Lab File

Source Code Management

Pavani B A

A866185824001

Lab File

Source Code Management

LAB REPORT – 1

Overview of Git:

Git is a distributed version control system that tracks versions of files. It is often used to control source code by programmers who are developing software collaboratively. Design goals of Git include speed, data integrity, and support for distributed, non-linear workflows — thousands of parallel branches running on different computers.

Step 1: Downloading Git

1. Open your web browser and navigate to the official Git website: https://git-scm.com.
2. On the homepage, you will see a "**Download**" button that automatically detects your OS. Click on the "Download" button to download the appropriate installer for your operating system (Windows, macOS, or Linux).
3. Alternatively, you can manually select your OS from the website to download a specific version.

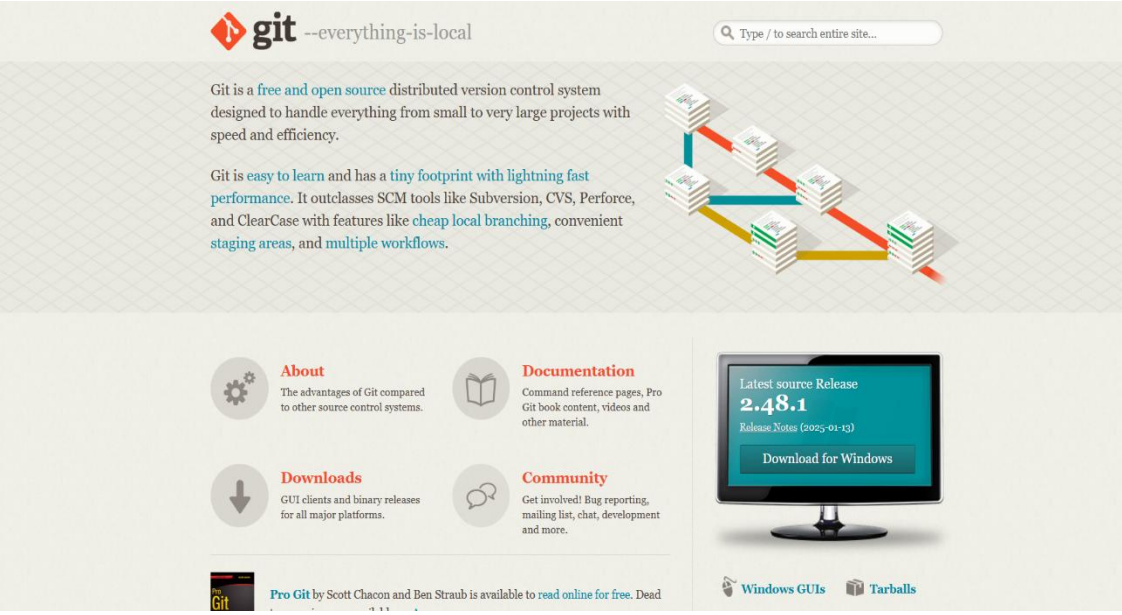


Figure - 1

Step 2: Running the Git Installer

Locate the downloaded **Git.exe** file and double-click to run it.

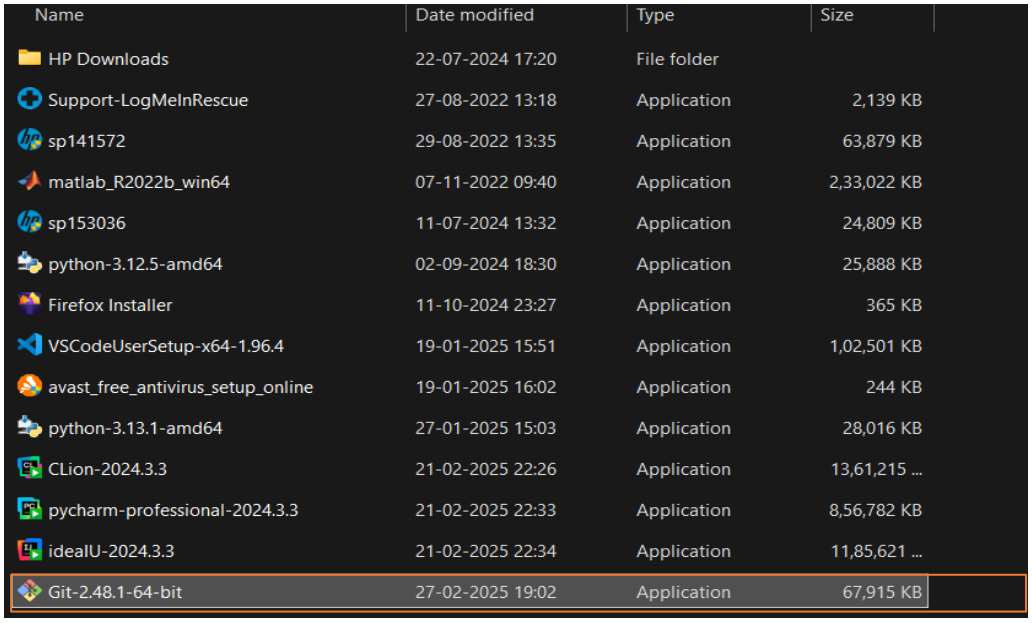


Figure – 2

Step 3: License (Terms and Conditions)

Read the **GNU** General Public License's terms and conditions and click on **Next**.

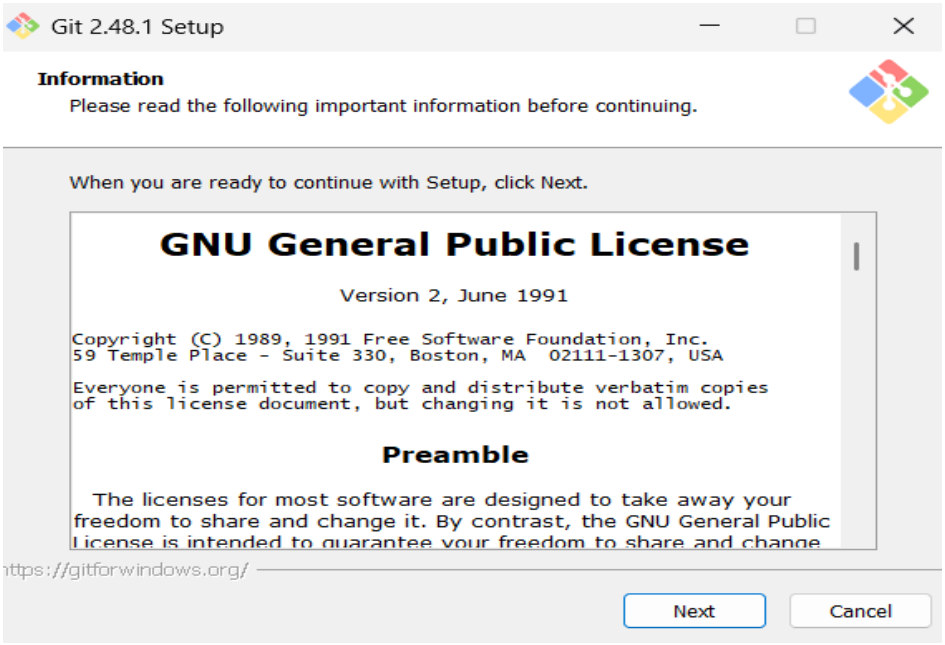


Figure - 3

Step 4: Choose Installation Location

Choose the installation location (default is **C:\Program Files\Git**) and click **Next**.

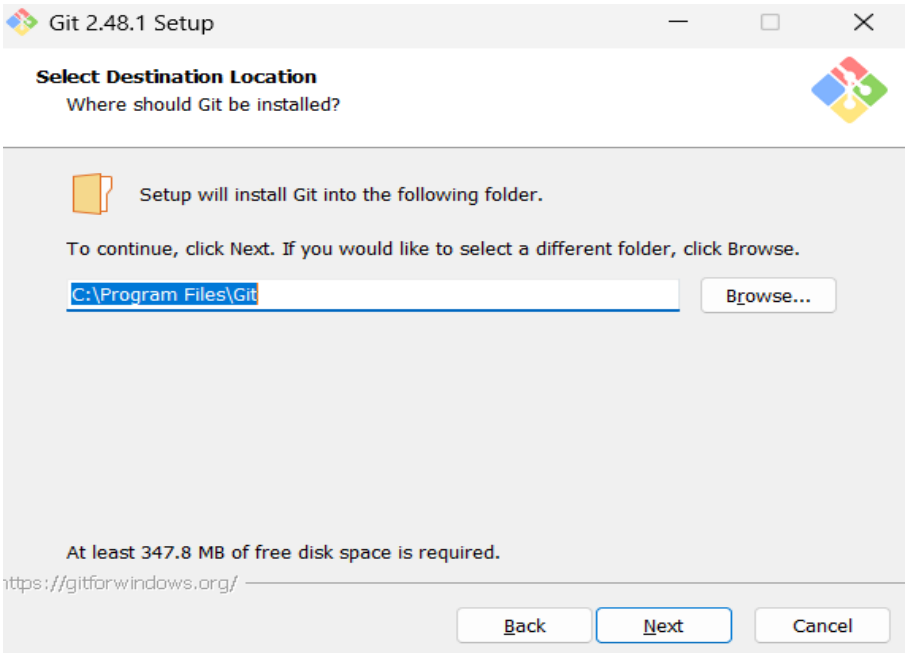


Figure – 4

Step 5: Select the Components

Select the components you want (default options are fine) and click **Next**.

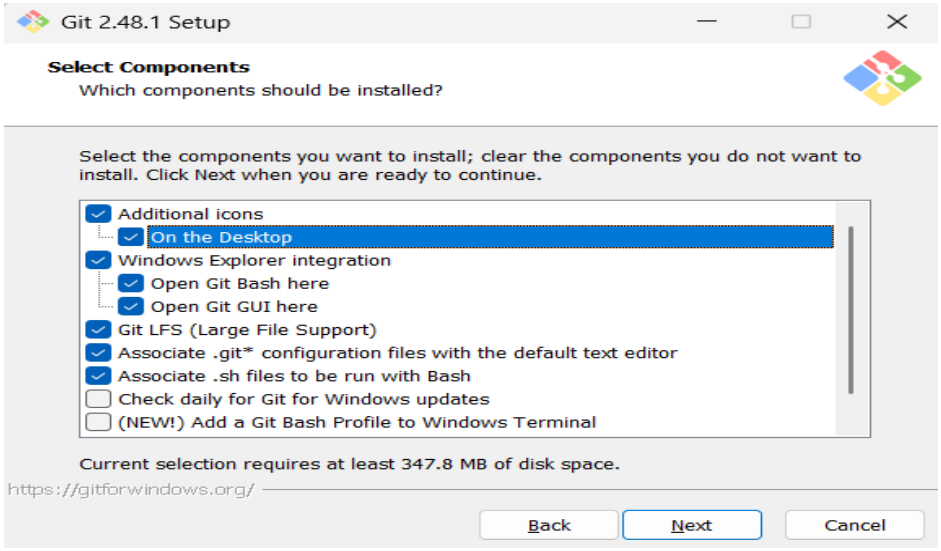


Figure – 5

Step 6: Select Start Menu Folder

Choose the Start Menu folder where Git shortcuts will be placed. By default, the folder is named **"Git"**. Keep the default name and click **Next** to Proceed.

