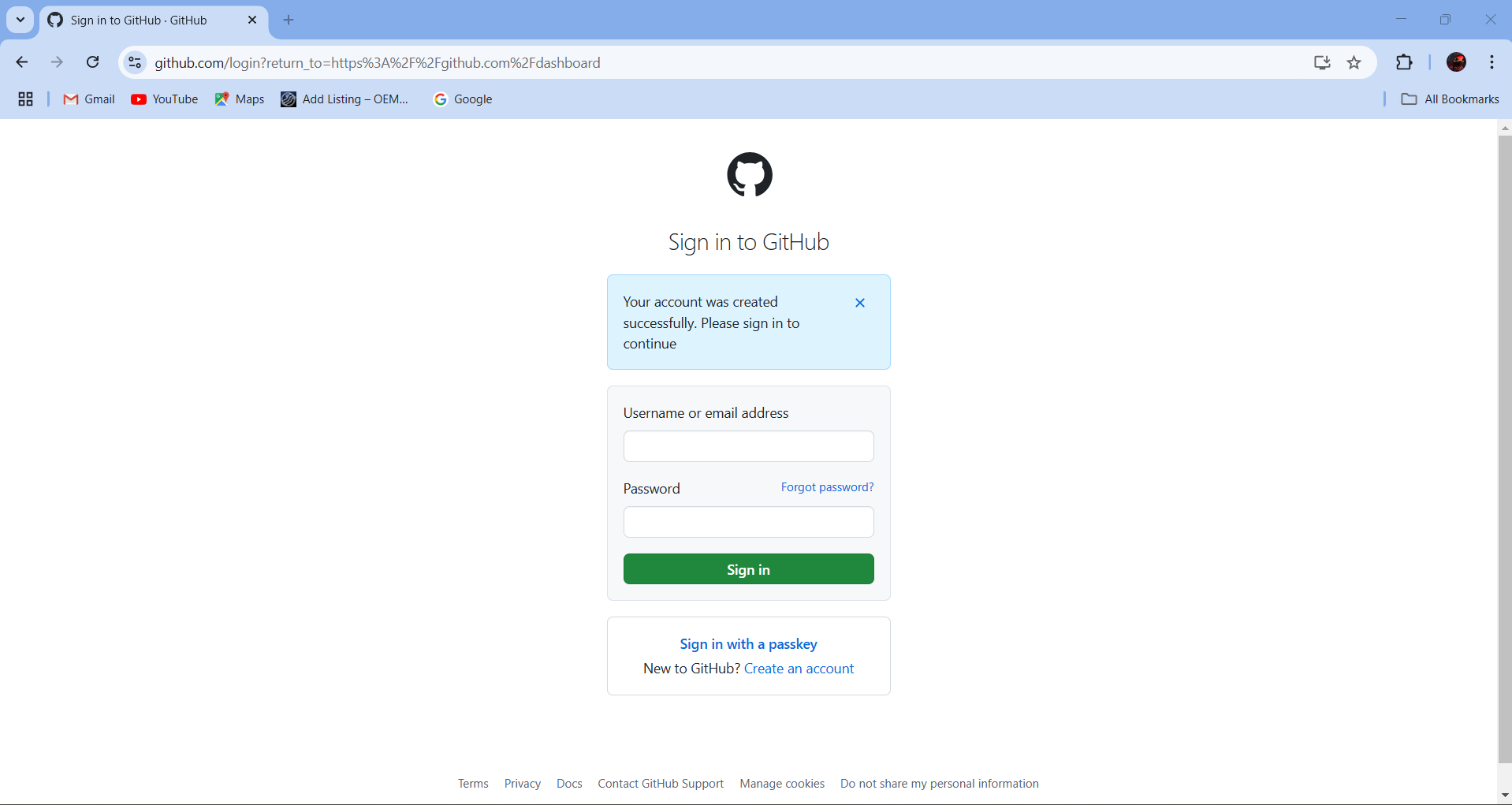


Step2: Sign in with your personal email id.

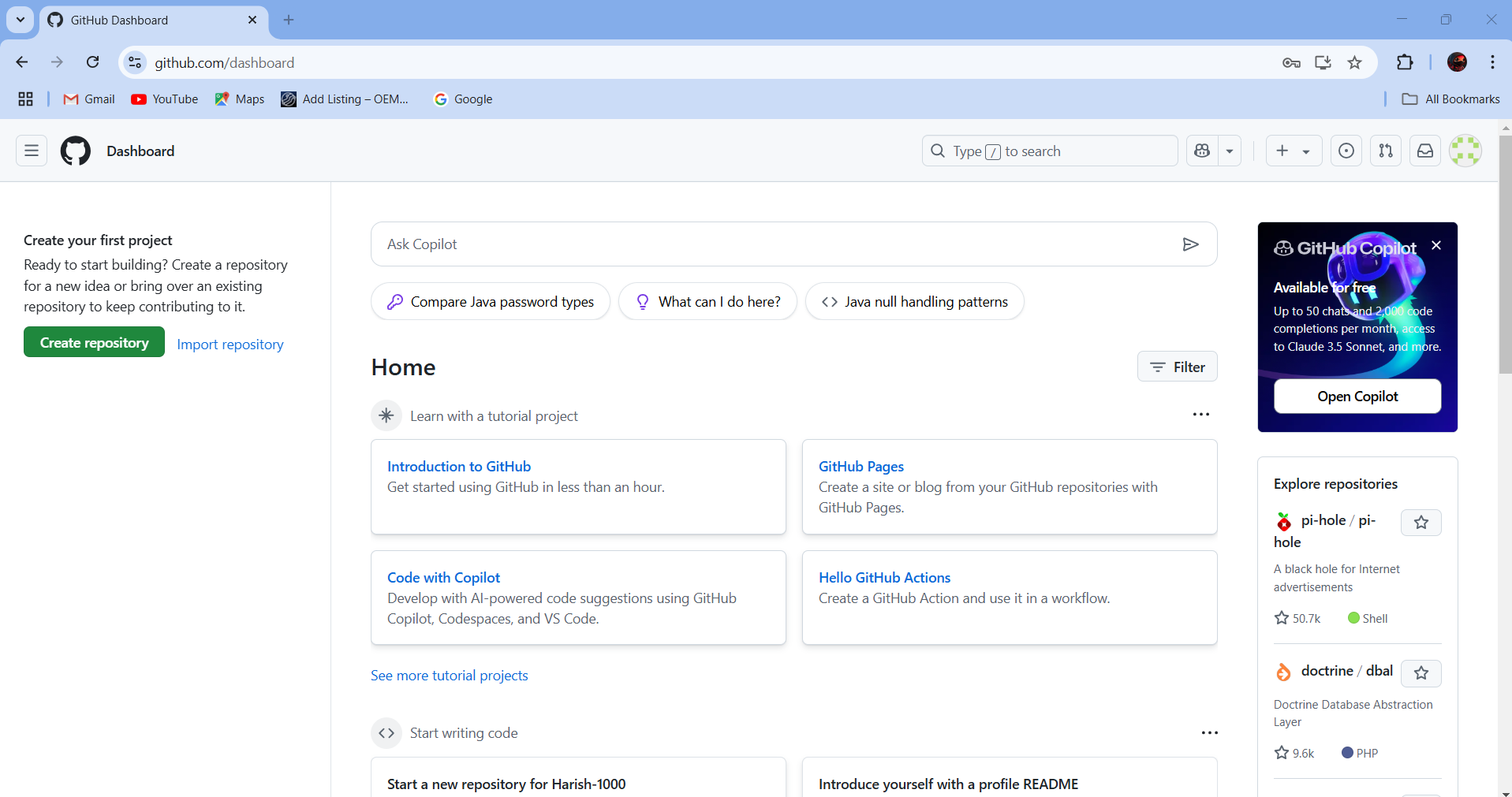
step3: And set the password and user name.

