Source Code Management (CS181)

CSE Batch 2021

Submitted by:

Student Name: Abhinav Gupta

Roll no: 2110990050

Group No.: G01

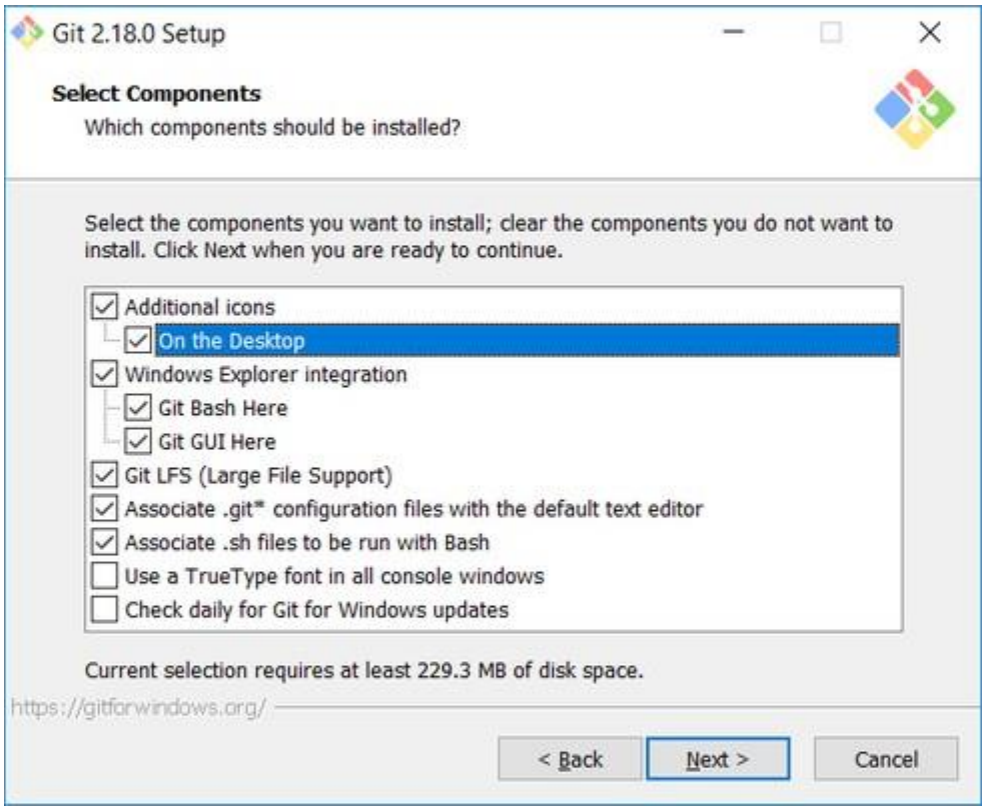
Installing and Configuring the Git client

The following sections list the steps required to properly install and configure the Git clients - Git Bash and Git GUI - on a Windows 7 computer. Git is also available for Linux and Mac. The remaining instructions here, however, are specific to the Windows installation.

Git installation

Download the Git installation program (Windows, Mac, or Linux) from http://git-scm.com/downloads. When running the installer, various screens appear (Windows screens shown). Generally, you can accept the default selections, ***except in the screens below where you do NOT want the default selections:***

In the **Select Components** screen, make sure **Windows Explorer Integration** is selected as shown:



Git 2.18.0 Setup

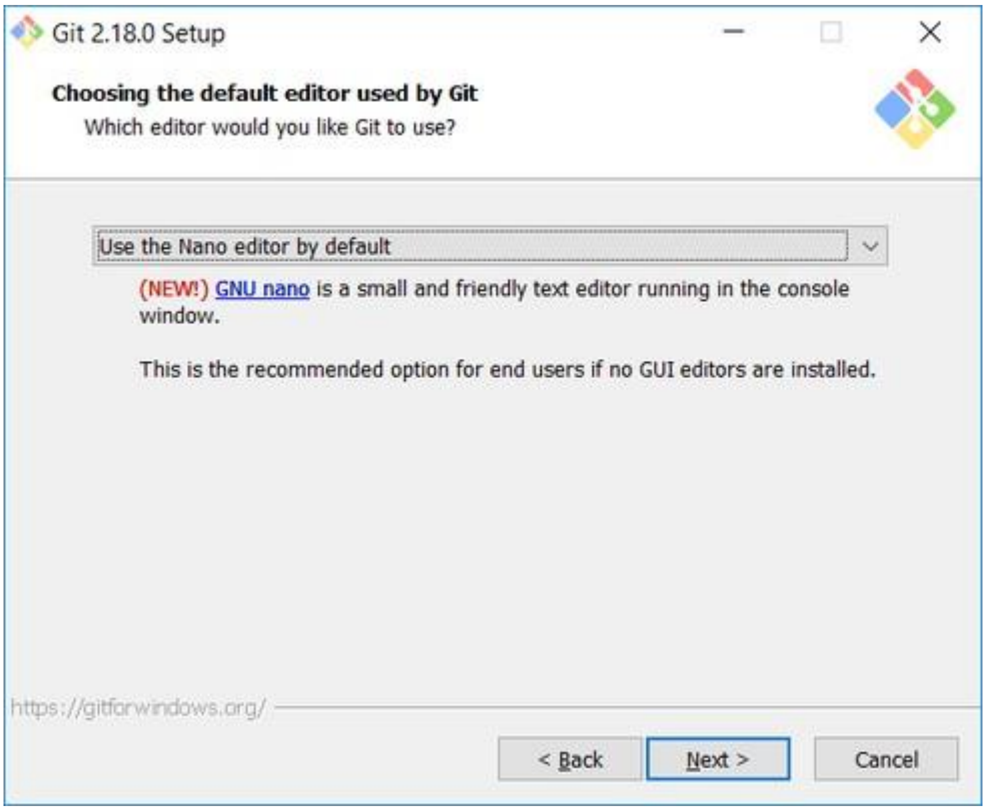
Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] Additional icons
  - [✓] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Git LFS (Large File Support)
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows
- [ ] Check daily for Git for Windows updates

Current selection requires at least 229.3 MB of disk space.



Choosing the default editor used by Git
Which editor would you like Git to use?

Use the Nano editor by default

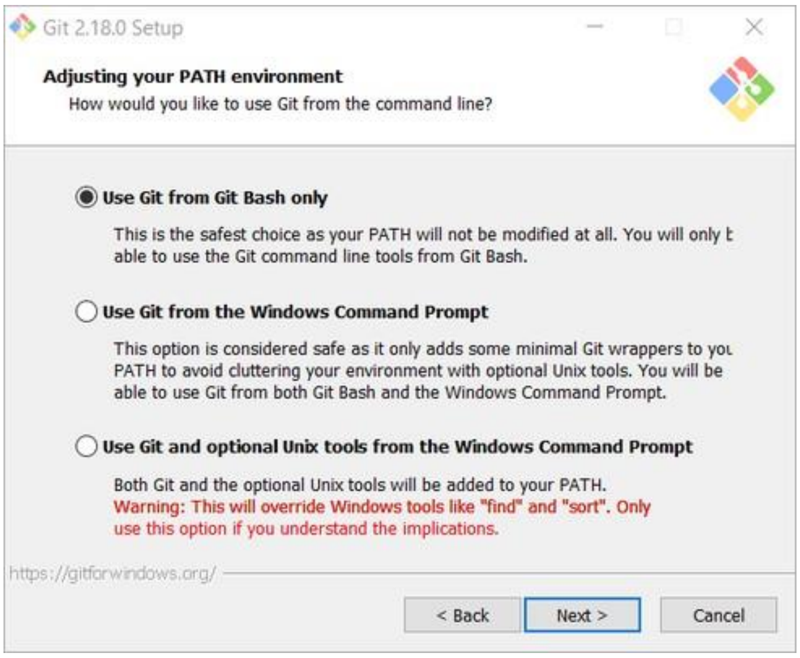
(NEW!) GNU nano is a small and friendly text editor running in the console window.

This is the recommended option for end users if no GUI editors are installed.

https://gitforwindows.org/

In the **Adjusting your PATH** screen, all three options are acceptable:

- **Use Git from Git Bash only**: no integration, and no extra commands in your command path
- **Use Git from the Windows Command Prompt**: adds flexibility - you can simply run git from a Windows command prompt, and is often the setting for people in industry - but this does add some extra commands.
- **Use Git and optional Unix tools from the Windows Command Prompt**: this is also a robust choice and useful if you like to use Unix commands like grep



Adjusting your PATH environment
How would you like to use Git from the command line?

● Use Git from Git Bash only

This is the safest choice as your PATH will not be modified at all. You will only b
able to use the Git command line tools from Git Bash.

○ Use Git from the Windows Command Prompt

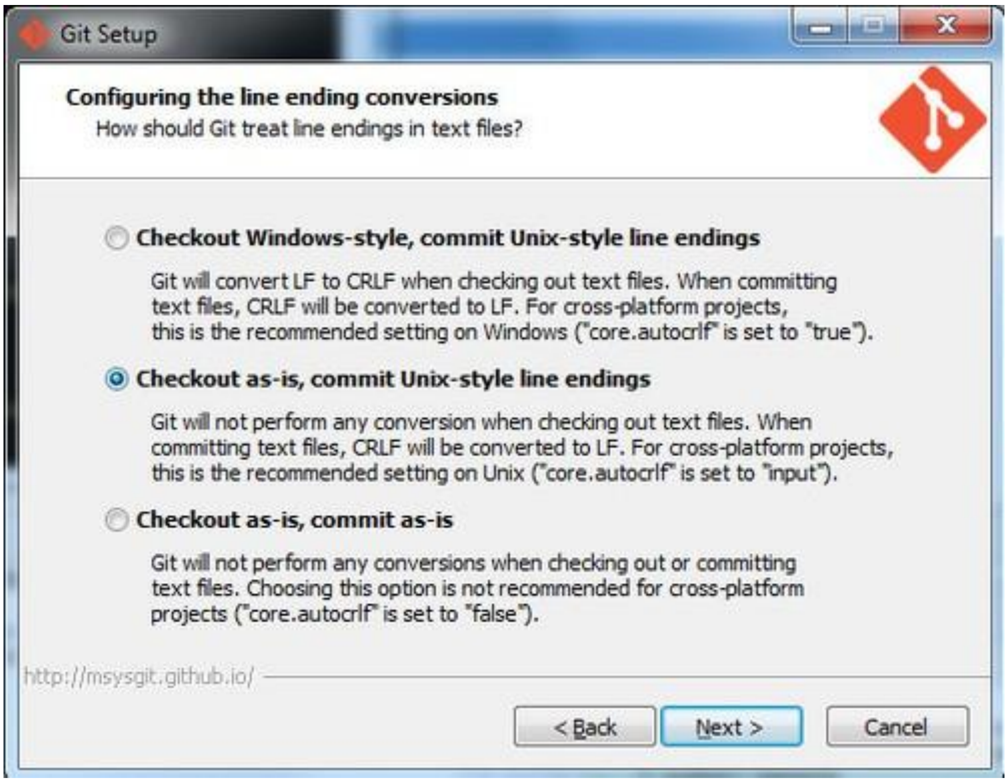
This option is considered safe as it only adds some minimal Git wrappers to you
PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

○ Use Git and optional Unix tools from the Windows Command Prompt

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

https://gitforwindows.org/

In the **Configuring the line ending** screen, select the middle option (Checkout as-is, commit Unix-style line endings) as shown. This helps migrate files towards the Unix-style (LF) terminators that most modern IDE's and editors support. The Windows convention (CR-LF line termination) is only important for Notepad (as opposed to Notepad++), but if you are using Notepad to edit your code you may need to ask your instructor for help.



Configuring the line ending conversions

How should Git treat line endings in text files?

( ) **Checkout Windows-style, commit Unix-style line endings**

Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

[✓ **Checkout as-is, commit Unix-style line endings**]

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

( ) **Checkout as-is, commit as-is**

Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

Setting up GitHub Account

The first steps in starting with GitHub are to create an account, choose a product that fits your needs best, verify your email, set up two-factor authentication, and view your profile.

There are several types of accounts on GitHub. Every person who uses GitHub has their own user account, which can be part of multiple organizations and teams. Your user account is your identity on GitHub.com and represents you as an individual

1. Creating an account

To sign up for an account on GitHub.com, navigate to https://github.com/ and follow the prompts.

To keep your GitHub account secure you should use a strong and unique password. For more information, see "Creating a strong password."

2. Choosing your GitHub product

You can choose GitHub Free or GitHub Pro to get access to different features for your personal account. You can upgrade at any time if you are unsure at first which product you want.

For more information on all of GitHub's plans, see "GitHub's products."

3. Verifying your email address

To ensure you can use all the features in your GitHub plan, verify your email address after signing up for a new account. For more information, see "Verifying your email address."

4. Configuring two-factor authentication

Two-factor authentication, or 2FA, is an extra layer of security used when logging into websites or apps. We strongly urge you to configure 2FA for the safety of your account. For more information, see "About two-factor authentication."

5. Viewing your GitHub profile and contribution graph

Your GitHub profile tells people the story of your work through the repositories and gist you've pinned, the organization memberships you've chosen to publicize, the contributions you've made, and the projects you've created. For more information, see "About your profile" and "Viewing contributions on your profile."

About Version Control:

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time.
benefits of version control systems:-

- **Generate Backups:** Creating a backup of the current version of that repository is perhaps the most important benefit of using a version control system. Having numerous backups on various machines is beneficial because it protects data from being lost in the event of a server failure.

- **Experiment:** When a team works on a software project, they frequently use clones of the main project to build new features, test them, and ensure that they work properly before adding them to the main project. This could save time as different portions of the code can be created simultaneously.

- **Keep History:** Keeping track of the changes in a code file would assist you and new contributors understand how a certain section of the code was created. How did it begin and evolve over time to get at its current state.

- **Collaboration:** One of the most important advantages of version control systems, particularly DVCS, is that it allowed us to participate to projects we enjoyed despite the fact that we were in separate countries

Local Version Control Systems

Many people's version-control method of choice is to copy files into another directory. This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.