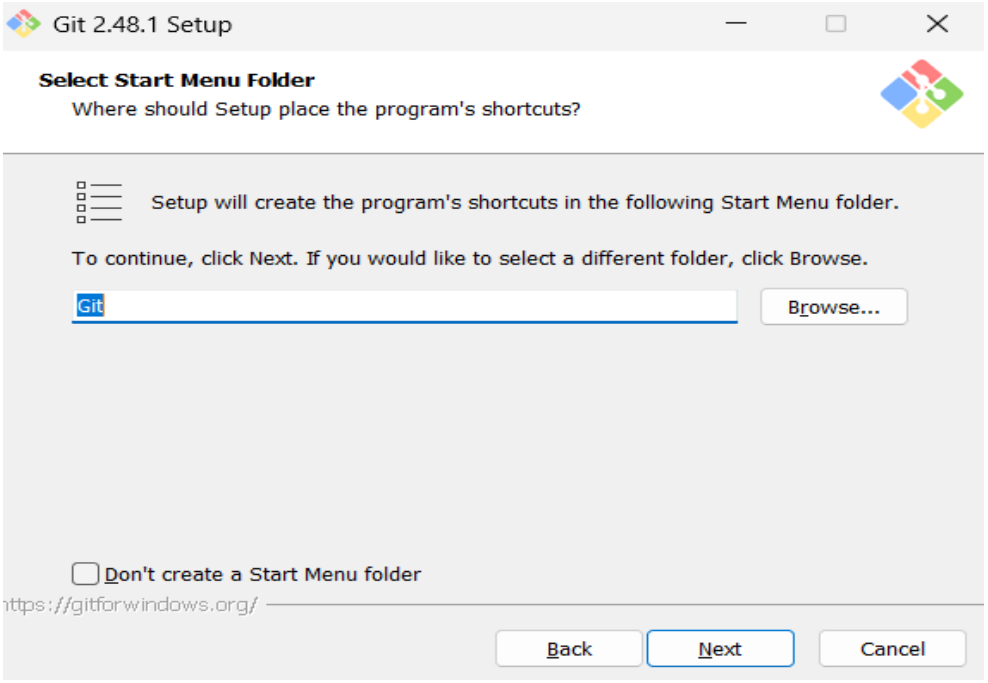


Figure – 6

Step 7: Choose the Text Editor

Choose a default text editor (select **Vim**) and Click **Next**.

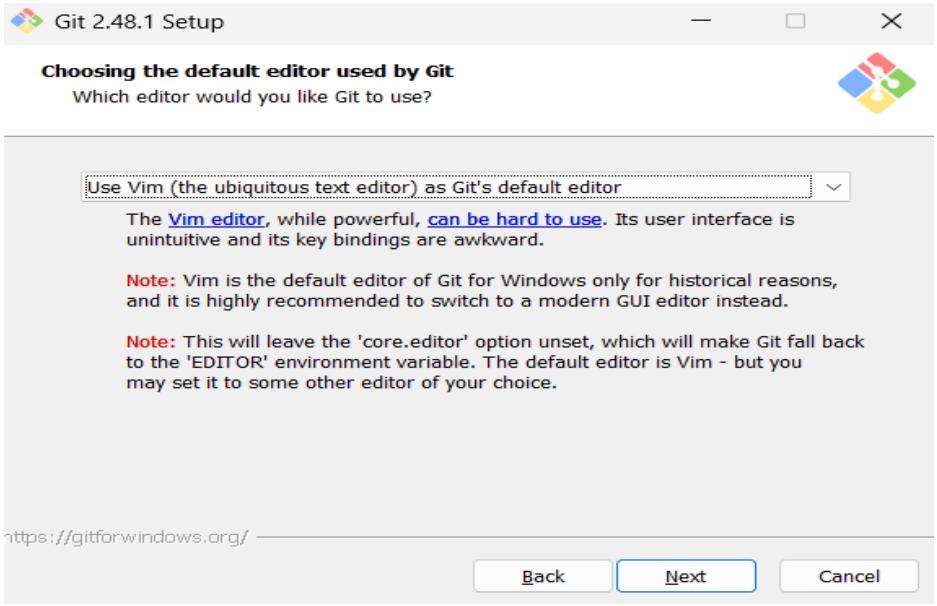


Figure – 7

Step 8: Adjusting Initial Branch Name

Choose the default name for the first branch when initializing a new Git repository. Go with **'Let Git Decide'** option setting the branch as **Master** branch and proceed with **Next**.

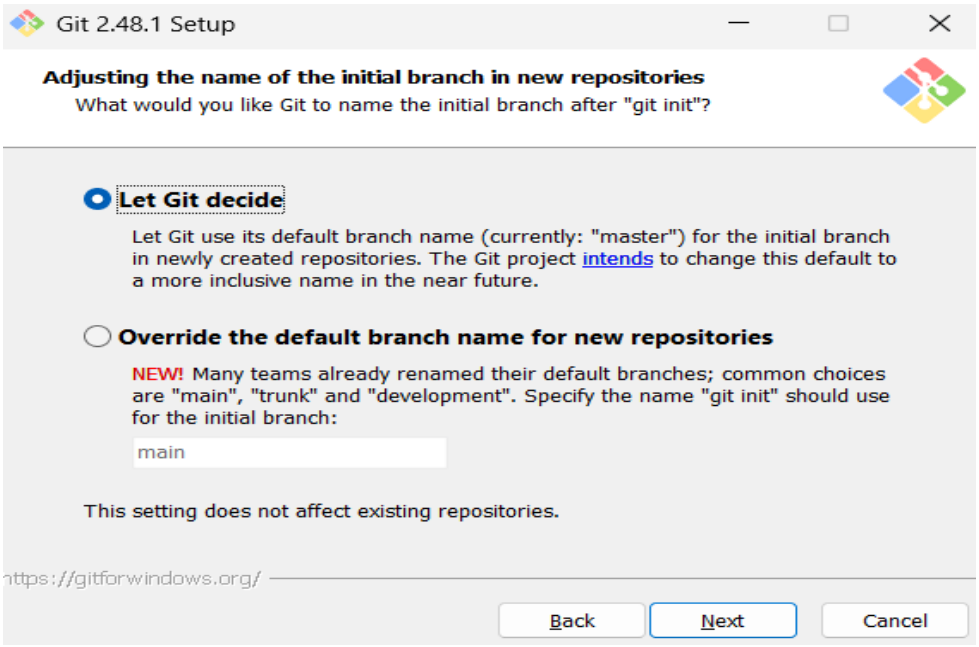


Figure – 8

Step 9: Adjusting PATH Environment

Select **Git from the command line and also from third-party software** (recommended). Click **Next**.

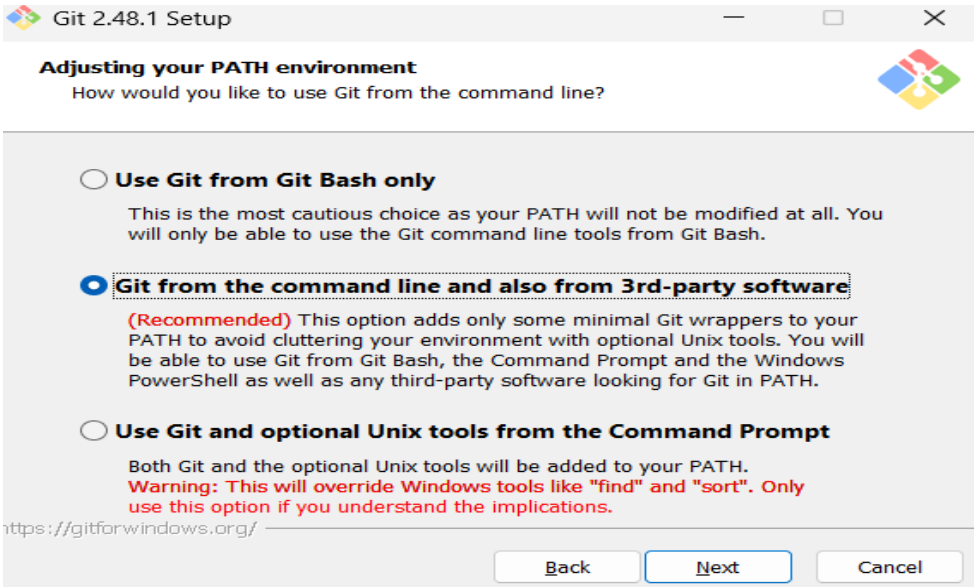


Figure – 9

Step 10: Choosing the SSH Executable

Select "**Use bundled OpenSSH**" for better compatibility and Click on **Next**.

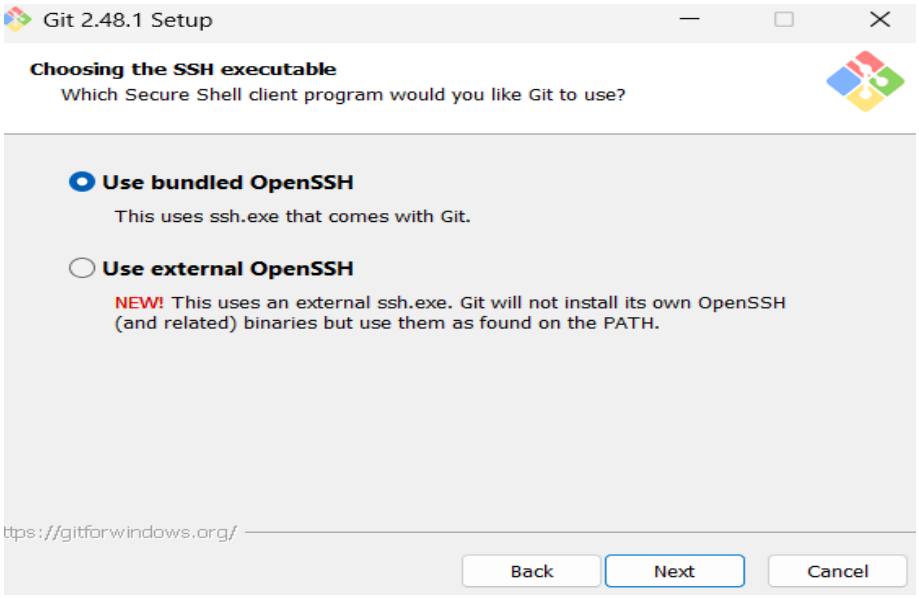


Figure - 10

Step 11: Choosing the HTTP Transport Background

Choose Use the **OpenSSL library** (default) and Click **Next**.

