Subject Name: **Source Code Management**

Subject Code: **CS181**

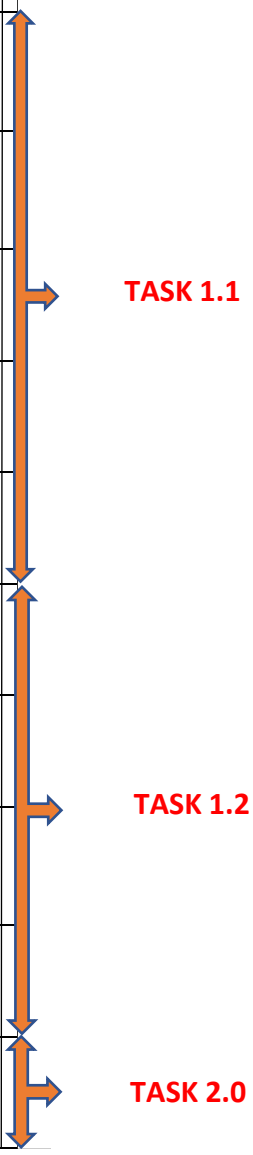
Cluster: **Beta**

Department: **DCSE**

Submitted By
Armaan Singh Bhasin
(2110990258)
G8-A

Submitted To:
Dr. Monit Kapoor

INDEX



INTRODUCTION

What is GIT and why is it used?

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.

What is the difference between GIT and GITHUB?

Git	GitHub
1. It is a software	1. It is a service
2. It is installed locally on the system	2. It is hosted on Web
3. It is a command line tool	3. It provides a graphical interface
4. It is a tool to manage different versions of edits, made to files in a git repository	4. It is a space to upload a copy of the Git repository
5. It provides functionalities like Version Control System Source Code Management	5. It provides functionalities of Git like VCS, Source Code Management as well as adding few of its own features

What is Repository?

A repository stores all of your project's files, as well as the revision history for each one. Within the repository, you may discuss and monitor your project's progress. The.git/ subdirectory within a project is a Git repository. This repository keeps track of any changes made to files in your project over time, creating a history. That is, if you delete the.git/ subdirectory, you are also deleting the history of your project.

What is Version Control System (VCS)?

Version Control Systems are the software tools for tracking/managing all the changes made to the source code during the project development. It keeps a record of every single change made to the code. It also allows us to turn back to the previous version of the code if any mistake is made in the current version. Without a VCS in place, it would not be possible to monitor the development of the project.

Types of VCS

- Local Version Control System
- Centralized Version Control System
- Distributed Version Control System

I. **Local Version Control System:** Local Version Control System is located in your local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost. If anything happens to a single version, all the versions made after that will be lost.

II. **Centralized Version Control System:** In the Centralized Version Control Systems, there will be a single central server that contains all the files related to the project, and many collaborators checkout files from this single server (you will only have a working copy). The problem with the Centralized Version Control Systems is if the central server crashes, almost everything related to the project will be lost.

III. **Distributed Version Control System:** In a distributed version control system, there will be one or more servers and many collaborators similar to the centralized system. But the

difference is, not only do they check out the latest version, but each collaborator will have an exact copy of the main repository on their local machines. Each user has their own repository and a working copy. This is very useful because even if the server crashes we would not lose everything as several copies are residing in several other computers.

Experiment No. 01

Aim: Setting up of Git Client

Theory:

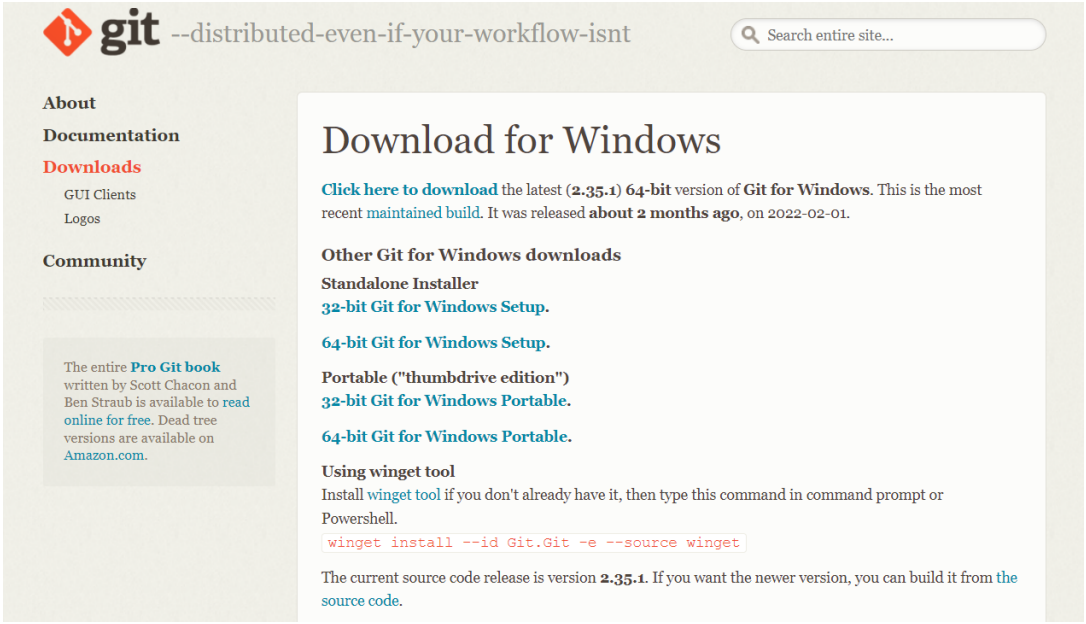
GIT –> It is basically used for pushing and pulling of code. We can use git and git-hub parallelly to work with multiple members or individually. We can make , edit , recreate ,copy or download any code on git hub using git.

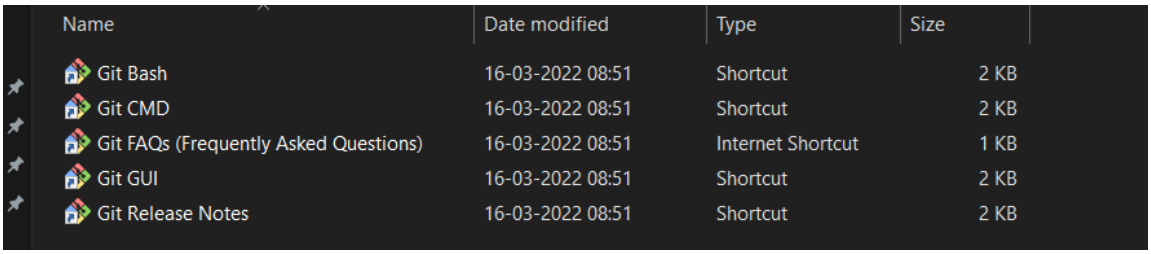
What is GIT ? –> It's a Version Control System(VCS) -> It is a software or we can say a server by which we are able to track all the previous changes in the code.

Advantages of GIT –>

Procedure: We can install Git on Windows, using the most official build which is available for download on the GIT's official website or by just typing (s c m git) on any search engine . We can go on https://git-scm.com/download/win and can select the platform and bit-version to download. And after clicking on your desired bit-version or ios it will start downloading automatically.

Snapshots of download:





Experiment No. 02

Aim: Setting up GitHub Account

Theory:

What is GitHub -> GitHub is a website and cloud-based service (client) that helps an individual or a developers to store and manage their code. We can also track as well as control changes to our or public code.

Advantages of GitHub -> GitHub's has a user-friendly interface and is easy to use .We can connect the git-hub and git but using some commands shown below in figure 001. Without GitHub we cannot use Git because it generally requires a host and if we are working for a project we need to share it will our team members, which can only be done by making a repository . Additionally , anyone can sign up and host a public code repository for free, which makes GitHub especially popular with open-source projects.

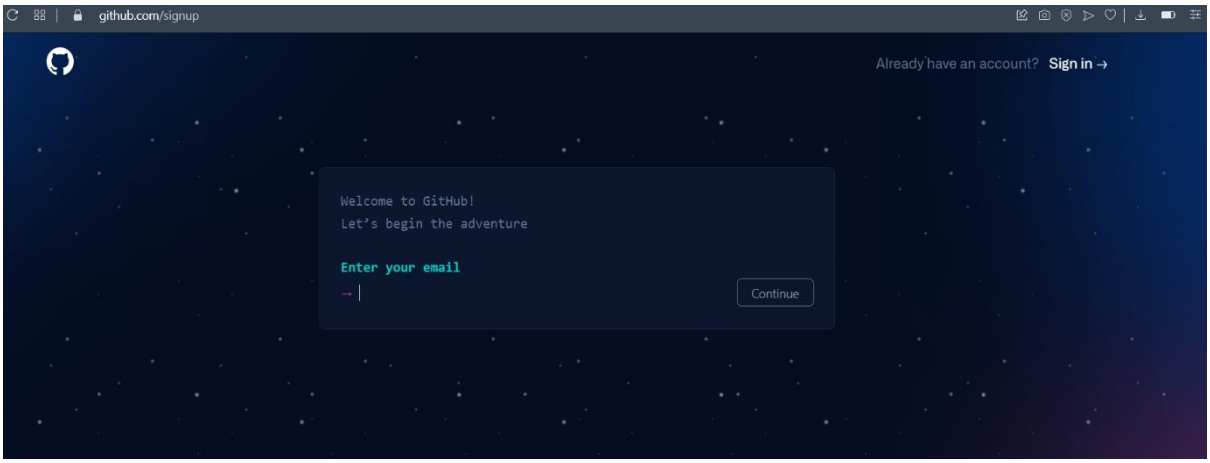
Procedure:-

Step1 :-

Google (any search engine)
Search for git-hub or (https://github.com/signup).

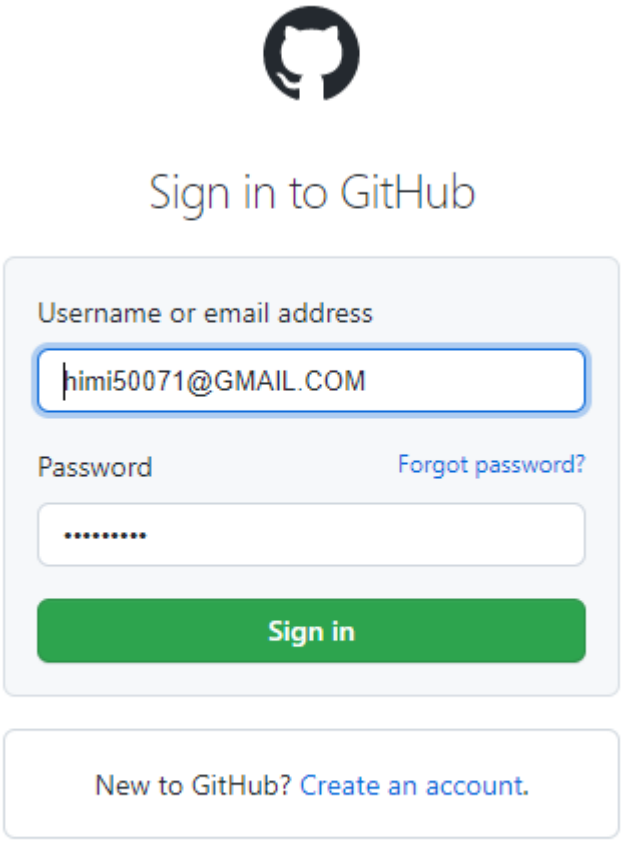
Step2 :-

Snapshots –

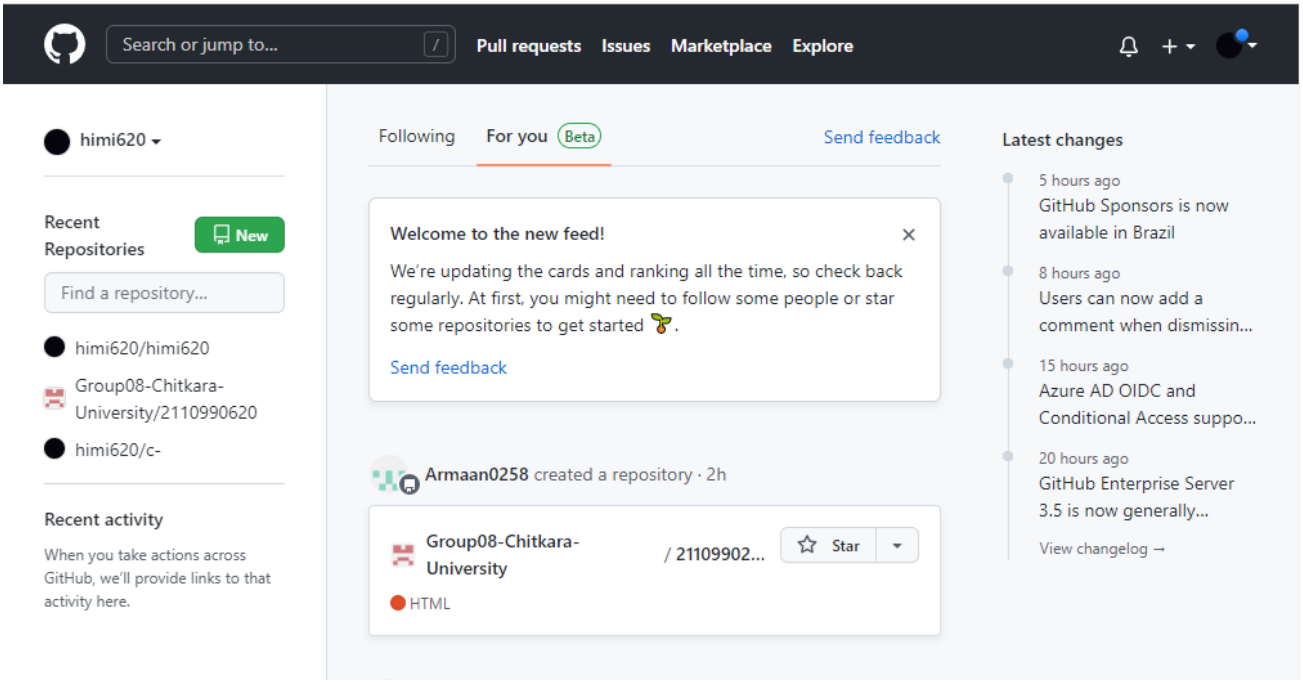


After visiting the link this type of interface will appear, if you already have account you can sign in and if not you can create.

Sign in into GIT-HUB :-



Interface of GitHub :-



To link GitHub account with Git bash –

For username:-

git config --global user.name “username in git-hub”

For user email:-

git config --global user.email “your email in git-hub”

To verify:-

git config user.name

git config user.email

Snapshot :-

```
DELL@DESKTOP-OLO2QO5 MINGW64 ~
$ git config --global user.email

DELL@DESKTOP-OLO2QO5 MINGW64 ~
$ git config --global user.email himi50071@gmail.com

DELL@DESKTOP-OLO2QO5 MINGW64 ~
$ git config --global user.name himi620
```

Experiment No. 03

Aim: Program to Generate log

Theory:-

Logs -> Logs are nothing but the history which we can see in git by using the code git log. It contains all the past commits, insertions and deletions in it which we can see any time.

✧ **Why logs ->** **Logs helps to check that what were the changes in the code or any other file and by whom. It also contains the number of insertions and deletions including at which time it was changed.**

First of all create a local repository using Git. For this, you have to make a folder in your device, right click and select "**Git Bash Here**". This opens the Git terminal. To create a new local repository, use the command "**git init**" and it creates a folder **.git**

✧ When we use GIT for the first time, we have to give the user name and email so that if I am going to change in project, it will be visible to all.

For this, we use command →

"git config --global user.name *Name*"

"git config --global user.email *email*"

For verifying the user's name and email, we use →

"git config --global user.name"

"git config --global user.email"

Some Important Commands:

- **ls** → It gives the file names in the folder.
- **ls -lart** → Gives the hidden files also.
- **git status** → Displays the state of the working directory and the staged snapshot.

- **touch filename →** This command creates a new file in the repository.
- **Clear →** It clears the terminal.
- **rm -rf .git →** It removes the repository.
- **git log →** displays all of the commits in a repository's history
- **git diff →** It compares my working tree to staging area.

✧ Now, we have to create some files in the repository. Suppose we created index.html Now type git status:

git log: *The git log command displays a record of the commits in a Git repository. By default, the git log command displays a commit hash, the commit message, and other commit metadata.*

Snapshots –

```
DELL@DESKTOP-OL02QO5 MINGW64 /c/himi620 (master)
$ git commit -a -m "some changes"
[master 1fb1501] some changes
 1 file changed, 2 insertions(+), 1 deletion(-)

DELL@DESKTOP-OL02QO5 MINGW64 /c/himi620 (master)
$ git status
On branch master
nothing to commit, working tree clean

DELL@DESKTOP-OL02QO5 MINGW64 /c/himi620 (master)
$ git log
commit 1fb15016716a1b369f8dd6d74d3c963e08837216 (HEAD -> master
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:54:34 2022 +0530

    some changes

commit 1ed483450dde05f901c637662d7113f04679ded6
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:49:12 2022 +0530

    newfile

commit 20a41ac2a79773df15aad78d0af15e4b23a8f343
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:32:53 2022 +0530

     modified this file

commit cdfbf538f52f22247247919201da97e454436422
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:23:18 2022 +0530

     commit this file

DELL@DESKTOP-OL02QO5 MINGW64 /c/himi620 (master)
$
```

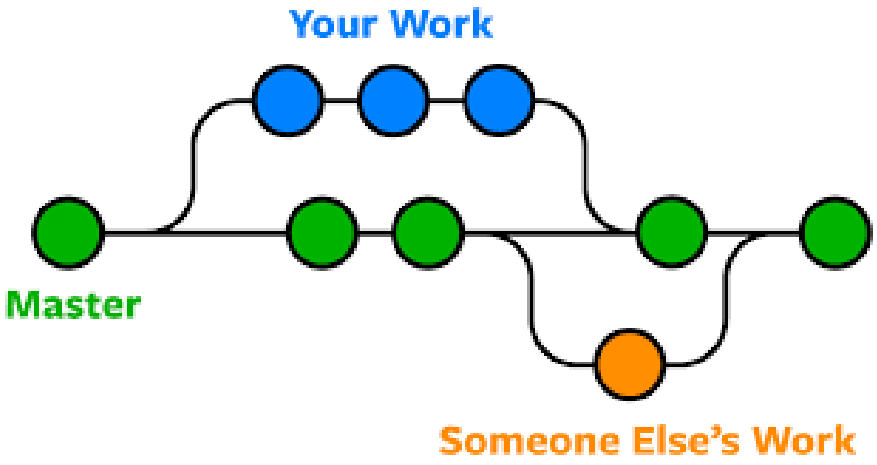
Experiment No. 04

Aim: Create and visualize branches

Create branches :-

✧ The main branch in git is called as master branch. But we can make branches out of this main master branch. All the files present in master can be shown in branch but the file which are created in branch are not shown in master branch. We can also merge both the parent (master) and child (other branches).

Branching: 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.



Let us see the command of it:

Firstly, add a new branch, let us suppose the branch name is activity1.
For this use command →

- **git branch name [**adding new branch**]**
- **git branch [**use to see the branch's names**]**
- **git checkout *branch name* [**use to switch to the given branch**]**
- In this you can see that firstly 'git branch' shows only one branch in green colour but when we add a new branch using 'git branch act1', it shows 2 branches but the green colour and star is on master. So, we have to switch to act1 by using 'git checkout act1'. If we use 'git branch', now you can see that the green colour and star is on act1. It means you are in activity1 branch and all the data of master branch is also on act1 branch. Use "ls" to see the files.
-
- Now add a new file in activity1 branch, do some changes in file and commit the file.