Centralized Version Control Systems

Centralized version control systems are based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy. Committing a change simply means recording the change in the central system.

Main benefits (CVCS):

- Centralized systems are typically easier to understand and use
- You can grant access level control on directory level
- performs better with binary files

Distributed Version Control Systems

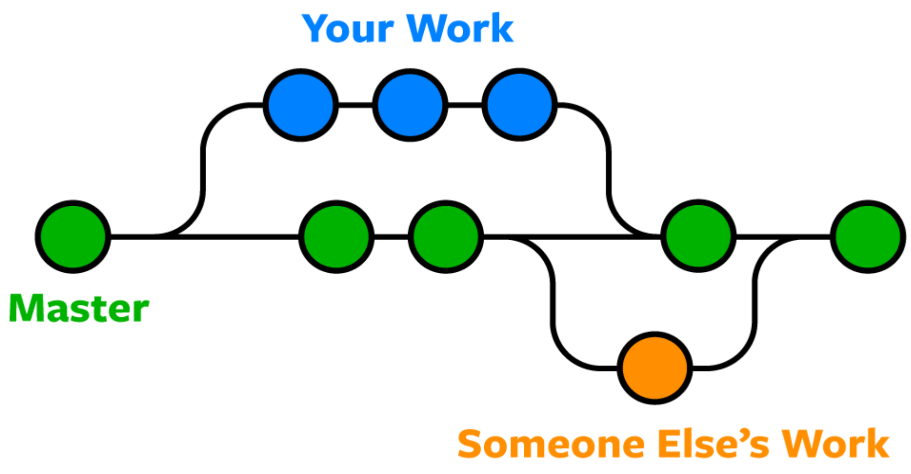
In distributed version control, every developer “clones” a copy of a repository and has the full history of the project on their own hard drive. This copy (or “clone”) has all of the metadata of the original.

Main benefits (DVCS):

- Performance of distributed systems is better
- Branching and merging is much easier
- With a distributed system, you don't need to be connected to the network all the time (complete code repository is stored locally on PC)

What is Git?

Git is a source code management technology used by DevOps. Git is a piece of software that allows you to track changes in any group of files. It is a free and open-source version control system that may be used to efficiently manage small to big projects.
Git is a version control system that allows numerous developers to collaborate on non-linear development projects.
Git is an example of a distributed version control system (DVCS) (hence Distributed Version Control System).



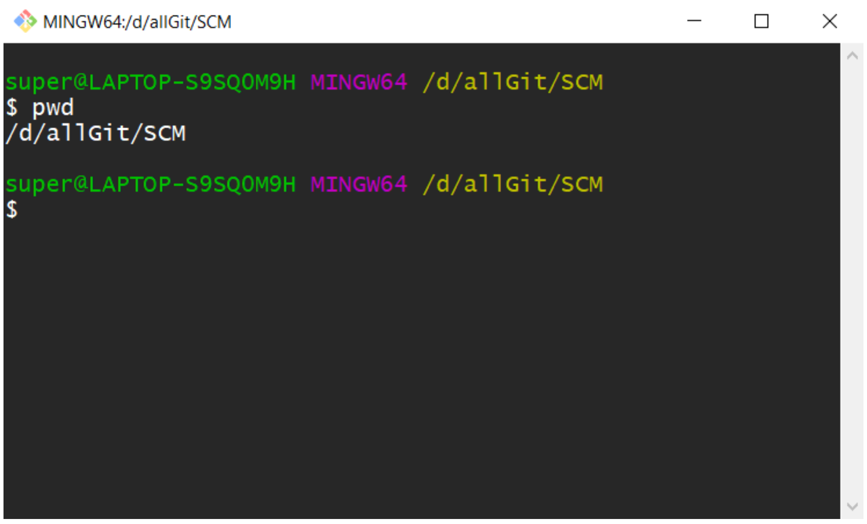
What is GitHub?

GitHub is a version management and collaboration tool for programming. It allows you and others to collaborate on projects from any location.

Important commands

1. pwd

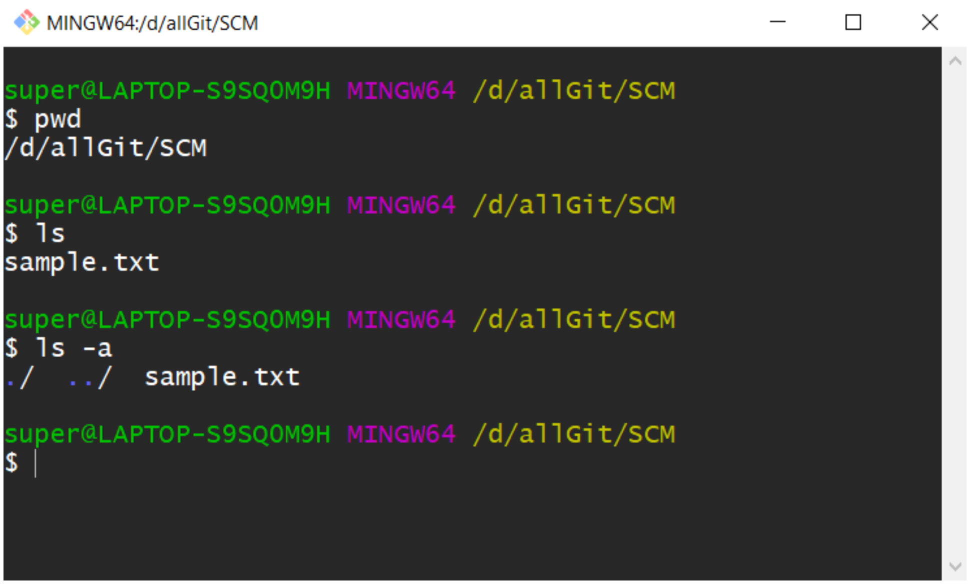
The Bash command pwd is used to print the 'present working directory'.



```
super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM
$ pwd
/d/allGit/SCM

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM
$
```

2. ls

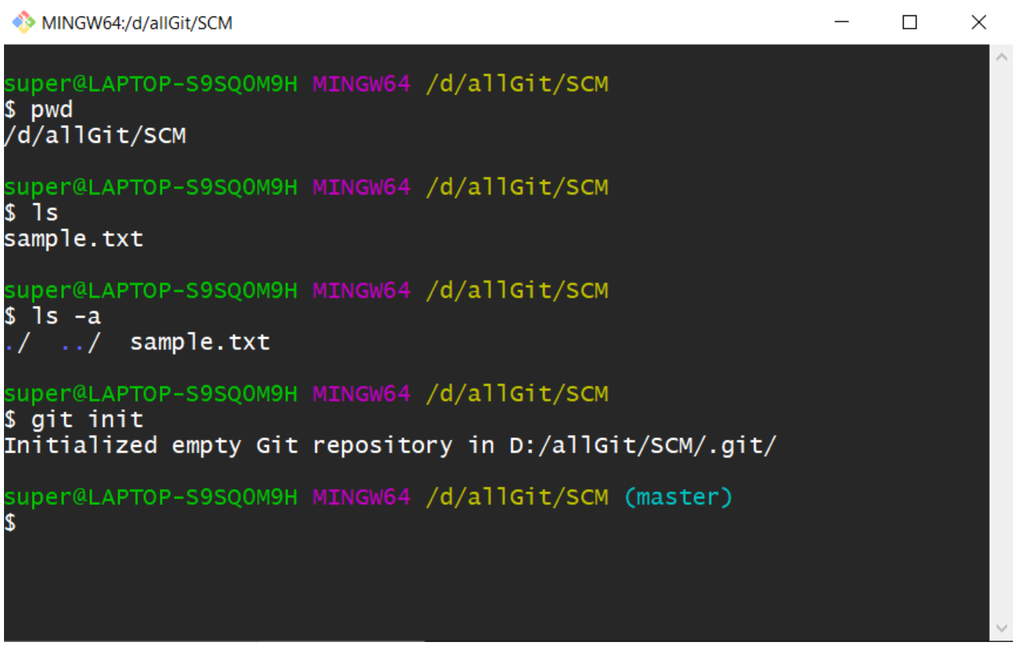
The ls command lists the current directory contents and by default will not show hidden files. If you pass it the -a flag, it will display hidden files. You can navigate into the .git directory like any other normal directory.



```
MINGW64:/d/allGit/SCM

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/SCM
$ pwd
/d/allGit/SCM

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/SCM
$ ls
sample.txt

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/SCM
$ ls -a
./  ../  sample.txt

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/SCM
$
```

3. git init

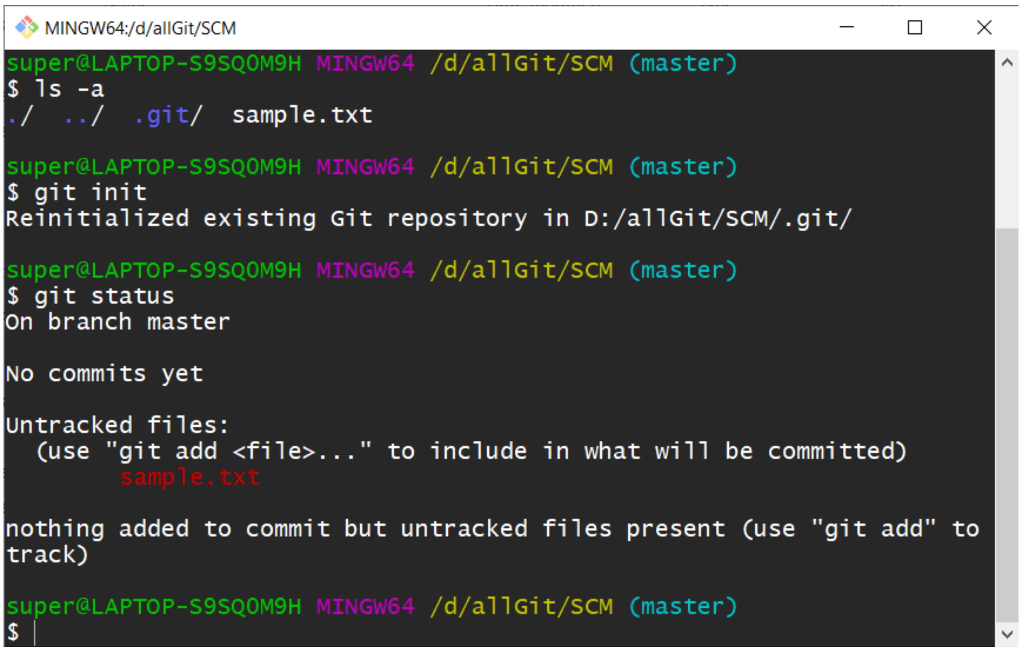
The git init command is used to generate a new, empty Git repository or to reinitialize an existing one. With the help of this command, a .git subdirectory is created, which includes the metadata, like subdirectories for objects and template files, needed for generating a new Git repository.



```
MINGW64:/d/allGit/SCM

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/SCM
$ pwd
/d/allGit/SCM

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/SCM
$ ls
sample.txt

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/SCM
$ ls -a
./  ../  sample.txt

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/SCM
$ git init
Initialized empty Git repository in D:/allGit/SCM/.git/

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/SCM (master)
$
```

4. git status

The git status command displays the state of the working directory and the staging area

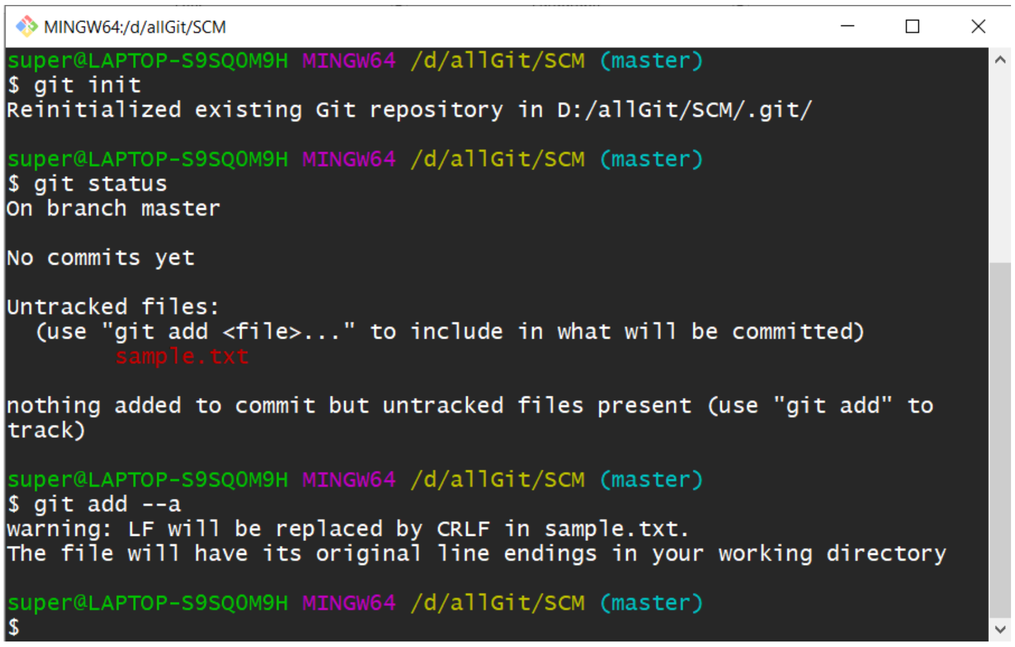


MINGW64:/d/allGit/SCM

```
super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ ls -a
./  ../  .git/  sample.txt

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git init
Reinitialized existing Git repository in D:/allGit/SCM/.git/

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git status
On branch master

No commits yet

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

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

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ 
```

5. git add --a

The git add --a command is used to add file contents to the Index (Staging Area). This command updates the current content of the working tree to the staging area. It also prepares the staged content for the next commit.