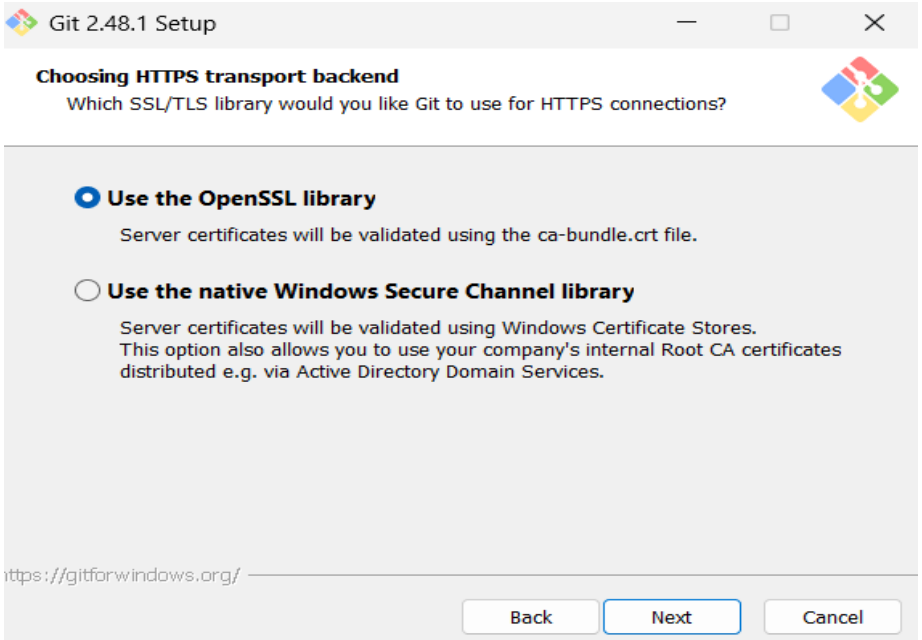


Figure – 11

Step 12: Configuring Line Ending Configs

Select **Checkout Windows-style, commit Unix-style line endings** (recommended) and Click **Next**.

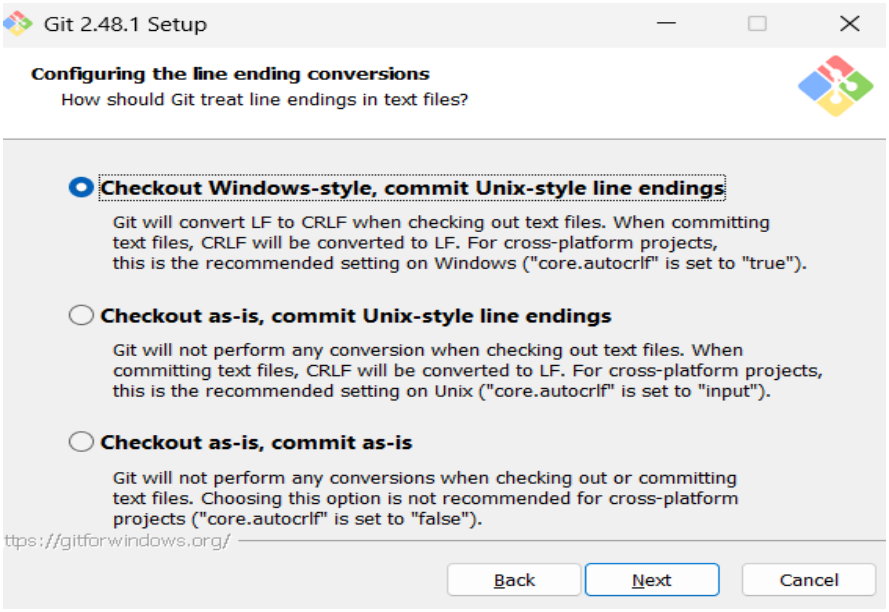


Figure - 12

Step 13: Configuring the Terminal Emulator

Select **Use MinTTY (default terminal for MSYS2**) and Click **Next**.

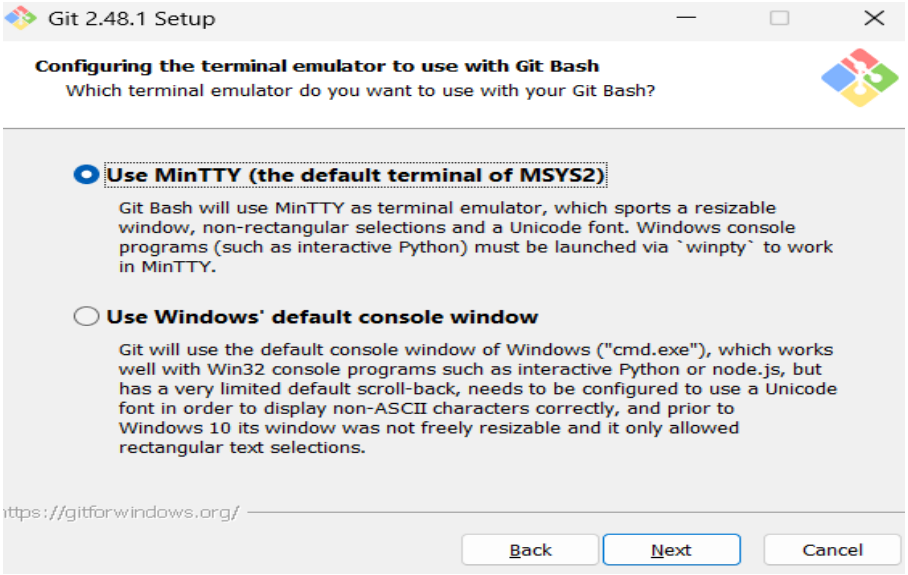


Figure- 13

Step 14: Choosing the Default Behaviour

Select **Fast-forward or Merge** (recommended) option and click **Next**.

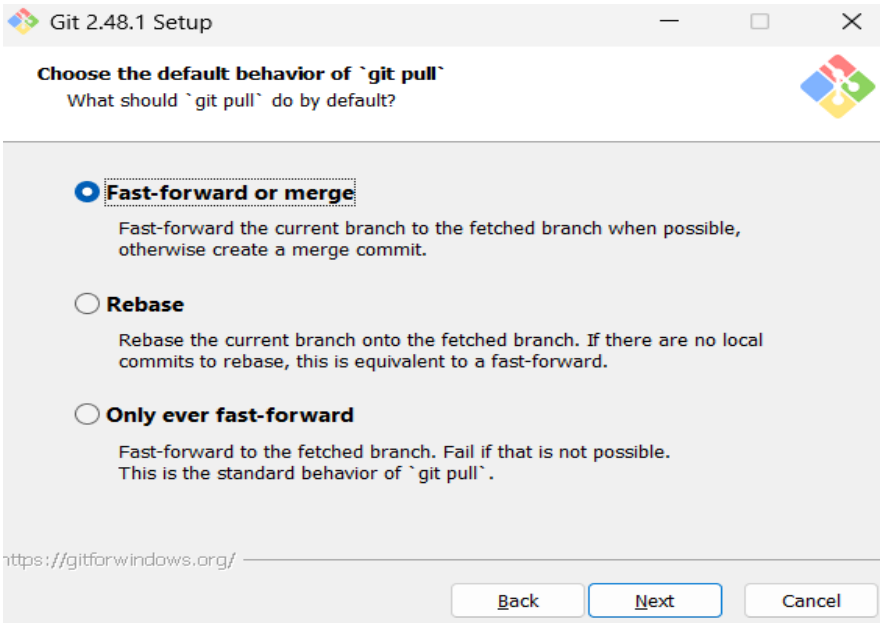


Figure – 14

Step 15: Choosing a Credential Helper

Select **Git Credential Manager** (recommended) and Click **Next**.

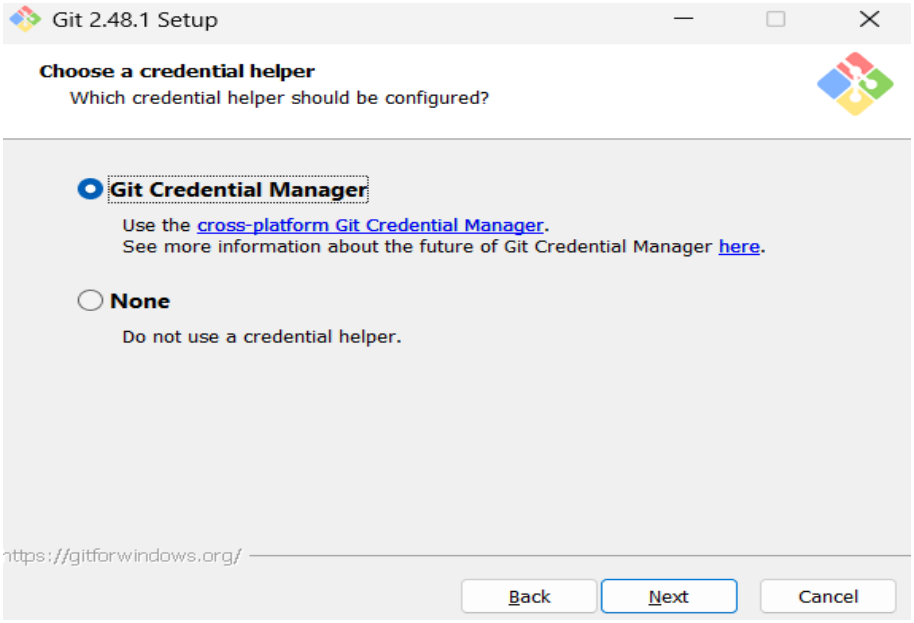


Figure – 15

Step 16: Configuring Extra Options

Select **Enable file system caching** (recommended) and Click on **Install**.