Syntax:-

1. For creating a new branch.
   git branch name of branch

Snapshots –

2. We can also check how many branches we have.
   git branch

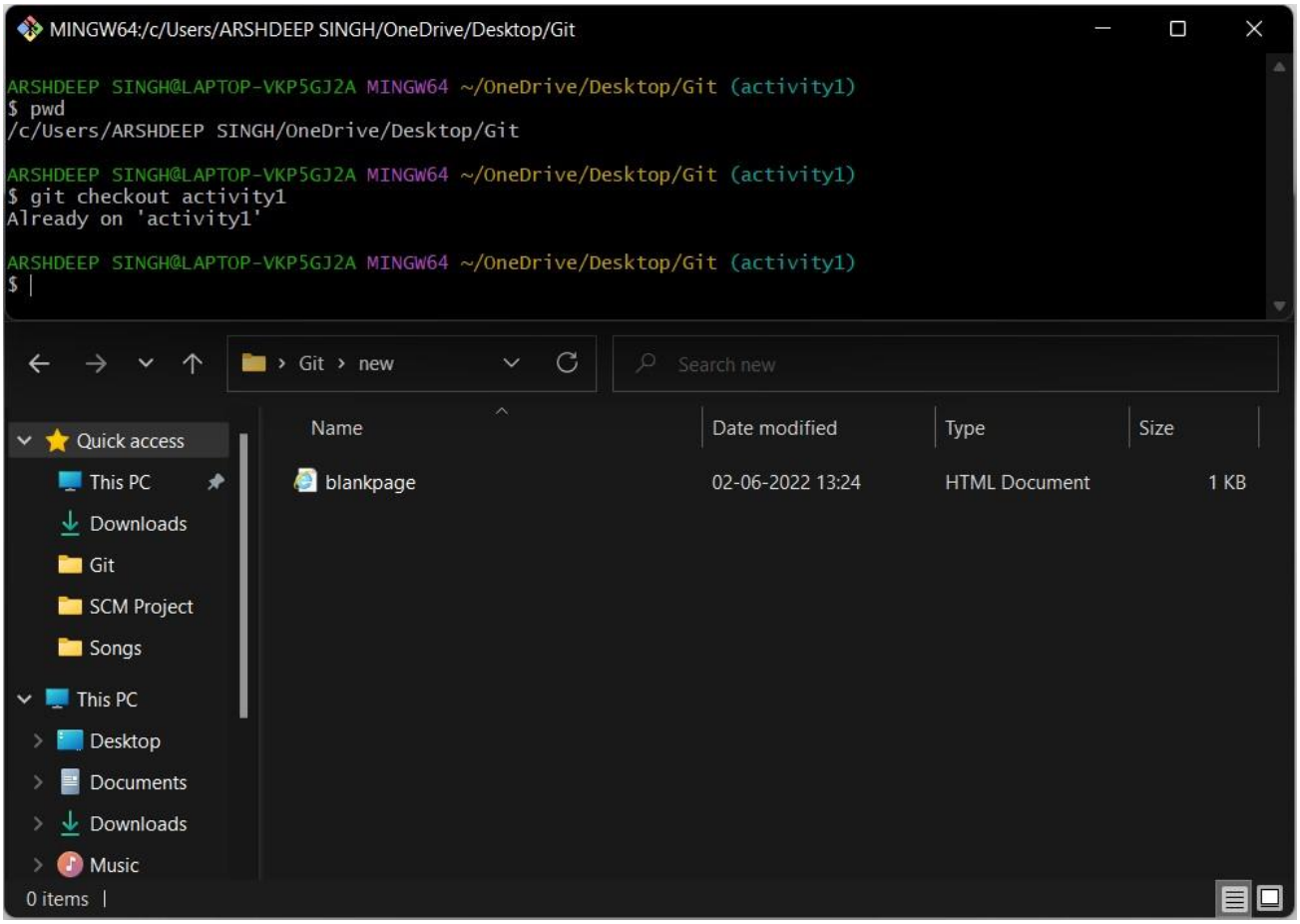
Snapshots :-

3. To change the present working branch.
git checkout name of branch.

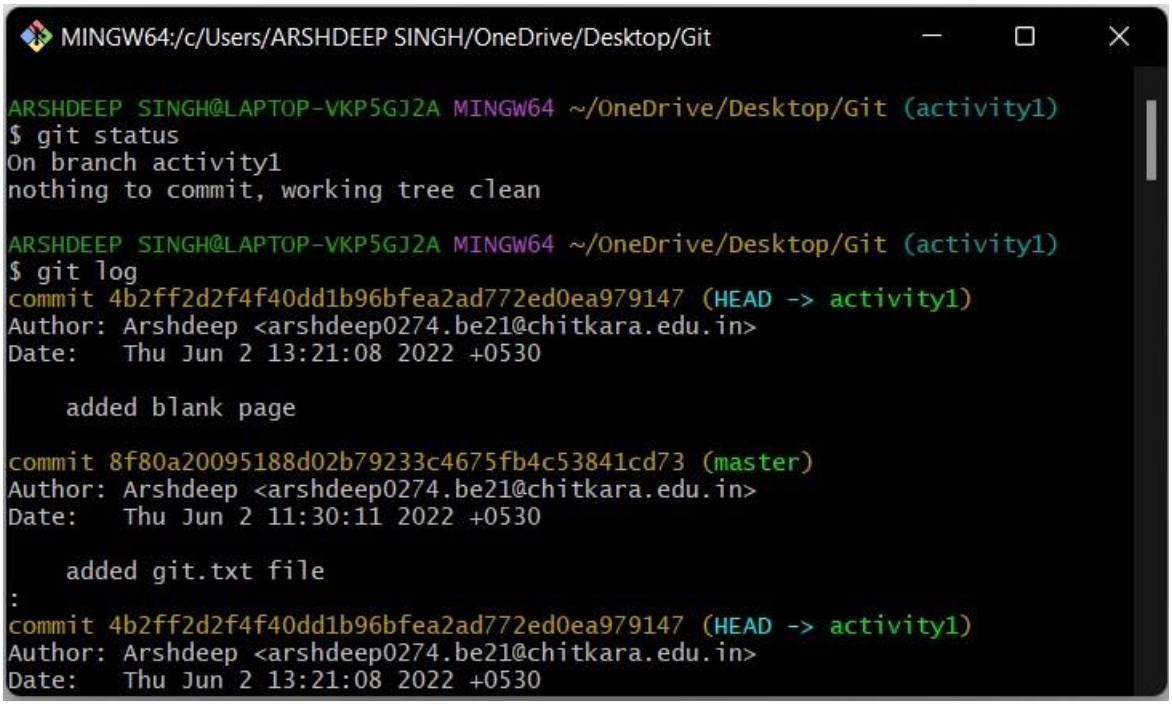
Snapshots –

Visualizing branches :-

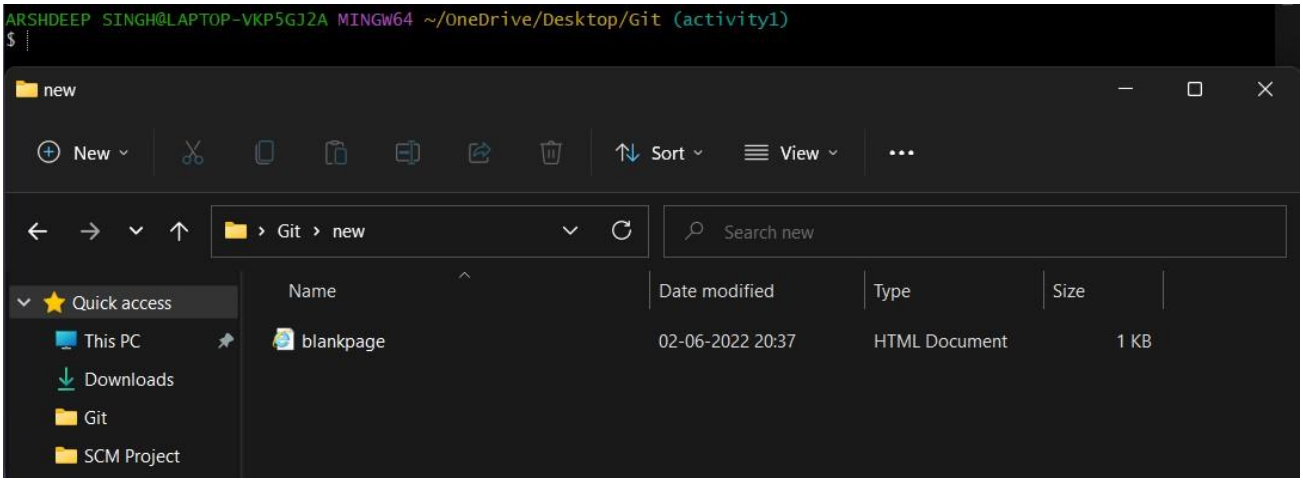
To visualize I have created a new file in a new branch activity 1 instead of master branch.



After this I have done the 3 step architecture which is tracking the file , send it to stagging area and finally we can role back to any previously saved version of this file.



After this we will change the branch from activity1 to master, but when I will switch to the master branch there will not be the same file in the master , it will not show the new file in the master branch.



In this way we can create and change different branches . We can also merge the branches

by using git merge command.

Experiment No. 05

Aim: Git lifecycle description

Theory:

Stages in GIT Life Cycle -> Files in a Git project have various stages like Creation, Modification, Refactoring, and Deletion and so on. Irrespective of whether this project is tracked by Git or not, these phases are still prevalent. However, when a project is under Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states:

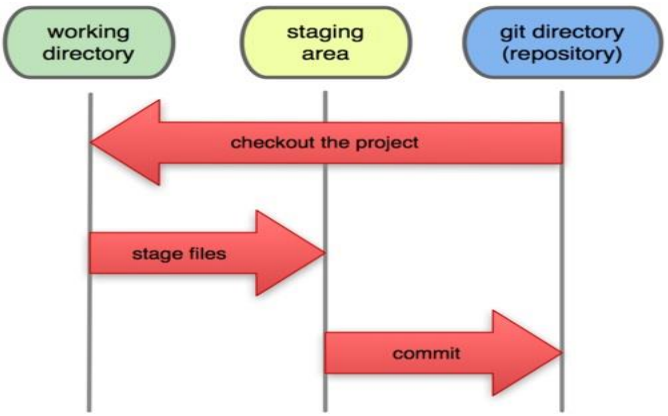
- Working directory
- Staging area
- Git directory

Working Directory ->
Consider a project residing in your local system. This project may or may not be tracked by Git. In either case, this project directory is called your Working directory.

Staging Area ->
Staging area is the playground where you group, add and organize the files to be committed to Git for tracking their versions.

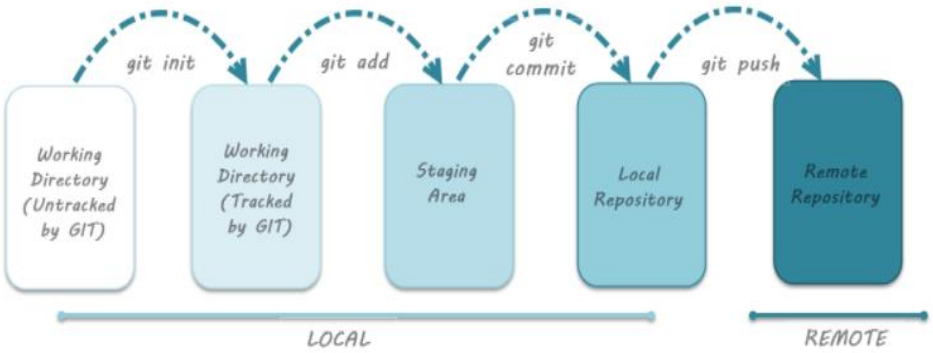
Git Directory ->
Now that the files to be committed are grouped and ready in the staging area, we can commit these files. So, we commit this group of files along with a commit message explaining what is the commit about. Apart from commit message, this step also records the author and time of the commit. Now, a snapshot of the files in the commit is recorded by Git. The information related to this commit is stored in the Git directory.

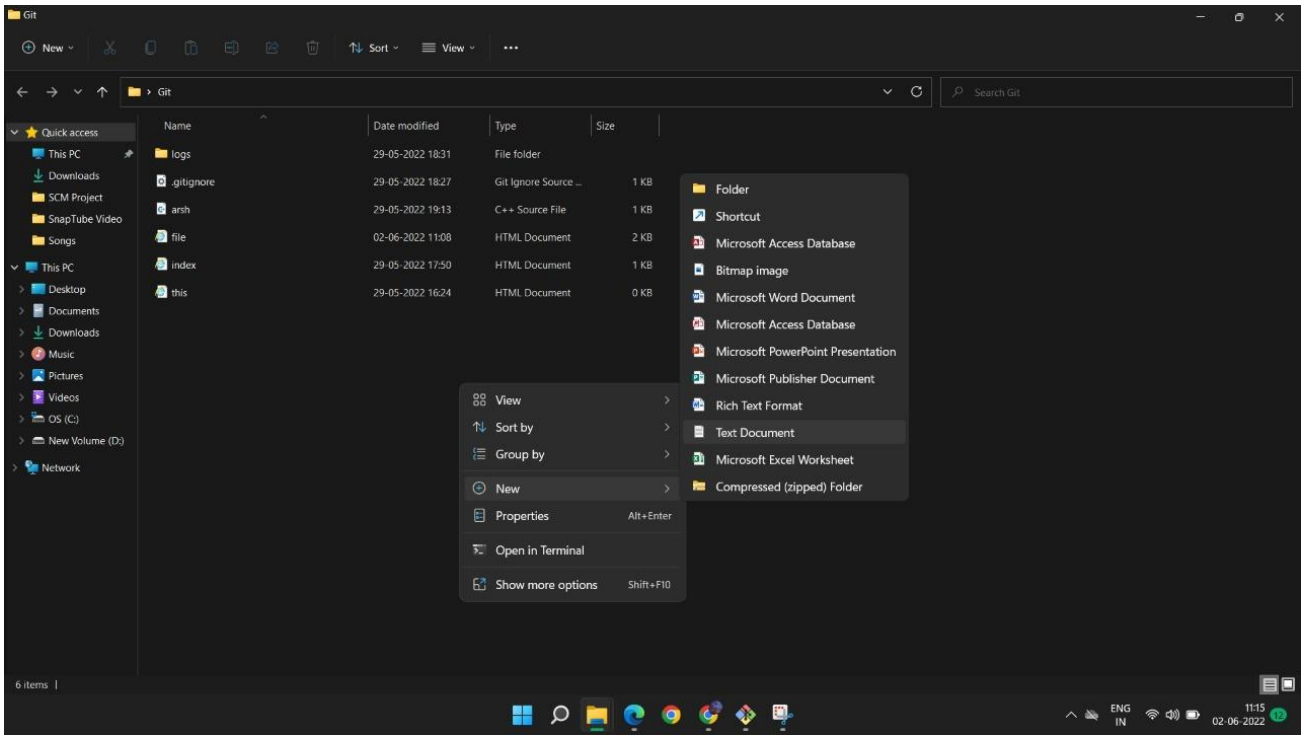
Remote Repository-> means mirror or clone of the local Git repository in GitHub. And pushing means uploading the commits from local Git repository to remote repository hosted in GitHub.

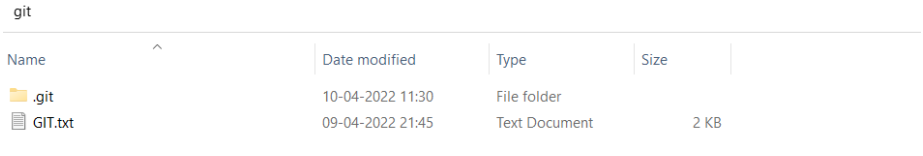

working directory
staging area
git directory (repository)
checkout the project
stage files
commit

Snapshots –

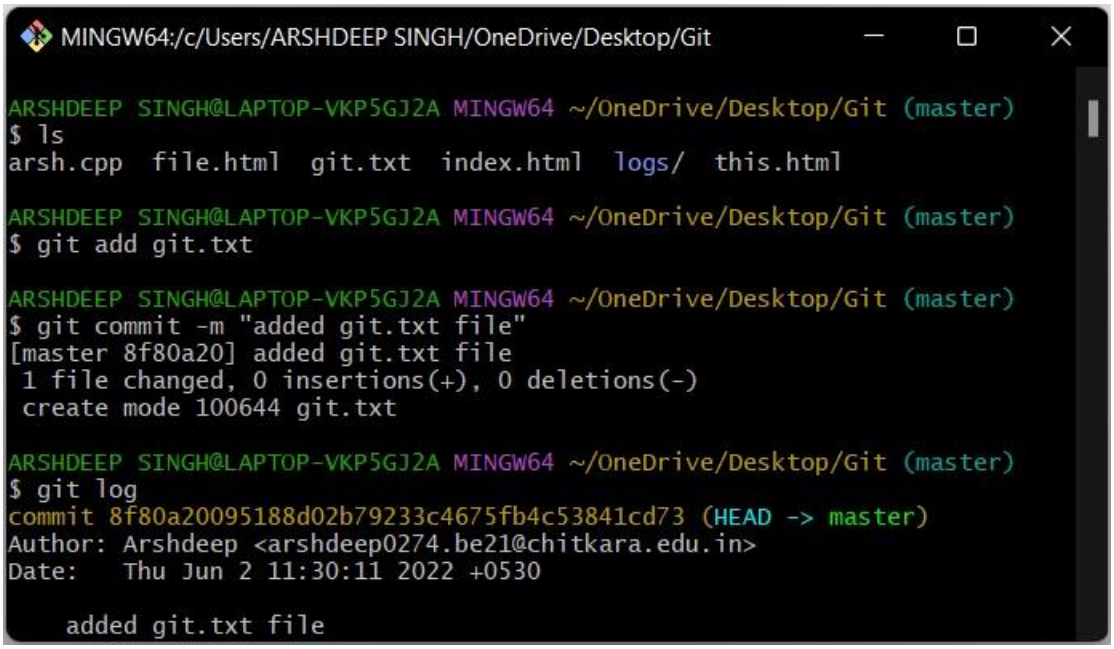

MINGW64:/c/Users/ARSHDEEP SINGH/OneDrive/Desktop/Git
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.
nothing to commit, working tree clean
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$


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


Git
New
Sort
View
Search Git
Quick access
This PC
Downloads
SCM Project
SnapTube Video
Songs
Desktop
Documents
Music
Pictures
Videos
OS (C:)
New Volume (D:)
Network
Name
Date modified
Type
Size
logs
29-05-2022 18:31
File folder
.gitignore
29-05-2022 18:27
Git Ignore Source ...
1 KB
arsh
29-05-2022 19:13
C++ Source File
1 KB
file
02-06-2022 11:08
HTML Document
2 KB
index
29-05-2022 17:50
HTML Document
1 KB
this
29-05-2022 16:24
HTML Document
0 KB
View
Sort by
Group by
New
Properties
Alt+Enter
Open in Terminal
Show more options
Shift+F10
Folder
Shortcut
Microsoft Access Database
Bitmap image
Microsoft Word Document
Microsoft Access Database
Microsoft PowerPoint Presentation
Microsoft Publisher Document
Rich Text Format
Text Document
Microsoft Excel Worksheet
Compressed (zipped) Folder
6 items
ENG
IN
11:15
02-06-2022


git
Name
Date modified
Type
Size
.git
10-04-2022 11:30
File folder
GIT.txt
09-04-2022 21:45
Text Document
2 KB


MINGW64:/c/Users/ARSHDEEP SINGH/OneDrive/Desktop/Git
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$ git init
Reinitialized existing Git repository in C:/Users/ARSHDEEP SINGH/OneDrive/Desktop/Git/.git/
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$


MINGW64:/c/Users/ARSHDEEP SINGH/OneDrive/Desktop/Git
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$ ls
arsh.cpp file.html git.txt index.html logs/ this.html
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$ git add git.txt
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$ git commit -m "added git.txt file"
[master 8f80a20] added git.txt file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 git.txt
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$ git log
commit 8f80a20095188d02b79233c4675fb4c53841cd73 (HEAD -> master)
Author: Arshdeep <arshdeep0274.be21@chitkara.edu.in>
Date: Thu Jun 2 11:30:11 2022 +0530
added git.txt file


MINGW64:/c/Users/ARSHDEEP SINGH/OneDrive/Desktop/Git
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)
nothing to commit, working tree clean
ARSHDEEP SINGH@LAPTOP-VKP5GJ2A MINGW64 ~/OneDrive/Desktop/Git (master)
$

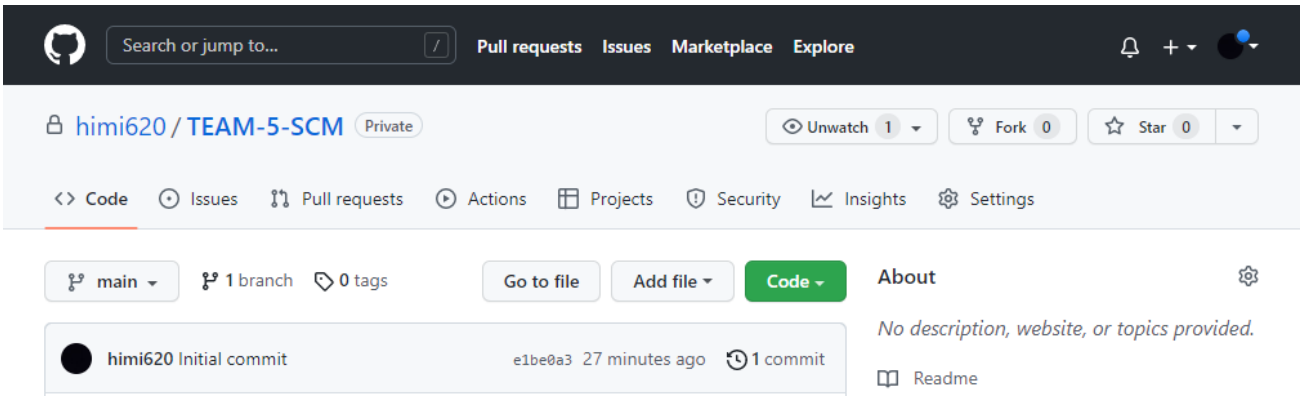
Experiment No. 06

Aim: Add collaborators on Github Repositoy

1. Ask for the username of the person you're inviting as a collaborator. If theydon't have a username yet, they can sign up for GitHub For more information

