CHITKARA UNIVERSITY

Subject Name: Source Code Management(SCM)

Subject Code : CS181

Cluster : Beta

Department : CSE

Submitted By:	**Submitted To:**
Name: Bhagya Babutta	**Dr. Monit Kapoor**
Roll No.: 2110990355	Dean(Beta)

INDEX

Task 2	Create Distributed Repository and add members Open and close a pull request Each members open pull request and closes in self repository Publish and print network graphs	

What is Git?

- Git is the most popular, open-source, widely used, and an example of distributed version control system (DVCS) used for handling the development of small and large projects in a more efficient and neat manner.
- It is most suitable when there are multiple people working on projects as a team and is used for tracking the project changes and efficiently supports the collaboration of the development process.
- With the help of the versioning system, the developer can identify who has made what changes and then run tests and fix bugs if any and then do necessary feature implementation. In case of any unforeseen circumstances, the code can be reverted to any of the previously working versions thereby saving huge efforts.

What is a version control system?

A VCS keeps track of the contributions of the developers working as a team on the projects. They maintain the history of code changes done and with project evolution, it gives an upper hand to the developers to introduce new code, fixes bugs, and run tests with confidence that their previously working copy could be restored at any moment in case things go wrong.

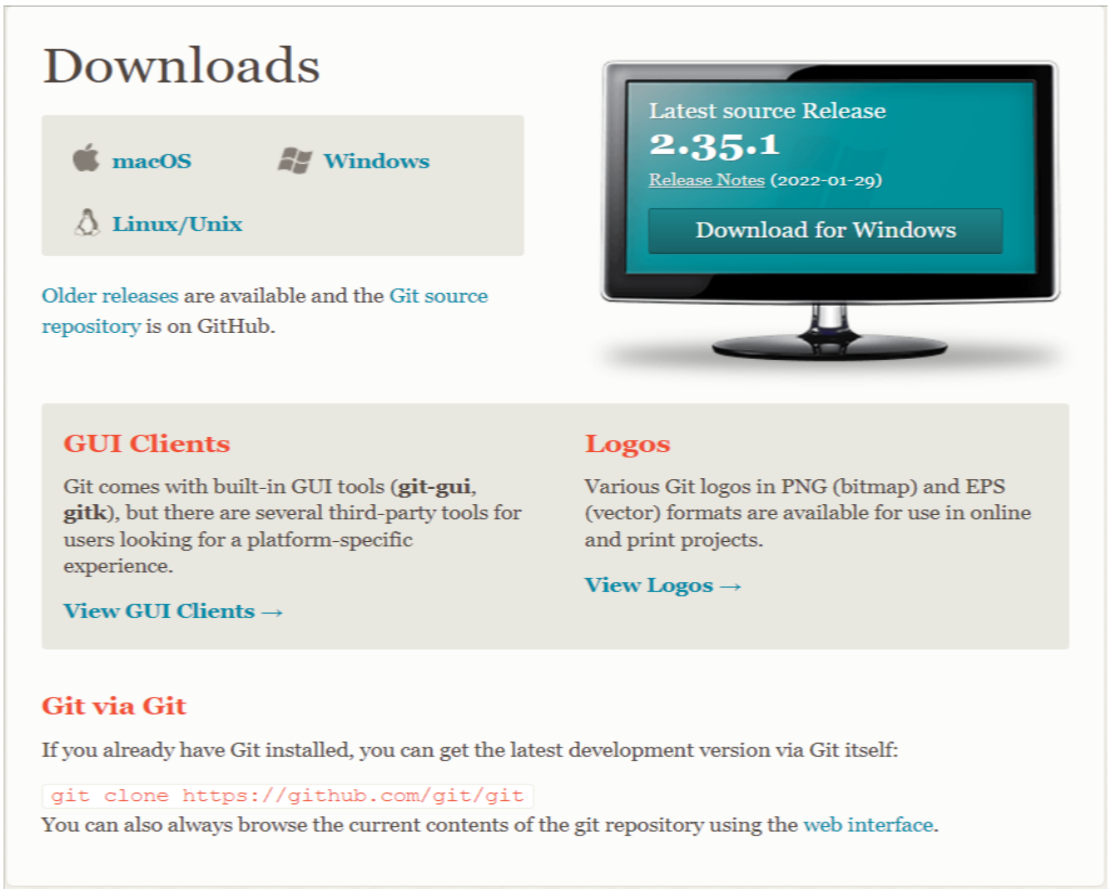
What is a Git Repository?