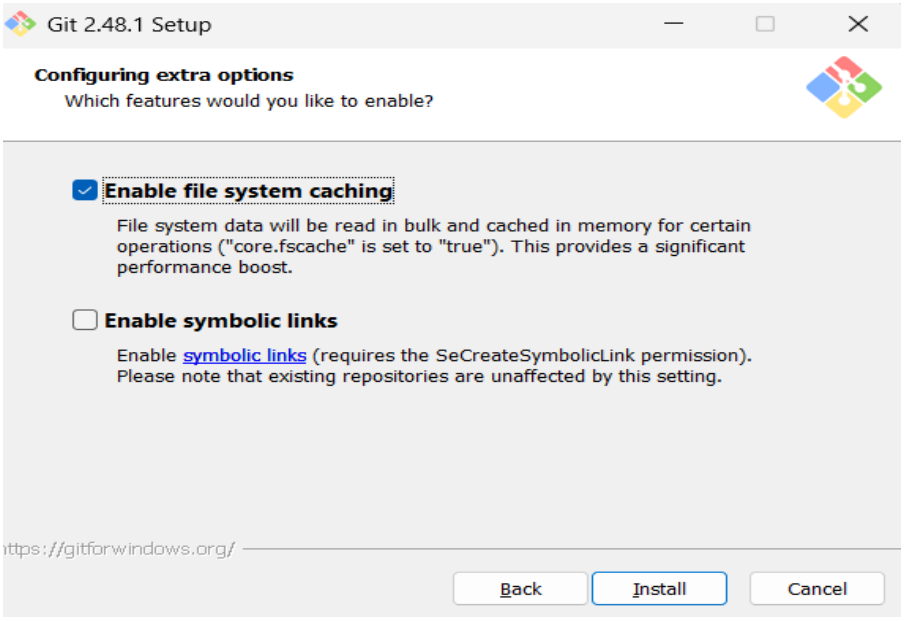


Figure – 16

Step 17: Installation Overview

A progress bar (**green bar**) will appear, indicating that Git is being installed. Wait for the installation to complete. This may take a few minutes.

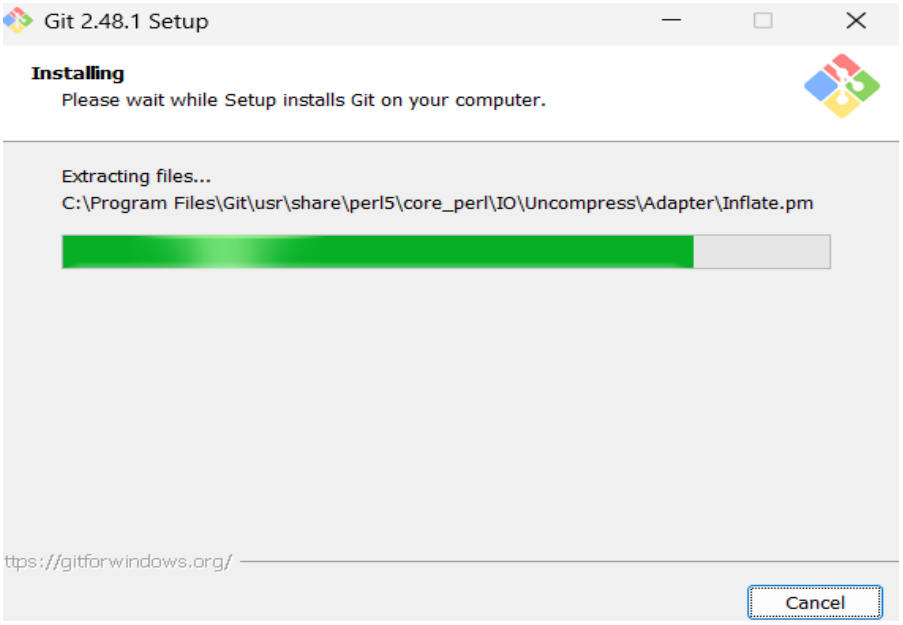


Figure – 17

Step 18: Completing the Git Set - Up Wizard

Once the installation is complete, **"Completing the Git Setup Wizard"** screen appears. Check the '**Launch Git bash'** option and Click on **Finish**.

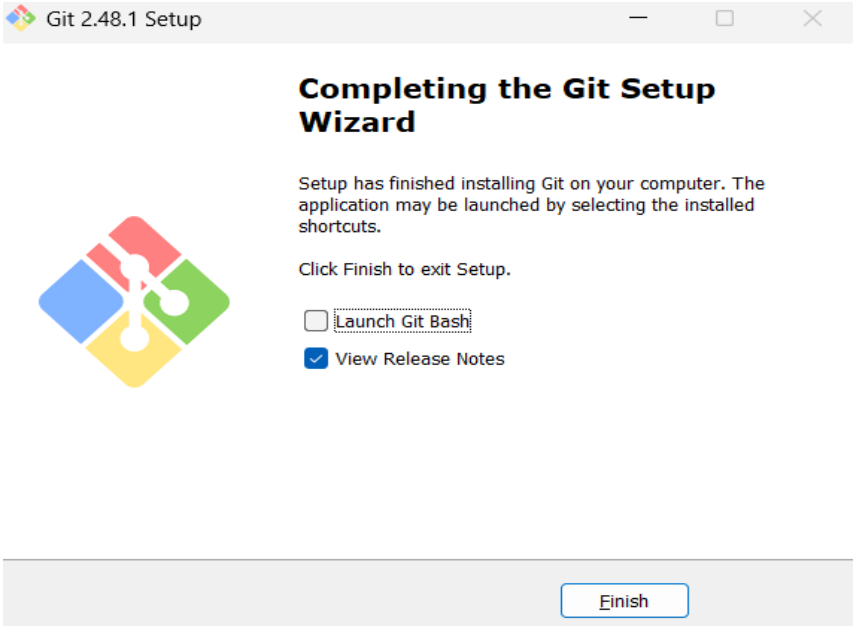


Figure – 18

LAB REPORT – 2

Step 1: Open Git Bash

Open **Git Bash** from the Start menu or by searching for it.

```
Pavani@Pavani-PC MINGW64 ~ (master)
$
```

Step 2: Configure Git

Set up your Git username and email (required for commits):

- **git config --global user.name "Your Name"**
- **git config --global user.email "your-email@example.com"**

```
Pavani@Pavani-PC MINGW64 ~ (master)
$ git config --global user.name "Pavani B A"

Pavani@Pavani-PC MINGW64 ~ (master)
$ git config --global user.email "pavani.a@s.amity.edu"

Pavani@Pavani-PC MINGW64 ~ (master)
$
```

Step 4: Verify Git Configurations

To check if the configurations were set correctly, run:

- **git config --list**

```
Pavani@Pavani-PC MINGW64 /c
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Pavani B A
user.email=pavani.a@s.amity.edu
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
filter.lfs.clean=git-lfs clean -- %f
safe.directory=D:/

Pavani@Pavani-PC MINGW64 /c
$
```

Figure – 4

Step 5: Change Directory

Change directory (**cd**) to your preferred location using the **'cd'** command.

```
Pavani@Pavani-PC MINGW64 /c
$ cd /c

Pavani@Pavani-PC MINGW64 /c
$
```

Figure – 5

Step 6: Print the Current Directory

To print the full path of your current Directory use the '**pwd'** command.

```
Pavani@Pavani-PC MINGW64 /c
$ cd /c

Pavani@Pavani-PC MINGW64 /c
$ pwd
/c

Pavani@Pavani-PC MINGW64 /c
$
```

Figure – 6

Step 7: Create a New Folder

To Create a new folder in the Directory, use the command: **mkdir** folder-name.

```
Pavani@Pavani-PC MINGW64 /c
$ cd /c

Pavani@Pavani-PC MINGW64 /c
$ pwd
/c

Pavani@Pavani-PC MINGW64 /c
$ mkdir GIT

Pavani@Pavani-PC MINGW64 /c
$
```

Figure – 7

Step 8: Listing the Files and Folders

To Display the list of all files and folders in the current directory use the **'ls'** command.

```
Pavani@Pavani-PC MINGW64 /c
$ cd /c

Pavani@Pavani-PC MINGW64 /c
$ pwd
/c

Pavani@Pavani-PC MINGW64 /c
$ mkdir GIT

Pavani@Pavani-PC MINGW64 /c
$ ls
'$Recycle.Bin'/             DumpStack.log.tmp  'Program Files'/        SLFNXT/                    System.sav@    hp/          sample/
 2006/                      GIT/               'Program Files (x86)'/  SWSetup/                   Users/         inetpub/     scm/
 Config.Msi/                OneDriveTemp/       ProgramData/           'Shree-Lipi nxt'/          Windows/       msys64/      shree-lipi
'Documents and Settings'@   PerfLogs/          Recovery/              'System Volume Information'/ hiberfil.sys  pagefile.sys  swapfile.sys

Pavani@Pavani-PC MINGW64 /c
$
```

Figure – 8

Step 9: Creating a File Inside the Folder

To create a C++ File inside the **Git** Folder, move inside the folder using the '**cd**' command and then use **'vi'** command to create a file.

```
Pavani@Pavani-PC MINGW64 /c
$ cd /c/GIT

Pavani@Pavani-PC MINGW64 /c/GIT
$ vi Hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT
$
```

Figure – 9

Step 10: Inside the VI Editor

Once typed Git opens the **'vi'** editor to create or edit a file named **Hello.cpp.** Press **i** to enter **INSERT** mode. Now start typing your code in the **vi** Editor.

```
#include <iostream>
using namespace std;
int main(){
        cout<<"Welcome to GIT!!"<<endl;
        return 0;
}
~
~
```

Figure – 10

Step 11: Exiting the VI Editor

Once done with the code Press **ESC** to exit **INSERT** mode and type **:wq** and press **Enter** to save and exit.

```
Pavani@Pavani-PC MINGW64 /c/GIT
$ vi Hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT
$
```

Figure – 11

Step 12: Display File Contents

To Display the contents of the CPP File use the **cat** Command as: **cat** filename.extension.

```
Pavani@Pavani-PC MINGW64 /c/GIT
$ vi Hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT
$ cat Hello.cpp
#include <iostream>
using namespace std;
int main(){
        cout<<"Welcome to GIT!!"<<endl;
        return 0;
}

Pavani@Pavani-PC MINGW64 /c/GIT
$
```

Figure – 12

Step 13: Initialize Git in Directory

To turn the directory into a Git repository, run: **git init**

```
Pavani@Pavani-PC MINGW64 /c/GIT
$ cd /c/GIT

Pavani@Pavani-PC MINGW64 /c/GIT
$ git init
Initialized empty Git repository in C:/GIT/.git/

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 13

Step 14: Check Git Status

The **git status** command is used to check for **untracked files**, along with other changes in the repository. You should see Hello.cpp as an **untracked file**.

```
Pavani@Pavani-PC MINGW64 /c/GIT
$ cd /c/GIT

Pavani@Pavani-PC MINGW64 /c/GIT
$ git init
Initialized empty Git repository in C:/GIT/.git/

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Hello.cpp

nothing added to commit but untracked files present (use "git add" to track)

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 14

Step 15: Add Files to Staging Area

To stage all newly created and modified files use the command: **git add .**

To confirm, check the status again using the command: **git status**

Now, all tracked files will appear as **staged**.