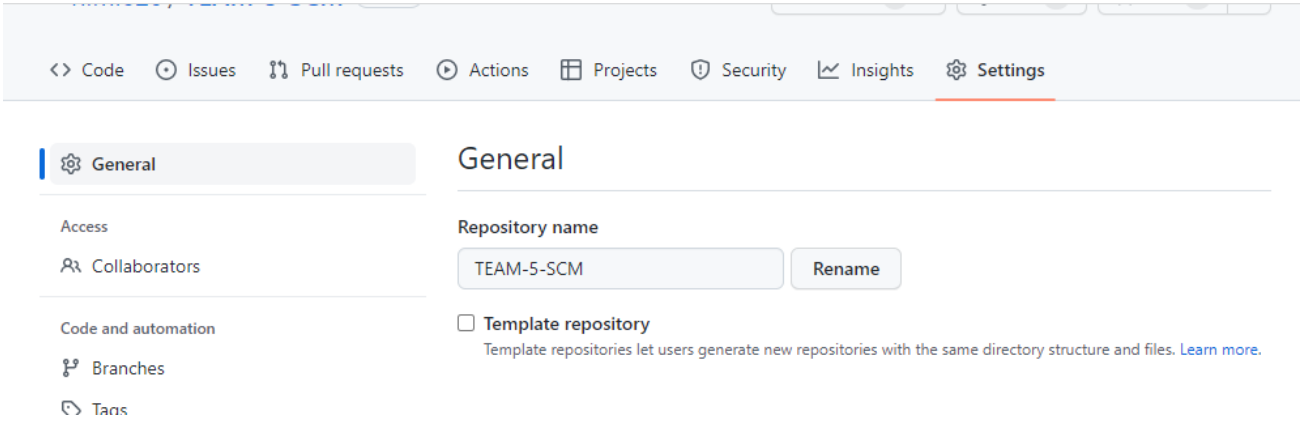
2. On GitHub.com, navigate to the main page of the repository

3. Under your repository name, click **Settings**

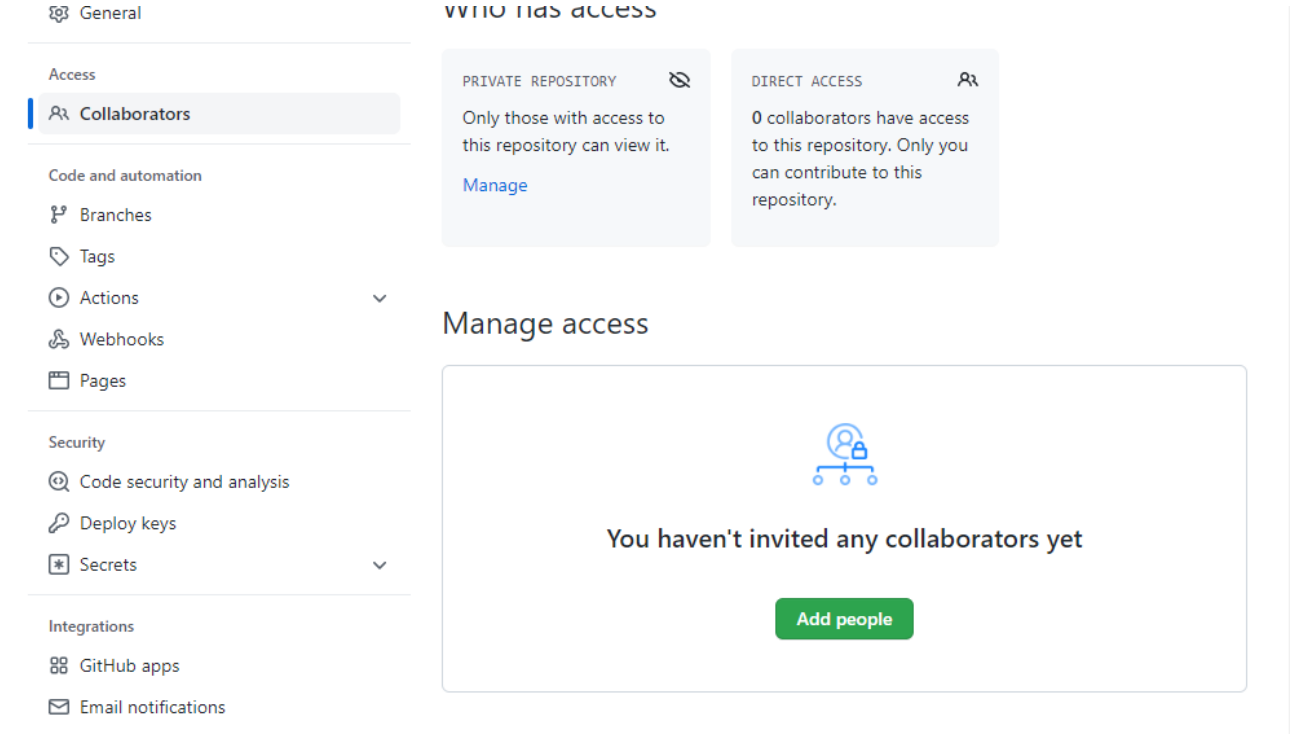


4. In the "Access" section of the sidebar, click **Collaborators & teams**.

5. Click **Invite a collaborator**..



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



Experiment No. 07

Aim: Fork and Commit

About forks

Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea. You can fork a repository to create a copy of the repository and make changes without affecting the upstream repository. For more information, see "Working with forks."

Propose changes to someone else's project

For example, you can use forks to propose changes related to fixing a bug. Rather than logging an issue for a bug you've found, you can:

- Fork the repository.
- Make the fix.
- Submit a pull request to the project owner.

Use someone else's project as a starting point for your own idea

Open source software is based on the idea that by sharing code, we can make better, more reliable software. For more information, see the "About the Open Source Initiative" on the Open Source Initiative. For more information about applying open source principles to your organization's development work on GitHub.com, see GitHub's white paper "An introduction to inner source."

When creating your public repository from a fork of someone's project, make sure to include a license file that determines how you want your project to be shared with others. For more information, see "Choose an open source license" at choosealicense.com.

For more information on open source, specifically how to create and grow an open source project, we've created Open Source Guides that will help you foster a healthy open source community by recommending best practices for creating and maintaining repositories for your open source project. You can also take a free GitHub Learning Lab course on maintaining open source communities.

Prerequisites

If you haven't yet, you should first set up Git. Don't forget to set up authentication to GitHub.com from Git as well.

Forking a repository

You might fork a project to propose changes to the upstream, or original, repository. In this case, it's good practice to regularly sync your fork with the upstream repository. To do this, you'll need to use Git on the command line. You can practice setting the upstream repository using the same octant/Spoon-Knife repository you just forked.

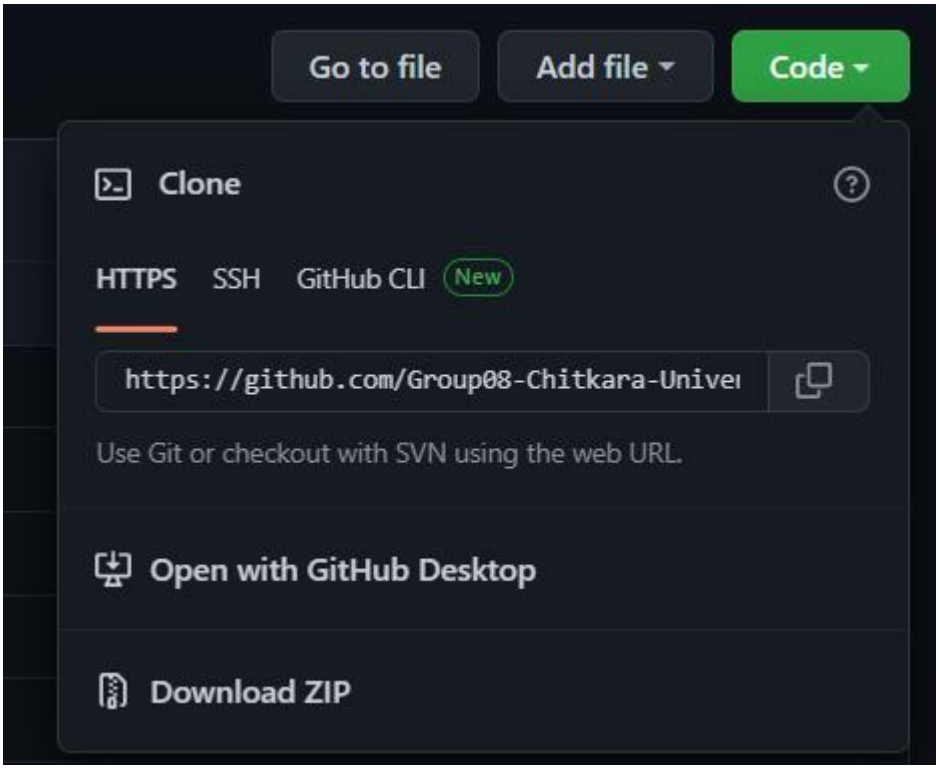
- On GitHub.com, navigate to the octant/Spoon-Knife repository.
- In the top-right corner of the page, click Fork.

Cloning your forked repository

Right now, you have a fork of the Spoon-Knife repository, but you don't have the files in that repository locally on your computer.

- On GitHub.com, navigate to your fork of the Spoon-Knife repository.
- Above the list of files, click Code

- To clone the repository using HTTPS, under "Clone with HTTPS", click. To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **Use SSH**, then click. To clone a repository using GitHub CLI, click **Use GitHub CLI**, then click .

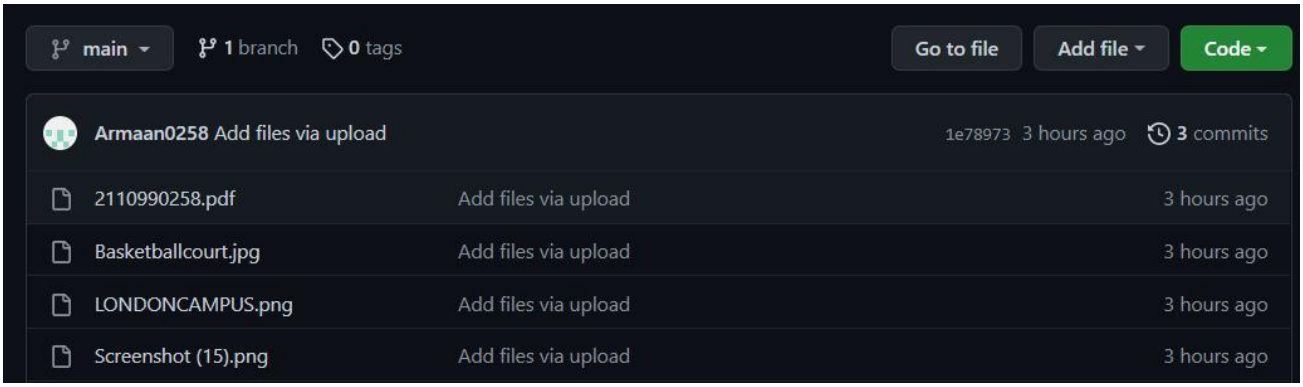


- Open Git Bash.
- Change the current working directory to the location where you want the cloned directory.
- Type git clone, and then paste the URL you copied earlier. It will look like this, with your GitHub username instead of YOUR-USERNAME:
- $ git clone https://github.com/*YOUR-USERNAME*/Spoon-Knife
- Press Enter. Your local clone will be created.
- $ git clone https://github.com/*YOUR-USERNAME*/Spoon-Knife
- > Cloning into `Spoon-Knife`...
- > remote: Counting objects: 10, done.
- > remote: Compressing objects: 100% (8/8), done.
- > remove: Total 10 (delta 1), reused 10 (delta 1)
- `> Unpacking objects: 100% (10/10), done.`

Configuring Git to sync your fork with the originalrepository

When you fork a project in order to propose changes to the original repository, you canconfigure Git to pull changes from the original, or upstream, repository into the local clone of your fork.

1. On GitHub.com, navigate to the octocat/Spoon-Knife repository.
2. Above the list of files, click Code



3. To clone the repository using HTTPS, under "Clone with HTTPS", click . To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **Use SSH**, then click . To clone a repository using GitHub CLI, click **Use GitHub CLI**, then click on

4. Open Git Bash.
5. Change directories to the location of the fork you cloned.
    - To go to your home directory, type just cdwith no other text.
    - To list the files and folders in your current directory, type ls.
    - To go into one of your listed directories, type cd your_listed_directory.
    - To go up one directory, type cd ...
6. Type git remote -vand press Enter. You'll see the current configured remoterepository for your fork.
7. $ git remote -v
8. > origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
   > origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
9. Type git remote add upstream, and then paste the URL you copied in Step 2 andpress Enter. It will look like this:
   $ git remote add upstream https://github.com/octocat/Spoon-Knife.git
10. To verify the new upstream repository you've specified for your fork, type git remote -vagain. You should see the URL for your fork as origin, and the URL for the original repository as upstream.

11. $ git remote -v
12. > origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
13. > origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
14. > upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (fetch)
15. > upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (push)

Now, you can keep your fork synced with the upstream repository with a few Gitcommands. For more information, see "Syncing a fork."

Next steps

You can make any changes to a fork, including:

- Creating branches: *Branches* allow you to build new features or test out ideas without putting your main project at risk.
- Opening pull requests: If you are hoping to contribute back to the original repository, you can send a request to the original author to pull your fork intotheir repository by submitting a pull request.

Find another repository to fork

Fork a repository to start contributing to a project. You can fork a repository to your user account or any organization where you have repository creation permissions. Formore information, see "Roles in an organization." If you have access to a private repository and the owner permits forking, you can fork the repository to your user account or any organization on GitHub Team where you have repository creation permissions. You cannot fork a private repository to an organization using GitHub Free. For more information, see "GitHub's products."

You can browse Explore to find projects and start contributing to open source repositories. For more information, see "Finding ways to contribute to open source onGitHub."

Celebrate

You have now forked a repository, practiced cloning your fork, and configured an upstream repository. For more information about cloning the fork and syncing the changes in a forked repository from your computer see "Set up Git." You can also create a new repository where you can put all your projects and share thecode on GitHub. For more information see, "Create a repository."

Each repository in GitHub is owned by a person or an organization. You can interact with the people, repositories, and organizations by connecting and following them onGitHub. For more information see "Be social." GitHub has a great support community where you can ask for help and talk to peoplefrom around the world. Join the conversation on Github Support Community.

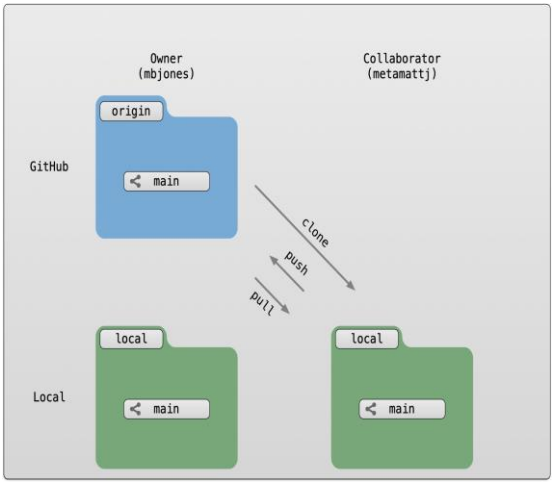
Experiment No. 08

Aim: Merge and Resolve conflicts created due to own activity and collaborators activity

Git is a great tool for working on your own, but even better for working with friends and colleagues. Git allows you to work with confidence on your own local copy of files with the confidence that you will be able to successfully synchronize your changes with the changes made by others.

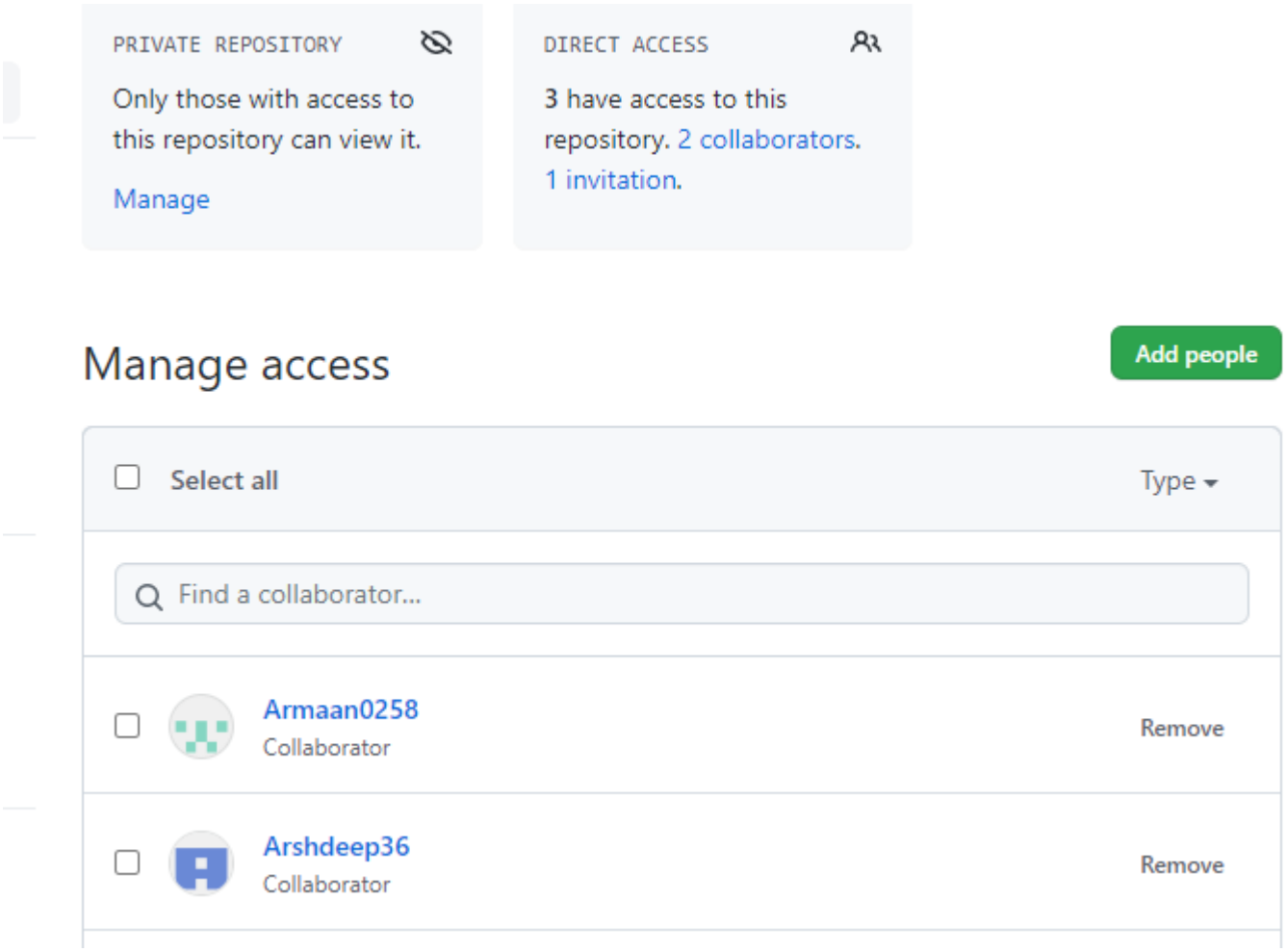
The simplest way to collaborate with Git is to use a shared repository on a hosting service such as GitHub, and use this shared repository as the mechanism to move changes from one collaborator to another. While there are other more advanced ways to sync git repositories, this "hub and spoke" model works really well due to its simplicity.

In this model, the collaborator will clone a copy of the owner's repository from GitHub, and the owner will grant them collaborator status, enabling the collaborator to directly pull and push from the owner's GitHub repository.



Collaborating with a trusted colleague without conflicts

We start by enabling collaboration with a trusted colleague. We will designate the Owner as the person who owns the shared repository, and the Collaborator as the person that they wish to grant the ability to make changes to their reposity. We start by giving that person access to our GitHub repository.



We will start by having the collaborator make some changes and share those with the Owner without generating any conflicts, In an ideal world, this would be the normal workflow. Here are the typical steps.

Step 1: Collaborator clone

To be able to contribute to a repository, the collaborator must clone the repository from the Owner's github account. To do this, the Collaborator should visit the github page for the Owner's repository, and then copy the clone URL. In R Studio, the Collaborator will create a new project from version control by pasting this clone URL into the appropriate dialog (see the earlier chapter introducing GitHub).

Step 2: Collaborator Edits

With a clone copied locally, the Collaborator can now make changes to the `index.Rmd` file in the repository, adding a line or statement somewhere noticeable near the top. Save your changes.

Step 3: Collaborator commit and push

To sync changes, the collaborator will need to add, commit, and push their changes to the Owner's repository. But before doing so, its good practice to pull immediately before committing to ensure you have the most recent changes from the owner. So, in R Studio's Git tab, first click the "Diff" button to open the git window, and then press the green "Pull" down arrow button. This will fetch any recent changes from the origin repository and merge them. Next, add the changed index.Rmd file to be committed by clicking the checkbox next to it, type in a commit message, and click 'Commit.' Once that finishes, then the collaborator can immediately click 'Push' to send the commits to the Owner's GitHub repository.

Step 4: Owner pull

Now, the owner can open their local working copy of the code in RStudio, and pull those changes down to their local copy. Congrats, the owner now has your changes!

Step 5: Owner edits, commit, and push

Next, the owner should do the same. Make changes to a file in the repository, save it, pull to make sure no new changes have been made while editing, and then add, commit, and push the Owner changes to GitHub.

Step 6: Collaborator pull

The collaborator can now pull down those owner changes, and all copies are once again fully synced. And you're off to collaborating.