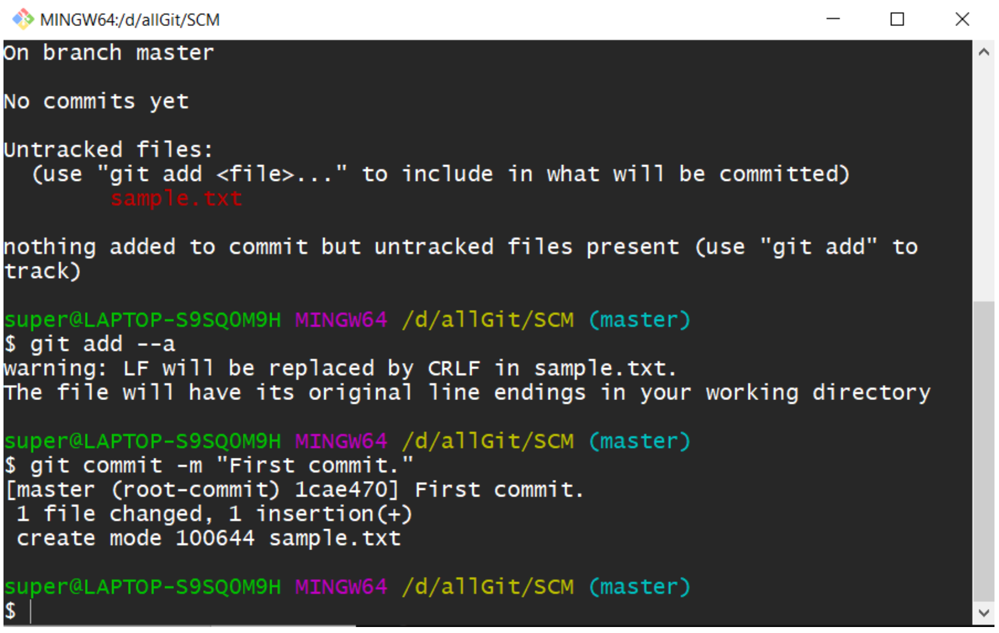


MINGW64:/d/allGit/SCM

```
super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git init
Reinitialized existing Git repository in D:/allGit/SCM/.git/

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git status
On branch master

No commits yet

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

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

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git add --a
warning: LF will be replaced by CRLF in sample.txt.
The file will have its original line endings in your working directory

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$
```

6. git commit -m “message”

The git commit -m command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as “safe” versions of a project



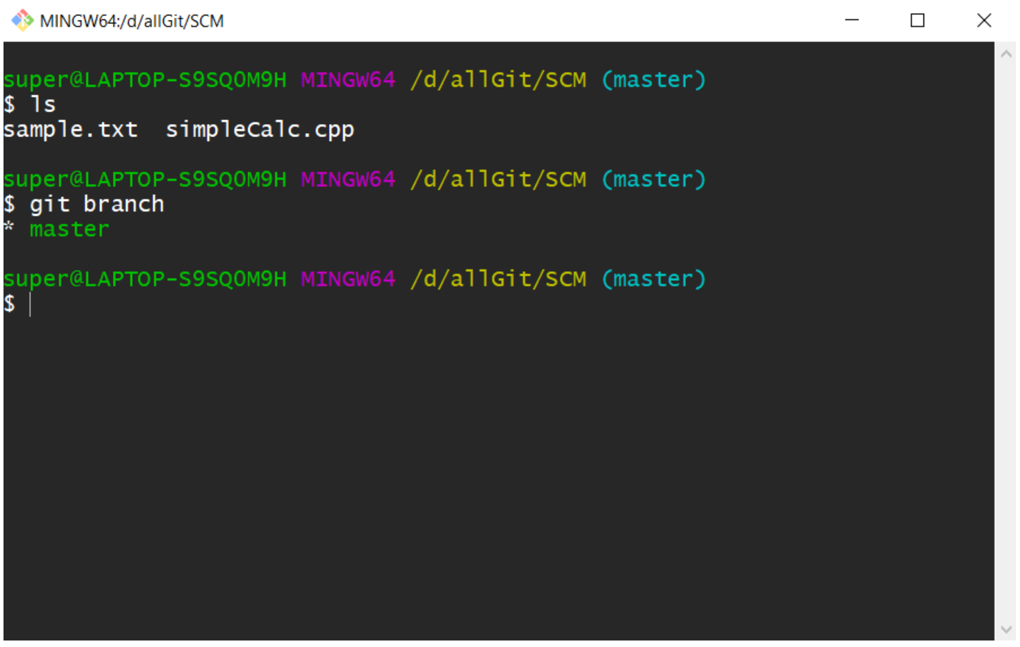
MINGW64:/d/allGit/SCM

```
On branch master

No commits yet

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

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

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git add --a
warning: LF will be replaced by CRLF in sample.txt.
The file will have its original line endings in your working directory

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git commit -m "First commit."
[master (root-commit) 1cae470] First commit.
 1 file changed, 1 insertion(+)
 create mode 100644 sample.txt

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ 
```

7. git branch

A branch in Git is an independent line of work (a pointer to a specific commit). It allows users to create a branch from the original code (master branch) and isolate their work. Branches allow you to work on different parts of a project without impacting the main branch.

The git branch command is used to List all of the branches in your repository.

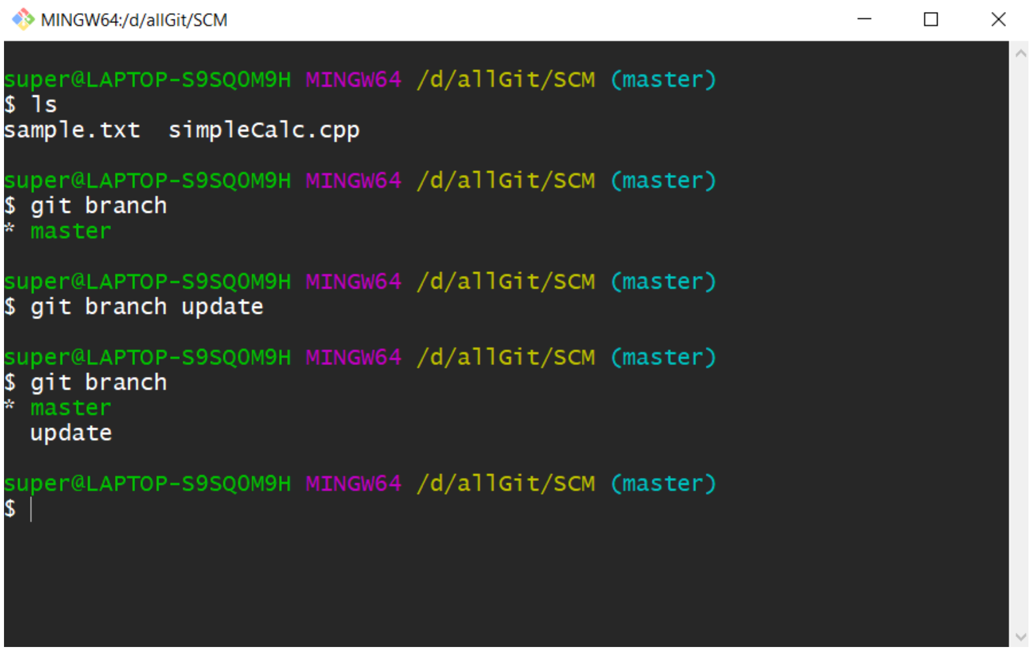


MINGW64:/d/allGit/SCM

```
super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ ls
sample.txt  simpleCalc.cpp

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git branch
* master

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ 
```

8. git branch <branch>

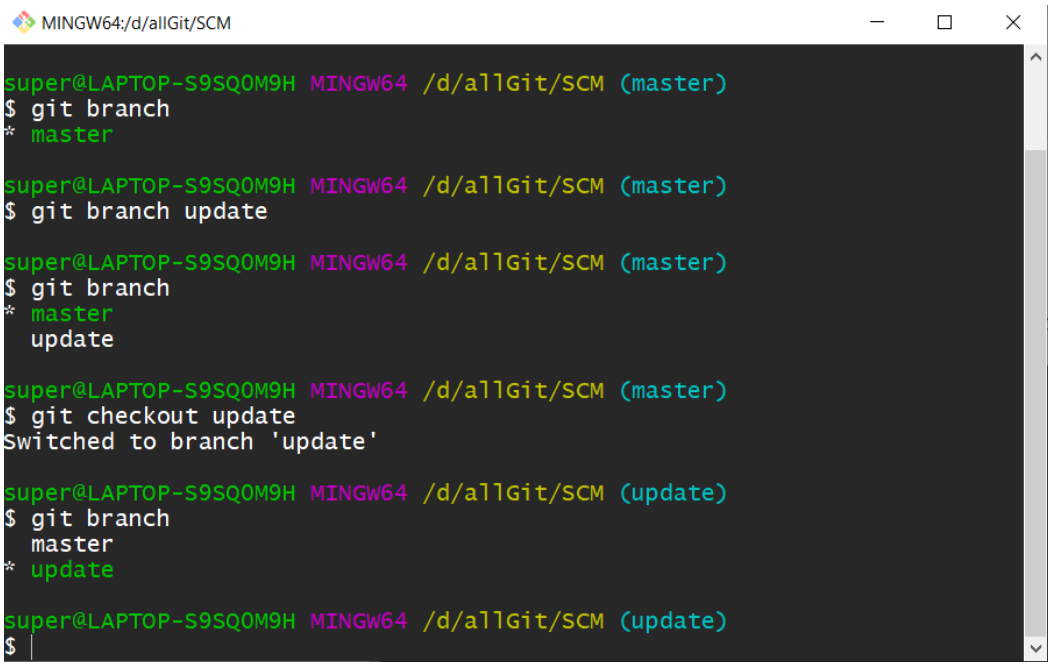
The git branch <branch> command is used to Create a new branch called <branch>.



```
MINGW64:/d/allGit/SCM
super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ ls
sample.txt  simpleCalc.cpp

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git branch
* master

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git branch update

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git branch
* master
  update

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ |
```

9. git checkout <branch name>

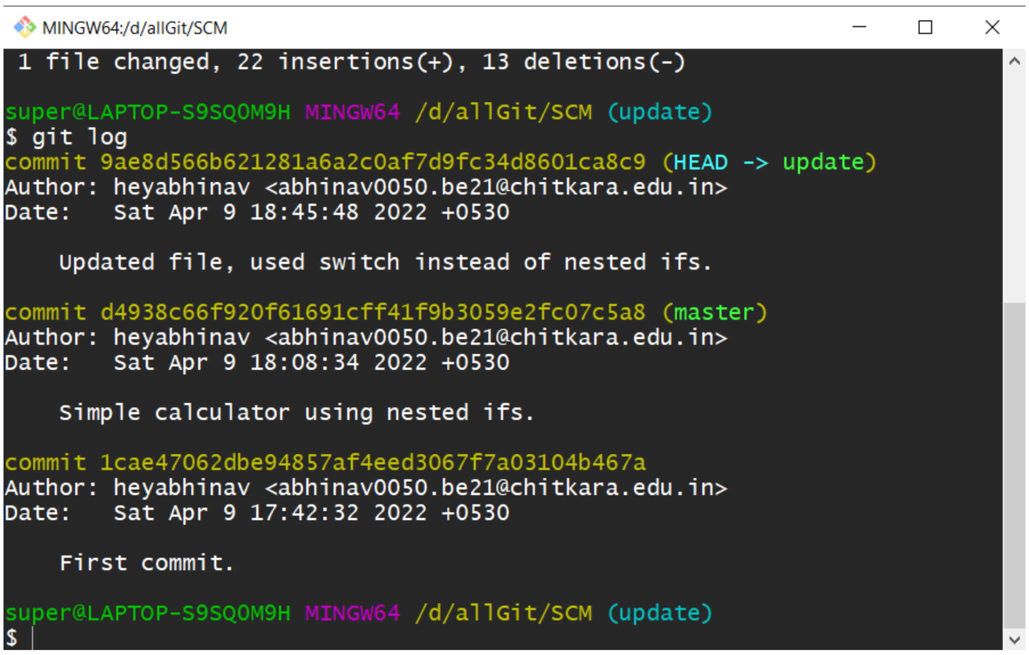
The git checkout command is used to switch the currently active branch.



```
MINGW64:/d/allGit/SCM
super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git branch
* master

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git branch update

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git branch
* master
  update

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git checkout update
Switched to branch 'update'

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (update)
$ git branch
  master
* update

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (update)
$ |
```

10. git log

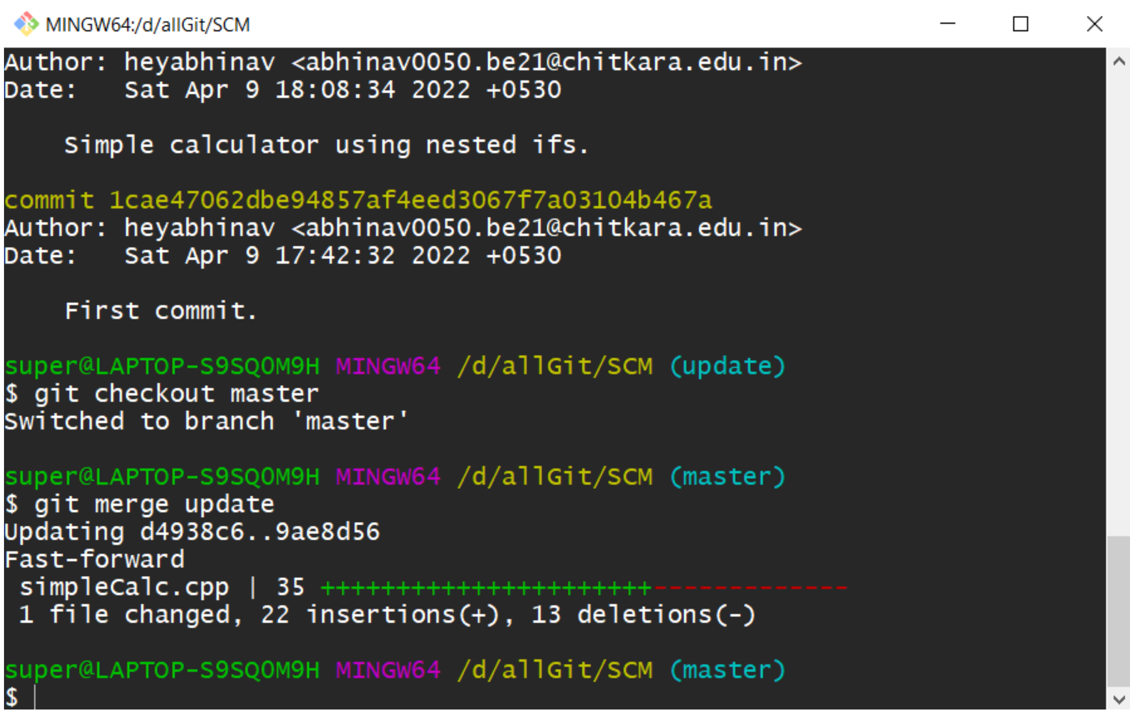
The git log command shows a list of all the commits made to a repository.