Figure – 15

Step 16: Commit the File

To save the changes in Git, commit the file with a message**: git commit -m "Initial commit: Added main.cpp"**

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git add .
warning: in the working copy of 'Hello.cpp', LF will be replaced by CRLF the next time Git touches it

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git commit -m"First commit"
[master (root-commit) 64eeeaa] First commit
 1 file changed, 6 insertions(+)
 create mode 100644 Hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 16

Source Code Management

LAB REPORT – 3

Step 1: Check Git Commit History

- The **git log** command displays the commit history in detail.
- It shows the commit hash, author, date, and commit message.

```
?avani@Pavani-PC MINGW64 /c/GIT (master)
$ git commit -m"First commit"
[master (root-commit) 64eeeaa] First commit
 1 file changed, 6 insertions(+)
 create mode 100644 Hello.cpp

?avani@Pavani-PC MINGW64 /c/GIT (master)
$ git log
commit 64eeeaa1662cf046e6d82428a8486478c01dcada (HEAD -> master)
Author: Pavani B A <pavani.a@s.amity.edu>
Date:   Sun Jun 1 08:40:00 2025 +0530

    First commit

?avani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 1

Step 2: View Git Log in One Line Format

- The **git log --oneline** command displays a compact version of the commit history.
- It only shows the commit hash and the commit message.

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git log
commit 64eeeaa1662cf046e6d82428a8486478c01dcada (HEAD -> master)
Author: Pavani B A <pavani.a@s.amity.edu>
Date:   Sun Jun 1 08:40:00 2025 +0530

    First commit

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git log --oneline
64eeeaa (HEAD -> master) First commit

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 2

Step 3: Modify the Hello.cpp File (First Change)

- Open the Hello.cpp file in a text editor using the **vi** command.
- Make a small change (e.g., add a new function or modify a print statement).
- Save the file and display it using the **cat** command.

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ vi Hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ cat Hello.cpp
#include <iostream>
using namespace std;
int main(){
        cout<<"Hello!!"<<endl;
        cout<<"Welcome to GIT!!"<<endl;
        return 0;
}

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 3

Step 4: Stage and Commit the First Change

Use **git add .** command to stage the modified file for commit **and git commit -m** to create a commit with a message describing the change.

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ cat Hello.cpp
#include <iostream>
using namespace std;
int main(){
        cout<<"Hello!!"<<endl;
        cout<<"Welcome to GIT!!"<<endl;
        return 0;
}

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git add .
warning: in the working copy of 'Hello.cpp', LF will be replaced by CRLF the next time Git touches it

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git commit -m"Second commit"
[master ced7957] Second commit
 1 file changed, 1 insertion(+)

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 4

Step 5: Modify the Hello.cpp File Again (Second Change)

- Make another change in the same Hello.cpp file.
- Example: Modify a different function or add a new comment.
- Save the file and commit it.

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ vi Hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ cat Hello.cpp
#include <iostream>
using namespace std;
int main(){
        cout<<"Hello!!"<<endl;
        cout<<"This is SCM"<<endl;
        cout<<"Welcome to GIT!!"<<endl;
        return 0;
}

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git add .
warning: in the working copy of 'Hello.cpp', LF will be replaced by CRLF the next time Git touches it

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git commit -m"Third commit"
[master 359f294] Third commit
 1 file changed, 1 insertion(+)

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 5

Step 6: View Git Log Again in One Line Format

This will now show the latest two commits along with previous commits.

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git commit -m"Third commit"
[master 359f294] Third commit