Challenge

Now that the instructors have demonstrated this conflict-free process, break into pairs and try the same with your partner. Start by designating one person as the Owner and one as the Collaborator, and then repeat the steps described above:

- Step 0: Setup permissions for your collaborator
- Step 1: Collaborator clones the Owner repository
- Step 2: Collaborator Edits the README file
- Step 3: Collaborator commits and pushes the file to GitHub
- Step 4: Owner pulls the changes that the Collaborator made
- Step 5: Owner edits, commits, and pushes some new changes
- Step 6: Collaborator pulls the owners changes from GitHub

Merge conflicts

So things can go wrong, which usually starts with a merge conflict, due to both collaborators making incompatible changes to a file. While the error messages from merge conflicts can be daunting, getting things back to a normal state can be straightforward once you've got an idea where the problem lies.

A merge conflict occurs when both the owner and collaborator change the same lines in the same file without first pulling the changes that the other has made. This is most easily avoided by good communication about who is working on various sections of each file, and trying to avoid overlaps. But sometimes it happens, and git is there to warn you about potential problems. And git will not allow you to overwrite one person's changes to a file with another's changes to the same file if they were based on the same version.

The main problem with merge conflicts is that, when the Owner and Collaborator both make changes to the same line of a file, git doesn't know whose changes take precedence. You have to tell git whose changes to use for that line.

How to resolve a conflict

Abort, abort, abort…

Sometimes you just made a mistake. When you get a merge conflict, the repository is placed in a 'Merging' state until you resolve it. There's a commandline command to abort doing the merge altogether:

```
git merge --abort
```

Of course, after doing that you stull haven't synced with your collaborator's changes, so things are still unresolved. But at least your repository is now usable on your local machine.

Checkout

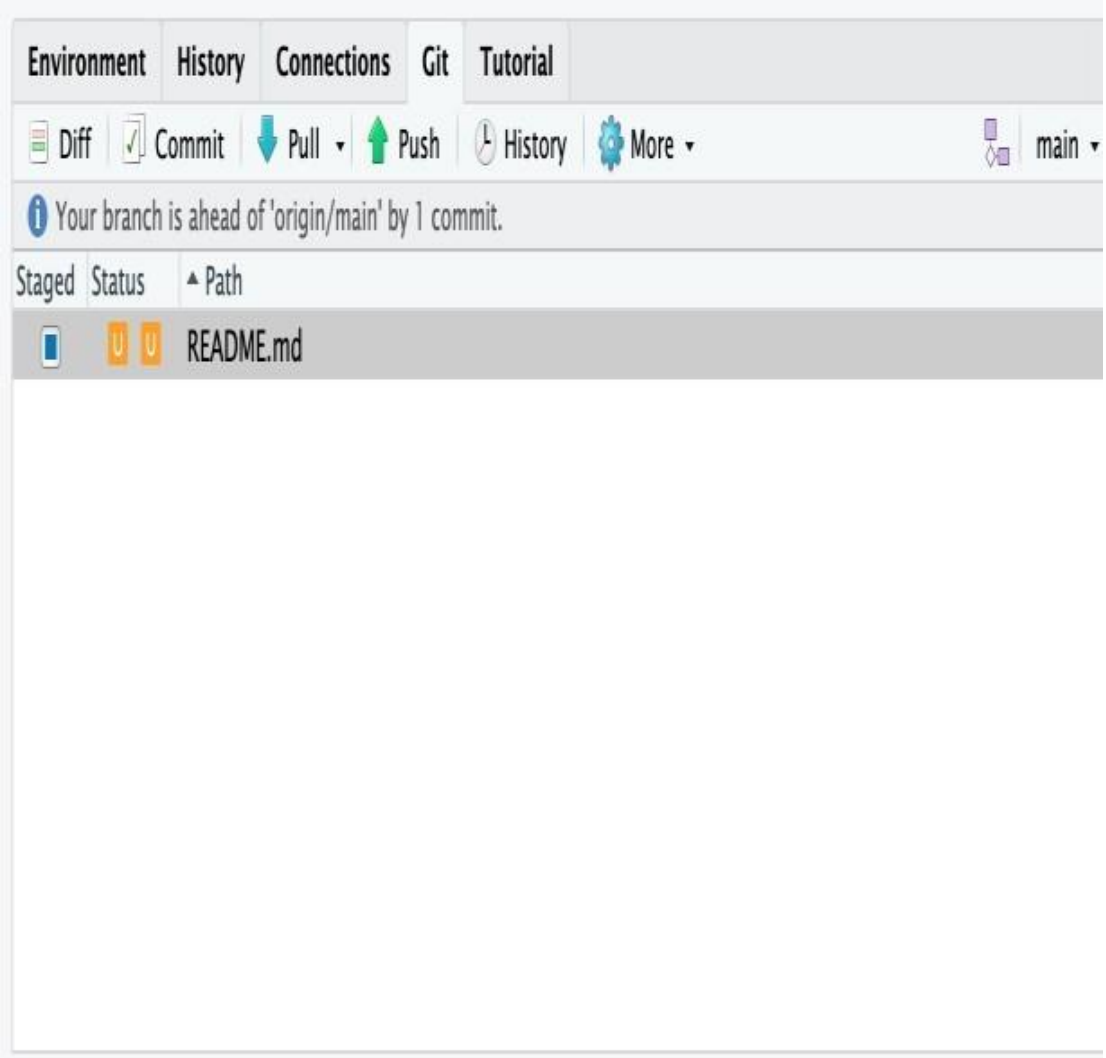
The simplest way to resolve a conflict, given that you know whose version of the file you want to keep, is to use the commandline **git** program to tell git to use either your changes (the person doing the merge), or their changes (the other collaborator).

- keep your collaborators file: `git checkout --theirs conflicted_file.Rmd`
- keep your own file: `git checkout --ours conflicted_file.Rmd`

Once you have run that command, then run `add`, `commit`, and `push` the changes as normal.

Pull and edit the file

But that requires the commandline. If you want to resolve from RStudio, or if you want to pick and choose some of your changes and some of your collaborator's, then instead you can manually edit and fix the file. When you pulled the file with a conflict, git notices that there is a conflict and modifies the file to show both your own changes and your collaborator's changes in the file. It also shows the file in the Git tab with an orange `U` icon, which indicates that the file is `Unmerged`, and therefore awaiting you help to resolve the conflict. It delimits these blocks with a series of less than and greater than signs, so they are easy to find:



To resolve the conflicts, simply find all of these blocks, and edit them so that the file looks how you want (either pick your lines, your collaborators lines, some combination, or something altogether new), and save. Be sure you removed the delimiter lines that started with <<<<<<<, =======, and >>>>>>>.

Once you have made those changes, you simply add, commit, and push the files to resolve the conflict

Producing and resolving merge conflicts

To illustrate this process, we're going to carefully create a merge conflict step by step, show how to resolve it, and show how to see the results of the successful merge after it is complete. First, we will walk through the exercise to demonstrate the issues.

Owner and collaborator ensure all changes are updated

First, start the exercise by ensuring that both the Owner and Collaborator have all of the changes synced to their local copies of the Owner's repository in Studio. This includes doing a `git pull` to ensure that you have all changes local, and make sure that the Git tab in RStudio doesn't show any changes needing to be committed.

Owner makes a change and commits

From that clean slate, the Owner first modifies and commits a small change inlcuding their name on a specific line of the README.md file (we will change line 4). Work to only change that one line, and add your username to the line in some form and commit the changes (but DO NOT push). We are now in the situation where the owner has unpushed changes that the collaborator can not yet see.

Collaborator makes a change and commits on the same line

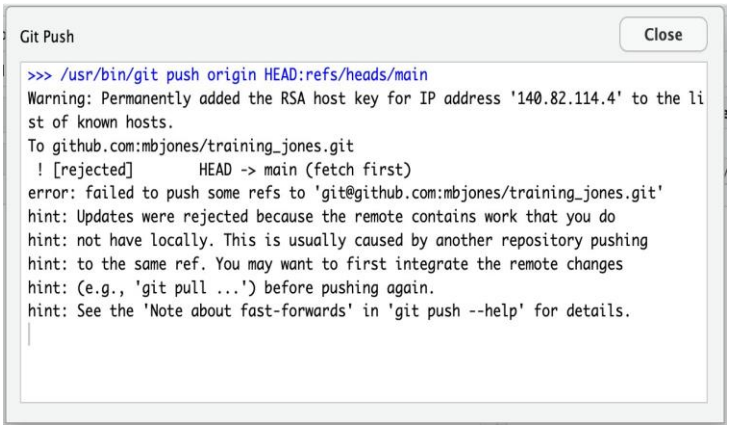
Now the collaborator also makes changes to the same (line 4) of the README.md file in their RStudio copy of the project, adding their name to the line. They then commit. At this point, both the owner and collaborator have committed changes based on their shared version of the README.md file, but neither has tried to share their changes via GitHub.

Collaborator pushes the file to GitHub

Sharing starts when the Collaborator pushes their changes to the GitHub repo, which updates GitHub to their version of the file. The owner is now one revision behind, but doesn't yet know it.

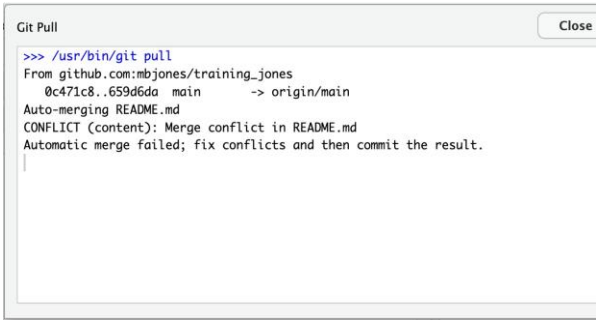
Owner pushes their changes and gets an error

At this point, the owner tries to push their change to the repository, which triggers an error from GitHub. While the error message is long, it basically tells you everything needed (that the owner's repository doesn't reflect the changes on GitHub, and that they need to pull before they can push).

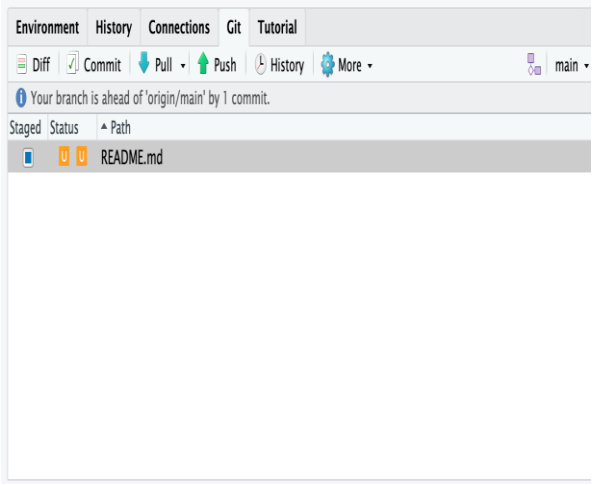


Owner pulls from GitHub to get Collaborator changes

Doing what the message says, the Owner pulls the changes from GitHub, and gets another, different error message. In this case, it indicates that there is a merge conflict because of the conflicting lines.

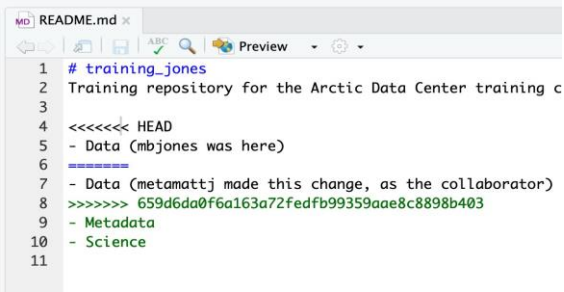


In the Git pane of RStudio, the file is also flagged with an orange 'U', which stands for an unresolved merge conflict.

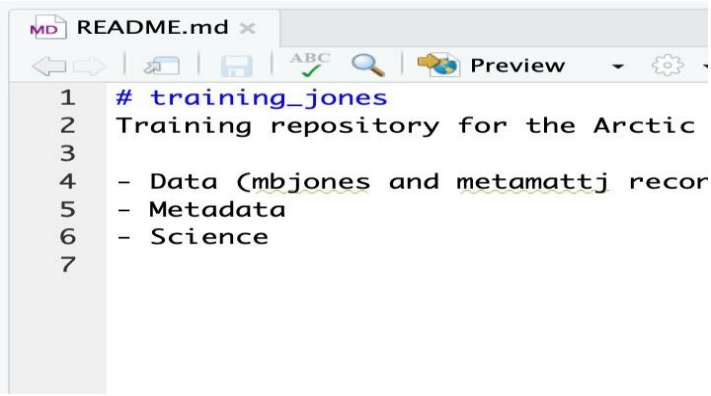


Owner edits the file to resolve the conflict

To resolve the conflict, the Owner now needs to edit the file. Again, as indicated above, git has flagged the locations in the file where a conflict occcurred with <<<<<<<, =======, and >>>>>>>. The Owner should edit the file, merging whatever changes are appropriate until the conflicting lines read how they should, and eliminate all of the marker lines with with <<<<<<<, =======, and >>>>>>>



Of course, for scripts and programs, resolving the changes means more than just merging the text – whoever is doing the merging should make sure that the code runs properly and none of the logic of the program has been broken.

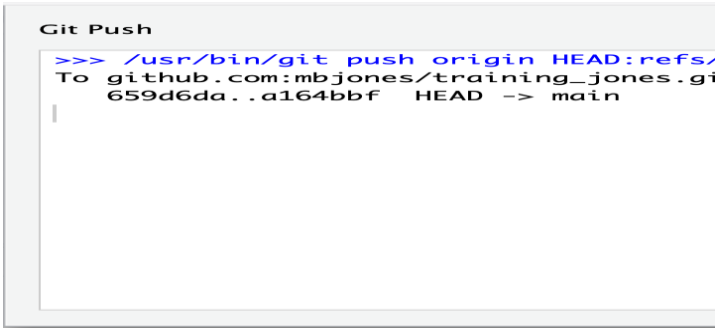


Owner commits the resolved changes

From this point forward, things proceed as normal. The owner first 'Adds' the file changes to be made, which changes the orange U to a blue M for modified, and then commits the changes locally. The owner now has a resolved version of the file on their system.

Owner pushes the resolved changes to GitHub

Have the Owner push the changes, and it should replicate the changes to GitHub without error.



Collaborator pulls the resolved changes from GitHub

Finally, the Collaborator can pull from GitHub to get the changes the owner made.

Both can view commit history

When either the Collaborator or the Owner view the history, the conflict, associated branch, and the merged changes are clearly visible in the history.

Merge Conflict Challenge

Now it's your turn. In pairs, intentionally create a merge conflict, and then go through the steps needed to resolve the issues and continue developing with the merged files. See the sections above for help with each of these steps:

- Step 0: Owner and collaborator ensure all changes are updated
- Step 1: Owner makes a change and commits
- Step 2: Collaborator makes a change and commits on the same line
- Step 3: Collaborator pushes the file to GitHub
- Step 4: Owner pushes their changes and gets an error
- Step 5: Owner pulls from GitHub to get Collaborator changes
- Step 6: Owner edits the file to resolve the conflict
- Step 7: Owner commits the resolved changes
- Step 8: Owner pushes the resolved changes to GitHub
- Step 9: Collaborator pulls the resolved changes from GitHub
- Step 10: Both can view commit history

Workflows to avoid merge conflicts

Some basic rules of thumb can avoid the vast majority of merge conflicts, saving a lot of time and frustration. These are words our teams live by:

- Communicate often
- Tell each other what you are working on
- Pull immediately before you commit or push
- Commit often in small chunks.

A good workflow is encapsulated as follows:

`Pull -> Edit -> Add -> Pull -> Commit -> Push`

Always start your working sessions with a pull to get any outstanding changes, then start doing your editing and work. Stage your changes, but before you commit, Pull again to see if any new changes have arrived. If so, they should merge in easily if you are working in different parts of the program. You can then Commit and immediately Push your changes safely. Good luck, and try to not get frustrated. Once you figure out how to handle merge conflicts, they can be avoided or dispatched when they occur, but it does take a bit of practice.

Experiment No. 09

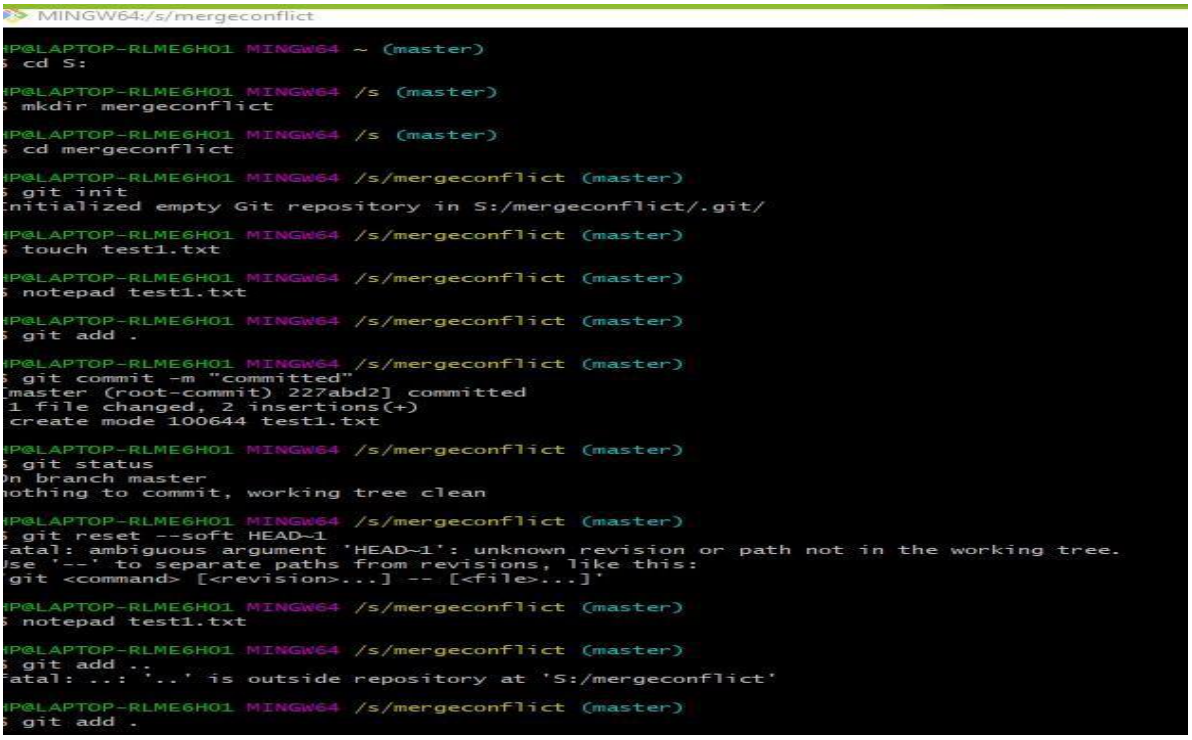
Aim: Reset and Revert

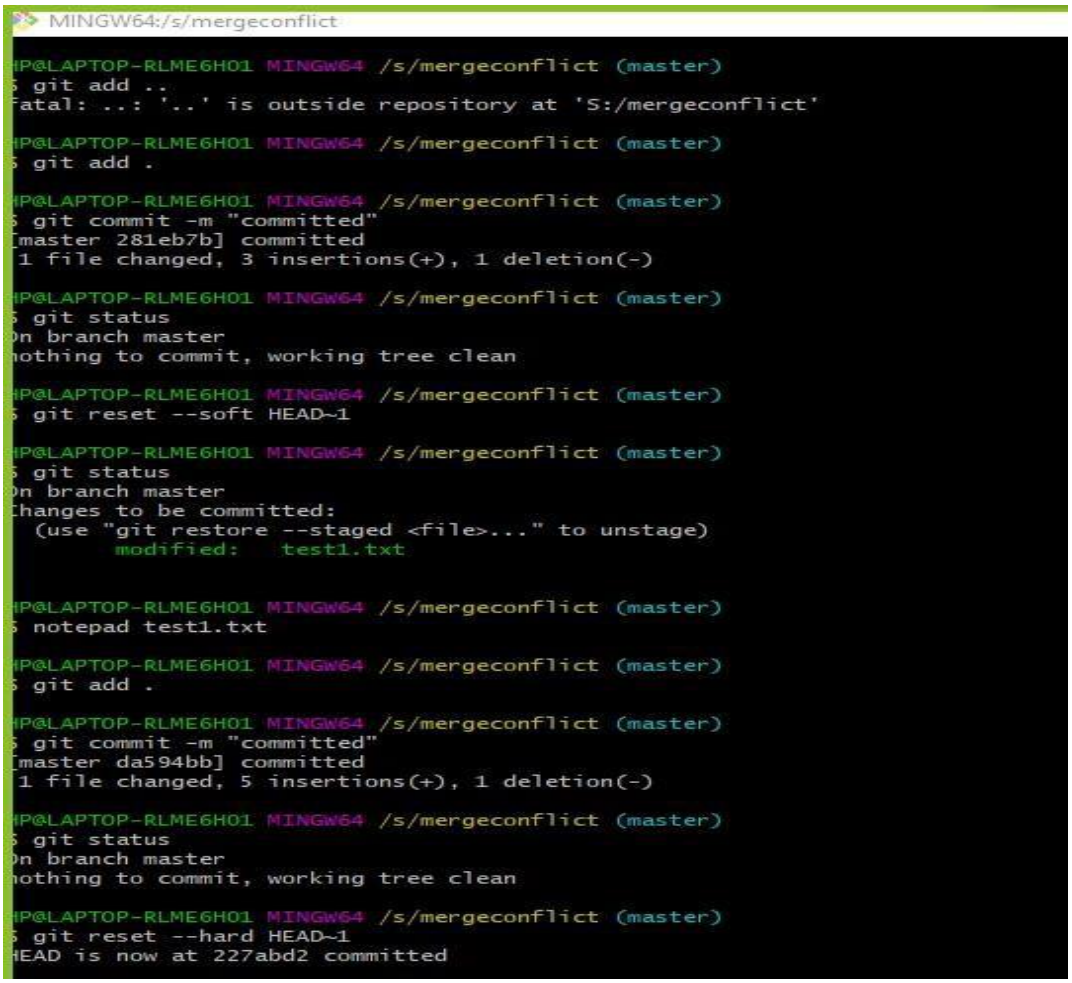
One of the lesser understood (and appreciated) aspects of working with Git is how easy it is to get back to where you were before—that is, how easy it is to undo even major changes in a repository. In this article, we'll take a quick look at how to reset, revert, and completely return to previous states, all with the simplicity and elegance of individual Git commands.