MINGW64:/d/allGit/SCM

```
 1 file changed, 22 insertions(+), 13 deletions(-)

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (update)
$ git log
commit 9ae8d566b621281a6a2c0af7d9fc34d8601ca8c9 (HEAD -> update)
Author: heyabhinav <abhinav0050.be21@chitkara.edu.in>
Date:   Sat Apr 9 18:45:48 2022 +0530

    Updated file, used switch instead of nested ifs.

commit d4938c66f920f61691cff41f9b3059e2fc07c5a8 (master)
Author: heyabhinav <abhinav0050.be21@chitkara.edu.in>
Date:   Sat Apr 9 18:08:34 2022 +0530

    Simple calculator using nested ifs.

commit 1cae47062dbe94857af4eed3067f7a03104b467a
Author: heyabhinav <abhinav0050.be21@chitkara.edu.in>
Date:   Sat Apr 9 17:42:32 2022 +0530

    First commit.

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (update)
$ 
```

11. git merge <branch name>

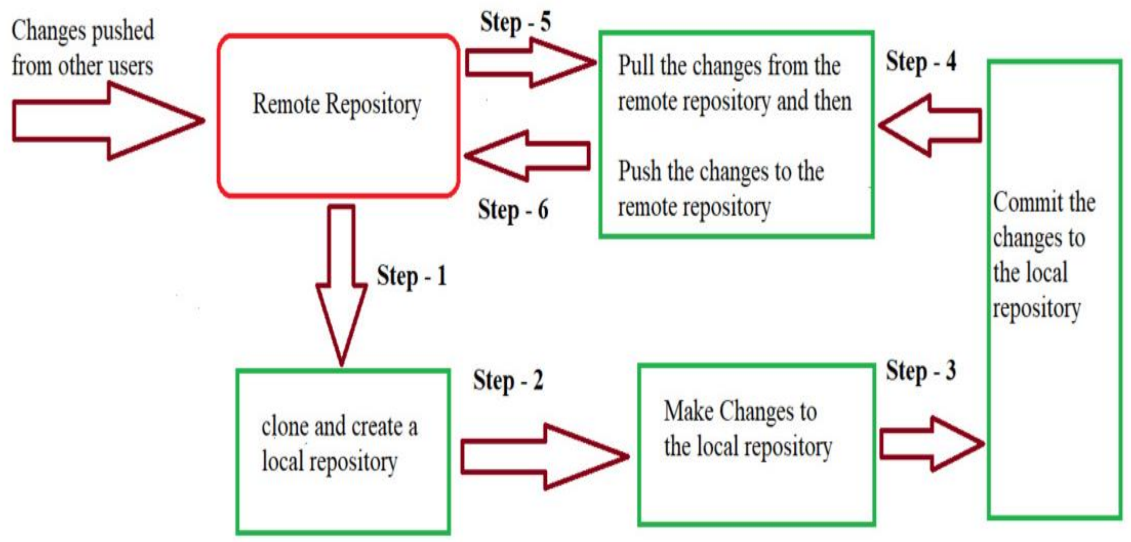
The git merge command is used to merge the branches.


MINGW64:/d/allGit/SCM

```
Author: heyabhinav <abhinav0050.be21@chitkara.edu.in>
Date:   Sat Apr 9 18:08:34 2022 +0530

    Simple calculator using nested ifs.

commit 1cae47062dbe94857af4eed3067f7a03104b467a
Author: heyabhinav <abhinav0050.be21@chitkara.edu.in>
Date:   Sat Apr 9 17:42:32 2022 +0530

    First commit.

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (update)
$ git checkout master
Switched to branch 'master'

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ git merge update
Updating d4938c6..9ae8d56
Fast-forward
 simpleCalc.cpp | 35 ++++++++++++++++++++++-------------
 1 file changed, 22 insertions(+), 13 deletions(-)

super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/SCM (master)
$ 
```

Git Life Cycle:-

Git is used in our day-to-day work, we use git for keeping a track of our files, working in a collaboration with our team, to go back to our previous code versions if we face some error. Git helps us in many ways. Let us look at the Life Cycle that git has and understand more about its life cycle. Let us see some of the basic steps that we follow while working with Git


Changes pushed from other users
Remote Repository
Step - 5
Pull the changes from the remote repository and then
Push the changes to the remote repository
Step - 6
Step - 4
Commit the changes to the local repository
Step - 1
clone and create a local repository
Step - 2
Make Changes to the local repository
Step - 3

- **In Step – 1**, We first clone any of the code residing in the remote repository to make our won local repository.
- **In Step-2** we edit the files that we have cloned in our local repository and make the neces sary changes in it.
- **In Step-3** we commit our changes by first adding them to our staging area and committin g them with a commit message.
- **In Step – 4 and Step-5** we first check whether there are any of the changes done in the re mote repository by some other users and we first pull that changes.
- If there are no changes we directly proceed with **Step – 6** in which we push our changes t o the remote repository and we are done with our work.

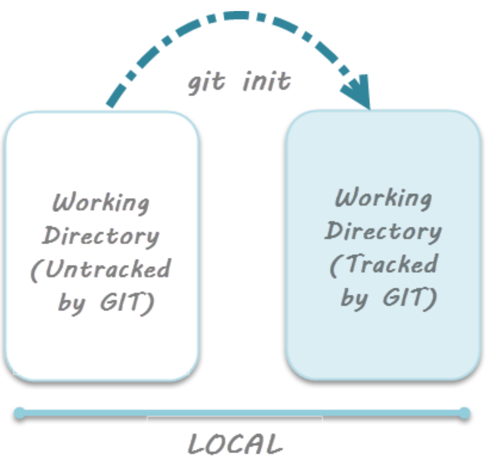
When a directory is made a git repository, there are mainly 3 states which make the essence of Git Version Control System. The three states are-

• Working directory
• Staging area
• Git directory

Working Directory

Whenever we want to initialize our local project directory to make it a git repository, we use the ***git init*** command. After this command, git becomes aware of the files in the project although it doesn’t track the files yet. The files are further tracked in the staging area.

git init


git init
Working Directory (Untracked by GIT)
Working Directory (Tracked by GIT)
LOCAL

Staging Area

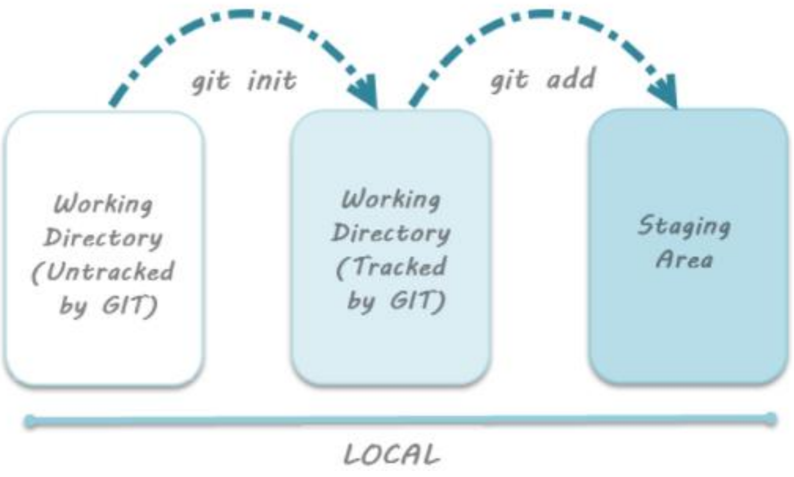
Now, to track the different versions of our files we use the command ***git add***. We can term a staging area as a place where different versions of our files are stored. ***git add*** command copies the version of your file from your working directory to the staging area. We can, however, choose which files we need to add to the staging area because in our working directory there are some files that we don't want to get tracked, examples include node modules, env files, temporary files, etc. Indexing in Git is the one that helps Git in understanding which files need to be added or sent. You can find your staging area in the ***.git*** folder inside the ***index*** file.

// to specify which file to add to the staging area

git add <filename>

// to add all files of the working directory to the staging area

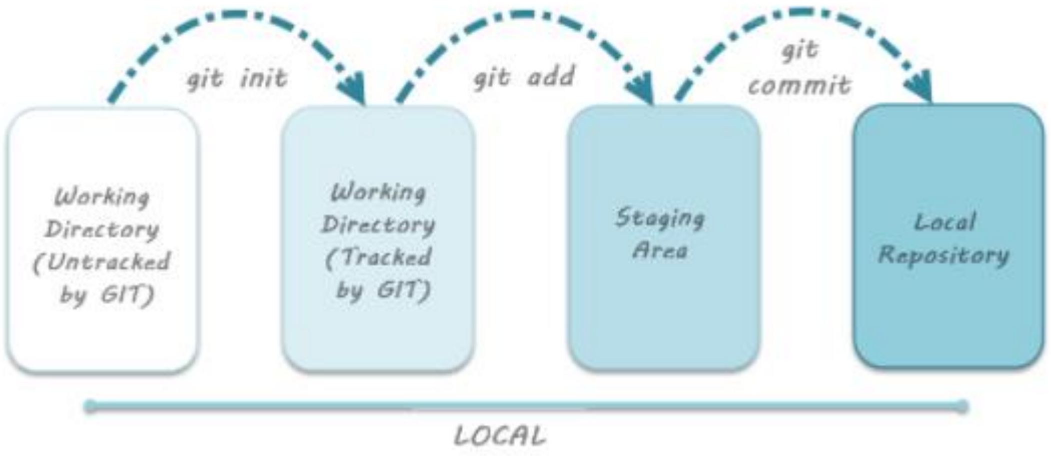
git add .


git init
git add
Working Directory (Untracked by GIT)
Working Directory (Tracked by GIT)
Staging Area
LOCAL

Git Directory

Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit our files using the ***git commit*** command. Commit helps us in keeping the track of the metadata of the files in our staging area. We specify every commit with a message which tells what the commit is about. Git preserves the information or the metadata of the files that were committed in a Git Directory which helps Git in tracking files and basically it preserves the photocopy of the committed files. Commit also stores the name of the author who did the commit, files that are committed, and the date at which they are committed along with the commit message.

git commit -m "<Commit Message>"


git init
git add
git commit
Working Directory (Untracked by GIT)
Working Directory (Tracked by GIT)
Staging Area
Local Repository
LOCAL

Source Code Management (CS181)

CSE Batch 2021

Submitted by:

Student Name: Abhinav Gupta

Roll no: 2110990050

Group No.: G01

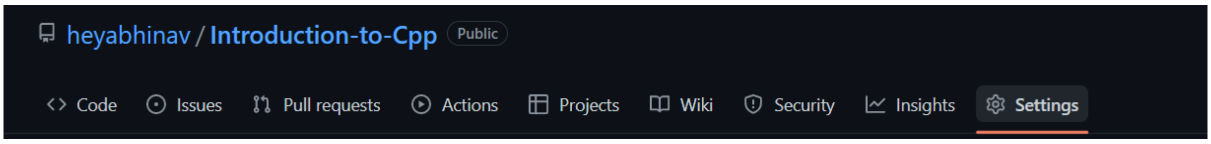
Add collaborators on GitHub Repo

What is GitHub Collaboration?
GitHub collaboration is a space where you can invite another developer to your repository and work together at an organization level, splitting the task and 2nd person will have the rights to add or merge files to the main repository without further permission.

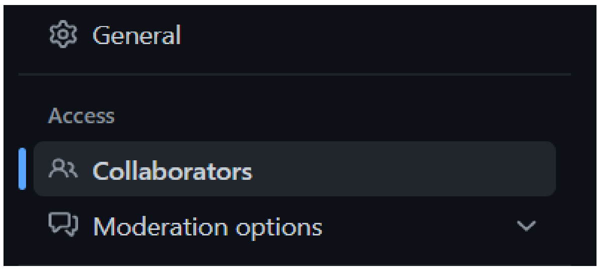
Let's invite a Collaborator
In this post, we will create a new repository and invite a collaborator to our repository by sending an invitation. A detailed procedure on how to invite a collaborator to your GitHub repository is mentioned below.

Step 1: On GitHub.com, navigate to the main page of the repository.

Step 2: Under your repository name, click **Settings**.

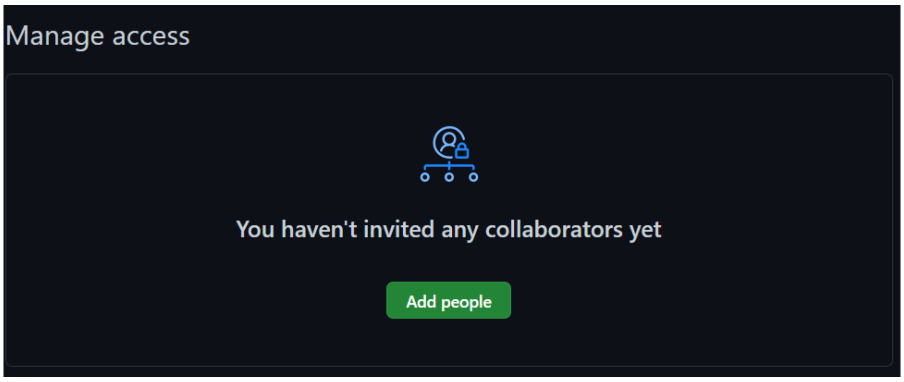


Step 3: In the "Access" section of the sidebar, click **Collaborators & teams**.