 1 file changed, 1 insertion(+)

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git log --oneline
359f294 (HEAD -> master) Third commit
ced7957 Second commit
64eeeaa First commit

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 6

Step 7: View Differences Between Commits

The **git diff** command shows the exact lines changed between each commits. You can compare between multiple commits. Example: First commit and Second commit or Second commit and Third commit or even multiple commits.

This shows changes between the First commit and Second commit.

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git diff 359f294 ced7957
diff --git a/Hello.cpp b/Hello.cpp
index 224f86a..a51177b 100644
--- a/Hello.cpp
+++ b/Hello.cpp
@@ -2,7 +2,6 @@
 using namespace std;
 int main(){
        cout<<"Hello!!"<<endl;
-       cout<<"This is SCM"<<endl;
        cout<<"Welcome to GIT!!"<<endl;
        return 0;
 }

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 7

This shows changes between the Second commit and Third commit.

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git diff

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git diff 64eeeaa ced7957
diff --git a/Hello.cpp b/Hello.cpp
index d346ac4..a51177b 100644
--- a/Hello.cpp
+++ b/Hello.cpp
@@ -1,6 +1,7 @@
 #include <iostream>
 using namespace std;
 int main(){
+       cout<<"Hello!!"<<endl;
        cout<<"Welcome to GIT!!"<<endl;
        return 0;
 }

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 8

Source Code Management

LAB REPORT – 4

Step 1: Sign in to GitHub

Open a web browser and go to github.com

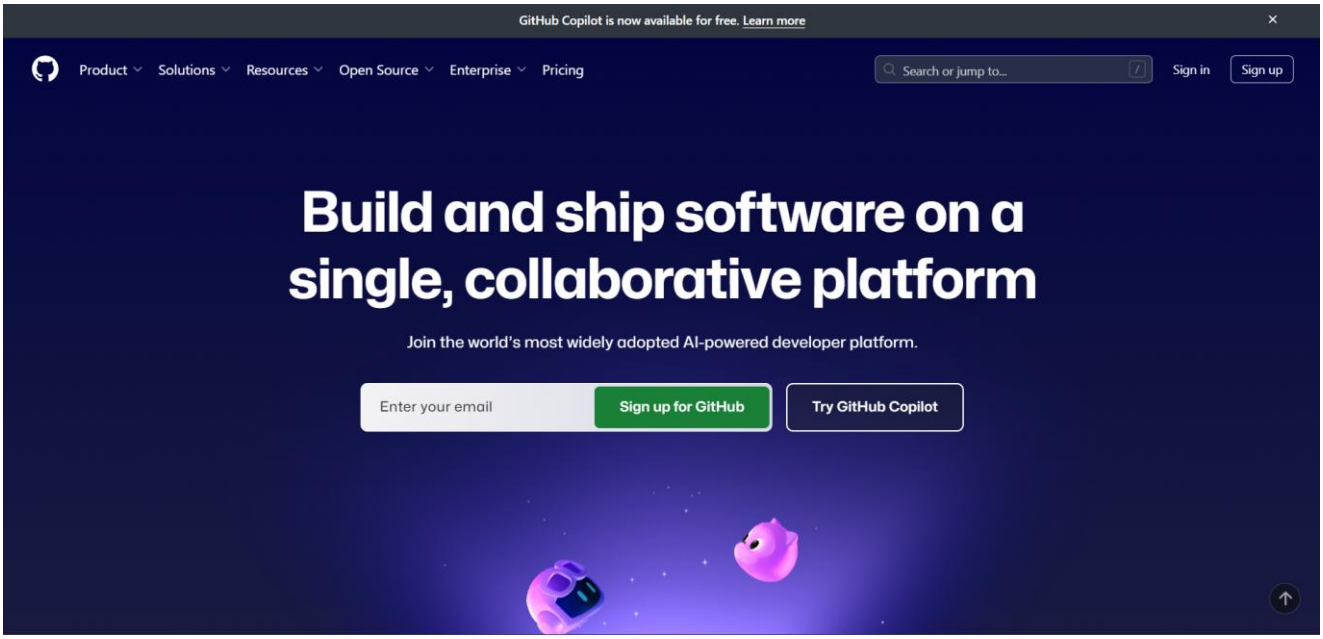


Figure – 1

Click Sign in and enter your credentials.

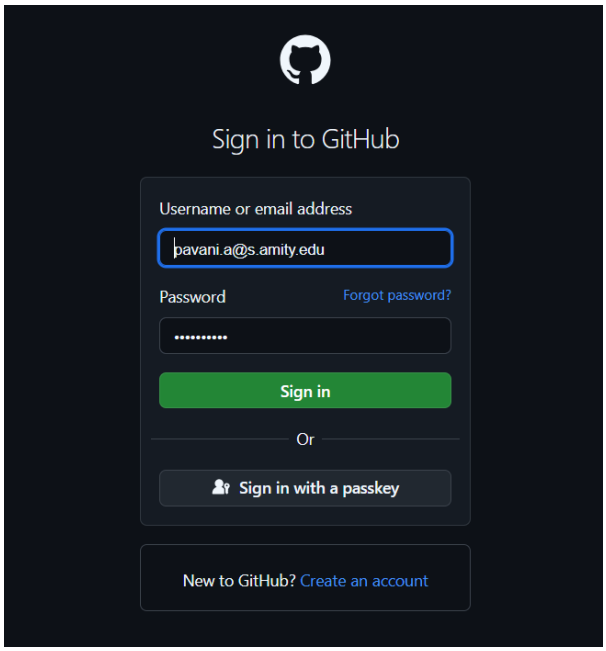


Figure – 2

Step 2: Creating a Repository

Click on the **"+"** icon (top-right corner) and select "**New repository**".

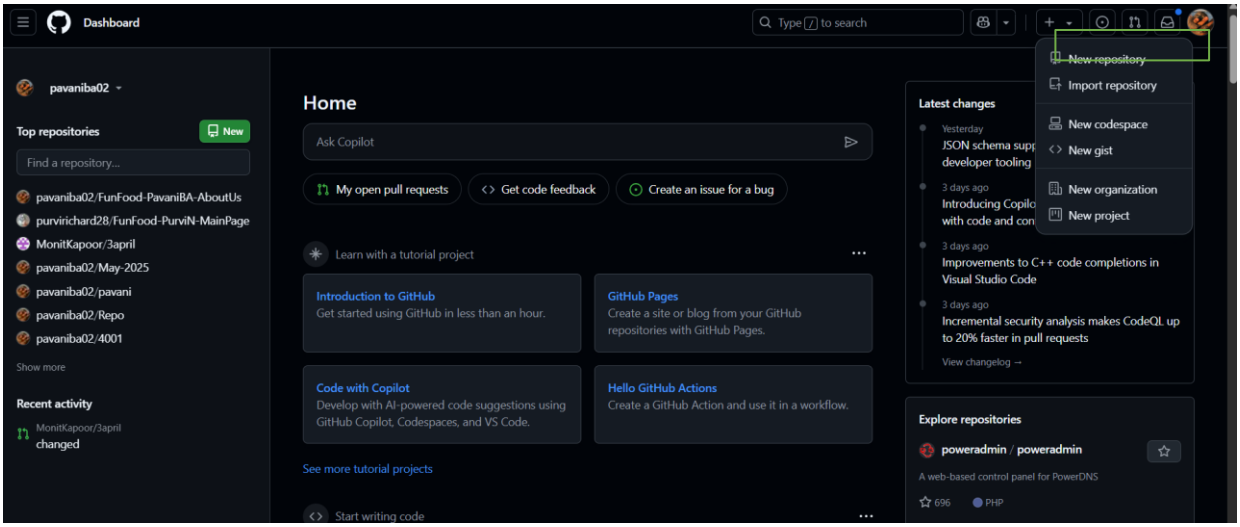


Figure – 3

In the **Repository name** field, enter the same name as your local folder. Select Public. **Do not** check "**Initialize this repository with a README**". Click **Create repository**.

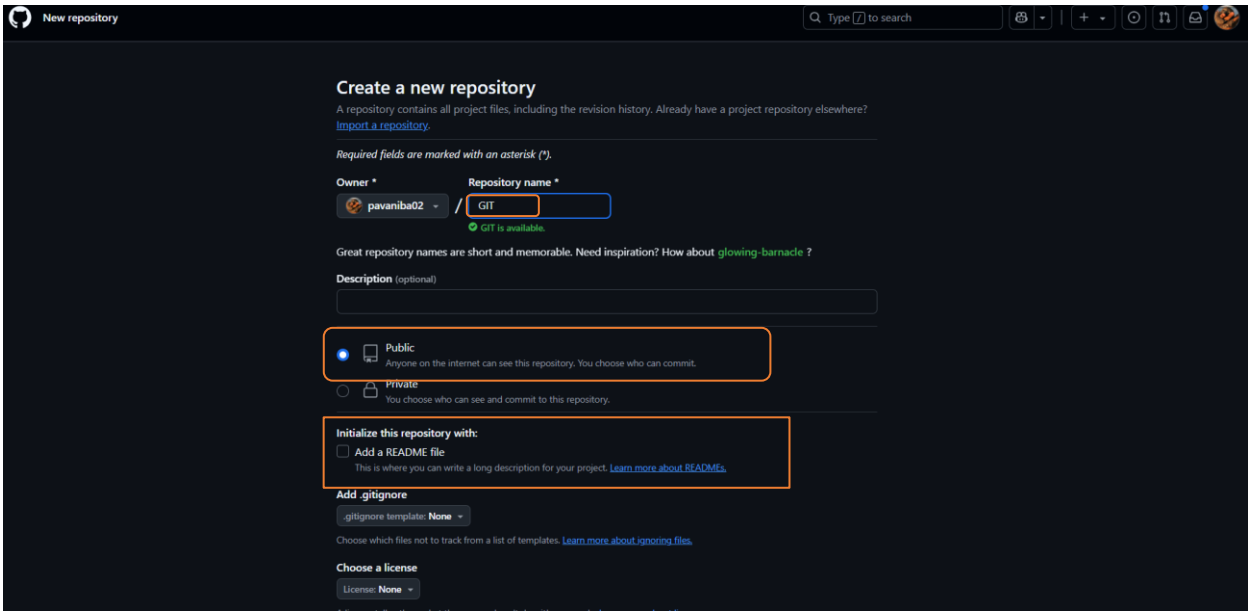


Figure – 4

Step 3: Connect Local Repository to GitHub

On the next page, copy the **HTTPS URL** under **"Quick setup"** it looks like (**https://github.com/yourusername/repositoryname.git**).

Add the GitHub repository as a remote:

- **git remote**
- **git remote add origin <repository-URL>**

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git remote

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git remote add origin https://github.com/pavaniba02/GIT

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git remote
origin

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 5

Step 4: Push Code To GitHub

Push the committed files to GitHub using the command: **git push -u origin master**

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git remote

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git remote add origin https://github.com/pavaniba02/GIT

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git remote
origin

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git push -u origin master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 770 bytes | 154.00 KiB/s, done.
Total 9 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/pavaniba02/GIT
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 6

Step 5: Verify Changes on GitHub

1. Open **GitHub** in your browser.
2. Go to your repository.
3. Refresh the page – your files should be visible in the repository.

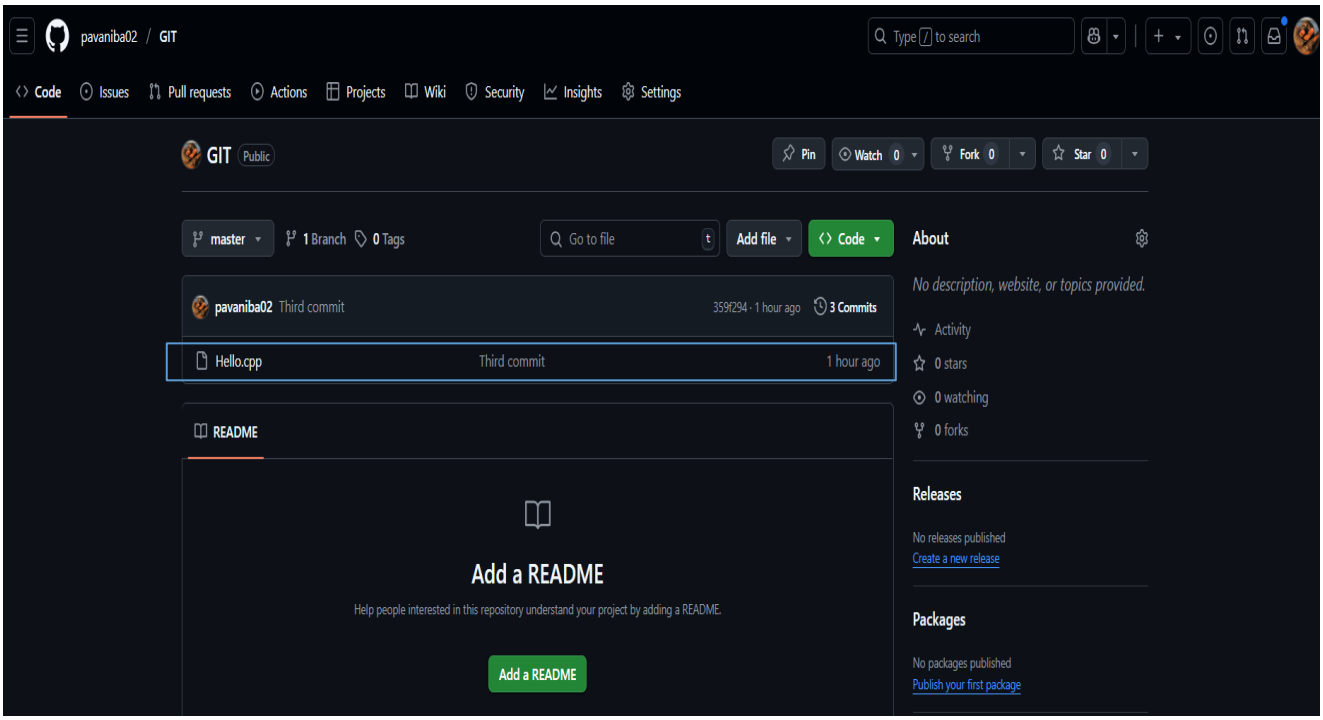


Figure – 7

Step 6: Edit the File Directly on GitHub

1. Click on Hello.cpp file in your GitHub repository.

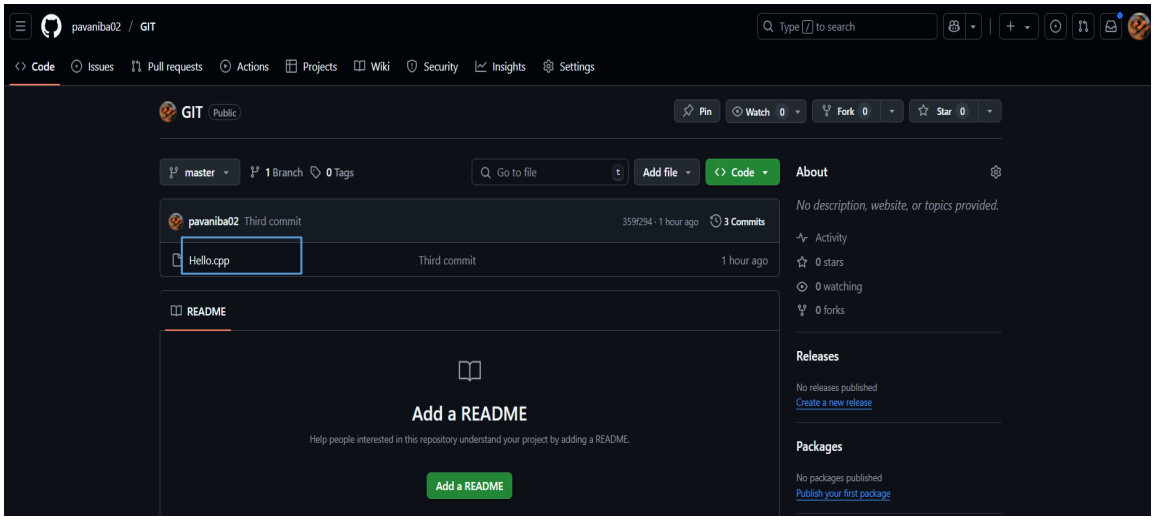


Figure – 8

2. Click the edit (pencil) icon in the top-right.

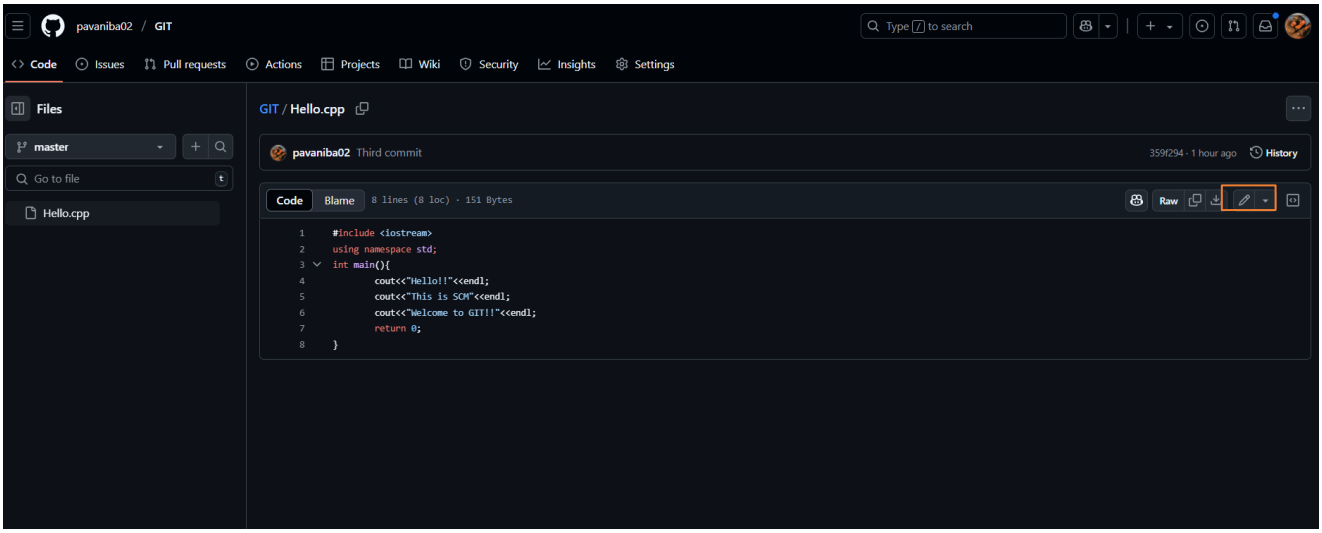


Figure – 9

3. Make some changes to the file, scroll down, enter a commit message, and click Commit changes.

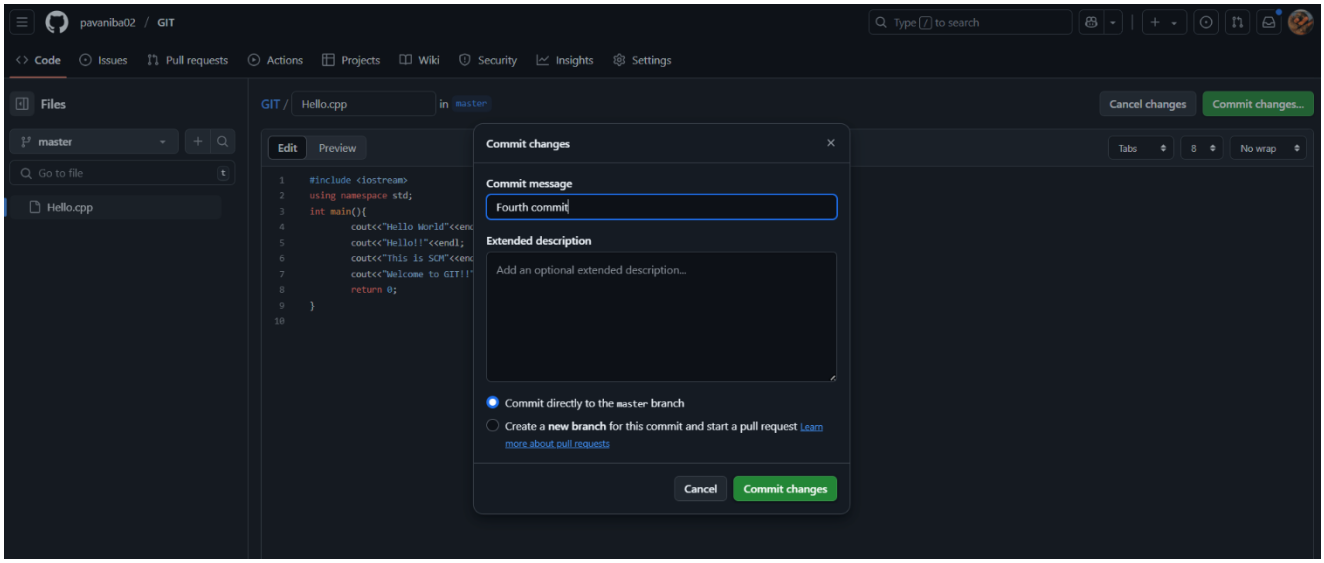


Figure – 10

Step 7: Pull Changes from GitHub to Local System

Open **Git Bash** in your project folder and Pull the latest changes from GitHub using the

command: **git pull**

The updated file will now be available on your local system.