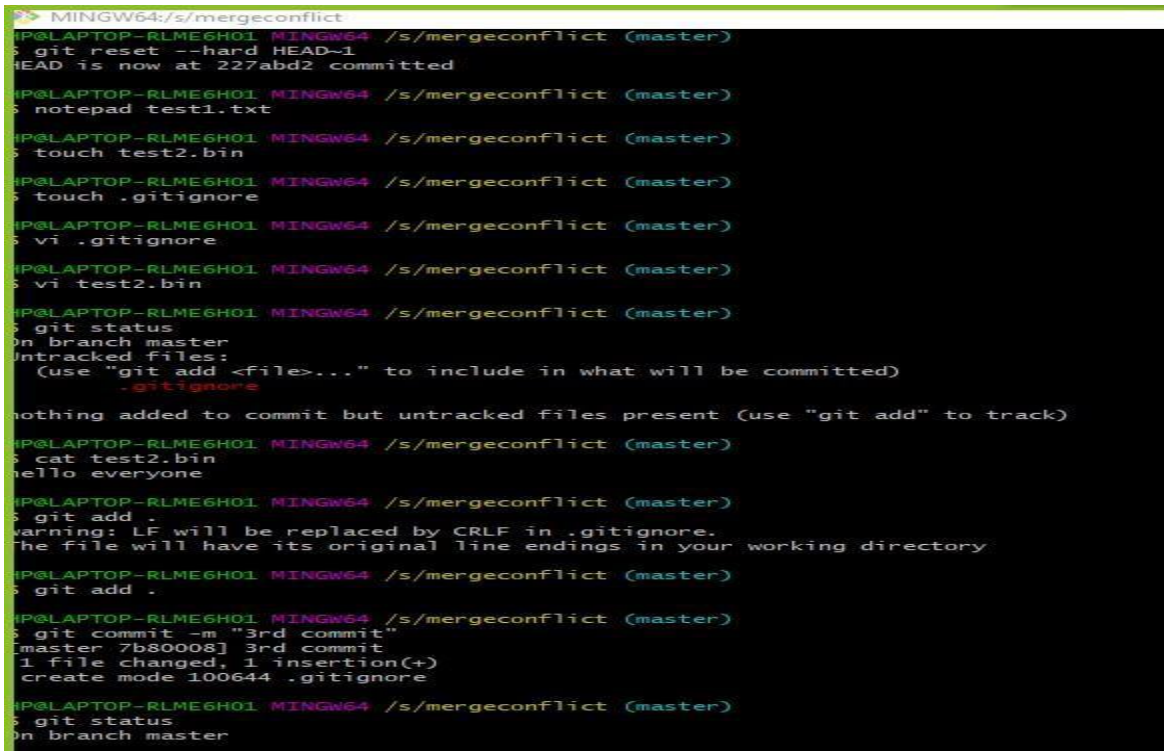
1. Git Reset :- Git reset is a powerful command that is used to undo local changes to thestate of a Git repo. Git reset operates on "The Three Trees of Git". These trees are the Commit History (HEAD), the Staging Index, and the Working Directory.

The easiest way to undo the last Git commit is to execute the “git reset” command with the“—soft” option that will preserve changes done to your files.

Git reset --hard , which will completely destroy any changes and remove them from the localdirectory.


MINGW64:/s/mergeconflict
P@LAPTOP-RLME6HO1 MINGW64 ~ (master)
$ cd S:
P@LAPTOP-RLME6HO1 MINGW64 /s (master)
$ mkdir mergeconflict
P@LAPTOP-RLME6HO1 MINGW64 /s (master)
$ cd mergeconflict
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git init
nitialized empty Git repository in S:/mergeconflict/.git/
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ touch test1.txt
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ notepad test1.txt
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git add .
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git commit -m "committed"
master (root-commit) 227abd2] committed
1 file changed, 2 insertions(+)
create mode 100644 test1.txt
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git status
n branch master
othing to commit, working tree clean
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git reset --soft HEAD~1
fatal: ambiguous argument 'HEAD~1': unknown revision or path not in the working tree.
se '--' to separate paths from revisions, like this:
git <command> [<revision>...] -- [<file>...]'
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ notepad test1.txt
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git add ..
fatal: ..: '..' is outside repository at 'S:/mergeconflict'
P@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git add .


MINGW64:/s/mergeconflict
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git add ..
fatal: ..: '..' is outside repository at 'S:/mergeconflict'
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git add .
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git commit -m "committed"
[master 281eb7b] committed
1 file changed, 3 insertions(+), 1 deletion(-)
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git status
On branch master
nothing to commit, working tree clean
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git reset --soft HEAD~1
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git status
On branch master
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
modified: test1.txt
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ notepad test1.txt
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git add .
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git commit -m "committed"
[master da594bb] committed
1 file changed, 5 insertions(+), 1 deletion(-)
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git status
On branch master
nothing to commit, working tree clean
HP@LAPTOP-RLME6HO1 MINGW64 /s/mergeconflict (master)
$ git reset --hard HEAD~1
HEAD is now at 227abd2 committed



2. Git ignore :- When sharing your code with others, there are often files orparts of your project, you do not want to share.

Git can specify which files or parts of your project should be ignored by Git usinga `.gitignore` file.

Git will not track files and folders specified in `.gitignore`. However, the `.gitignore` file itself **IS** tracked by Git

A Project report

on

“Project Title”

with

Source Code Management

(CS181)

Submitted by

Team Member 1	Himanshu sharma	2110990620
Team Member 2	Armaan singh bhasin	2110990258
Team Member 3	Harul arora	2110990600
Team Member 3	Arshdeep singh	2110990274

Department of Computer Science & Engineering

Chitkara University Institute of Engineering and Technology, Punjab

Jan- June
(2021-22)

Institute/School Name	**Chitkara University Institute of Engineering and Technology**		
Department Name	**Department of Computer Science & Engineering**		
Programme Name	**Bachelor of Engineering (B.E.), Computer Science & Engineering**		
Course Name	**Source Code Management**	Session	**2021-22**
Course Code	**CS181**	Semester/Batch	**2nd/2021**
Vertical Name	**Beta**	Group No	G-08
Course Coordinator	**Dr. Navjeet Kaur**		
Faculty Name	**Dr. Monit Kapoor**		

Submission

Name: Armaan Singh Bhasin
Date: 2/06/2022

CHITKARA UNIVERSITY

Table of Content

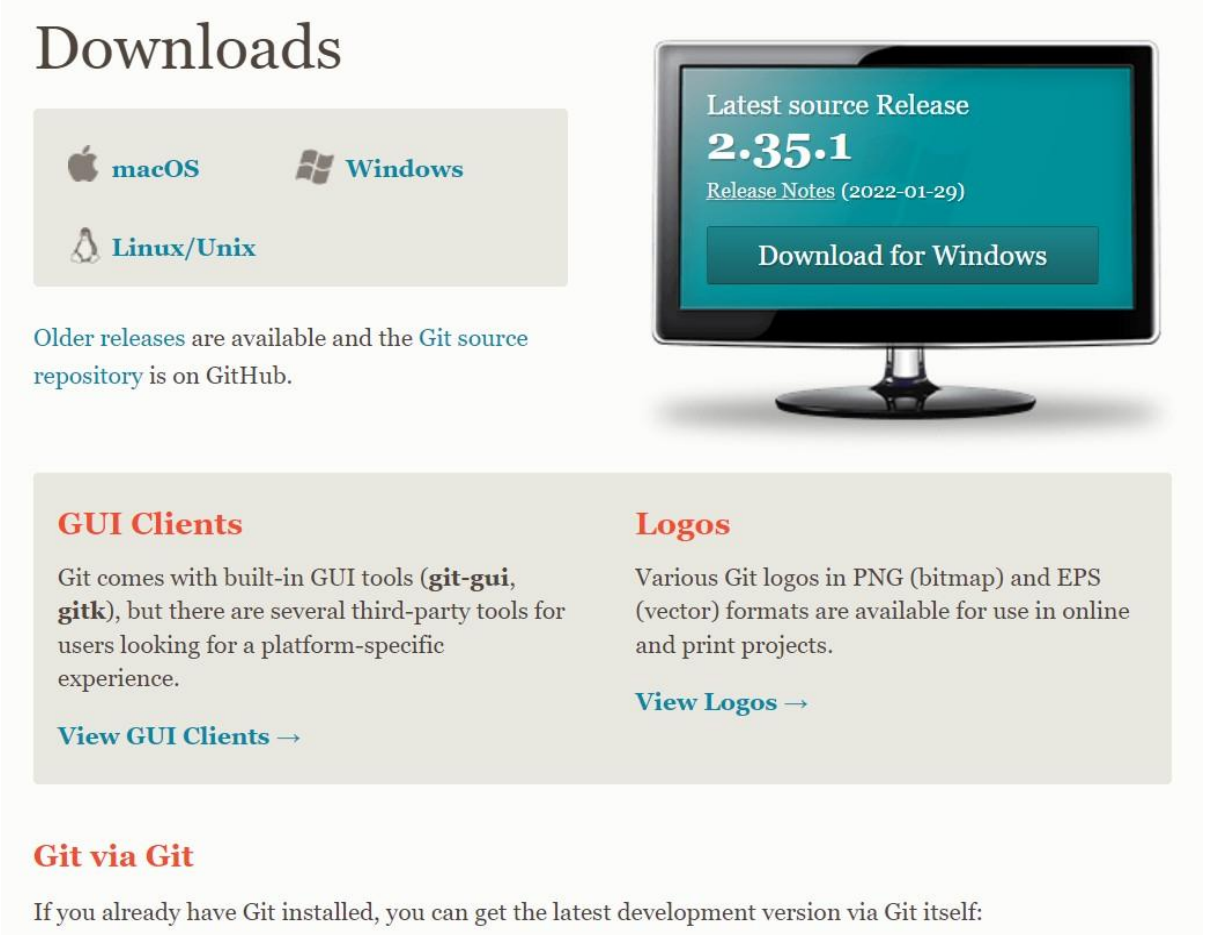
1. Version control with Git

Steps:-

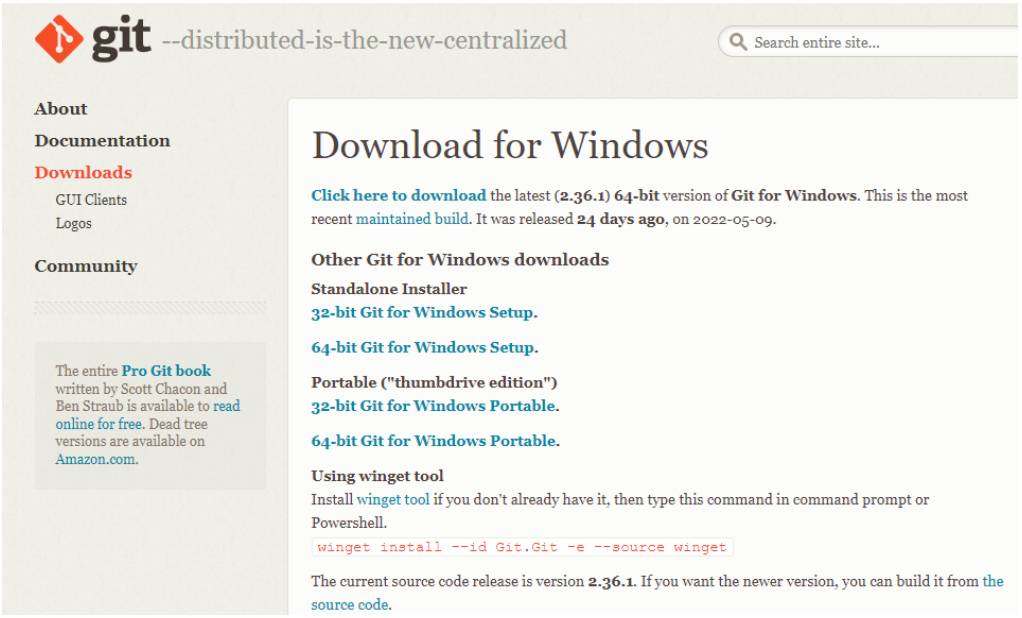
To download the Git installer, visit the Git official site and go to thedownload page.

The link for the download page is https://git-scm.com/downloads

The page looks like as:



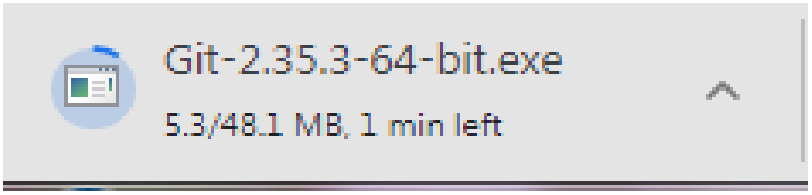
Click on the package given on the page as **download 2.23.0 for windows**. The download will start after selecting the package.



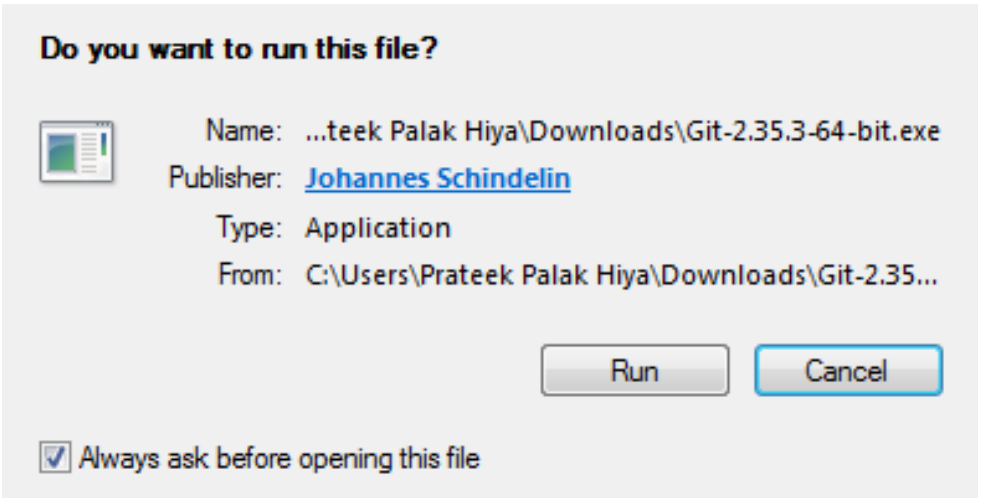
Now, the Git installer package has been downloaded.

❖ Click on the download for which you want to download.The page looks like as: -

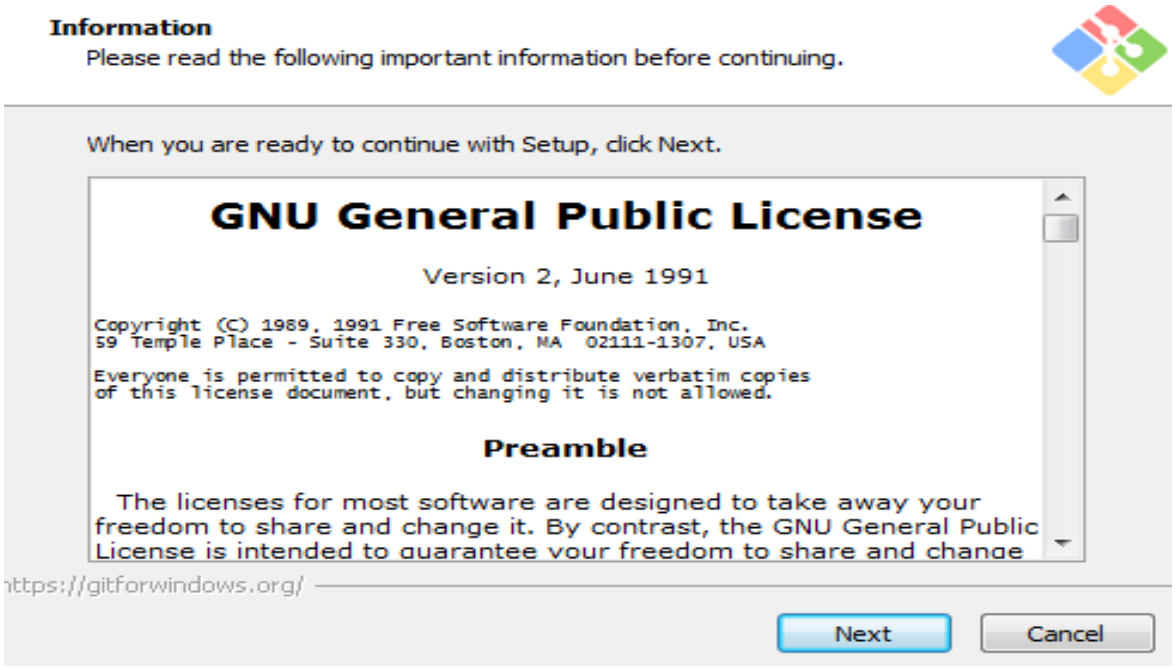
❖ Downloading will start here.



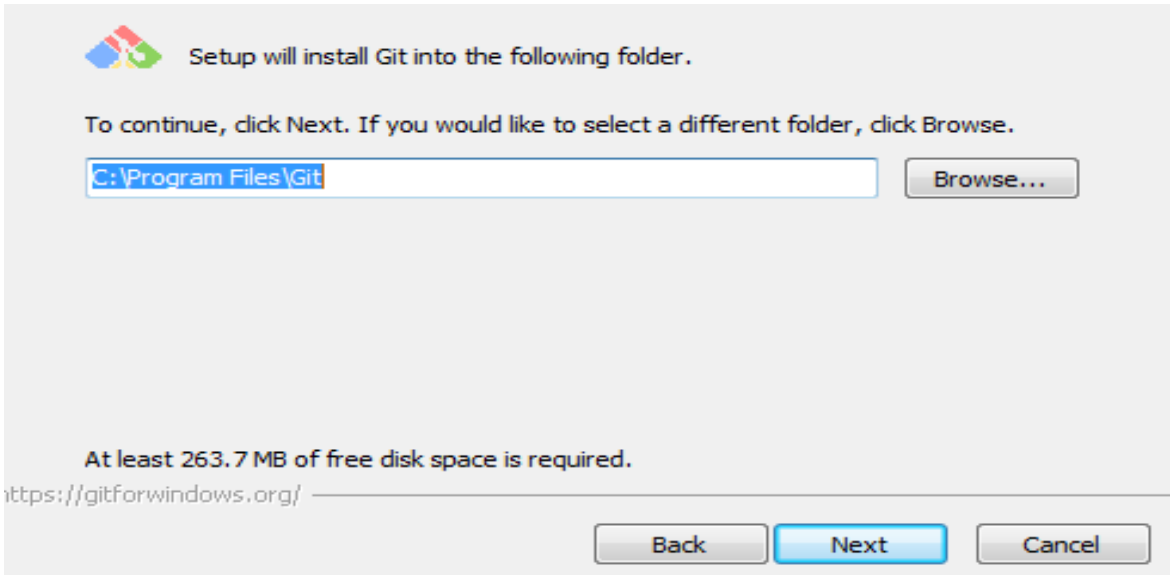
❖ Now click on run as shown here.


Do you want to run this file?
Name: ...teek Palak Hiya\Downloads\Git-2.35.3-64-bit.exe
Publisher: Johannes Schindelin
Type: Application
From: C:\Users\Prateek Palak Hiya\Downloads\Git-2.35...
Run
Cancel
Always ask before opening this file

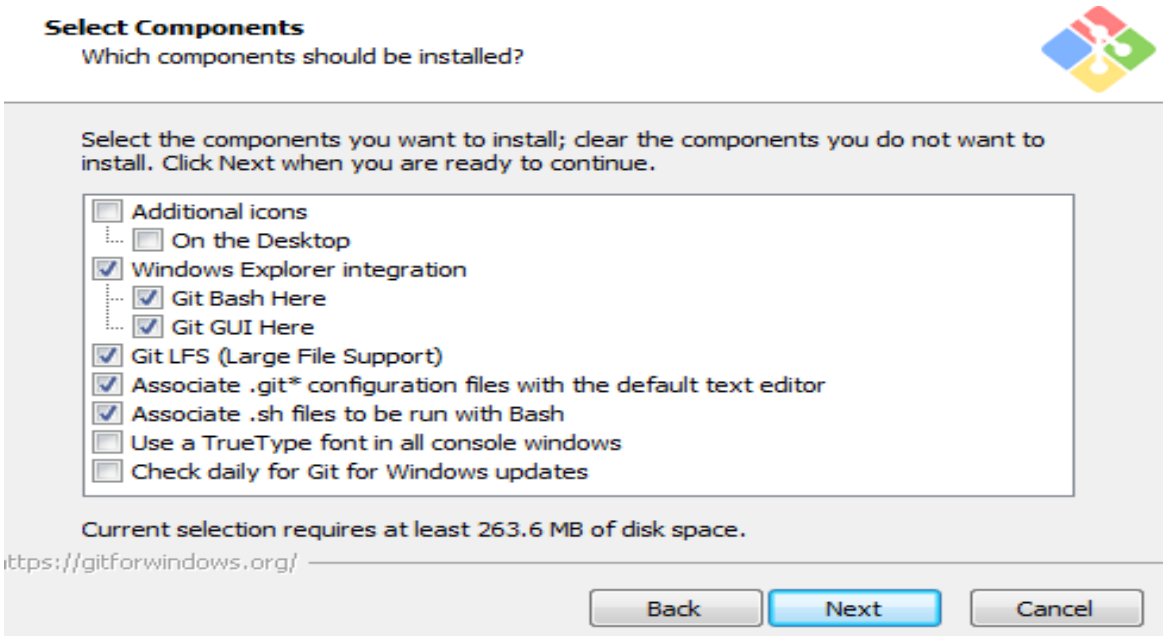
❖ Do next as shown below.