step4: Then solve the puzzle.

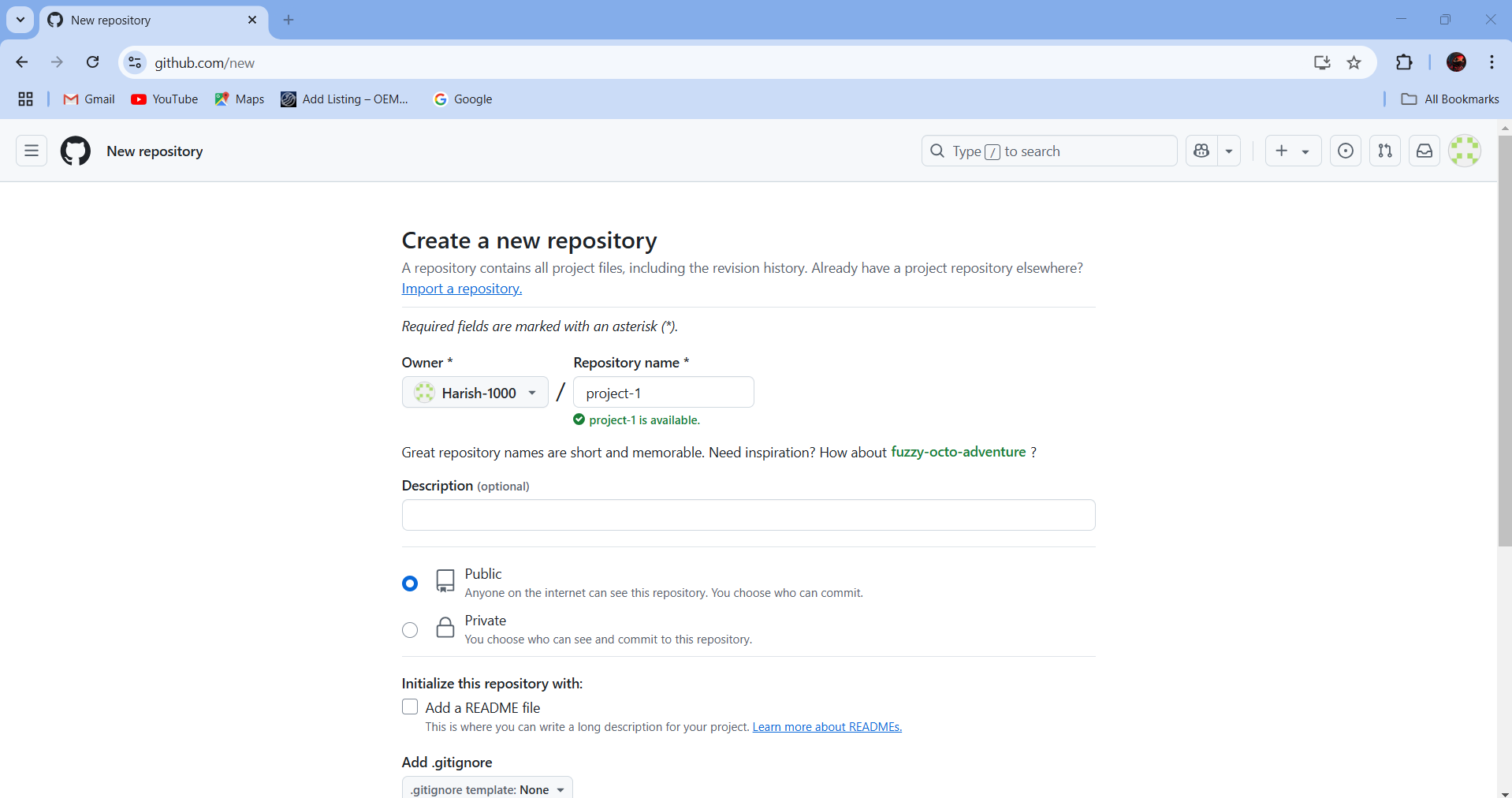
step5: Now enter the Username/email and password then login.



step6: Now you can see the Dashboard of GitHub.



step7: Create a repository by keeping it as a public.



step8: Create a file and write the any contend in it and commit changes (Save).