```
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/pavaniba02/GIT
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), 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), 924 bytes | 77.00 KiB/s, done.
From https://github.com/pavaniba02/GIT
   359f294..3950c90  master     -> origin/master
Updating 359f294..3950c90
Fast-forward
 Hello.cpp | 1 +
 1 file changed, 1 insertion(+)

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 11

Use git log to see the changes in your local repository file.

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git log
commit 3950c90525b7c3ae8b71152a184c4c7593669aec (HEAD -> master, origin/master)
Author: Pavani B A <pavani.a@s.amity.edu>
Date:   Sun Jun 1 12:58:05 2025 +0530

    Fourth commit

commit 359f294116fdeaa34f8ef1ac439d16a08d72c608
Author: Pavani B A <pavani.a@s.amity.edu>
Date:   Sun Jun 1 11:06:36 2025 +0530

    Third commit

commit ced7957ff1c23207fa4bb570983499365e7f3c42
Author: Pavani B A <pavani.a@s.amity.edu>
Date:   Sun Jun 1 11:04:22 2025 +0530

    Second commit

commit 64eeeaa1662cf046e6d82428a8486478c01dcada
Author: Pavani B A <pavani.a@s.amity.edu>
Date:   Sun Jun 1 08:40:00 2025 +0530

    First commit

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 12

LAB REPORT – 5

Step 1: Create a New Branch

Use the following command to create a new branch named **dev** and switch to it:

```
git checkout -b dev
```

```
    First commit

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git commit -m"first commit"
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git checkout -b dev
Switched to a new branch 'dev'

Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$
```

Figure - 1

Step 2: Make Changes in the dev Branch

Open the **hello.cpp** file and make some changes.

```
Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$ vi hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$ cat hello.cpp
#include <iostream>
using namespace std;
int main(){
        cout<<"Hello World"<<endl;
        cout<<"Hello!!"<<endl;
        cout<<"This is SCM"<<endl;
        cout<<"Welcome to GIT!!"<<endl;
        return 0;

}

Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$ ls
Hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$ fg
vi hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$
```

Figure – 2

Step 3: Stage and Commit Changes

- **git add .**
- **git commit -m "Added a new file in dev branch"**

```
Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$ git commit -m "Added a line"
[dev 7965753] Added a line
 1 file changed, 2 insertions(+)

Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$ git status
On branch dev
nothing to commit, working tree clean

Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$
```

Figure – 3

Step 4: Switch Back to master Branch

git checkout master

```
Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$ git status
On branch dev
nothing to commit, working tree clean

Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 4

Step 5: Merge dev into master

If there are no conflicts, this will merge the changes from the dev branch into master.

git merge dev

```
Pavani@Pavani-PC MINGW64 /c/GIT (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ git merge dev
Updating 3950c90..7965753
Fast-forward
 Hello.cpp | 2 ++
 1 file changed, 2 insertions(+)

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure - 5

Step 6: Verify the Merge

Use cat command to check is the files are merged.

cat hello.cpp

```
Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ ls
Hello.cpp

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$ cat hello.cpp
//Made some changes
#include <iostream>
using namespace std;
int main(){
        cout<<"Hello World"<<endl;
        cout<<"Hello!!"<<endl;
        cout<<"This is SCM"<<endl;
        cout<<"Welcome to GIT!!"<<endl;
        return 0;

}

Pavani@Pavani-PC MINGW64 /c/GIT (master)
$
```

Figure – 6

Step 7: Run the Git Merge Tool

Use git mergetool to open the conflict screen. Close it using **escape :wqa**

git mergetool

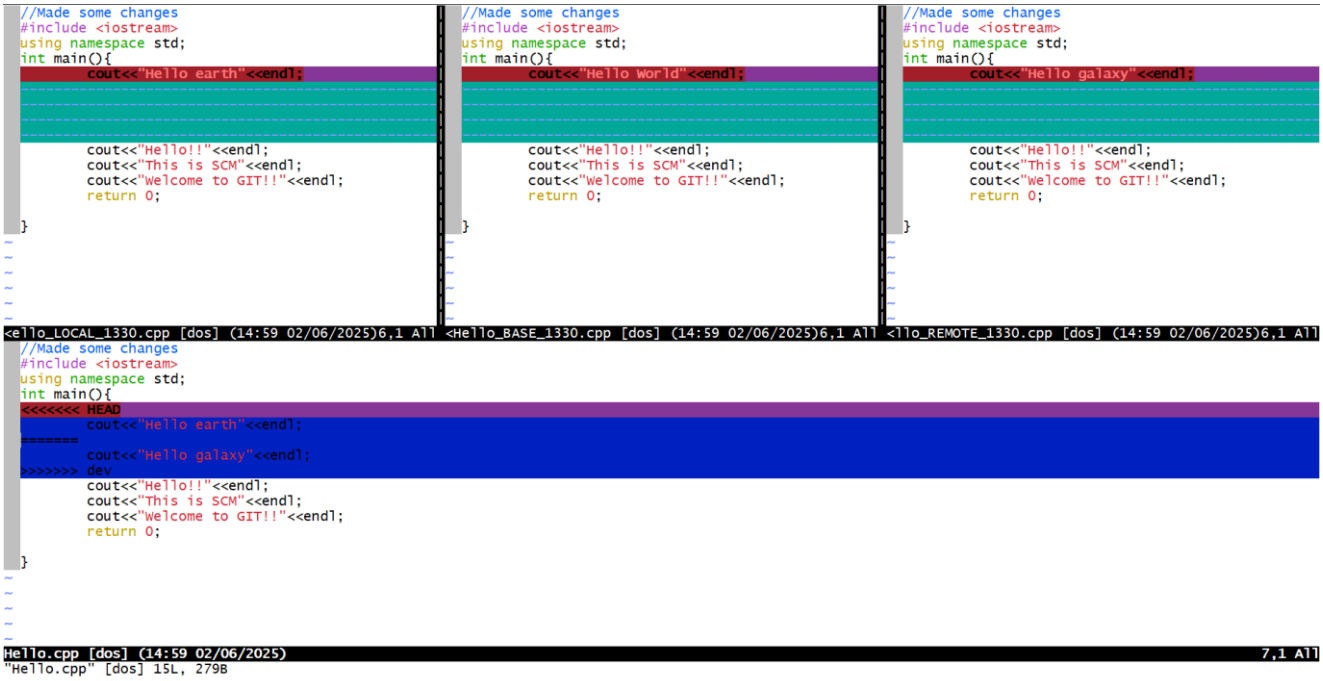


Figure – 7

Step 8: Creating a .gitignore File

The **.gitignore** file tells Git to ignore specific files or directories that do not need to be tracked, such as log files, build directories, or system files.

```
touch .gitignore
```

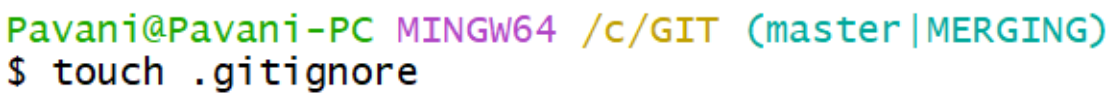


Figure – 8

Step 9: Viewing Hidden Files and Folders

By default, files that start with a dot (.) are **hidden** in Unix-based systems, including Git Bash.

```
ls -a
```

```
Pavani@Pavani-PC MINGW64 /c/GIT (master|MERGING)
$ touch .gitignore

Pavani@Pavani-PC MINGW64 /c/GIT (master|MERGING)
$ ls -a
./  ../  .git/  .gitignore  Hello.cpp  Hello.cpp.orig

Pavani@Pavani-PC MINGW64 /c/GIT (master|MERGING)
$
```