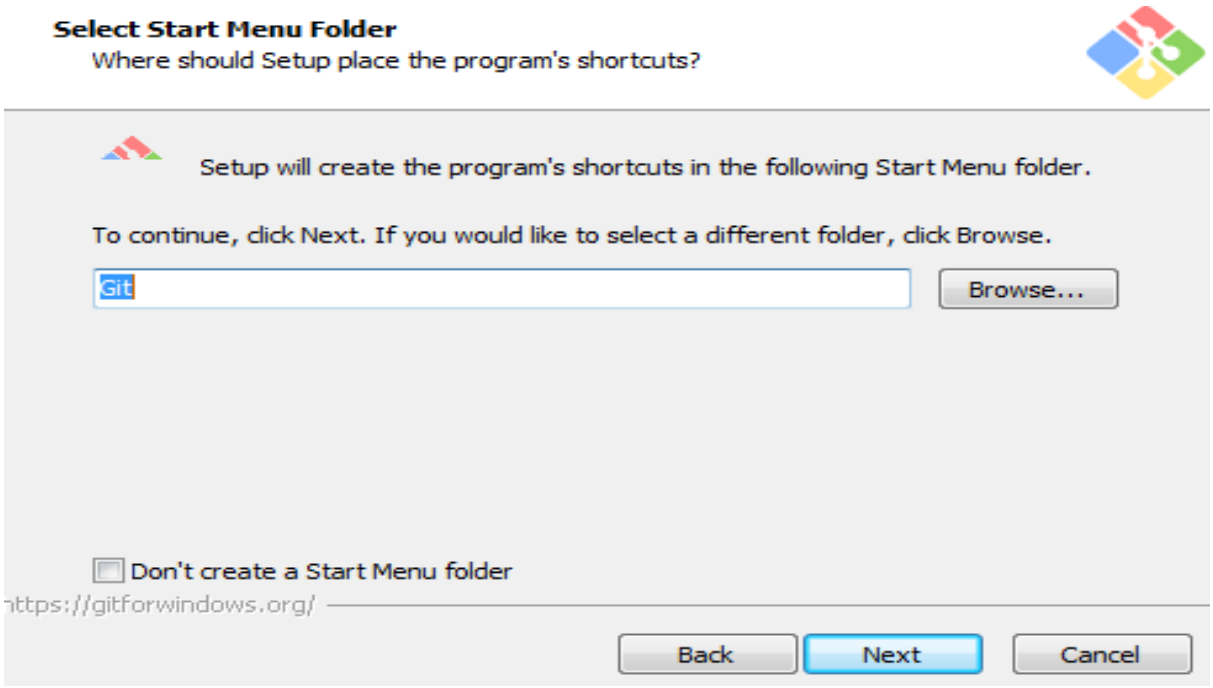
❖ Select the folder you want to do and do next.



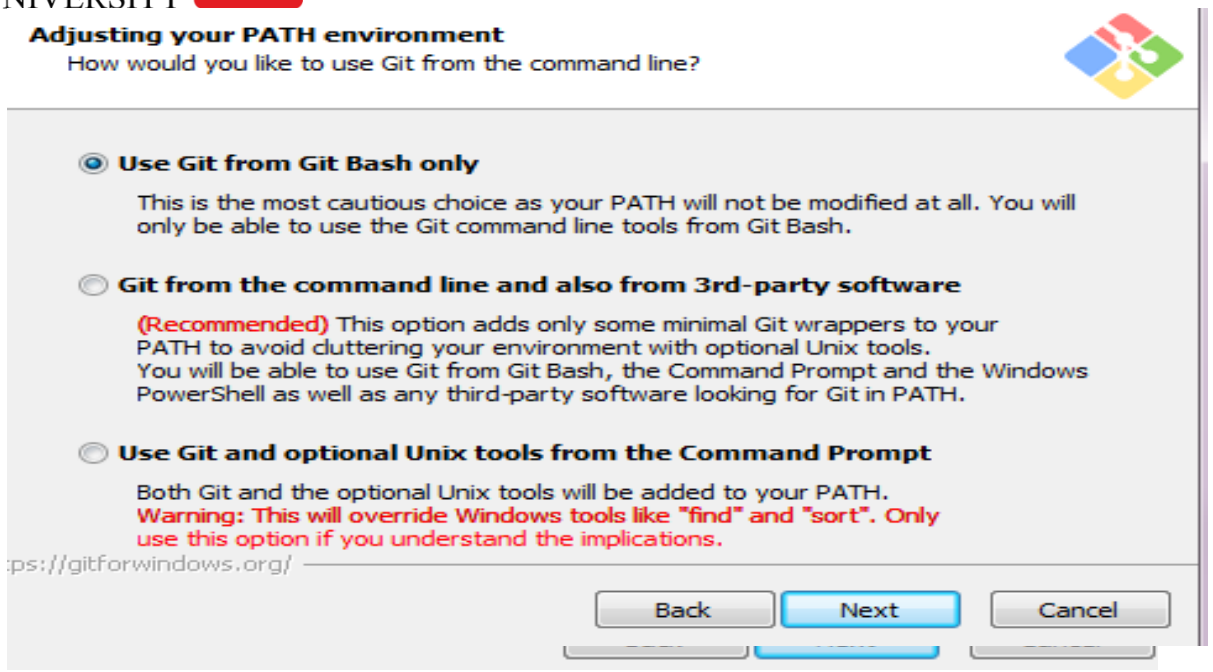
❖ Simply click on the next button as it automatically selects the required file.The page looks like as: -



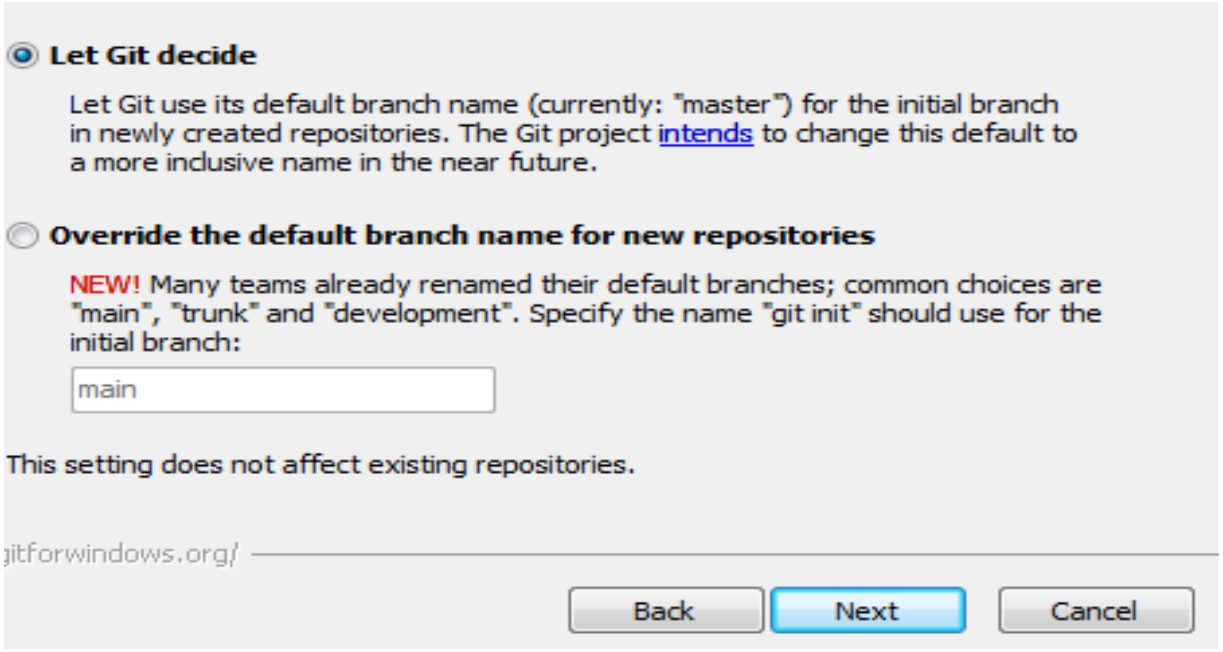
❖ Do next



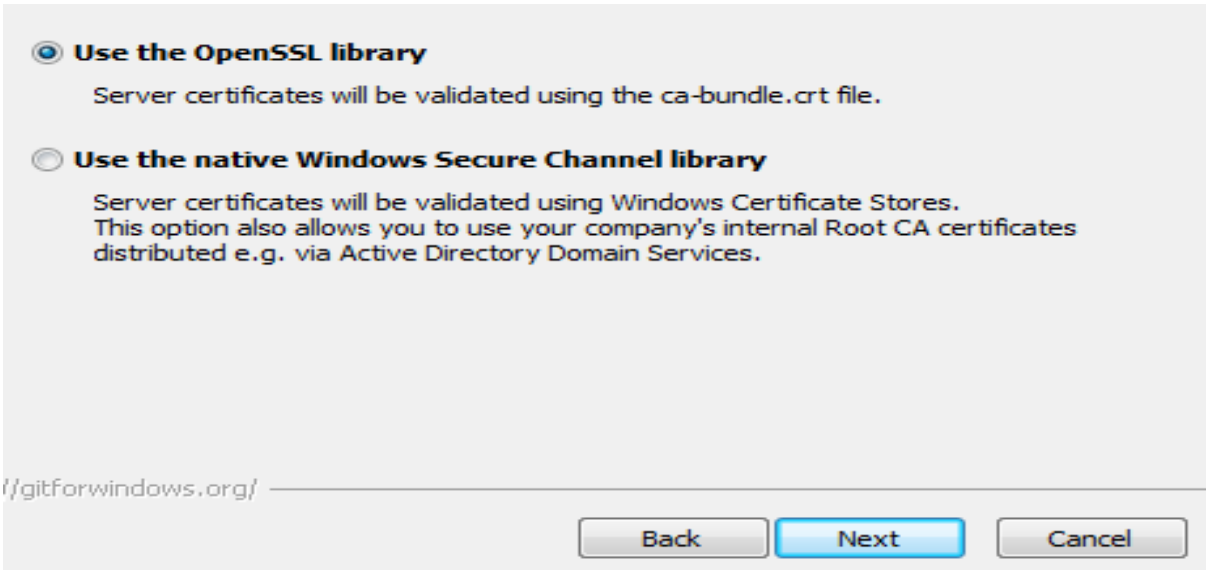
❖ Use vim as a Git default editor.

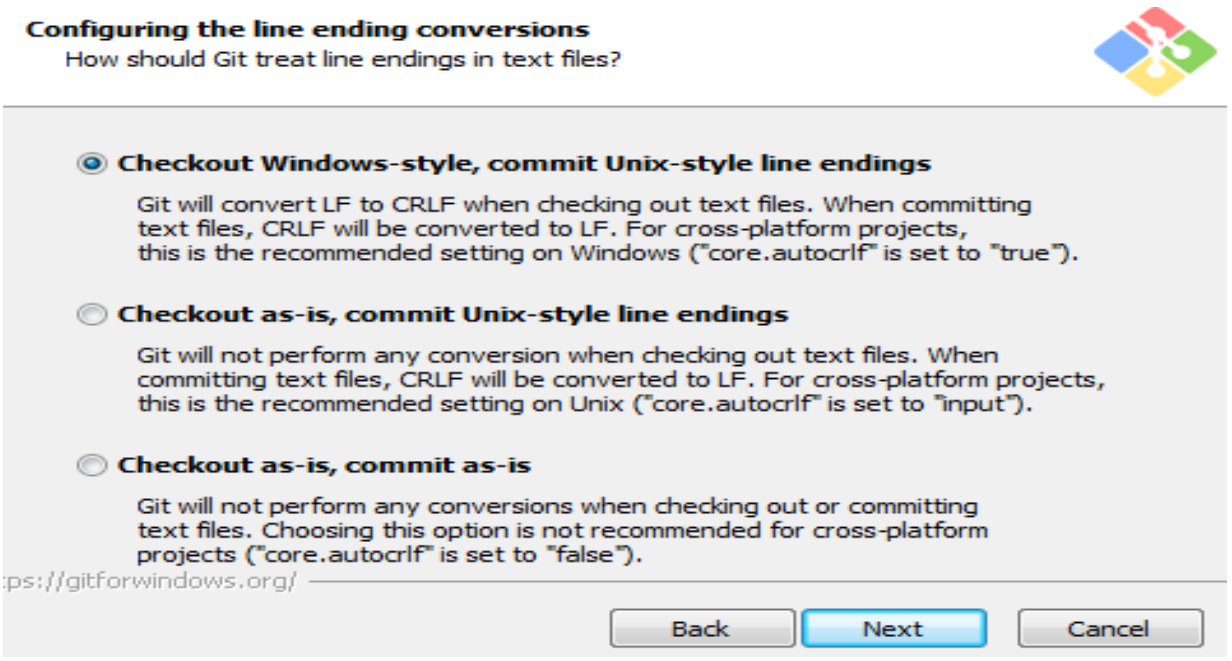


❖ Do next.

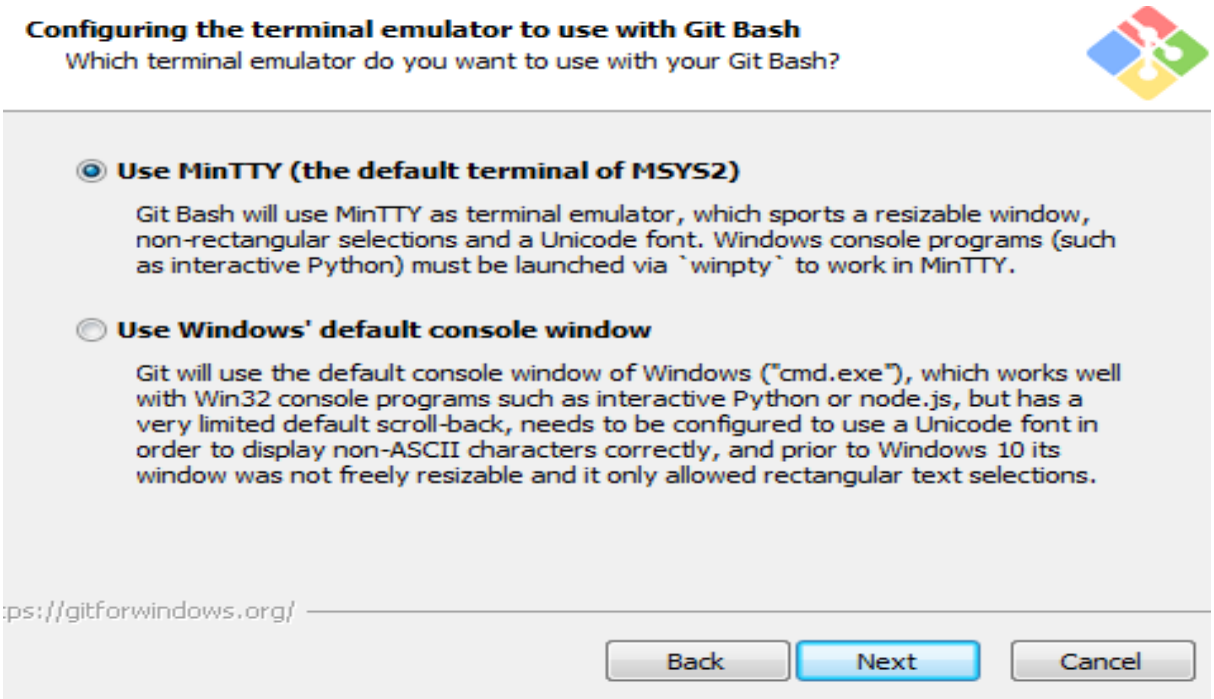


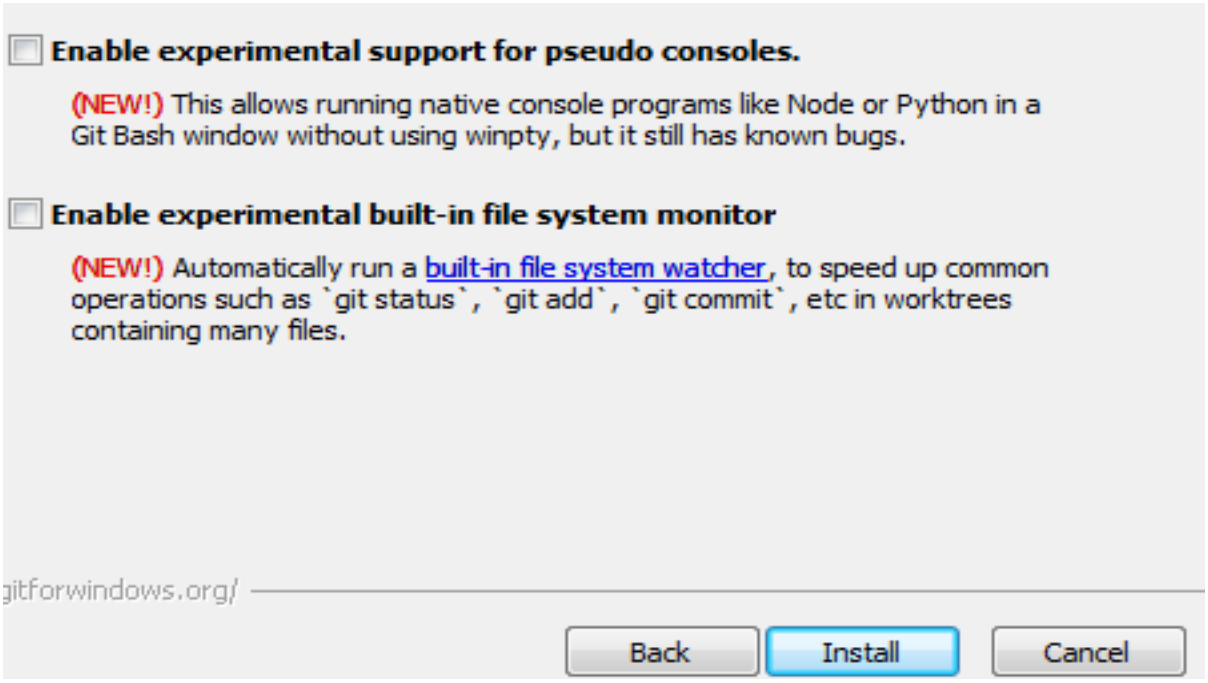
❖ Use Git from Git Bash only, as shown .


Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates
distributed e.g. via Active Directory Domain Services.
//gitforwindows.org/
Back
Next
Cancel


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").
ps://gitforwindows.org/
Back
Next
Cancel

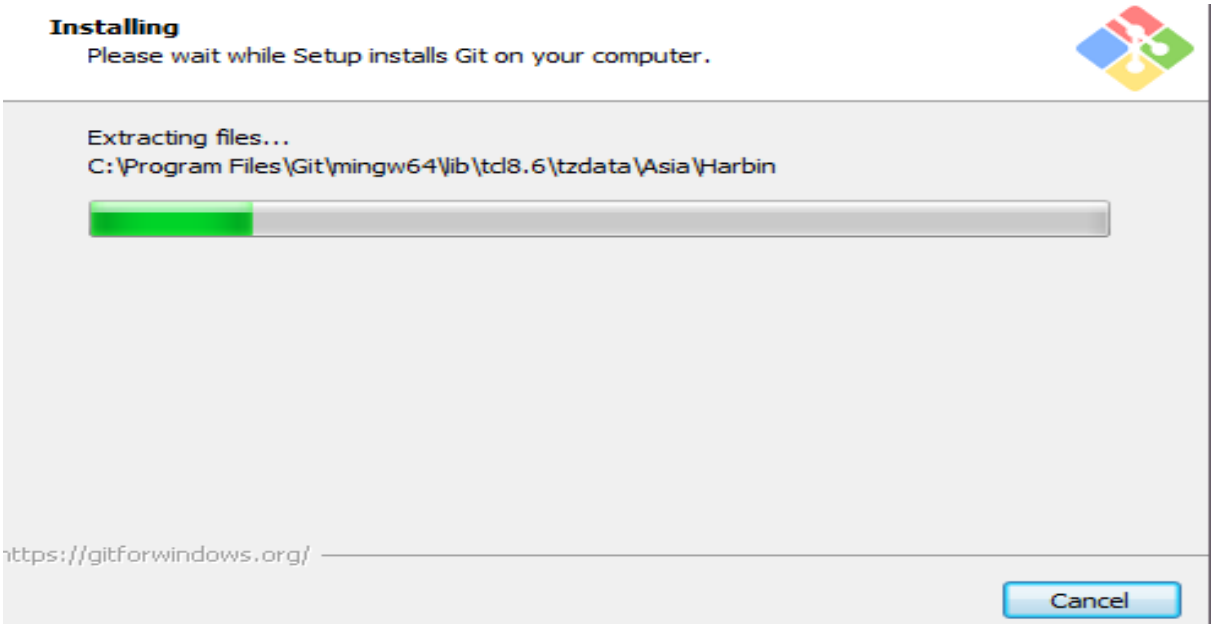
❖ Choose default.