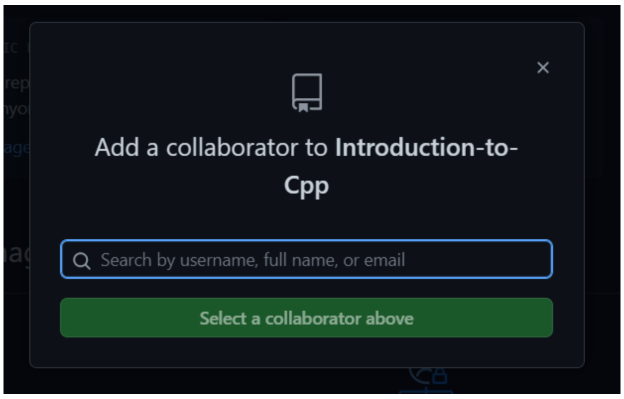


Step 4: Enter your GitHub account password for confirmation.

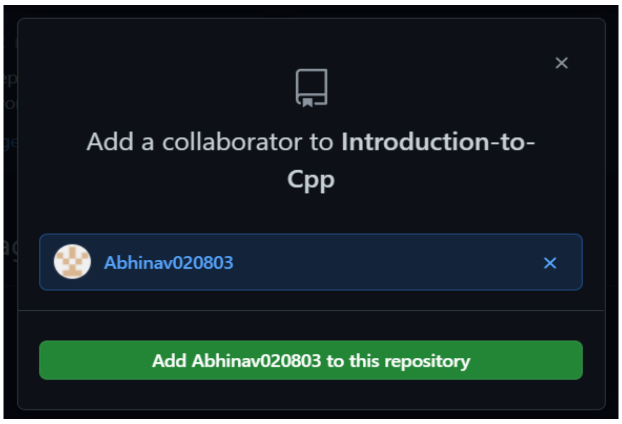
Step 5: Click on **Add people**.



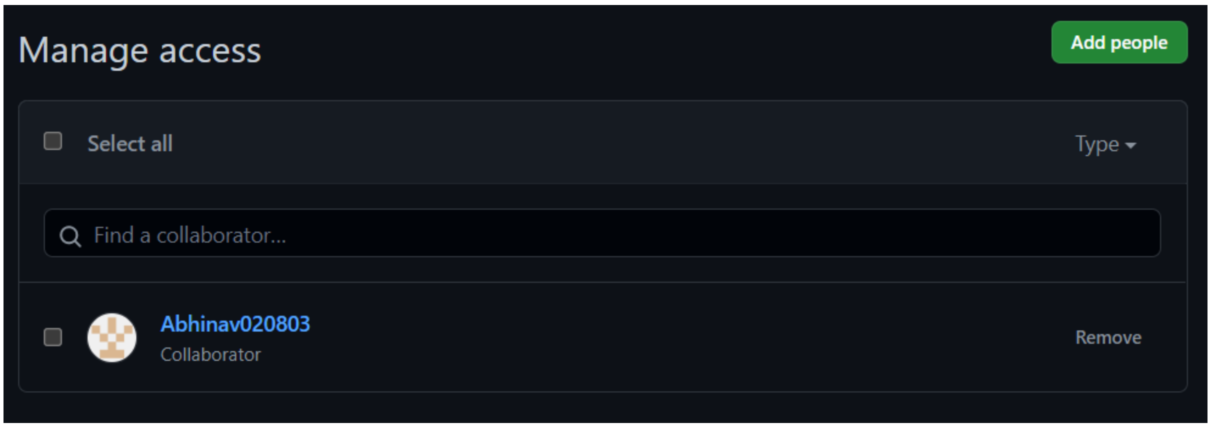
Step 6: In the search field, start typing the name of person you want to invite, then click a name in the list of matches.


Add a collaborator to **Introduction-to-Cpp**
Search by username, full name, or email
Select a collaborator above

Step 7: Click **Add NAME to REPOSITORY**.


Add a collaborator to **Introduction-to-Cpp**
Abhinav020803
Add Abhinav020803 to this repository

Step 8: The user will receive an email inviting them to the repository. Once they accept your invitation, they will have collaborator access to your repository.



How to delete a collaborator from GitHub Repository?

Go to the Repository, Click on Manage access under settings tab, you can see a list of collaborators under Mange access, on right side of each listing there is a delete icon

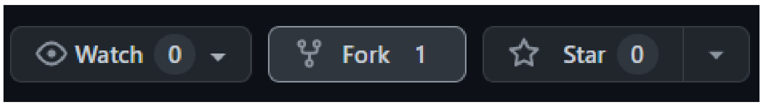
Fork and Commit

What is Forking in GitHub?
A fork is a copy of a repository that we manage. Forks let us make changes to a project without affecting the original repository. We can fetch updates from or submit changes to the original repository with pull requests.

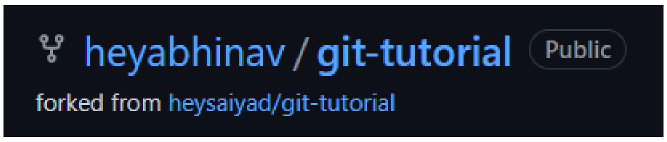
Forking a repository

Step 1: On GitHub.com, navigate to the repository you wish to fork.

Step 2: In the top-right corner of the page, click **Fork**.



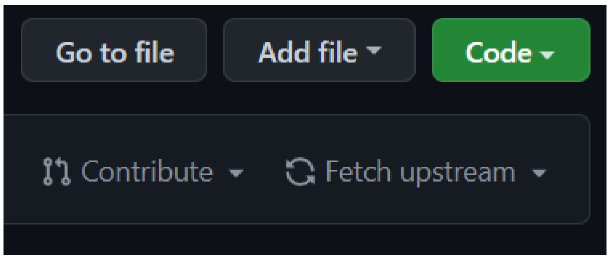
Step 3: Now go to your profile and see the latest forked repository in your repository tab.



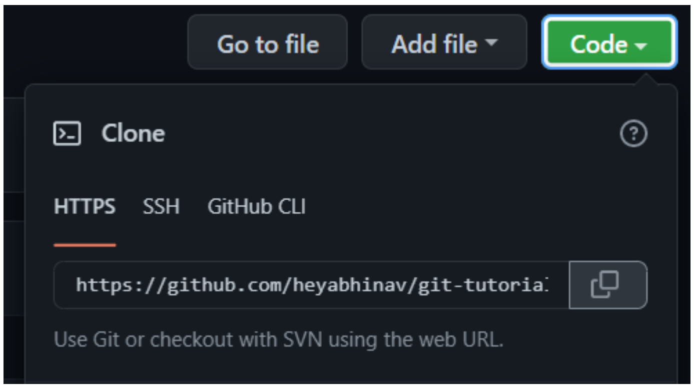
Cloning forked repository

Step 1: On GitHub.com, navigate to forked repository.

Step 2: Above the list of files, click **Code**.



Step 3: To clone the repository using HTTPS, under "Clone with HTTPS", copy the link.



```
https://github.com/heyabhinav/git-tutoria
```

Use Git or checkout with SVN using the web URL.

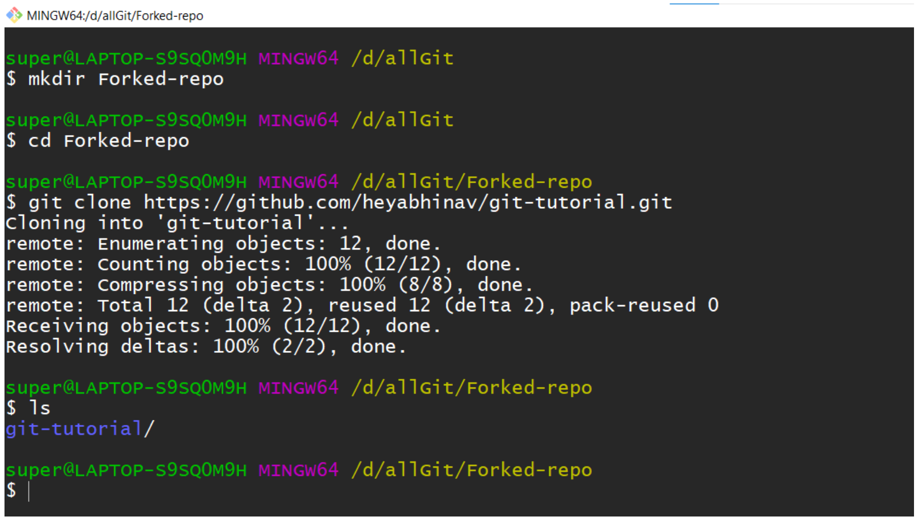
Step 4: Open Git Bash.

Step 5: Change the working directory to the location where you want to clone the repo.

Step 6: Type **git clone**, and then paste the URL you copied earlier. It will look like this, with your GitHub username instead of YOUR-USERNAME:

$ git clone https://github.com/heyabhinav/git-tutorial.git

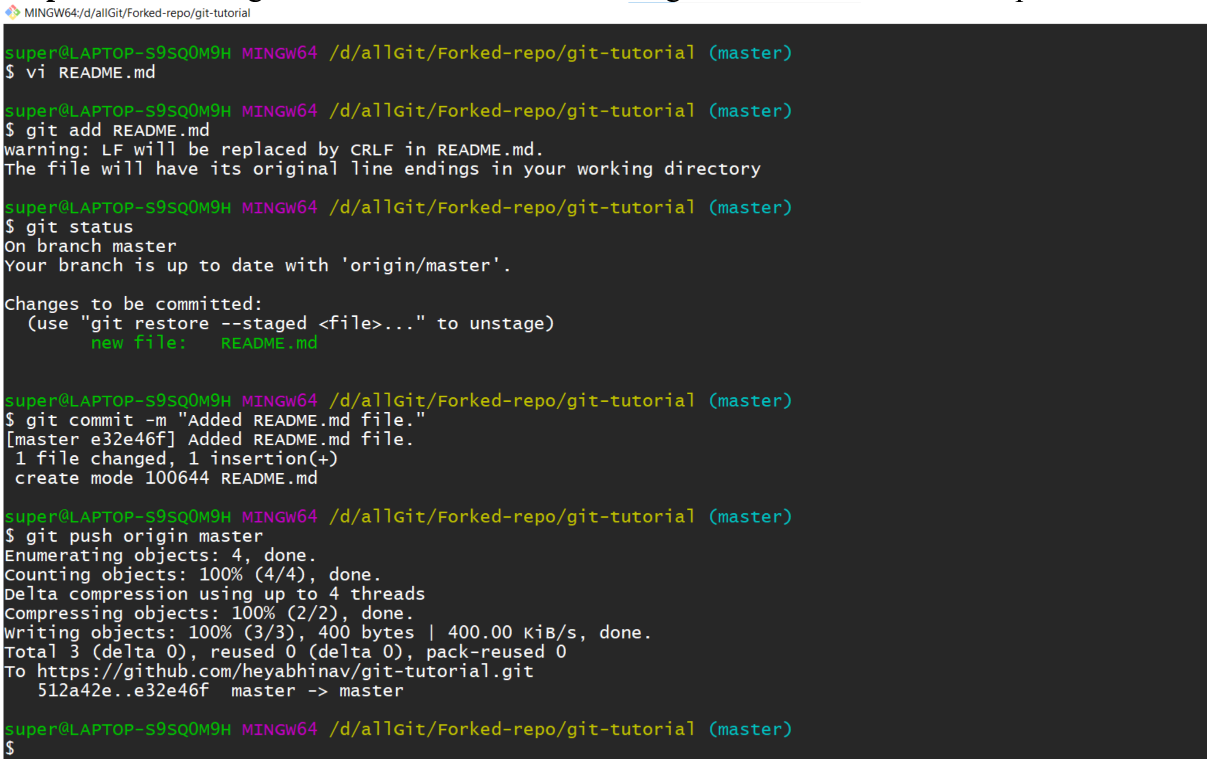
Step 7: Press Enter. Your local clone will be created.



```
MINGW64:/d/allGit/Forked-repo

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit
$ mkdir Forked-repo

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit
$ cd Forked-repo

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo
$ git clone https://github.com/heyabhinav/git-tutorial.git
Cloning into 'git-tutorial'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 12 (delta 2), reused 12 (delta 2), pack-reused 0
Receiving objects: 100% (12/12), done.
Resolving deltas: 100% (2/2), done.

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo
$ ls
git-tutorial/

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo
$ |
```

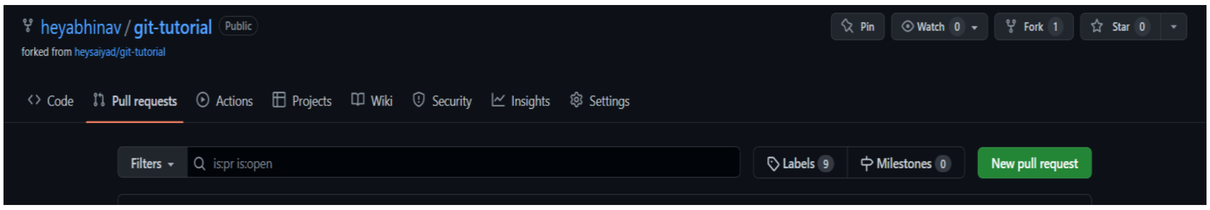
Creating a pull request

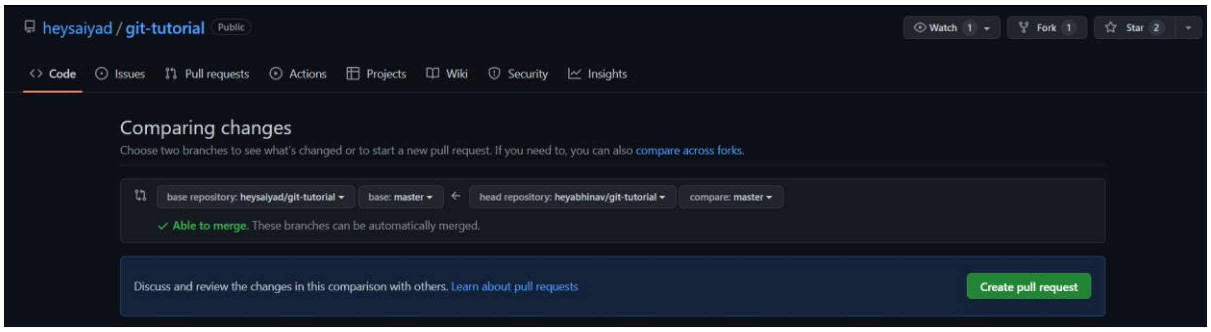
Step 1: Make changes to the file and it's time to stage and commit the file and push.



```
MINGW64:/d/allGit/Forked-repo/git-tutorial

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/git-tutorial (master)
$ vi README.md

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/git-tutorial (master)
$ git add README.md
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/git-tutorial (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   README.md


super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/git-tutorial (master)
$ git commit -m "Added README.md file."
[master e32e46f] Added README.md file.
 1 file changed, 1 insertion(+)
 create mode 100644 README.md

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/git-tutorial (master)
$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 400 bytes | 400.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/heyabhinav/git-tutorial.git
   512a42e..e32e46f  master -> master

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/git-tutorial (master)
$
```

Step 2: Now on the main branch, go to **Pull requests** section and click on **New pull request** button.