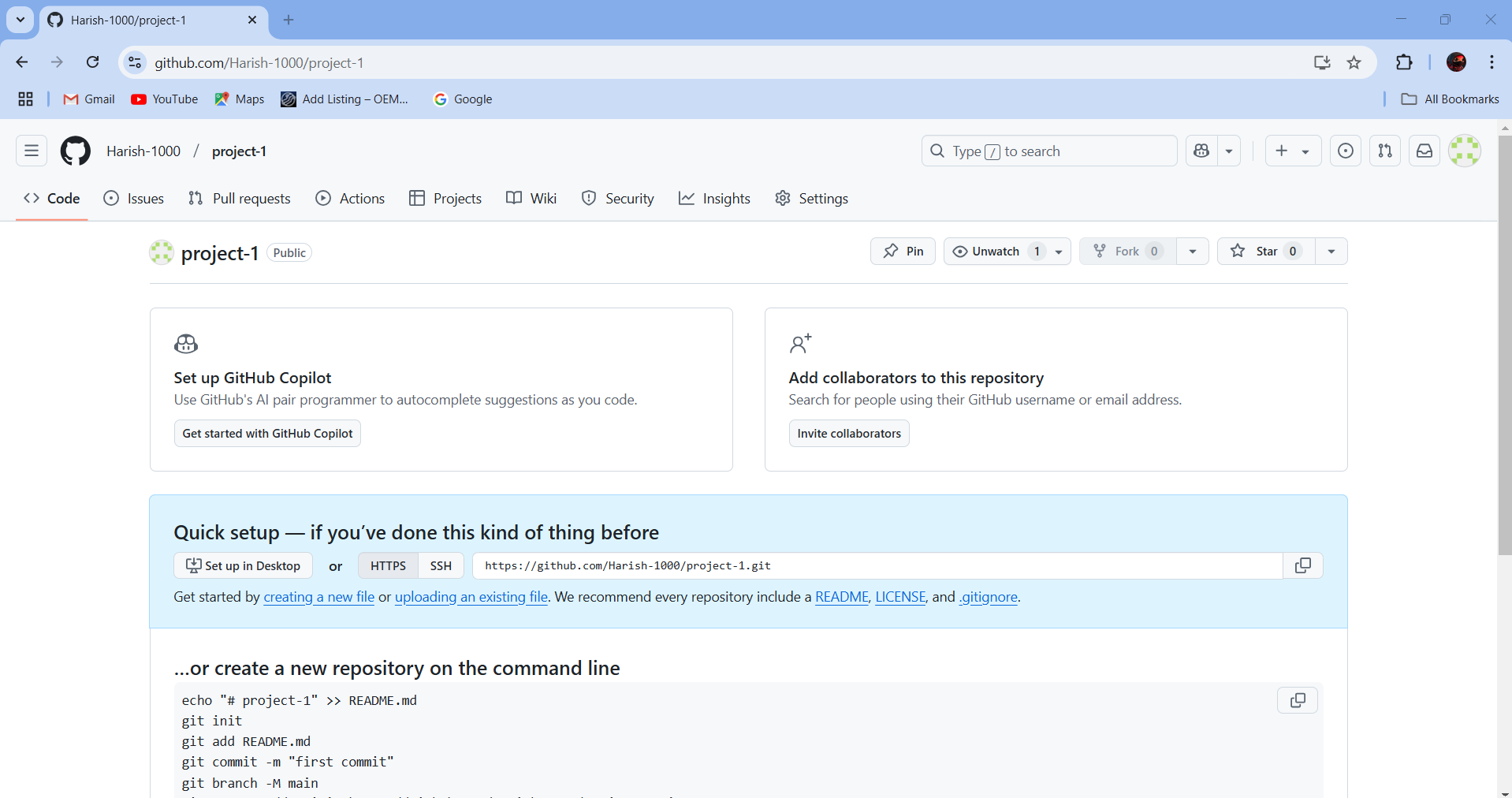


Fig: creating a file

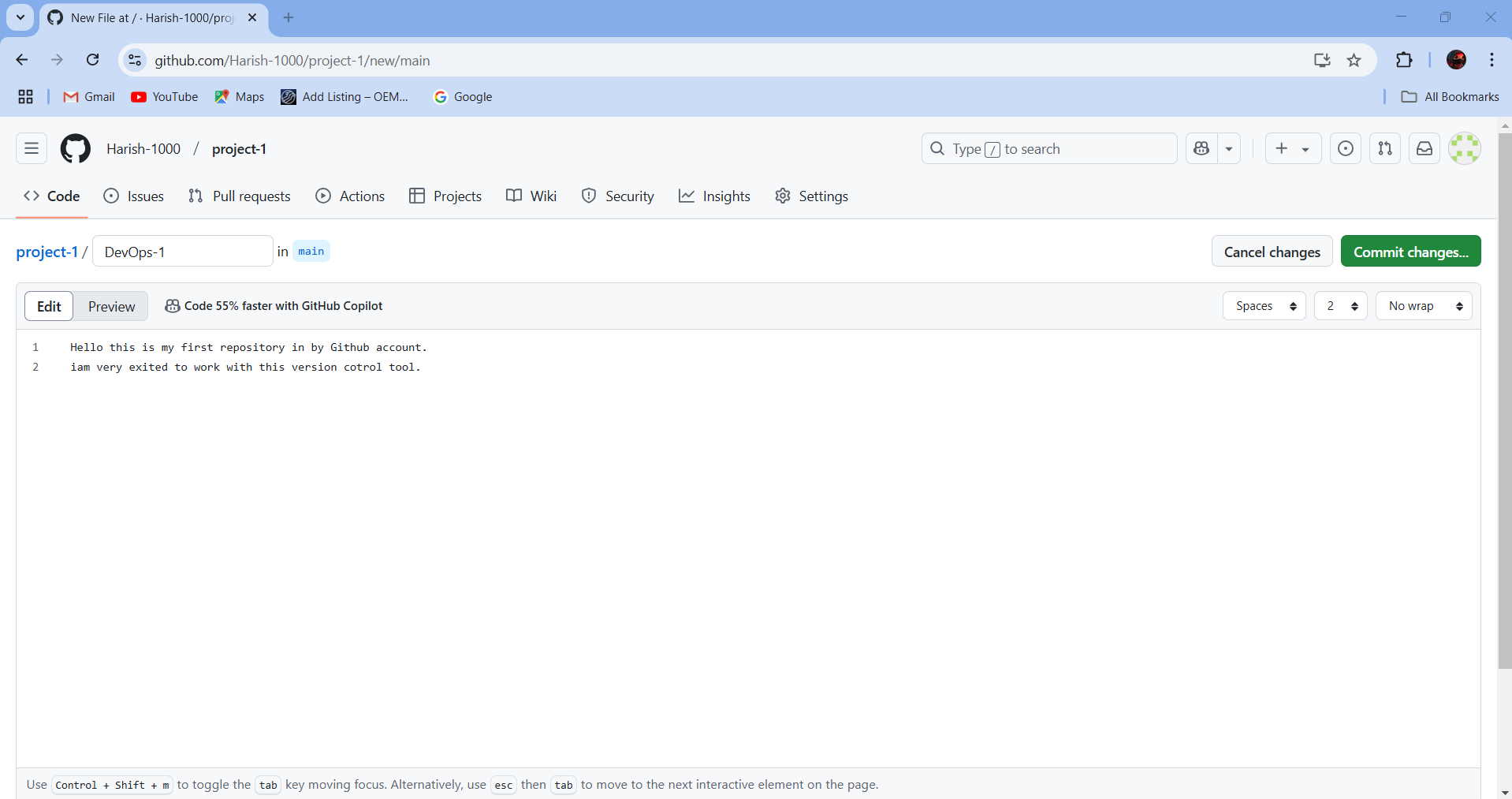