A repository is a file structure where git stores all the project-based files. Git can either stores the files on the local or the remote repository.

Experiment No. 01

Aim: Setting up the git client.

Git Installation: Download the Git installation program (Windows, Mac, or Linux) from [Git - Downloads (git-scm.com)](git-scm.com).



Downloads

macOS Windows
Linux/Unix

Older releases are available and the Git source repository is on GitHub.

Latest source Release
2.35.1
Release Notes (2022-01-29)
Download for Windows

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

View GUI Clients →

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

View Logos →

Git via Git

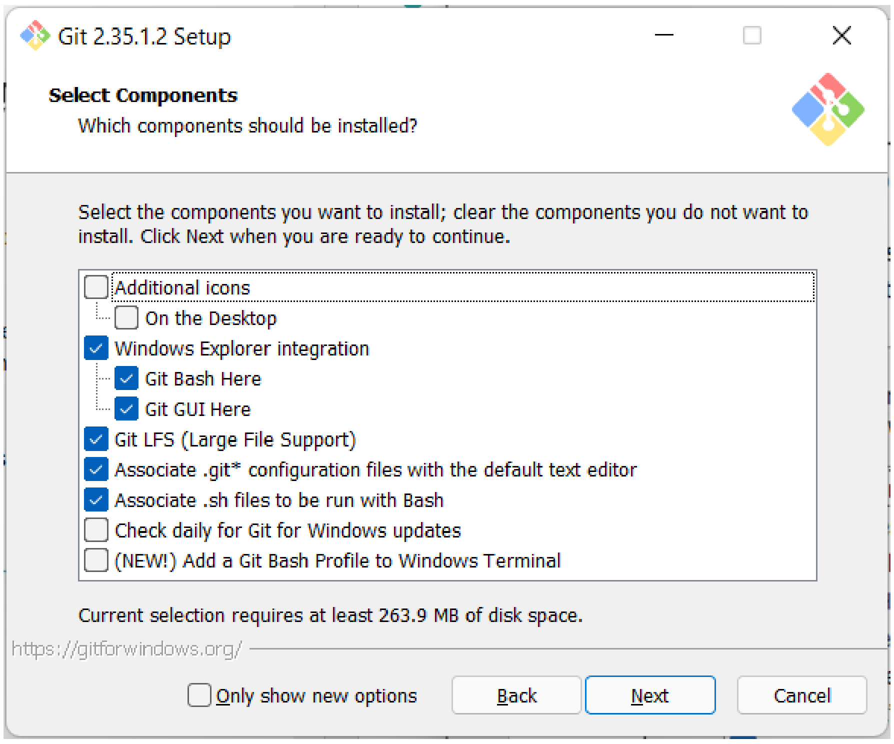
If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the web interface.

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 2.35.1.2 Setup

Select Components

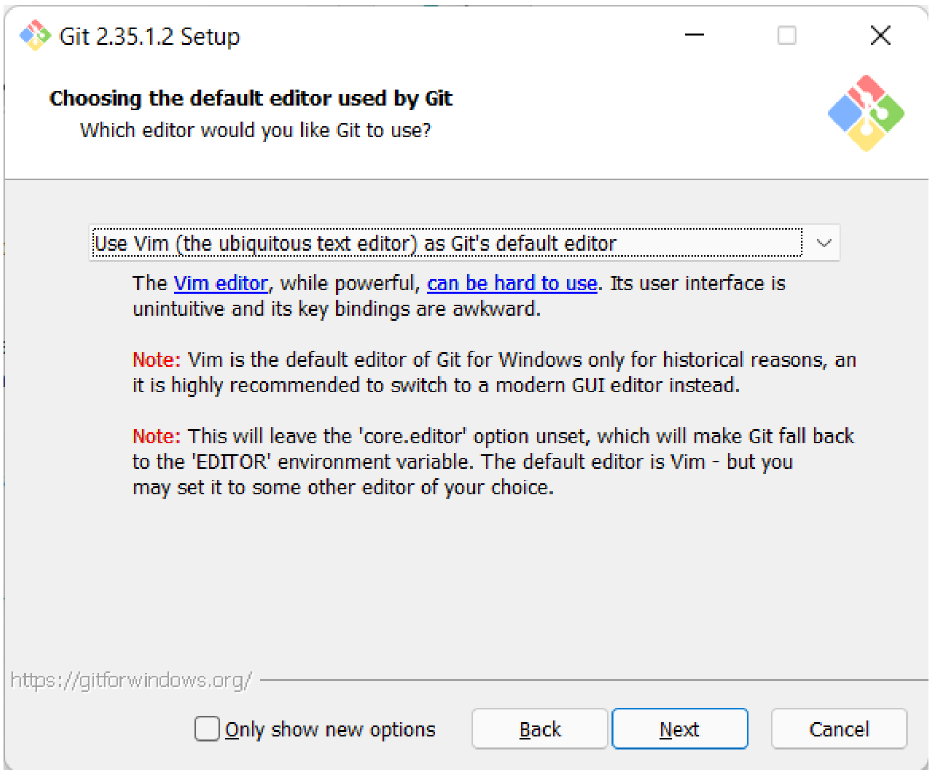
Which components should be installed?