Comparing changes

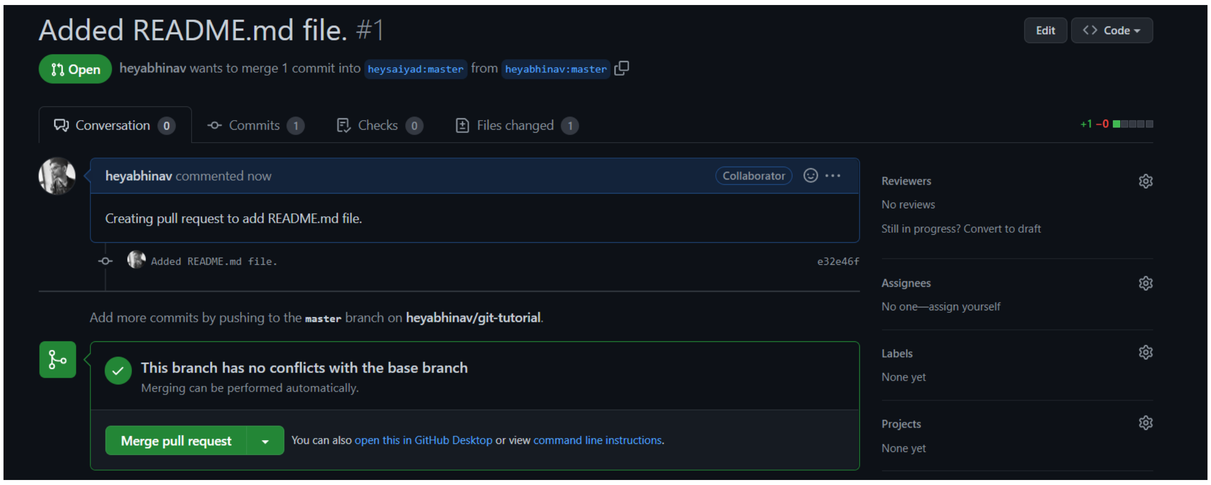
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. Learn about pull requests

Step 3: Make sure to add a good comment in the merge and explain what's your commit is all about.

Step 4: In few Organisations, there will be Github actions scheduled for auto-check and any one of the code reviewers will review the file and approve the changes or suggest any issue with your pull request.



Added README.md file. #1

heyabhinav wants to merge 1 commit into heysaiyad:master from heyabhinav:master

heyabhinav commented now

Creating pull request to add README.md file.

Added README.md file. e32e46f

Add more commits by pushing to the master branch on heyabhinav/git-tutorial.

This branch has no conflicts with the base branch

Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Merge and Resolve conflicts due to own activity and collaborators activity

Understanding GitHub Conflicts

Let's imagine a case where two developers working on one repository by creating 2 different branch and same code line and both of them pulled the file to main repository, Now the GitHub is good understanding the code and merge the conflicts automatically but it fails understanding if developer has put any comments. Now Let's look into How to resolve merge conflicts in GitHub.

Let's resolve merge conflict:

Step 1: Create a branch from the master/base branch and checkout that branch using **git checkout <branch name>** command.

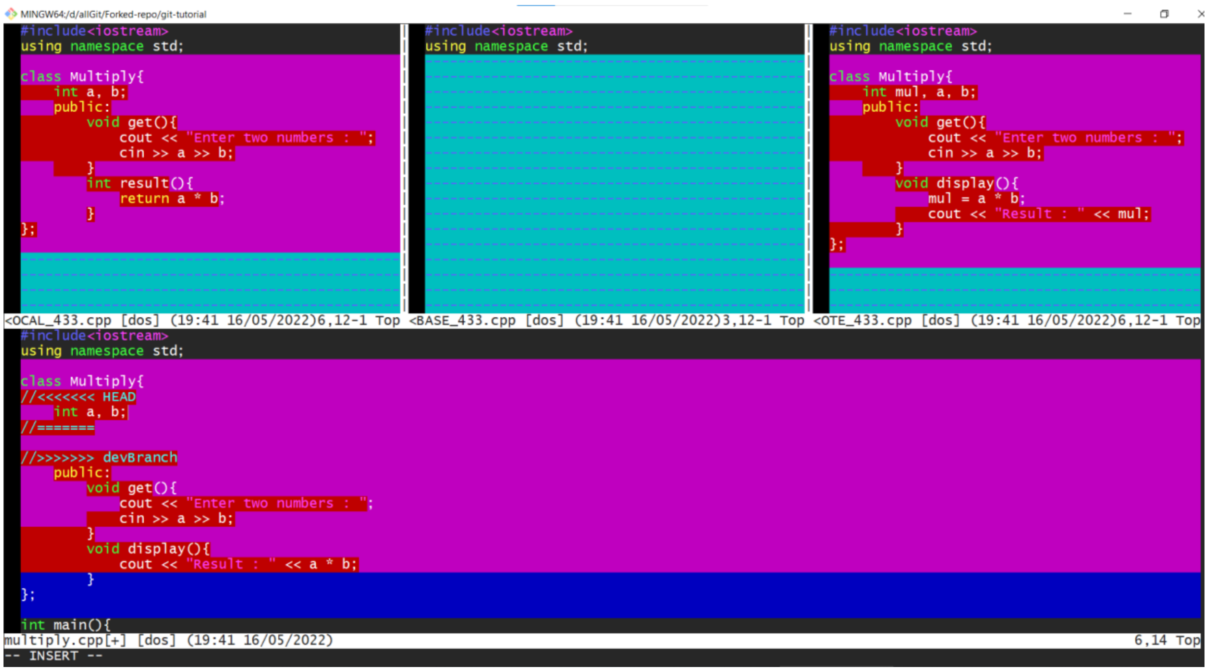
Step 2: Modify the files you wished to, then stage that file and then commit.

Step 3: Now, checkout to base branch, make changes to files if you wish else merge the branch to base branch using "**git merge <branch to be merged>**" command.

```
super@LAPTOP-S9SQOM9H MINGW64 /d/allGit/Forked-repo/git-tutorial (master)
$ git merge devBranch
Auto-merging multiply.cpp
CONFLICT (content): Merge conflict in multiply.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

Git hasn't automatically created a new merge commit. It has paused the process while you resolve the conflict.

Step 4: If you want to use a graphical tool to resolve these issues, you can run **git mergetool**, which fires up an appropriate visual merge tool and walks you through the conflicts



MINGW64:/d/allGit/Forked-repo/git-tutorial

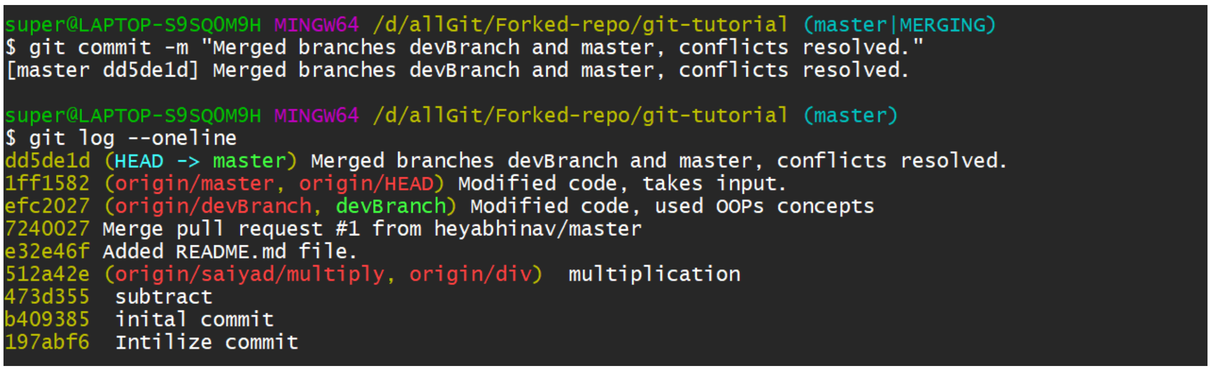
```
#include<iostream>
using namespace std;

class Multiply{
    int a, b;
    public:
        void get(){
            cout << "Enter two numbers : ";
            cin >> a >> b;
        }
        int result(){
            return a * b;
        }
};
<OCAL_433.cpp [dos] (19:41 16/05/2022)6,12-1 Top
```

```
#include<iostream>
using namespace std;
<BASE_433.cpp [dos] (19:41 16/05/2022)3,12-1 Top
```

```
#include<iostream>
using namespace std;

class Multiply{
    int mul, a, b;
    public:
        void get(){
            cout << "Enter two numbers : ";
            cin >> a >> b;
        }
        void display(){
            mul = a * b;
            cout << "Result : " << mul;
        }
};
<OTE_433.cpp [dos] (19:41 16/05/2022)6,12-1 Top
```

```
#include<iostream>
using namespace std;

class Multiply{
//<<<<<<< HEAD
    int a, b;
//=======

//>>>>>>> devBranch
    public:
        void get(){
            cout << "Enter two numbers : ";
            cin >> a >> b;
        }
        void display(){
            cout << "Result : " << a * b;
        }
};

int main(){
multiply.cpp[+] [dos] (19:41 16/05/2022)                                6,14 Top
-- INSERT --
```

Step 5: After exiting merge tool, if you're happy with that, and you verify that everything that had conflicts has been staged, you can type **git commit** to finalize the merge commit.



```
super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/git-tutorial (master|MERGING)
$ git commit -m "Merged branches devBranch and master, conflicts resolved."
[master dd5de1d] Merged branches devBranch and master, conflicts resolved.

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/git-tutorial (master)
$ git log --oneline
dd5de1d (HEAD -> master) Merged branches devBranch and master, conflicts resolved.
1ff1582 (origin/master, origin/HEAD) Modified code, takes input.
efc2027 (origin/devBranch, devBranch) Modified code, used OOPs concepts
7240027 Merge pull request #1 from heyabhinav/master
e32e46f Added README.md file.
512a42e (origin/saiyad/multiply, origin/div)  multiplication
473d355  subtract
b409385  inital commit
197abf6  Intilize commit
```

You can modify commit message with details about how you resolved the merge and explain why you did the changes you made if these are not obvious.

Reset and Revert

How to reset a Git commit

Let's start with the Git command reset. Practically, you can think of it as a "rollback"—it points your local environment back to a previous commit. By "local environment," we mean your local repository, staging area, and working directory.

Take a look at Figure 1. Here we have a representation of a series of commits in Git. A branch in Git is simply a named, movable pointer to a specific commit. In this case, our branch *master* is a pointer to the latest commit in the chain.

```
$ git log --oneline
b764644 File with three lines
7c709f0 File with two lines
9ef9173 File with one line
```

What happens if we want to roll back to a previous commit. Simple—we can just move the branch pointer. Git supplies the reset command to do this for us. For example, if we want to reset master to point to the commit two back from the current commit, we could use either of the following methods:

```
$ git reset 9ef9173 (using an absolute commit SHA1 value 9ef9173)
or
$ git reset current~2 (using a relative value -2 before the "current" tag)
```

Figure 2 shows the results of this operation. After this, if we execute a git log command on the current branch (master), we'll see just the one commit.

```
$ git log –oneline
9ef9173 File with one line
```

The git reset command also includes options to update the other parts of your local environment with the contents of the commit where you end up. These options include: hard to reset the commit being pointed to in the repository, populate the working directory with the contents of the commit, and reset the staging area; soft to only reset the pointer in the repository; and mixed (the default) to reset the pointer and the staging area.

Using these options can be useful in targeted circumstances such as git reset -- hard . This overwrites any local changes you haven't committed. In effect, it resets (clears out) the staging area and overwrites content in the working directory with the content from the commit you reset to. Before you use the hard option, be sure that's what you really want to do, since the command overwrites any uncommitted changes.

How to revert a Git commit

The net effect of the git revert command is similar to reset, but its approach is different. Where the reset command moves the branch pointer back in the chain (typically) to "undo" changes, the revert command adds a new commit at the end of the chain to "cancel" changes. The effect is most easily seen by looking at Figure 1 again. If we add a line to a file in each commit in the chain, one way to get back to the version with only two lines is to reset to that commit, i.e., *git reset HEAD~1*

Another way to end up with the two-line version is to add a new commit that has the third line removed—effectively cancelling out that change. This can be done with a git revert command, such as:

$ git revert HEAD

Because this adds a new commit, Git will prompt for the commit message:

```
Revert "File with three lines"

This reverts commit
b764644bad524b804577684bf74e7bca3117f554.