Fig: writing the content and Commit changes.

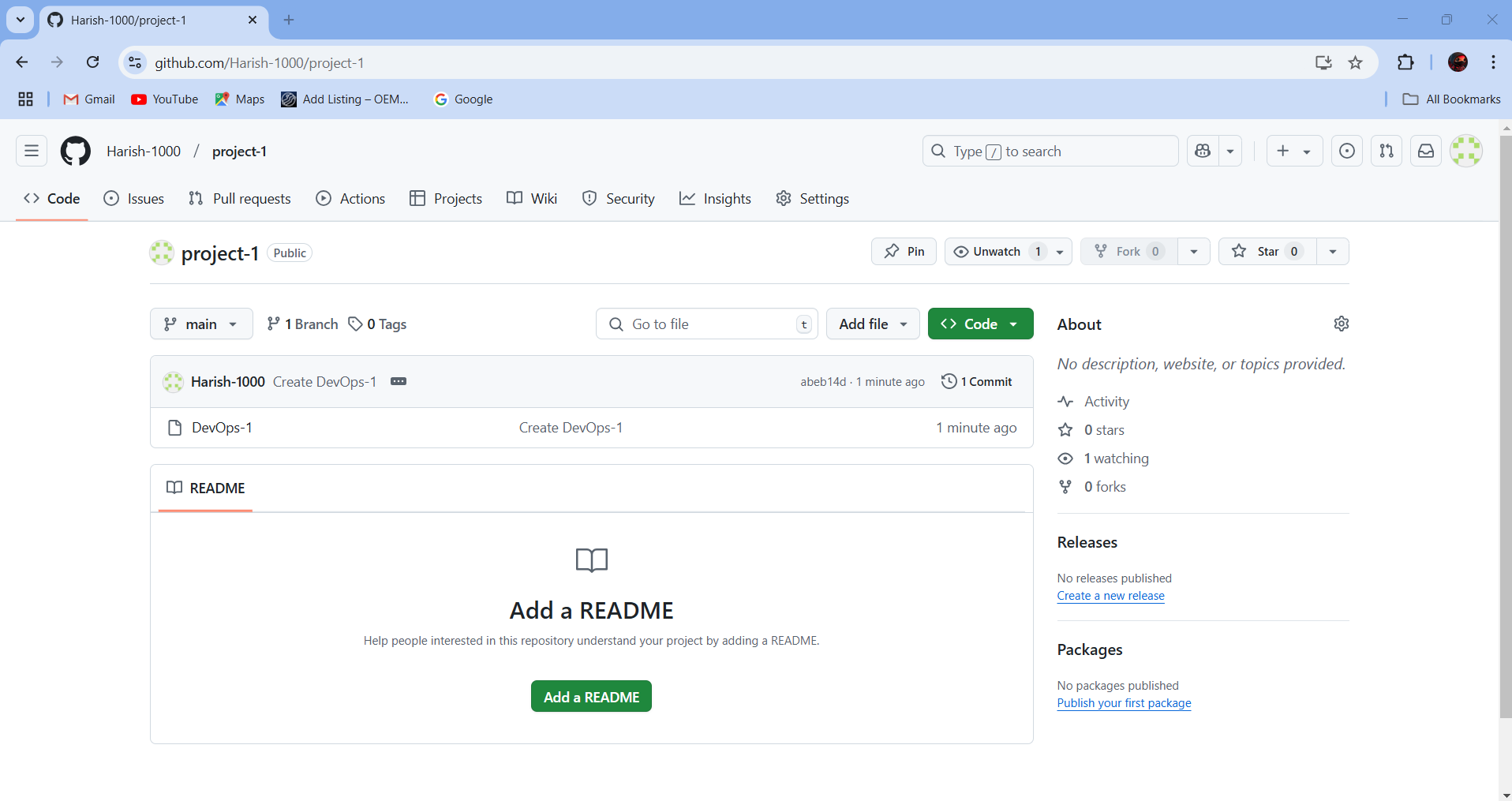
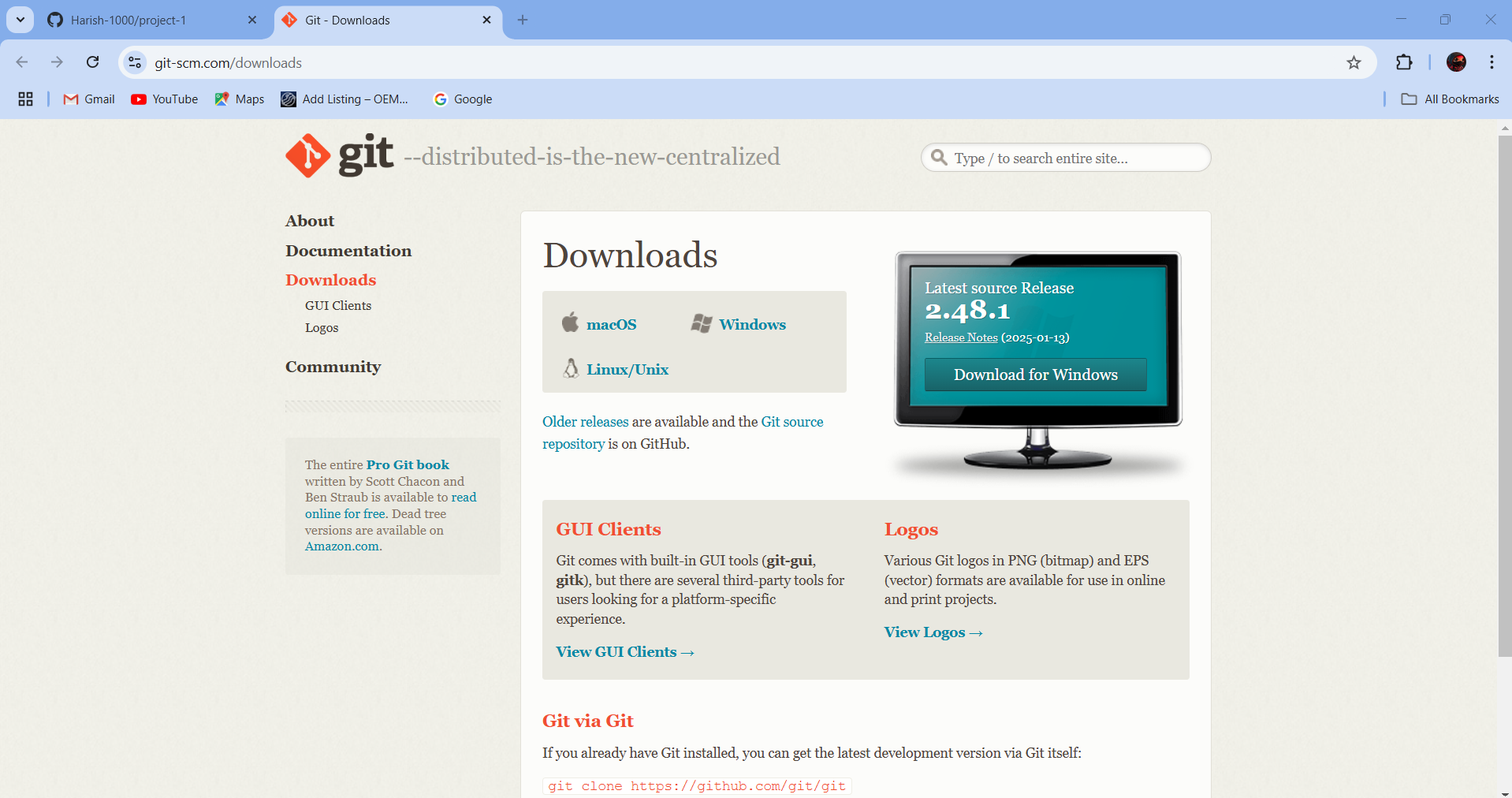