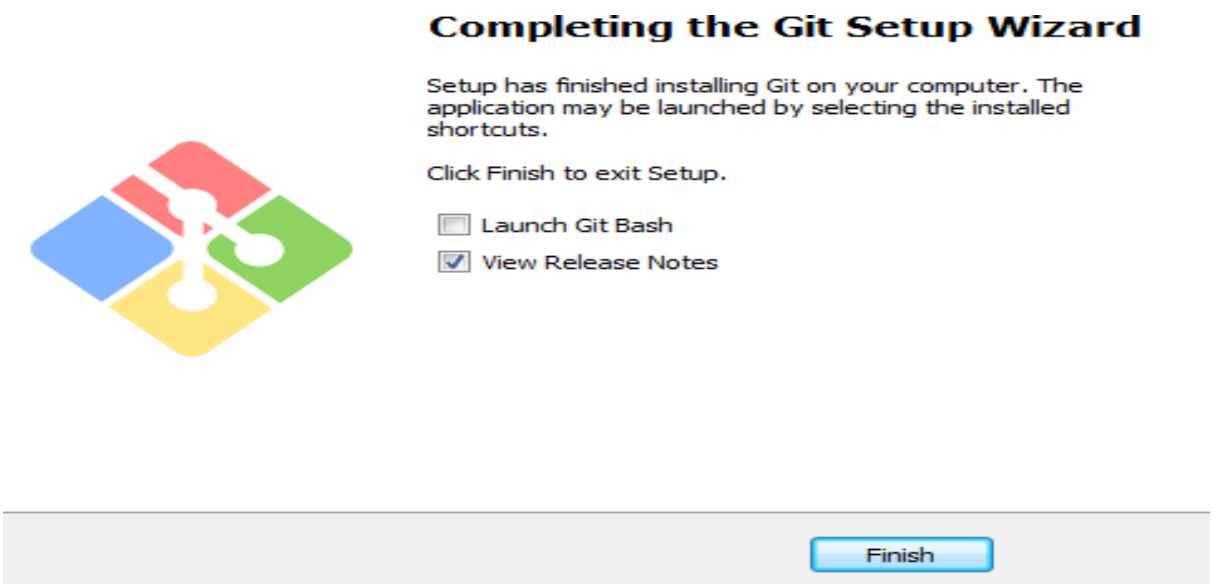




❖ The Git is getting download in your system. The page lookslike as:

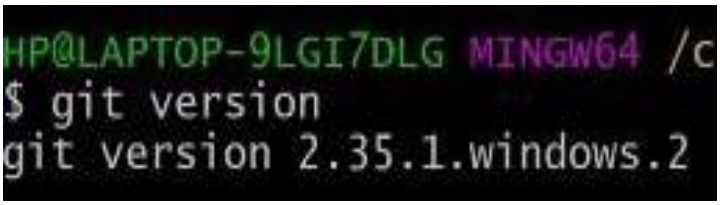
-





- You can check that Git is install by simply type git - -versionin

The page looks like as: -



2. Problem Statement

Aim: Write a program to accept N numbers from the user and store them in an array. Then, accept another number from the user and search that using Linear Search.

PROGRAM CODE:-

```
#ProjectMCP.cpp > main()
#include<iostream>
using namespace std;

class arr
{
    public:
        void create();
        void display();
};
void arr::create()
{
    int i,n,flag=0,j;
 cout<<"enter the no. of elements in an array"<<endl;
    cin>>j;
    int a[j];
    cout<<"enter the "<<j<<" elements"<<endl;
    for(i=0;i<j;i++)
    {
    cin>>a[i];
    }
}
void arr::display()
{
    int flag=0,n,i,j,a[100];
cout<<"enter no. to be checked"<<endl;
    cin>>n;
    for(i=0;i<j;i++)
    {
        if(a[i]==n)
        {
            flag=1;
            cout<<"index of no. is "<<i<<endl;
            break;
        }
        else{
            flag=0;
        }
    }
    if(flag==1)
    {
        cout<<"YES no. is present";
    }
    else
    {
 cout<<"NO no. is not present";
    }
}
int main()
{
    arr ll;
    ll.create();
    ll.display();
}
```

OUTPUT:

```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS S:\babuji> cd "s:\babuji\" ; if ($?) { g++ 1stProjectMCP.cpp -o 1stProjectMCP } ; if ($?) { .\1stProjectMCP }
enter the no. of elements in an array
3
enter the 3 elements
4
7
2
enter no. to be checked
6
NO no. is not present
PS S:\babuji>
```

3. Objective

Through our project, The Awd clone we learnt the core concepts of HTML& JavaScript. We learnt Developer Skills in JavaScript.

With the help of DOM & Event Fundamentals we created a more authentic working for the project webpage.

We also observed how to Manipulate CSS Styles, Handling Click Events. Working with Class Object was a great experience, Dry Running the program helped us to figure out the common mistake one makes during such projects.

We learnt new concepts with every step on the project. The users can refresh their mind by playing this user-friendly music player website. The code is written in a very simple manner so that even the basic users don't face any difficulties going through the code.

This AWD Project covers all concepts from basic to advance in HTML & JavaScript. So, this project is a one stop solution for everyone who wants to learn Web-Coding and have a keen interest in developing such Fun Projects.

4. Concepts and Commands

Codes of git with concept with the concepts:

- **git status**
  To check the work done

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

- **git config user.name**
  To verify linked mail
- **git config --global user.name**
  To link repo with GitHub username

```
DELL@DESKTOP-OLO2QO5 MINGW64 ~
$ git config --global user.name
himi620
```

- **git config --global user.email**
  To link repo with GitHub mail
- **git config user.email**
  To verify linked username

```
DELL@DESKTOP-OLO2QO5 MINGW64 ~
$ git config user.email
himi50071@gmail.com
```

- **git init**
- To make folder git ready

```
DELL@DESKTOP-OLO2QO5 MINGW64 ~
$ git init
Initialized empty Git repository in C:/Users/DELL/.git/
```

- **git add –a**
  To push all the files to repo
- **git add filename**
  To push a particular file to repo

```
$ git add .
```

- **git branch name**
  To make a new branch
- **git checkout name**
- To switch between branch

- **git branch -d branch name**
  To delete branch (**Soft delete** because it ask to merge **)**
- **git branch -D branch name**
  To delete branch **(Hard delete** because it don't ask to merge**)**

```
$ git branch -D new2
Deleted branch new2 (was eedecc5).
```

- **git branch**
  To see number of branches

```
$ git branch
  activity1
* master
```

- **git branch -m new branch name**
  To rename a branch (we need to be in that branch)

```
$ git branch -m activity
```

- **git branch -r**
  To see number of branches

```
$ git branch -r
  origin/master
```

- **git log**
  used to check the history of the work done also contains a checksum

```
DELL@DESKTOP-OLO2QO5 MINGW64 /c/himi620 (master)
$ git commit -a -m "some changes"
[master 1fb1501] some changes
 1 file changed, 2 insertions(+), 1 deletion(-)

DELL@DESKTOP-OLO2QO5 MINGW64 /c/himi620 (master)
$ git status
On branch master
nothing to commit, working tree clean

DELL@DESKTOP-OLO2QO5 MINGW64 /c/himi620 (master)
$ git log
commit 1fb15016716a1b369f8dd6d74d3c963e08837216 (HEAD -> master
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:54:34 2022 +0530

    some changes

commit 1ed483450dde05f901c637662d7113f04679ded6
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:49:12 2022 +0530

    newfile

commit 20a41ac2a79773df15aad78d0af15e4b23a8f343
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:32:53 2022 +0530

     modified this file

commit cdfbf538f52f22247247919201da97e454436422
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:23:18 2022 +0530

     commit this file

DELL@DESKTOP-OLO2QO5 MINGW64 /c/himi620 (master)
$
```

- **git log –oneline**

  To get log in short

```
$ git log --oneline
fd0efed (HEAD -> activity, origin/master) THIS IS A MIRROR REPOSITORY
fe7d352  ADDED TimeComplexity.java
dd363c1 ADDED PATTERNS-3
ae00b88 ADDED PATTERNS-1
74662e6 ADDED RECURSION-3
c13802c ADDED REDCURSION-2
e5b1b7a Recursion-1
432f5f2 ADDED OOPS-1
188840a ADDED OOPS-1
e6af6b0 ADDED QUEUES
188e36d ADDED BACKTRACKING
c9f0778 ADDED HASHMAPS
eebface ADDED GRAPHS
a6aad9d ADDED STRINGS
cb391eb ADDED STACK
```

- **git log --online –graph**

  To get log in graph format

```
$ git log --oneline --graph
* fd0efed (HEAD -> activity, origin/master) THIS IS A MIRROR REPOSITORY
* fe7d352  ADDED TimeComplexity.java
* dd363c1 ADDED PATTERNS-3
* ae00b88 ADDED PATTERNS-1
* 74662e6 ADDED RECURSION-3
* c13802c ADDED REDCURSION-2
* e5b1b7a Recursion-1
* 432f5f2 ADDED OOPS-1
* 188840a ADDED OOPS-1
* e6af6b0 ADDED QUEUES
* 188e36d ADDED BACKTRACKING
* c9f0778 ADDED HASHMAPS
* eebface ADDED GRAPHS
* a6aad9d ADDED STRINGS
* cb391eb ADDED STACK
```

- **pwd**

  Present working Directory

```
$ pwd
/e/Coding-Ninjas-java-
```

- **git commit -m ”any message”**
  To be done after staging

```
$ git commit -m "added files"
On branch activity
nothing to commit, working tree clean
```

- **git merge branch name**
  To merge sub branch with master

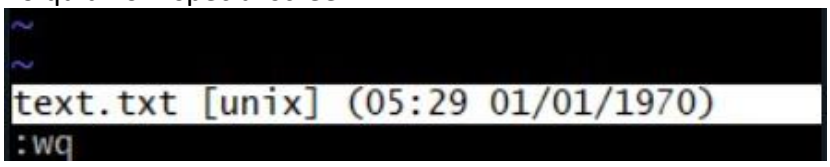
```
$ git merge 'activity'
Already up to date.
```

- **mv old-file-name new-file-name**
  To rename a flolder (we have to do staging for this)
- **git mv old-file-name new-file-name**
  To rename a folder (no need for staging)

```
$ rm -rf SetRepresentation.java
```

- **git restore --staged filed name**
  To reverse to to previous version
- **git mergetool**
  To remove merge conflict i.e. when content in master and branch is different.

- **:wq**
  To quit from special screen



- **rm -rf .git**
  To delete whole git folder
- **rm -rf filename**

To delete a particular file

```
$ rm -rf SetRepresentation.java
```

- **git remote add origin "link-of-repo-we-made-on-github**

  To make new remote

```
$ git remote add origin https://github.com/LORDFLACKO0087/Awd_codes.git
```

- **git push -u master remote-nam**

  To make data visible on sscloud

```
$ git remote add origin https://github.com/aaryansood2512chitkara/2110990020_AARYAN-SOOD.git
```

- **git pull link-of-repo-on-github**

  To make changes done on cloud visible on the system.

- **git remote -v**

  to see the location where remote is being stored

```
$ git remote -v
origin  ssh://git@github.com:Group08-Chitkara-University/2110990020.git (fetch)
origin  ssh://git@github.com:Group08-Chitkara-University/2110990020.git (push)
```

- **touch filename**

  To make css/html/c++ files

```
$ touch new.txt
```

- **git revert checksum**

  It's a forwar moving undo operation that offers a safe mode of undoing changes

```
Revert "new files added"

This reverts commit 397d1f2481f4683be0ec631e533fcc041e1ed2e9.

# 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:
#       deleted:    Team-Project-File
#
# Untracked files:
#       Team-Project-File/
```

- **touch .gitignore**
- **git reset --hard checksum**

  All the commits which was rested is deleted in the working directory along with the commit history.

```
$ git reset --hard  fd0efed81cc5ed66b28e79324755b4b37633bd78
HEAD is now at fd0efed THIS IS A MIRROR REPOSITORY
```

- **git reset --mixed checksum**

  Reset commit files doesn't get deleted it goes untracked changes (red colour on git status)

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ../2110990217_Ansh-Wadhwa/
        new.txt
```

- **git reset --soft checksum**
  Resseted commit files doesn't get deleted it goes to staging area (green colour)

```
$ git reset --soft  fd0efed81cc5ed66b28e79324755b4b37633bd78
```

git reset --any-type ~ n(no. of commits to be changed)
To delete no. of commits

```
$ git log --oneline
fd0efed (HEAD -> master, origin/master, activity) THIS IS A MIRROR REPOSITORY
fe7d352  ADDED TimeComplexity.java
dd363c1 ADDED PATTERNS-3
ae00b88 ADDED PATTERNS-1
74662e6 ADDED RECURSION-3
c13802c ADDED REDCURSION-2
e5b1b7a Recursion-1
432f5f2 ADDED OOPS-1
188840a ADDED OOPS-1
e6af6b0 ADDED QUEUES
188e36d ADDED BACKTRACKING
c9f0778 ADDED HASHMAPS
eebface ADDED GRAPHS
a6aad9d ADDED STRINGS
cb391eb ADDED STACK
```

- **git remote**
  To see the name of origin formed

```
$ git remote
origin
```

- **git remote rename old-file-name new-file-name**
  To rename remote

```
$ git remote rename origin origindev
```

- **git remote remove name**
  To delete remote

```
$ git remote remove origin
```

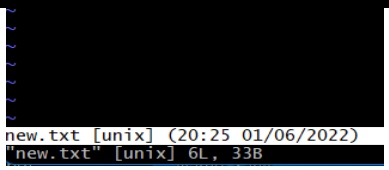
- **cat file-name**

To see the data stored in file in git bash without opening the file

```
$ git remote remove origindev
```

- **vi file-name**
  To edit content of file in git bash without opening the file

5. Workflow and discussion

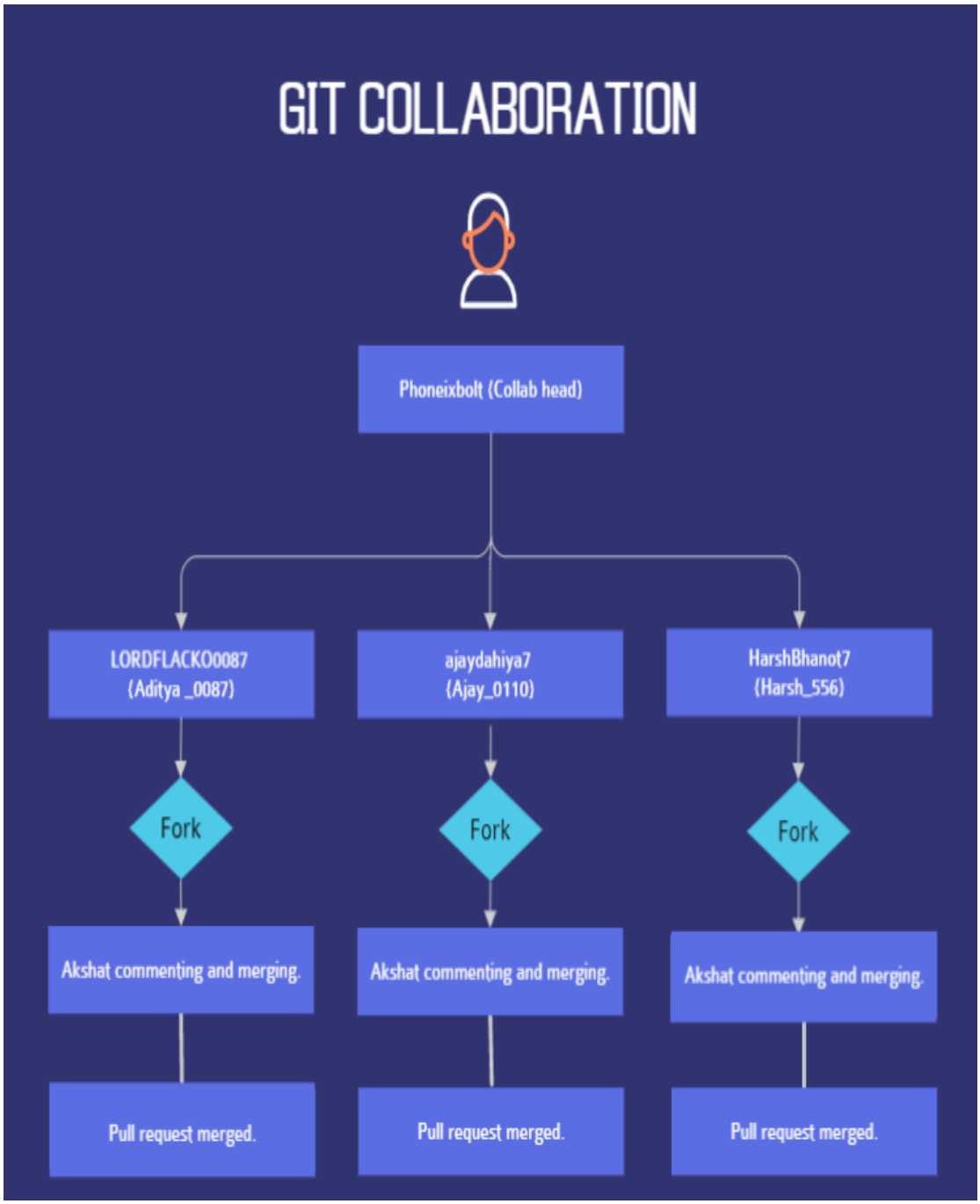


Open source szoftware is based on the idea that by sharing code, we can make better, more reliable software. For more information, see the "About the Open Source Initiative"on the Open Source Initiative. For more information about applying open source principles to your organization'sdevelopment work on GitHub.com, see GitHub's white paper "An introduction to innersource."

When creating your public repository from a fork of someone's project, make sure toinclude a license file that determines how you want your project to be shared with others. For more information, see "Choose an open source license" at choosealicense.com.

For more information on open source, specifically how to create and grow an open source project, we've created Open Source Guides that will help you foster a healthy open source community by recommending best practices for creating and maintaining repositories for your open source project. You can also take a free GitHub Learning Lab course on maintaining open source communities.

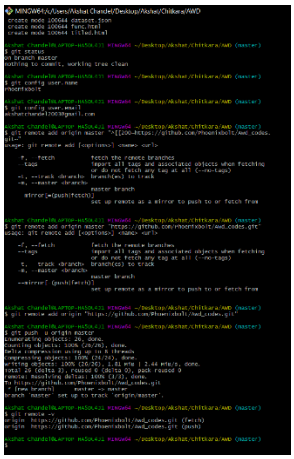
Forking



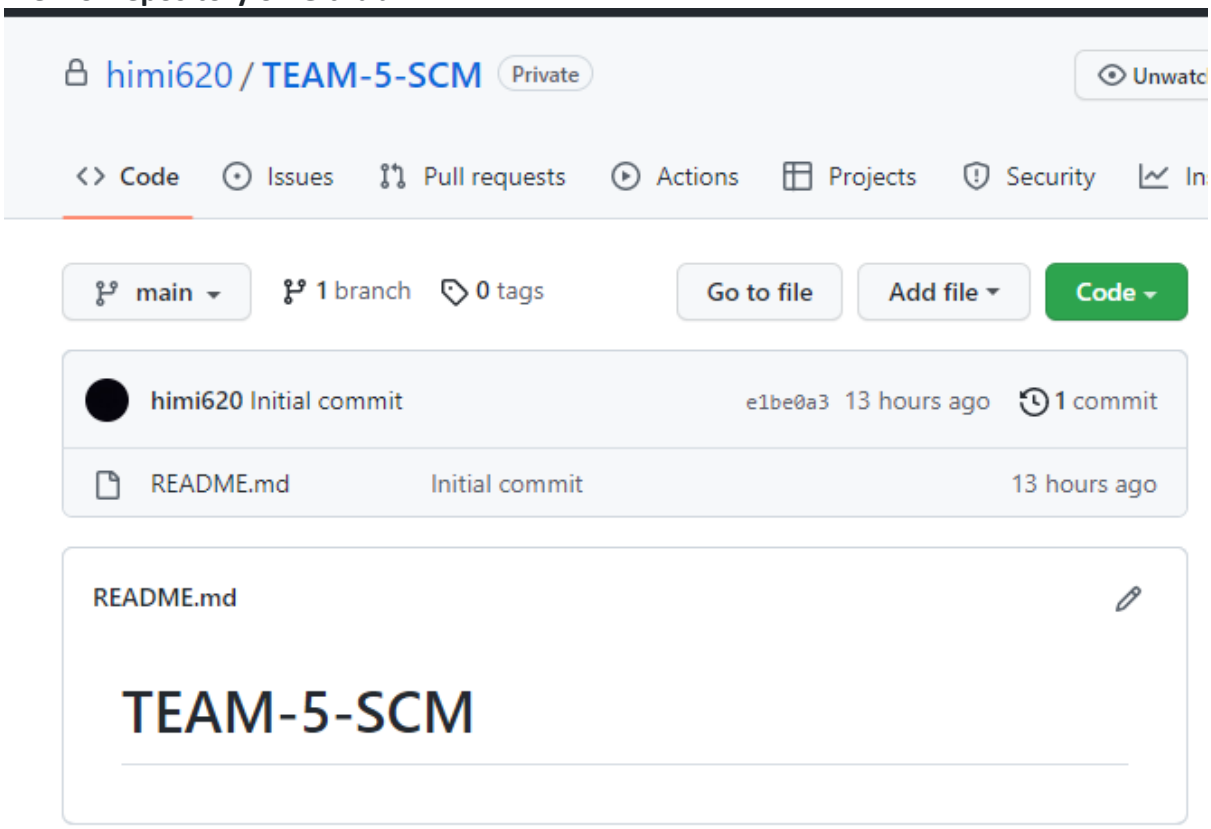
I make a new repository

himi620 / TEAM-5-SCM Private

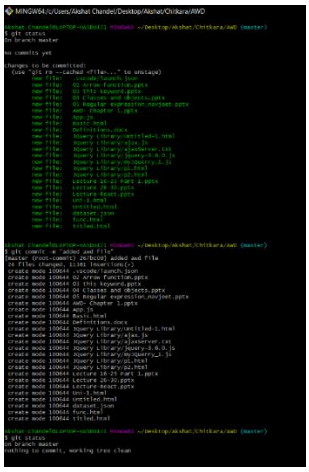
Armaan pushed it to cloud



View of repository on Github



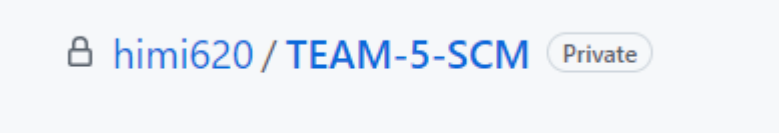
Armaan added more files



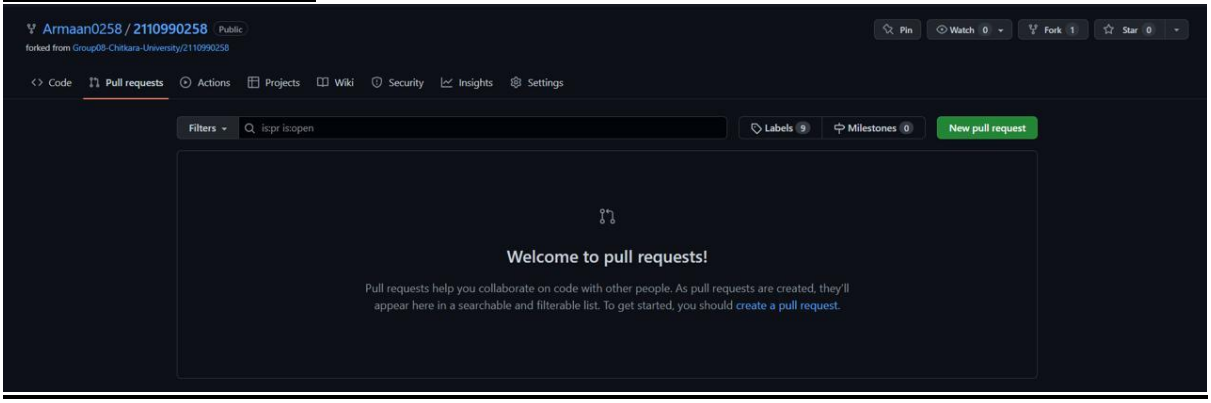
git log after forking and collaborating