# Please enter the commit message for your changes.
Lines starting
# with '#' will be ignored, and an empty message aborts
the commit.
# On branch master
# Changes to be committed:
#       modified:   file1.txt
#
```

Figure 3 (below) shows the result after the revert operation is completed.

If we do a git log now, we'll see a new commit that reflects the contents before the previous commit

```
$ git log --oneline
11b7712 Revert "File with three lines"
b764644 File with three lines
7c709f0 File with two lines
9ef9173 File with one line
```

Here are the current contents of the file in the working directory:

```
$ cat <filename>
Line 1
Line 2
```

Introduction:

This task is performed in the group of four. Each one of us made it possible to work on this project as if we are doing an open source contribution.

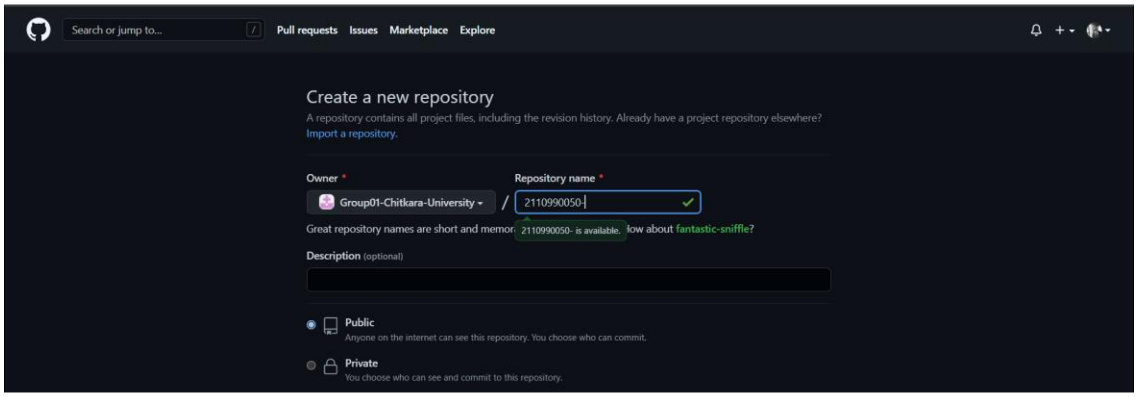
Each one of us create his/her repo and rest of the three contributors in the repo, firstly forked that repo and then clone it in our local machine and then make a new branch and made some changes in the existing file in master branch in the repo and then push it from your local system.

And finally make pull request to the owner of the repo in whose repo we want to make the changes.

Creating a repository:

Step 1: Went to GitHub and created new Repository, click on the + sign on top right side and drop down will appear click on New Repository.

Step 2: Specify the Name of the Project, make it public or private. Then click on Create repository. You can also see by default branch name is main if you want you can change it.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * Group01-Chitkara-University / Repository name * 2110990050-

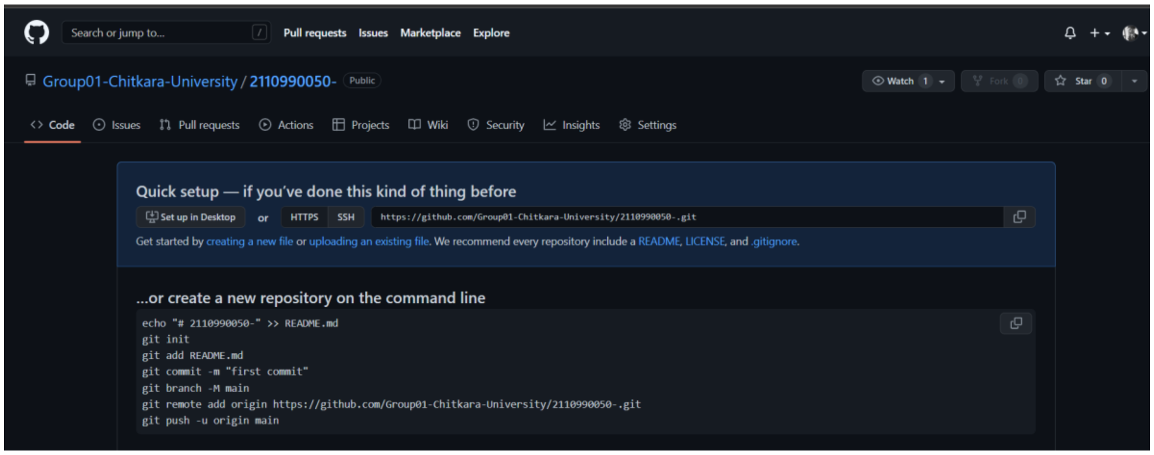
Great repository names are short and memor 2110990050- is available. łow about fantastic-sniffle?

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.

Step 3: Now a distributed repository is created on GitHub. Now owner of the repo can add collaborators to the repo or anyone can fork it and open pull request and contribute.



Group01-Chitkara-University / 2110990050- Public

Quick setup — if you've done this kind of thing before

https://github.com/Group01-Chitkara-University/2110990050-.git

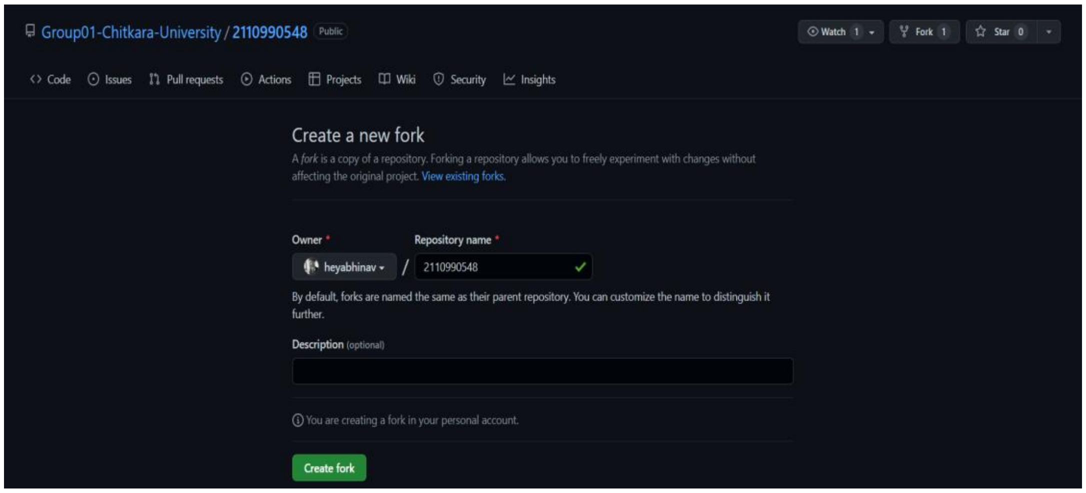
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# 2110990050-" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Group01-Chitkara-University/2110990050-.git
git push -u origin main
```

Fork and Cloning the repo:

Step 1: Create fork by selecting yourself as owner, and click on create fork.



Group01-Chitkara-University / 2110990548 Public

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.

Owner * heyabhinav / Repository name * 2110990548

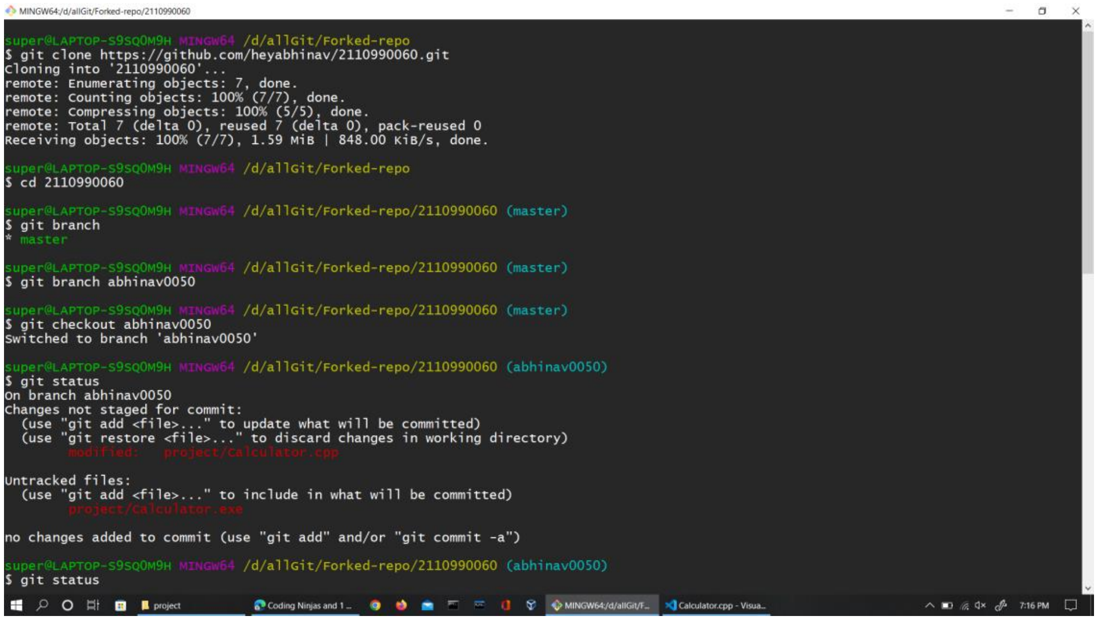
By default, forks are named the same as their parent repository. You can customize the name to distinguish it further.

Description (optional)

You are creating a fork in your personal account.

Create fork

Step 2: Cloning in your remote system and making a branch to add your changes and finally pulling a pull request.

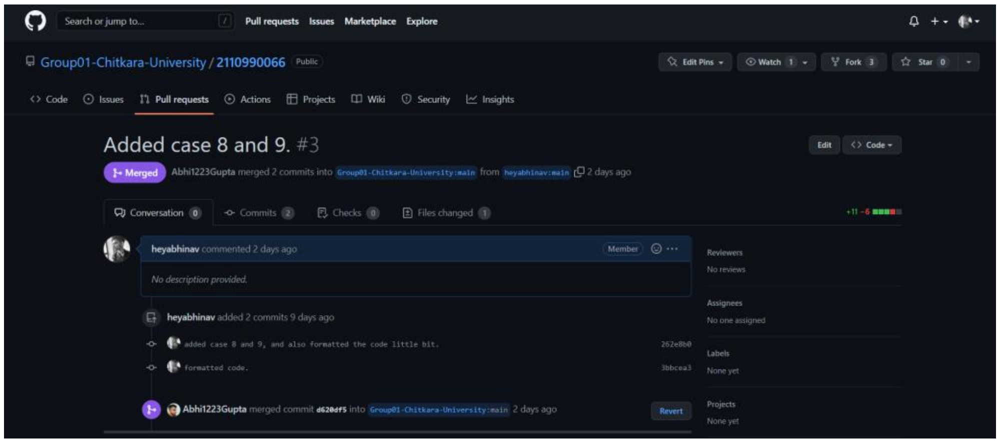


```
super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo
$ git clone https://github.com/heyabhinav/2110990060.git
Cloning into '2110990060'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 0), reused 7 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), 1.59 MiB | 848.00 KiB/s, done.

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo
$ cd 2110990060

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/2110990060 (master)
$ git branch
* master

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/2110990060 (master)
$ git branch abhinav0050

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/2110990060 (master)
$ git checkout abhinav0050
Switched to branch 'abhinav0050'

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/2110990060 (abhinav0050)
$ git status
On branch abhinav0050
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   project/Calculator.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        project/Calculator.exe

no changes added to commit (use "git add" and/or "git commit -a")

super@LAPTOP-S9SQ0M9H MINGW64 /d/allGit/Forked-repo/2110990060 (abhinav0050)
$ git status
```

Step 3: Create a pull request by clicking on compare and pull request.

Open and close pull request: Overview of the pull request sent by the contributors.

2110990066: Added switch case 8 and 9 to his switch.cpp file.



Group01-Chitkara-University / 2110990066 Public

Added case 8 and 9. #3

Merged Abhi1223Gupta merged 2 commits into `Group01-Chitkara-University:main` from `heyabhinav:main` 2 days ago

Conversation 0 | Commits 2 | Checks 0 | Files changed 1 | +11 −6

heyabhinav commented 2 days ago — Member

No description provided.

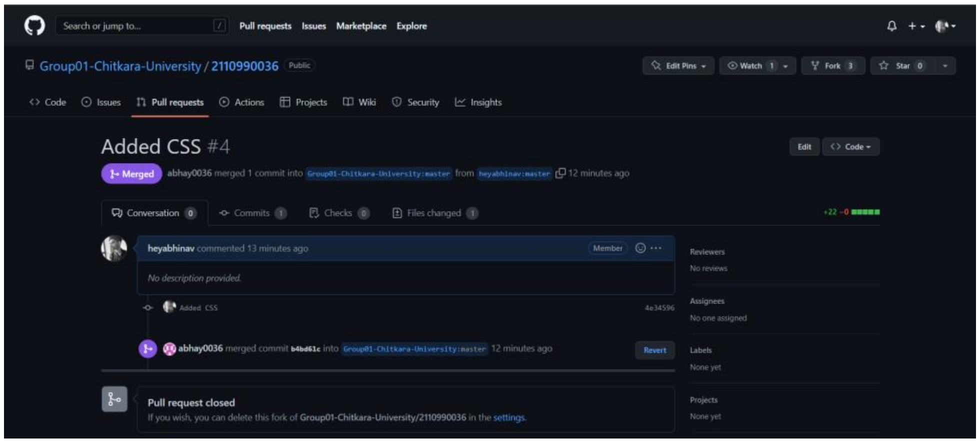
heyabhinav added 2 commits 9 days ago

```
added case 8 and 9, and also formatted the code little bit.    262e8b0
formatted code.                                                3bbcea3
```

Abhi1223Gupta merged commit `d620df5` into `Group01-Chitkara-University:main` 2 days ago — Revert

Reviewers: No reviews
Assignees: No one assigned
Labels: None yet
Projects: None yet

2110990036: Added CSS to their messmenu.html file.