Fig: File is created.

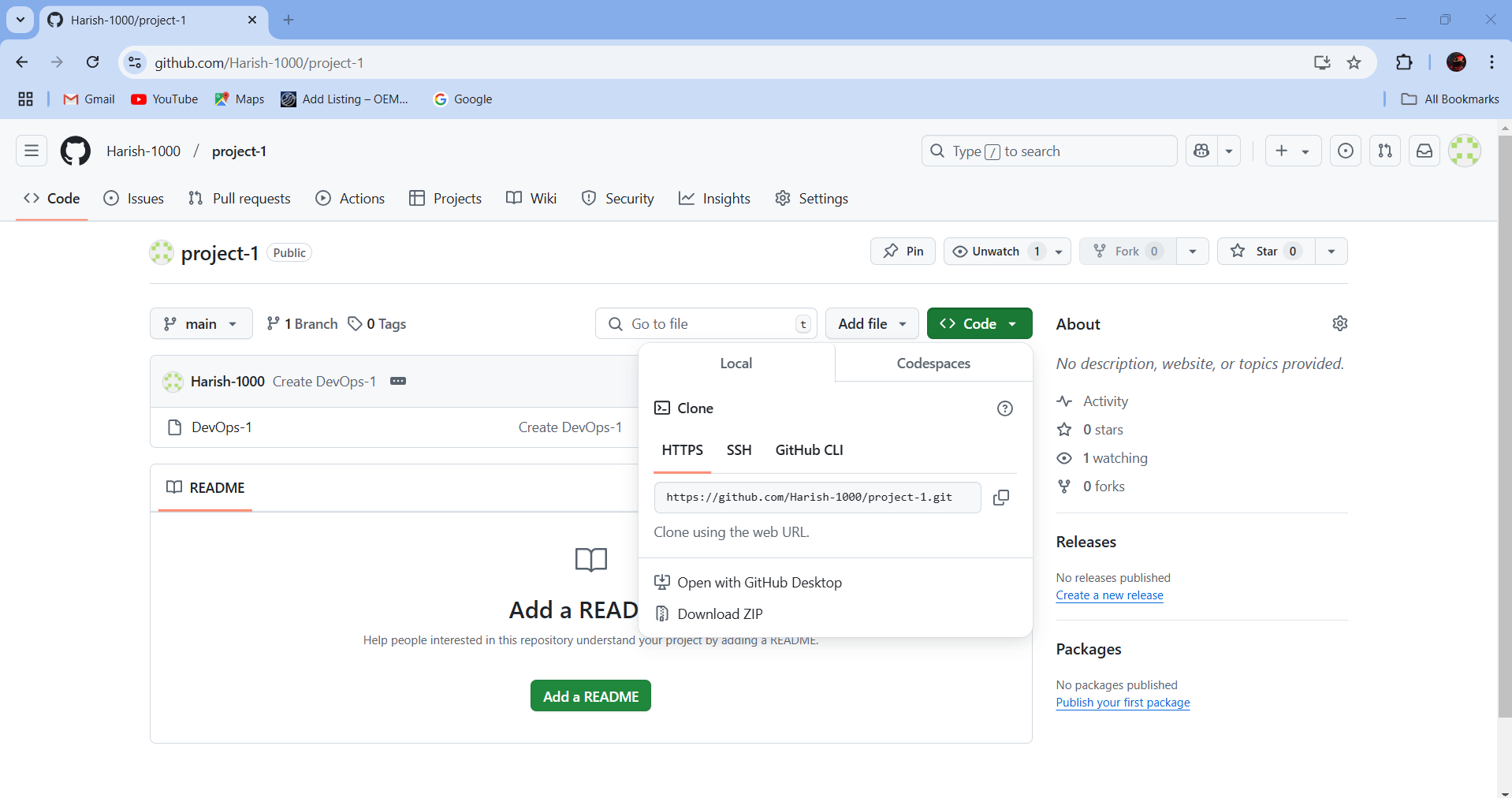
Up to now we worked on the remote (or) centralized repository by login and connecting to the internet, in order to work with it from locally we have to install Git software tool in our local machine.