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
- [x] Windows Explorer integration
  - [x] Git Bash Here
  - [x] Git GUI Here
- [x] Git LFS (Large File Support)
- [x] Associate .git* configuration files with the default text editor
- [x] Associate .sh files to be run with Bash
- [ ] Check daily for Git for Windows updates
- [ ] (NEW!) Add a Git Bash Profile to Windows Terminal

Current selection requires at least 263.9 MB of disk space.

In choosing the default editor used by Git dialog, it is recommended that you select default VIM editor- although there are better modern editors available. If you want you can also choose notepad++ or visual studio it depends on user's prferences.


Git 2.35.1.2 Setup

Choosing the default editor used by Git

Which editor would you like Git to use?

Use Vim (the ubiquitous text editor) as Git's default editor

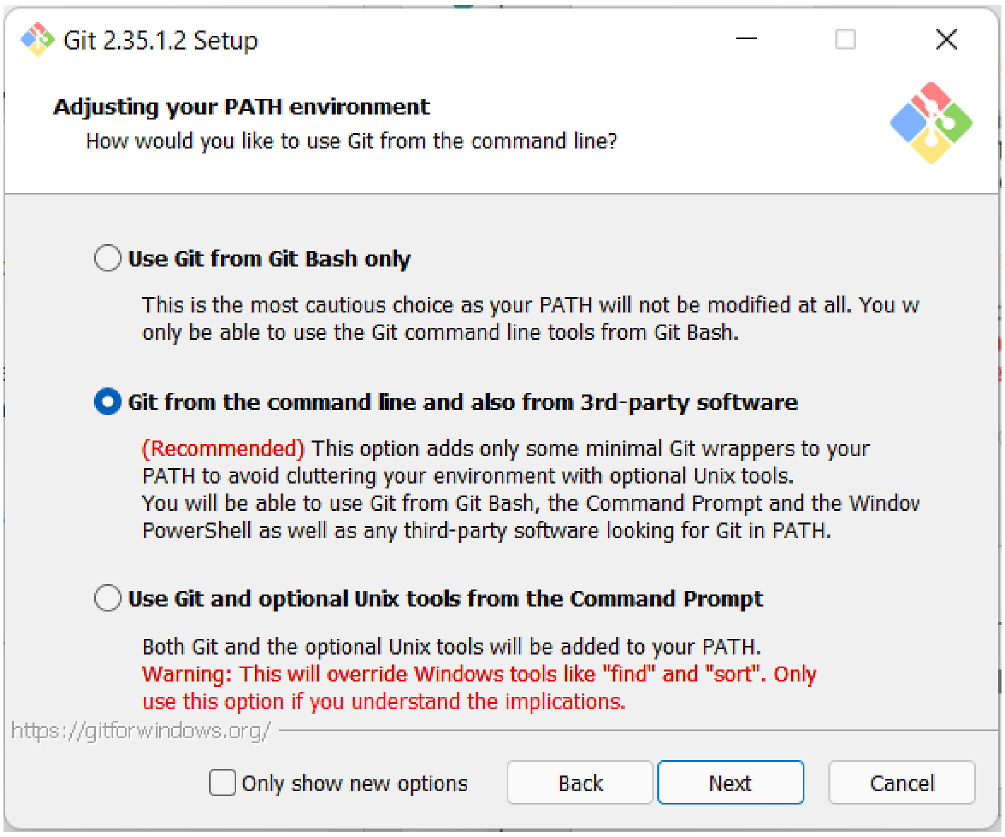
The Vim editor, while powerful, can be hard to use. Its user interface is unintuitive and its key bindings are awkward.