Group01-Chitkara-University / 2110990036 Public

Added CSS #4

Merged abhay0036 merged 1 commit into `Group01-Chitkara-University:master` from `heyabhinav:master` 12 minutes ago

Conversation 0 | Commits 1 | Checks 0 | Files changed 1 | +22 −0

heyabhinav commented 13 minutes ago — Member

No description provided.

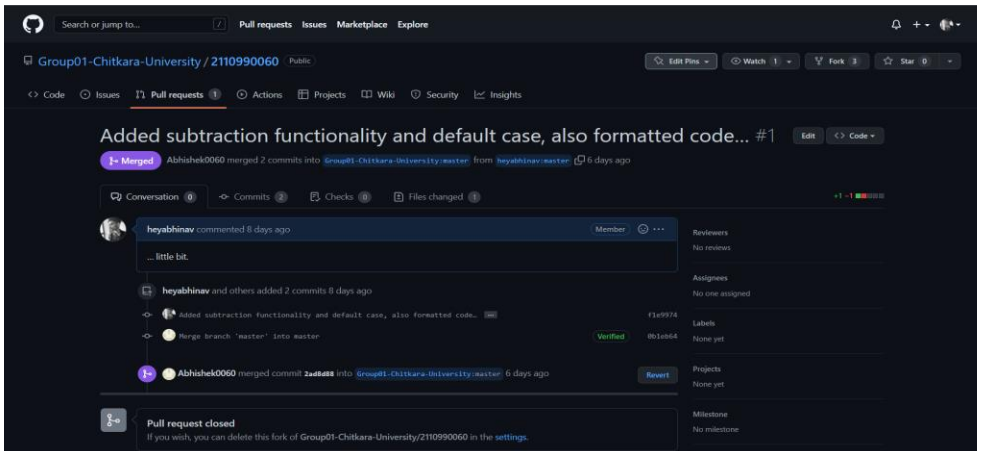
```
Added CSS    4e34596
```

abhay0036 merged commit `b4bd61c` into `Group01-Chitkara-University:master` 12 minutes ago — Revert

Pull request closed
If you wish, you can delete this fork of Group01-Chitkara-University/2110990036 in the settings.

Reviewers: No reviews
Assignees: No one assigned
Labels: None yet
Projects: None yet

2110990060: Added subtraction and default case to their calculator program.



Group01-Chitkara-University / 2110990060 Public

Added subtraction functionality and default case, also formatted code... #1

Merged Abhishek0060 merged 2 commits into `Group01-Chitkara-University:master` from `heyabhinav:master` 6 days ago

Conversation 0 | Commits 2 | Checks 0 | Files changed 1 | +1 −1

heyabhinav commented 8 days ago — Member

... little bit.

heyabhinav and others added 2 commits 8 days ago

```
Added subtraction functionality and default case, also formatted code… ...    f1e9974
Merge branch 'master' into master                           Verified          0b1eb64
```

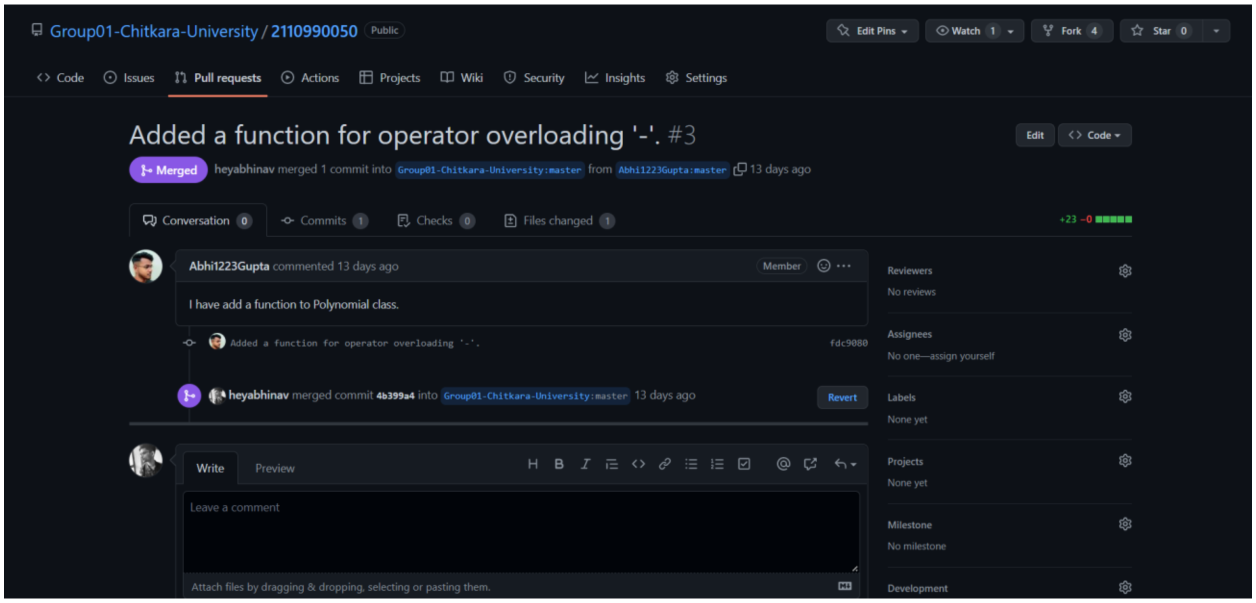
Abhishek0060 merged commit `2ad8d88` into `Group01-Chitkara-University:master` 6 days ago — Revert

Pull request closed
If you wish, you can delete this fork of Group01-Chitkara-University/2110990060 in the settings.

Reviewers: No reviews
Assignees: No one assigned
Labels: None yet
Projects: None yet
Milestone: No milestone

Closed pull requests from group members:

2110990066: Pull request from 211099066 added operator overloading function to my Polynomial class which help in performing subtraction operation between two polynomial object.



Group01-Chitkara-University / 2110990050 Public

Added a function for operator overloading '-'. #3

Merged heyabhinav merged 1 commit into `Group01-Chitkara-University:master` from `Abhi1223Gupta:master` 13 days ago

Conversation 0 · Commits 1 · Checks 0 · Files changed 1 · +23 −0

Abhi1223Gupta commented 13 days ago · Member

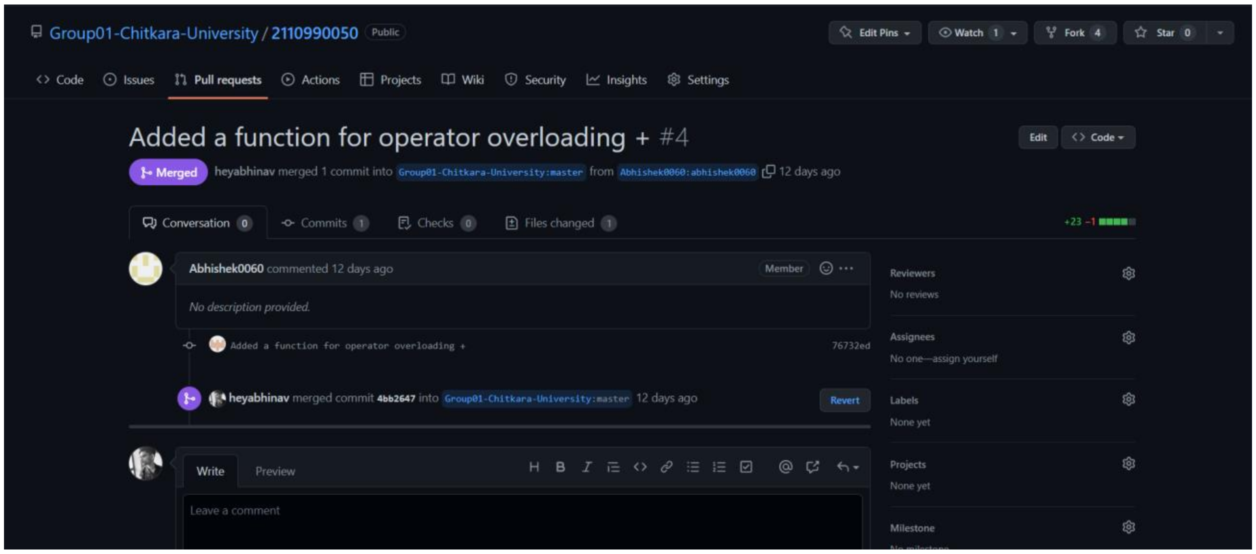
I have add a function to Polynomial class.

`Added a function for operator overloading '-'.` fdc9080

heyabhinav merged commit 4b399a4 into `Group01-Chitkara-University:master` 13 days ago

Reviewers: No reviews
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Development

2110990060: Pull request from 211099066 added operator overloading function to my Polynomial class which help in performing addition operation between two polynomial object.



Group01-Chitkara-University / 2110990050 Public

Added a function for operator overloading + #4

Merged heyabhinav merged 1 commit into `Group01-Chitkara-University:master` from `Abhishek0060:abhishek0060` 12 days ago

Conversation 0 · Commits 1 · Checks 0 · Files changed 1 · +23 −1

Abhishek0060 commented 12 days ago · Member

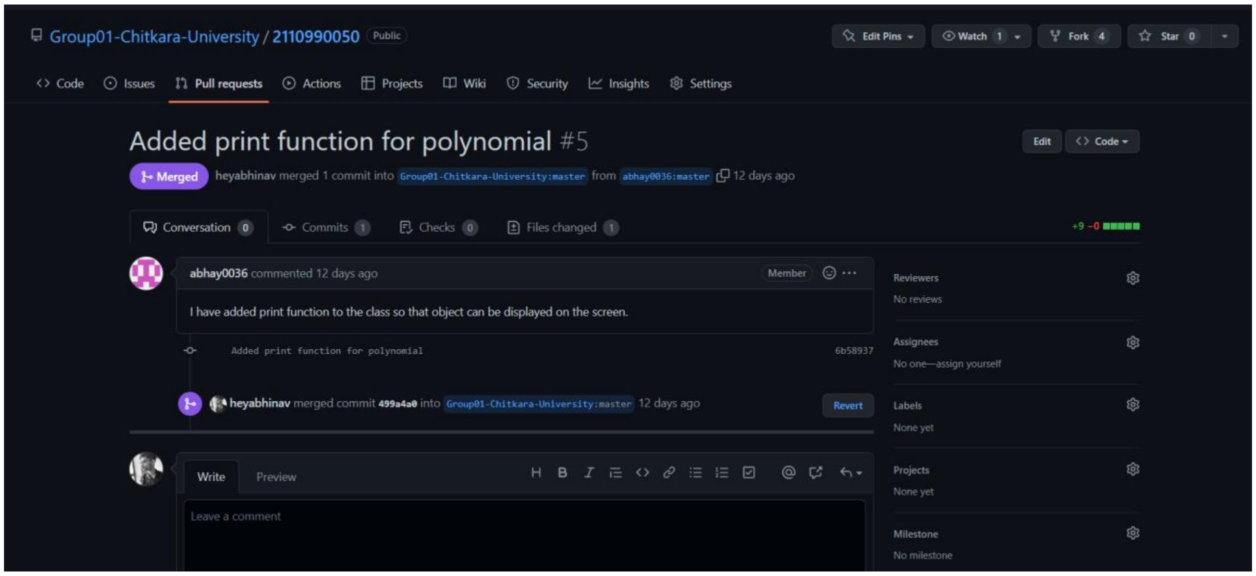
No description provided.

`Added a function for operator overloading +` 76732ed

heyabhinav merged commit 4bb2647 into `Group01-Chitkara-University:master` 12 days ago

Reviewers: No reviews
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone

<u>2110990036</u>: Pull request from 2110990036, added print function my polynomial class, which can be used to display polynomial in the output screen.



Added print function for polynomial #5

Merged heyabhinav merged 1 commit into Group01-Chitkara-University:master from abhay0036:master 12 days ago

abhay0036 commented 12 days ago

I have added print function to the class so that object can be displayed on the screen.

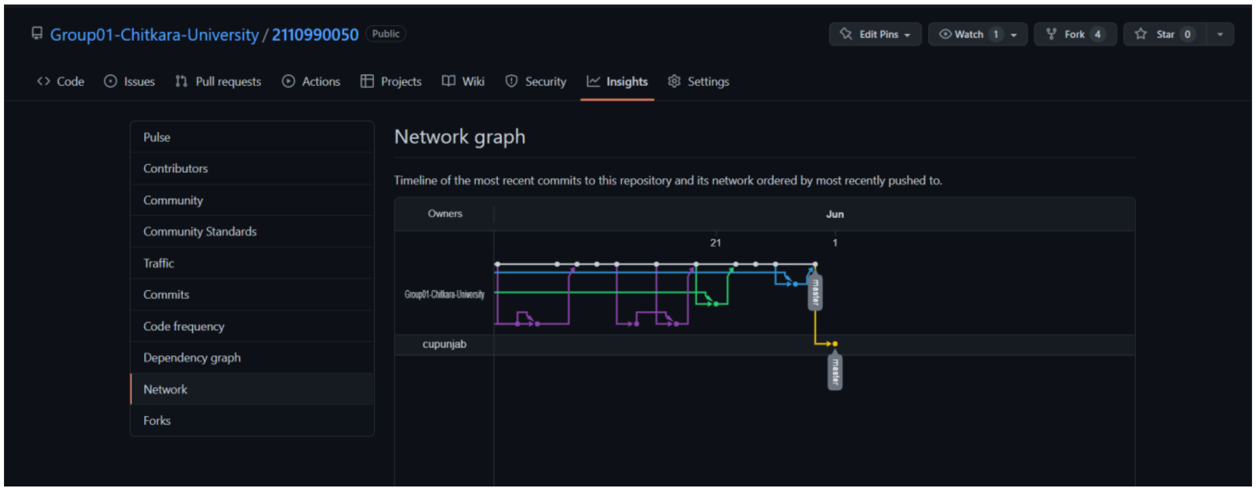
```
Added print function for polynomial                6b58937
```

heyabhinav merged commit 499a4a0 into Group01-Chitkara-University:master 12 days ago

Leave a comment

<u>Network graphs</u>

To get network graph of own or ones repo go to **Insight** section. Then click on **Network** tab.



Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.