**Step1:** Search for Git Download for windows in any browser.



If you downloaded git then it’s depended software’s like Git Bash, Git GUI are also installed automatically in our local machine.

By using Git Bash we can run the Git commands not only bash we can also run the Git commands from CMD, power shell and Visual studio.

**Step2:** Now in order to update the existing file (DevOps-1) which is created in our repository first we have to clone that repository (project-1) from the Git bash software tool using git clone command.

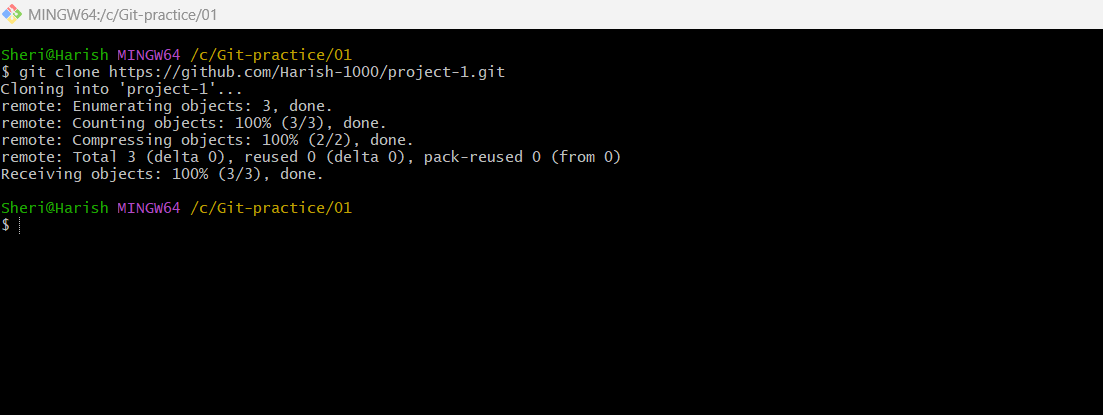
Fig: copy the URL.

Fig: Cloning of remote repository file to local machine.

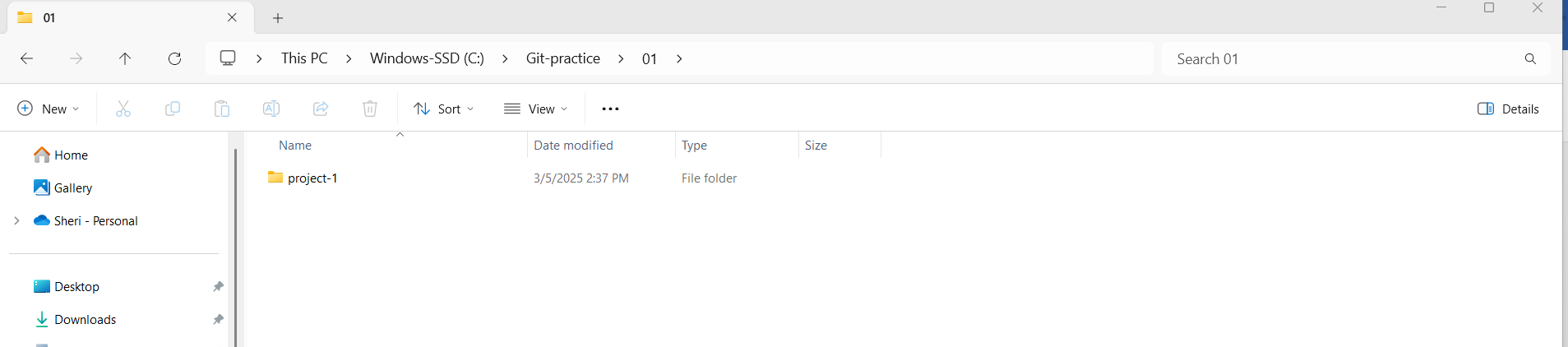
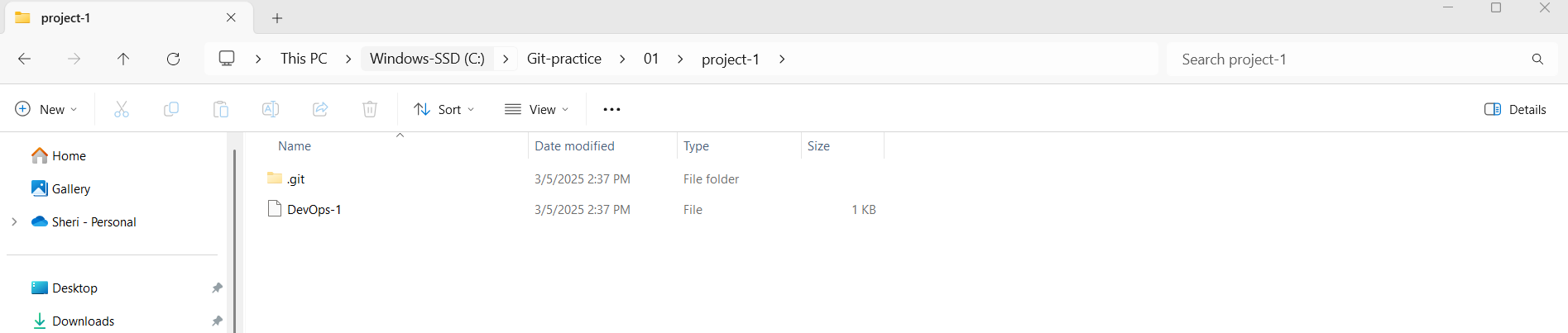


Fig: The remote repository is cloned into the local machine successfully.