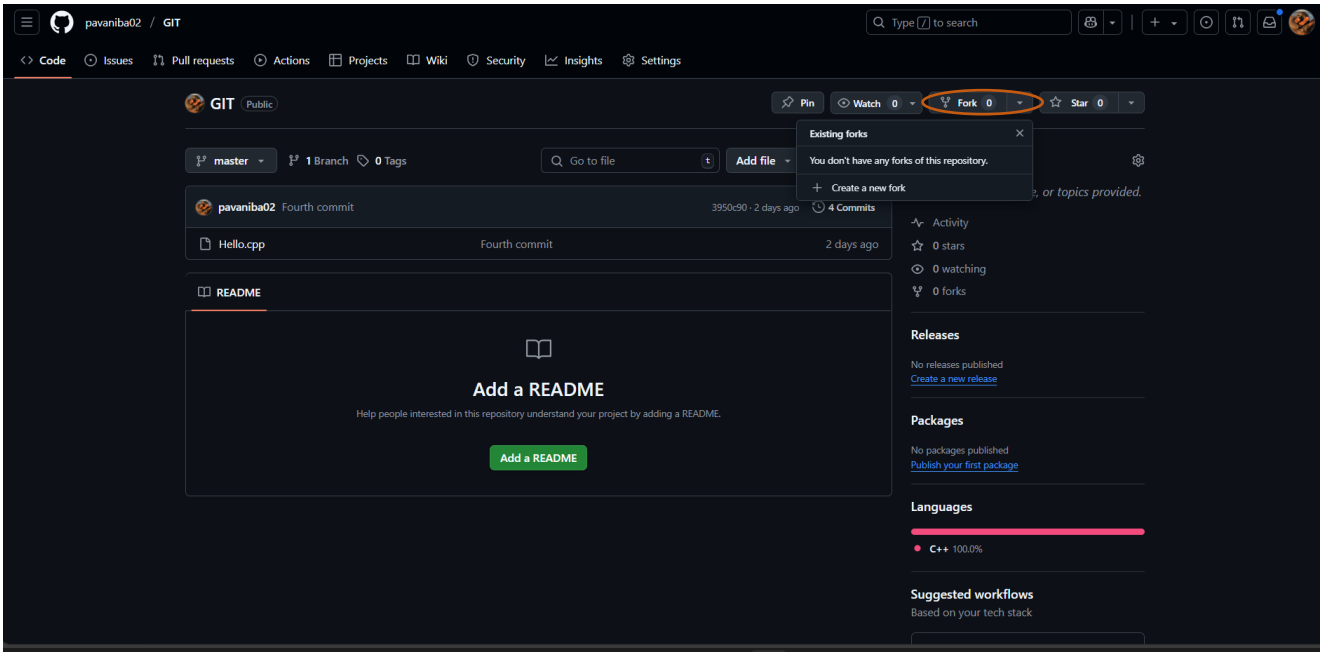
Figure – 8

LAB REPORT – 6

Step 1: Fork a Repository on GitHub

- Go to any public repository on GitHub (e.g., https://github.com/octocat/Hello-World).
- Click on the **"Fork"** button (top right corner).
- This creates a copy of the repository under **your GitHub account**.

Figure - 1



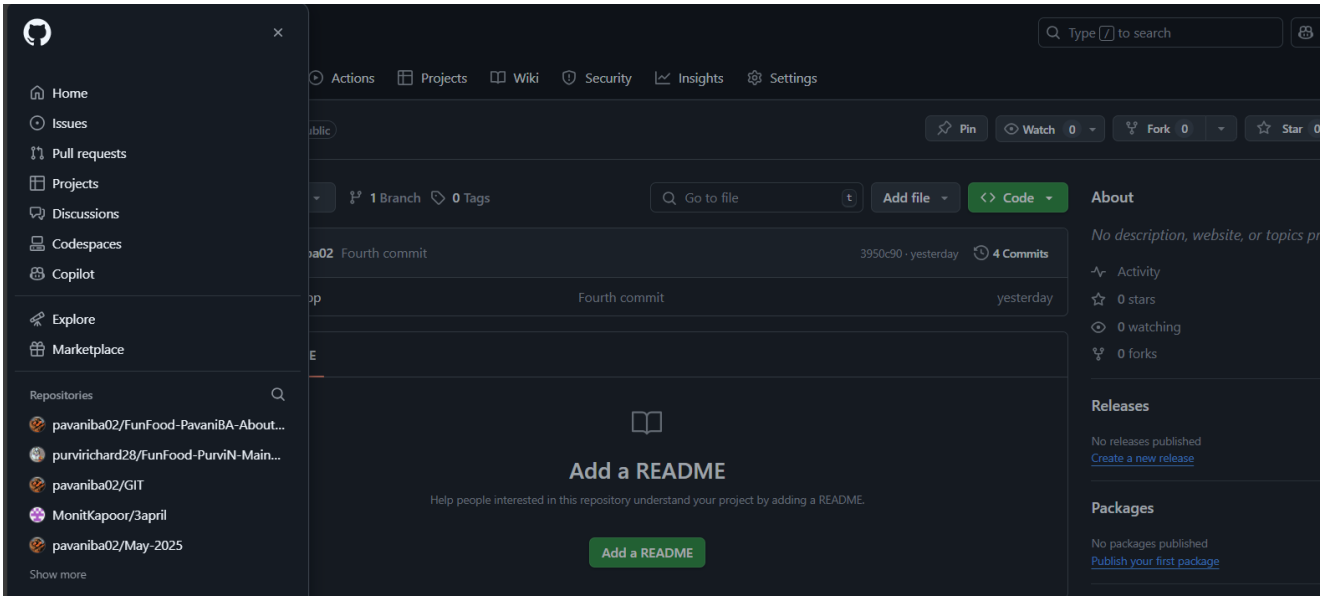


Figure – 2

Step 2: Clone the Forked Repository Locally

Replace your-**username** with your actual GitHub username.

```
Pavani@Pavani-PC MINGW64 /c/GIT (master|MERGING)
$ git clone https://github.com/pavaniba02/GIT.git
Cloning into 'GIT'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 12 (delta 3), reused 8 (delta 2), pack-reused 0 (from 0)
Receiving objects: 100% (12/12), done.
Resolving deltas: 100% (3/3), done.

Pavani@Pavani-PC MINGW64 /c/GIT (master|MERGING)
```

git clone https://github.com/your-username/Hello-World.git

Figure – 3

Step 3: Change Directory to the Cloned Repo

```
Pavani@Pavani-PC MINGW64 /c/GIT (master|MERGING)
$ cd Hello-World

Pavani@Pavani-PC MINGW64 /c/GIT/Hello-World (master|MERGING)
$
```

cd Hello-World

Figure – 4

Step 4: Add a New File or Modify Existing One

Vi hello.cpp

```
Pavani@Pavani-PC MINGW64 /c
$ cd Hello-World

Pavani@Pavani-PC MINGW64 /c/Hello-World
$ git init
Initialized empty Git repository in C:/Hello-World/.git/

Pavani@Pavani-PC MINGW64 /c/Hello-World (master)
$ vi hello.cpp

Pavani@Pavani-PC MINGW64 /c/Hello-World (master)
$
```

Figure – 5

Step 5: Stage and Commit Your Changes

git add .

git commit -m "First Commit"

```
Pavani@Pavani-PC MINGW64 /c/Hello-World (master)
$ git add .
warning: in the working copy of 'hello.cpp', LF will be replaced by CRLF the next time Git touches it

Pavani@Pavani-PC MINGW64 /c/Hello-World (master)
$ git commit -m
error: switch `m' requires a value

Pavani@Pavani-PC MINGW64 /c/Hello-World (master)
$ git commit -m "first commit"
[master (root-commit) 8835161] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello.cpp

Pavani@Pavani-PC MINGW64 /c/Hello-World (master)
$
```

Figure – 6

Step 6: Push Changes to Your Forked GitHub Repo

This updates **your forked repository** on GitHub with your changes.

git push origin master

```
Pavani@Pavani-PC MINGW64 /c/GIT (master
$ git push origin master
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 545 bytes | 545.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object
To https://github.com/pavaniba02/GIT
   3950c90..a0a1ec8  master -> master
```

Figure – 7

Step 7: Create a Pull Request

If you want your changes to be added to the original repository:

1. Go to your forked repo on GitHub.
2. Click **"Contribute" > "Open Pull Request"**.
3. Submit your pull request for review.

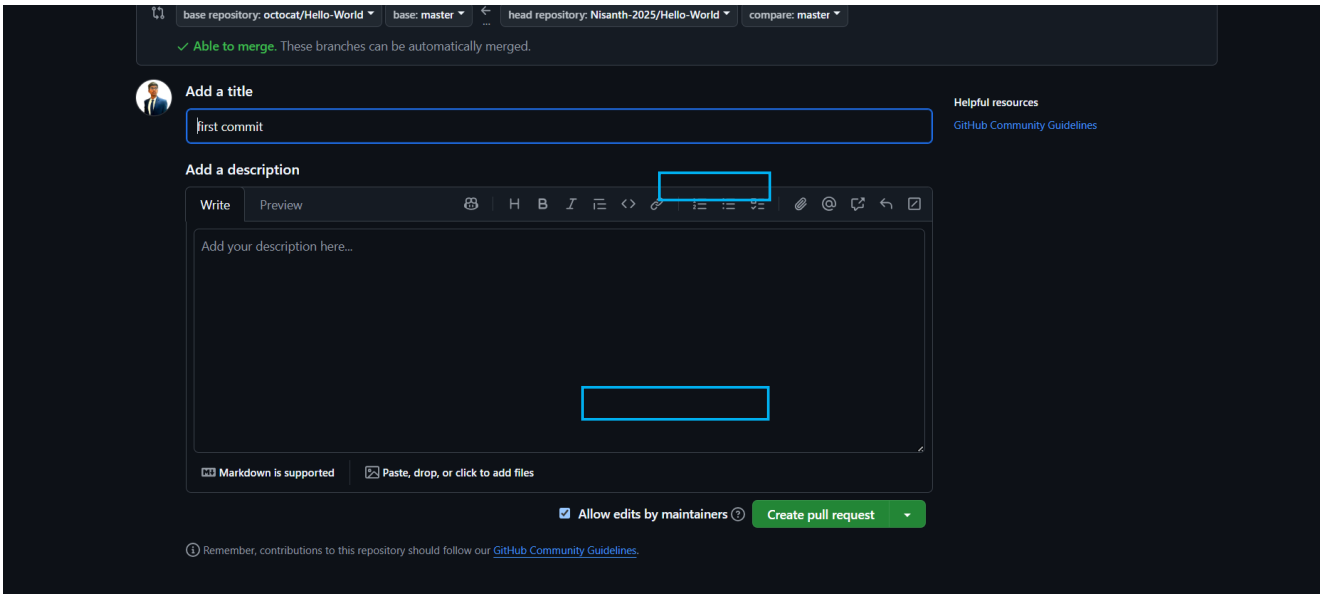


Figure - 9