```
DELL@DESKTOP-OLO2QO5 MINGW64 /c/himi620 (master)
$ git commit -a -m "some changes"
[master 1fb1501] some changes
 1 file changed, 2 insertions(+), 1 deletion(-)

DELL@DESKTOP-OLO2QO5 MINGW64 /c/himi620 (master)
$ git status
On branch master
nothing to commit, working tree clean

DELL@DESKTOP-OLO2QO5 MINGW64 /c/himi620 (master)
$ git log
commit 1fb15016716a1b369f8dd6d74d3c963e08837216 (HEAD -> master
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:54:34 2022 +0530

    some changes

commit 1ed483450dde05f901c637662d7113f04679ded6
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:49:12 2022 +0530

    newfile

commit 20a41ac2a79773df15aad78d0af15e4b23a8f343
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:32:53 2022 +0530

     modified this file

commit cdfbf538f52f22247247919201da97e454436422
Author: himi620 <himi50071@gmail.com>
Date:   Thu Jun 2 10:23:18 2022 +0530

     commit this file

DELL@DESKTOP-OLO2QO5 MINGW64 /c/himi620 (master)
$
```

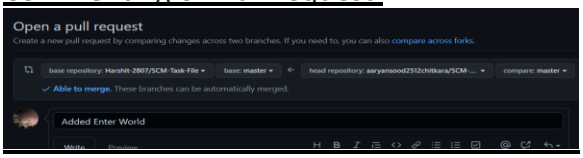
Codes Repo forked from Phoenixbolt:



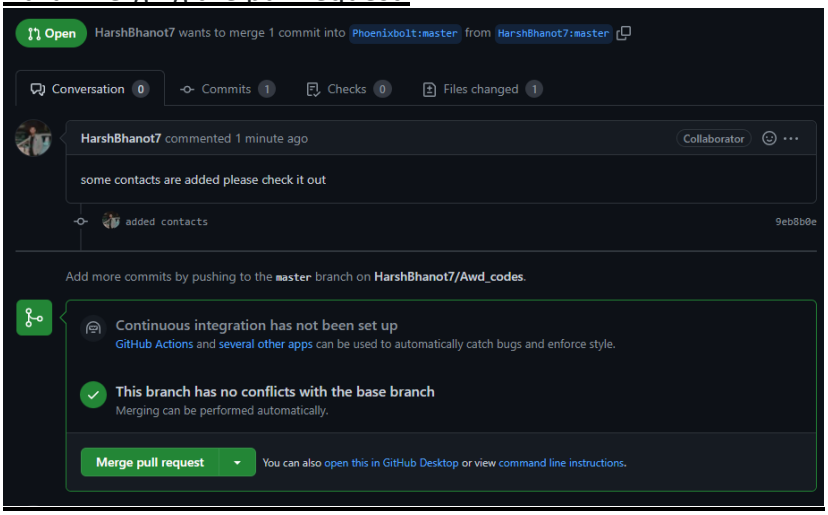
Pull request by Armaan:



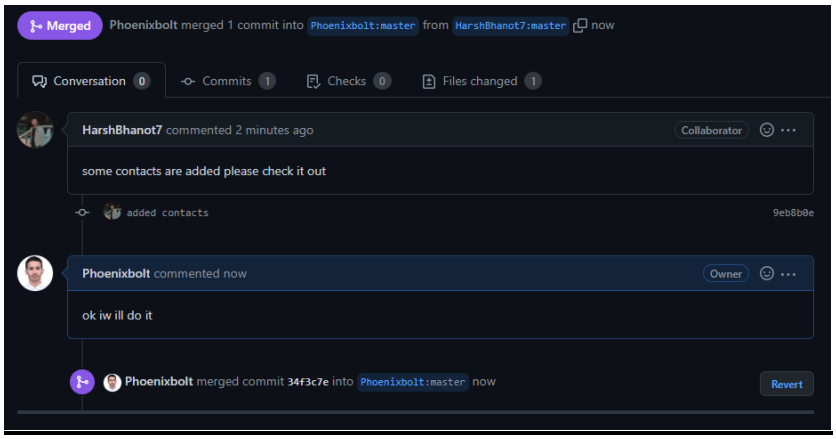
Commenting on Pull request:



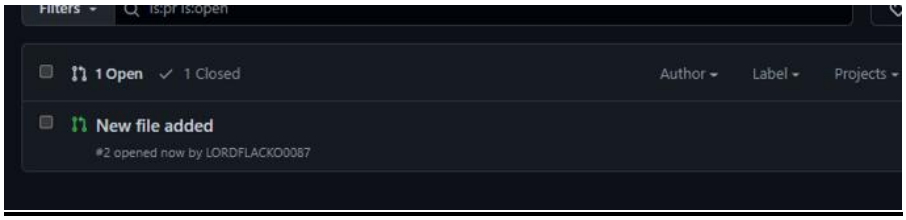
Harul merging the pull request:



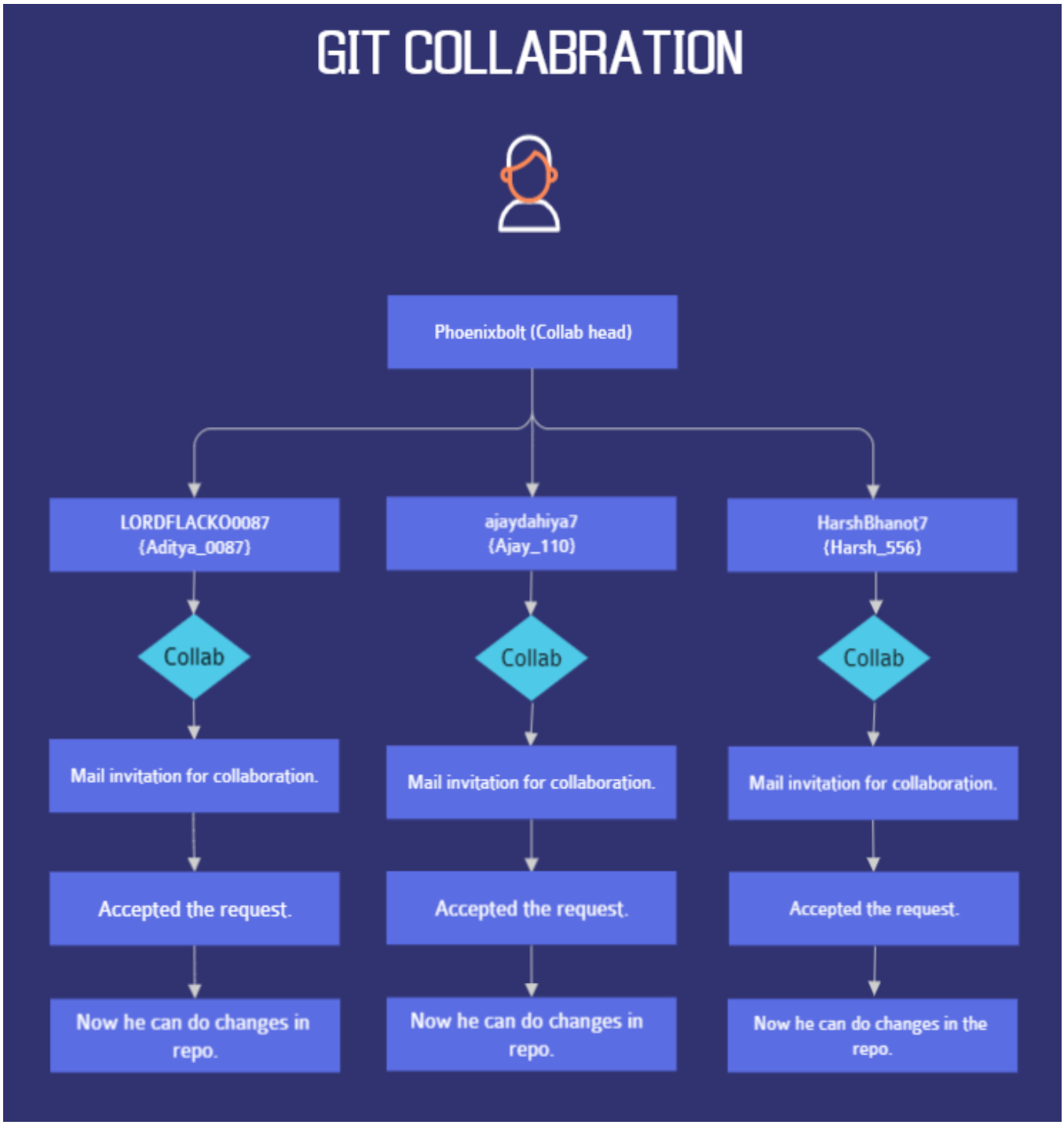
Pull request Merged requested by Harshit :



Creating a pull request to merge the Changes in forked Repo

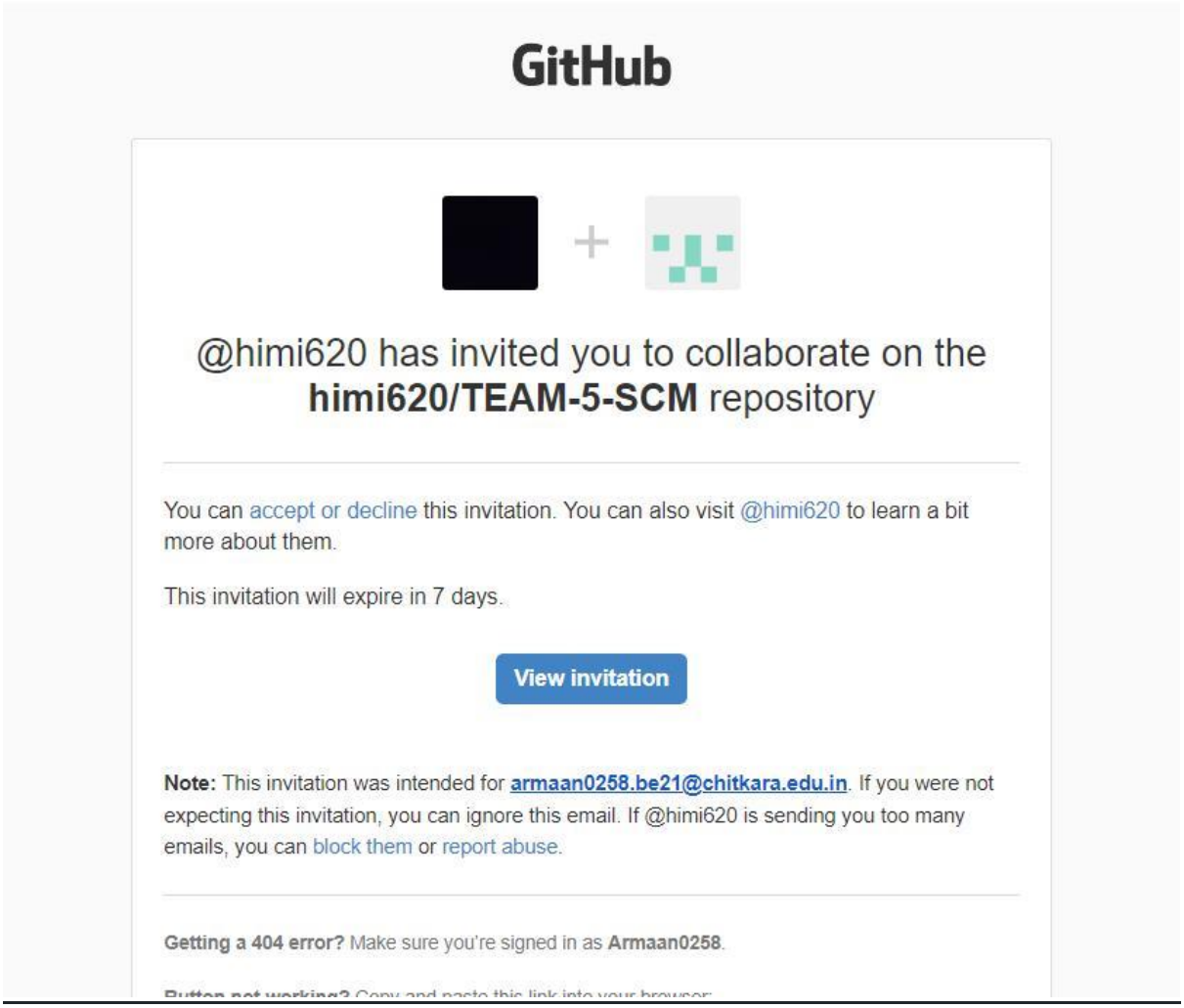


Collaboration



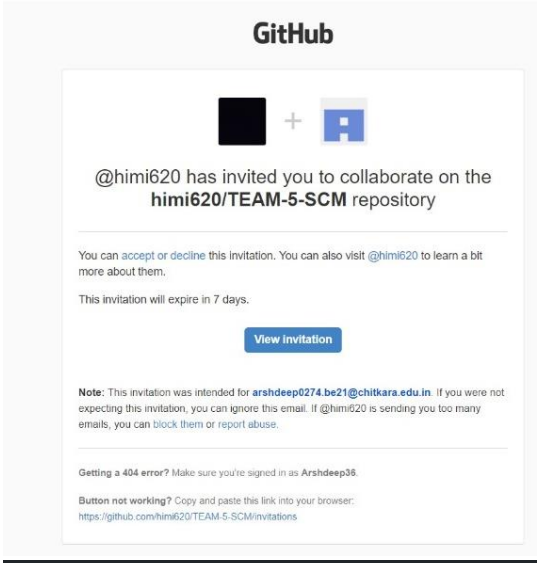
Collabing with team members

This is the invitation mail sent to armaan to add collaboration in Team-Project-File



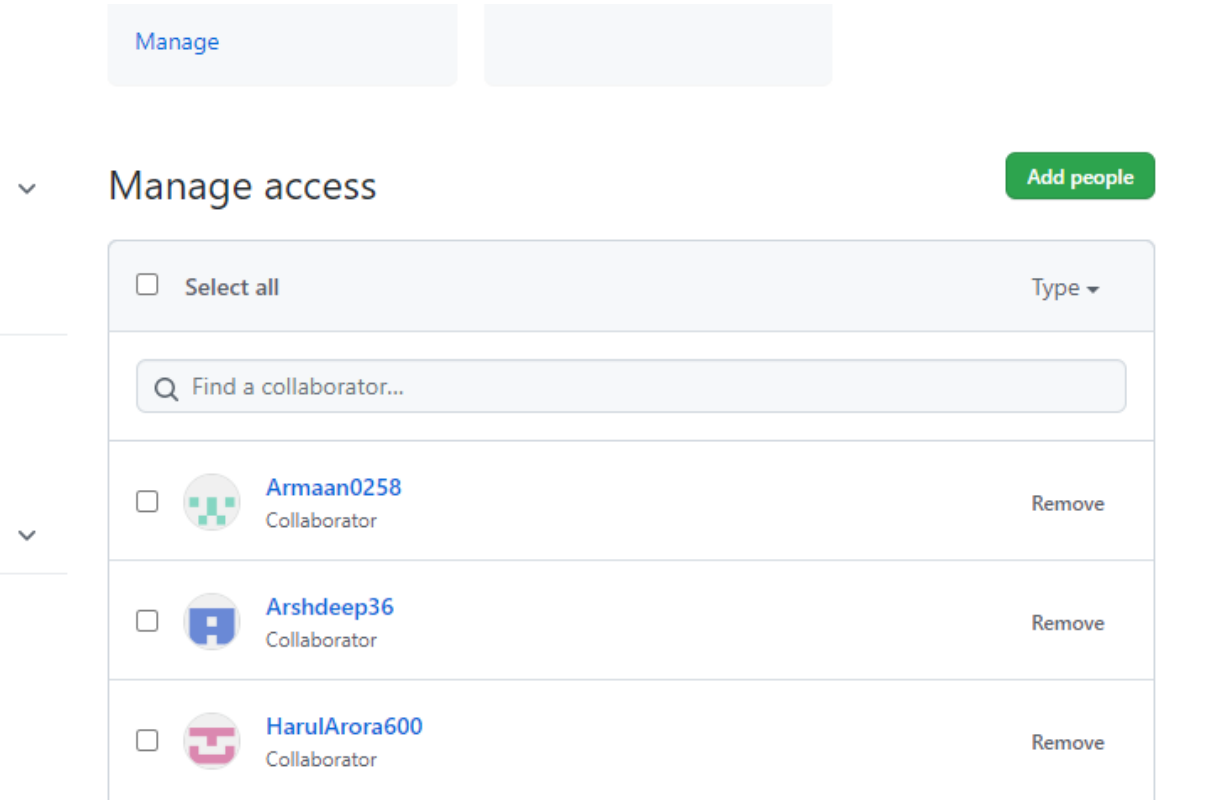
collab with arshdeep-

This is the invitation mail sent to Ajay to add collaboration in Team-Project-File



collab with Harul-

This is the invitation mail sent to add collaboration in Team-Project-File



6. Reference

Reference for few code snips of CSS and HTML were taken from Coding Ninjas video lectures and from few crash courses available on YouTube.
Reference for few code snippets of JavaScript were taken from the Udemy Courses available online.

Links used for reference

- https://www.coursera.org/learn/introduction-git-github
- https://www.freecodecamp.org/news/git-and-github-crash-course/
- https://www.udemy.com/course/github-ultimate/
- https://www.udemy.com/course/git-started-with-github/?LSNPUBID=JVFxdTr9V80&ranEAID=JVFxdTr9V80&ranMID=39197&ranSiteID=JVFxdTr9V80-GmotHk.p_rwC77qokoBs_w&utm_medium=udemyads&utm_source=aff-campaign