The git server is installed in our local machine as shown in below figure.



The **.git folder** represents the git server which is inatalled in our local machine after cloning the remote repository (project-1).

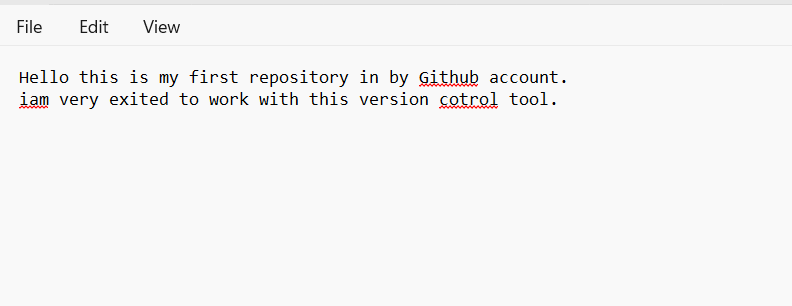


Fig: Content of the file (DevOps-1).

Now edit the existing contend which is present in the file (DevOPs-1) and push it to the remote repository.

While pushing the updated file to the remote repository (project-1) it under goes three main phases or staging areas as shown in below figure.

Remote repo

Or

Server repo

Commit area

Working area

Git commit

Index area

Git add .

Fig: Different Phases of Git commands.

Step3: Edit the existing file content (DevOps) from local machine.

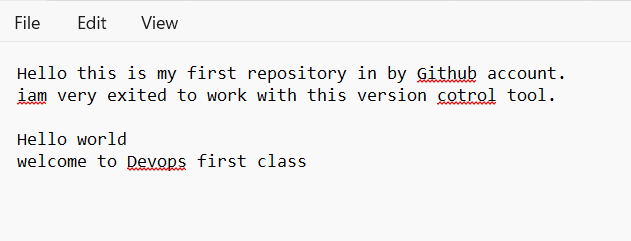


Fig: Edited content

Step4: Create a file from our local machine using git Bash.

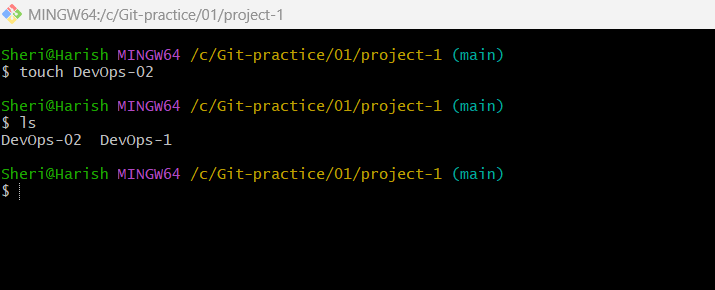


Fig: File (DevOps-02) is created.

This edited file (DevOps-1) and new file (DevOps-02) both are present in our local machine only.

**Step5**: Now push both Edited file (DevOps-1) & New file (DevOps-02) into the server (remote repository).

Do **git status** to show the phase or staging area of our repository or directory.

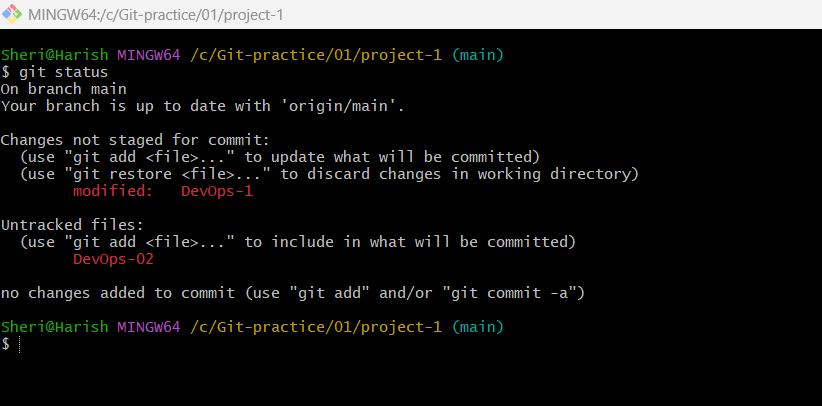


Fig: Working area of directory. (Indicates with red color).

Modified or Untracked file or directories are present at working area and indicated with Red color as show above figure.

In order to track the untracked files we use the **git add .** Command. Then it movie files to the index area as show below figure.

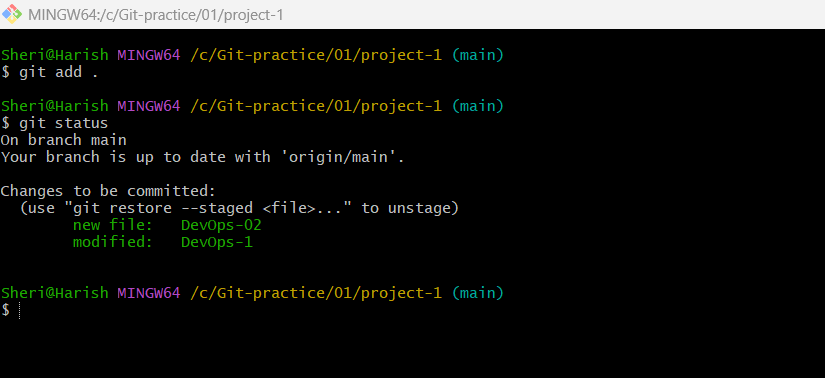
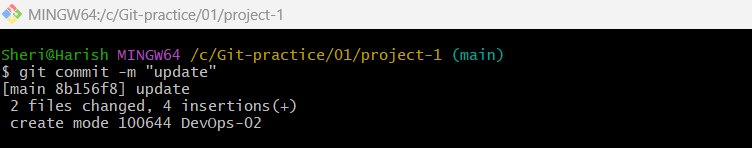


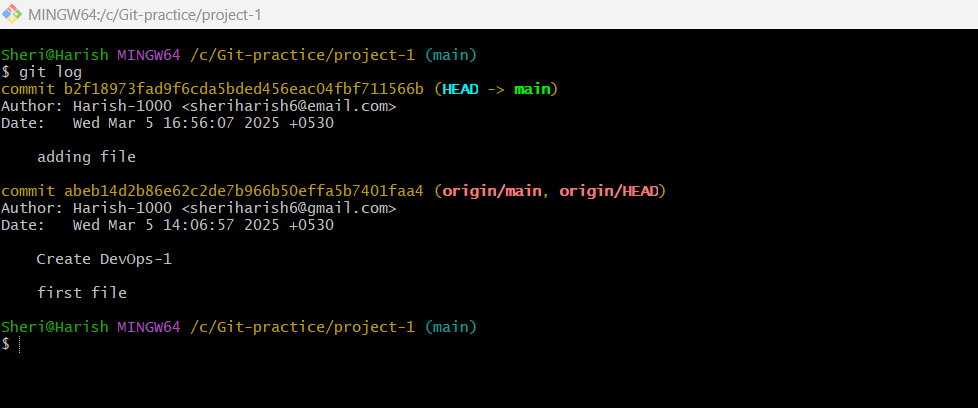
Fig: Index area of files.

Now the files are at index stage or area, in order to movie these file to the commit area we have to do **Git commit –m “any text”,** so that now they move to the commit area as shown in below figure.

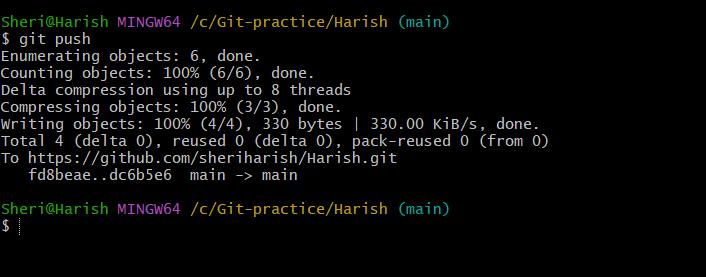


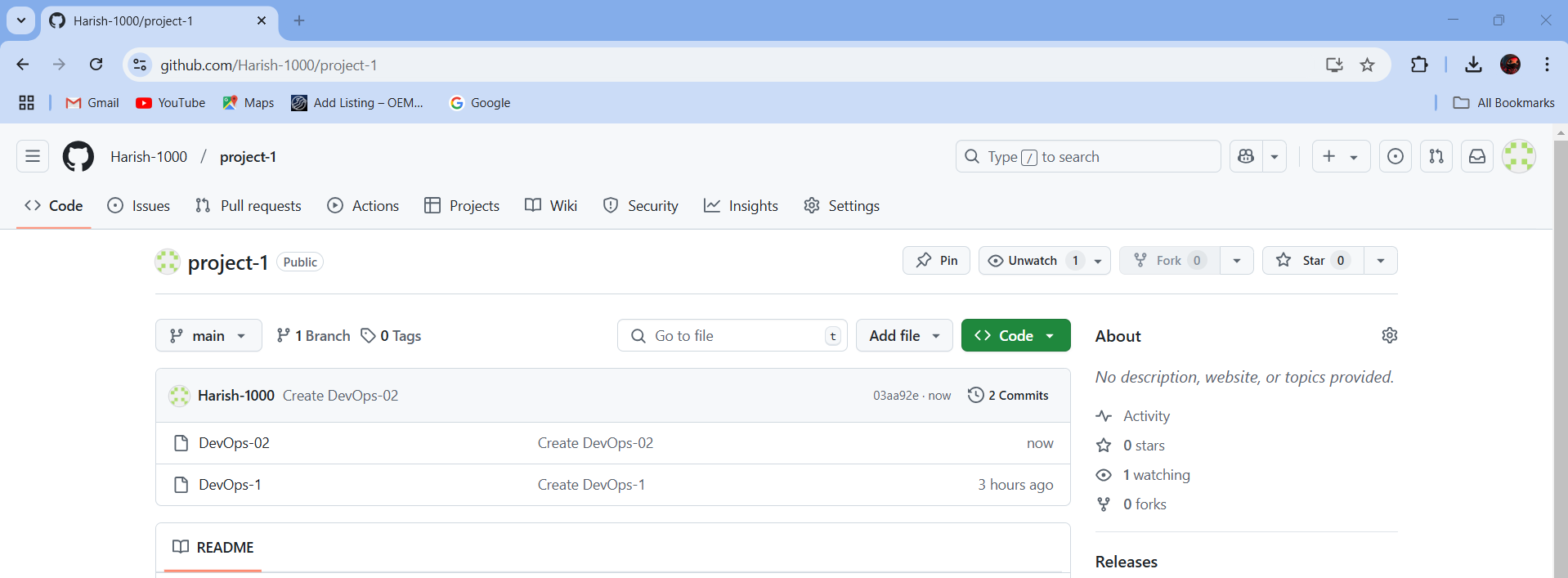
As a result the files are moved to the commit area.

Now do the “**git log”** it will show th commit history of a Git repository as shown on below figure.



Step5: Now push the file to the remote git repository (project-1).





Successfully we updated (DevOps-1) and added new file (DevOps-02) to the remote repository (Project-1).

In above task we cloned the existing project from the git and worked on it and again pushed it into the git.

The flow chat for above task is given below.

Starting point

git clone <http/gitlink>

git repository (server)

git push

git commit –m “message”

git add .

Edit or create new file