Note: Vim is the default editor of Git for Windows only for historical reasons, an it is highly recommended to switch to a modern GUI editor instead.

Note: This will leave the 'core.editor' option unset, which will make Git fall back to the 'EDITOR' environment variable. The default editor is Vim - but you may set it to some other editor of your choice.

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

1. Use Git from Git Bash only: no integration, and no extra command in your command path.
2. Use Git from the windows Command Prompt: add 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.
3. 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 like commands like grep.


Git 2.35.1.2 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

( ) **Use Git from Git Bash only**

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

[✓ **Git from the command line and also from 3rd-party software**]

(Recommended) This option adds only some minimal Git wrappers to your
PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windov
PowerShell as well as any third-party software looking for Git in PATH.

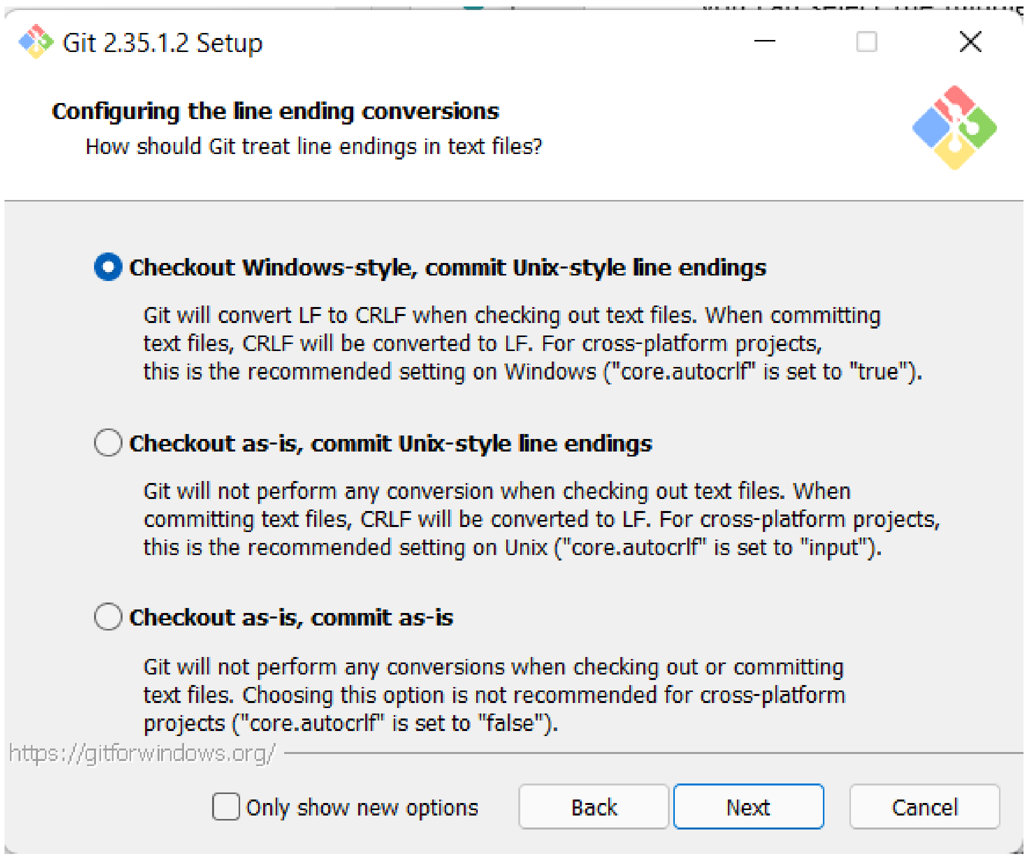
( ) **Use Git and optional Unix tools from the Command Prompt**

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/

☐ Only show new options

In the Configuring the line ending screen, by default first option will be selected but if you want you can select the middle option (Checkout-as-is, commit Unix-style line endings). 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.



Git 2.35.1.2 Setup

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").

After this just click on next until install option comes and then press install.

Configuring Git to ignore certain files:

This part is extra important and required so that your repository does not get cluttered with garbage files.

By default, Git tracks all files in a project. Typically, this is not what you want; rather, you want Git to ignore certain files such as .bak files created by an editor or .class files created by the Java compiler. To have Git automatically ignore particular files, create a file named .gitignore (note that the filename begins with a dot) in the C:\users\name folder (where name is your MSOE login name).

NOTE: The .gitignore file must NOT have any file extension (e.g. .txt). Windows normally tries to place a file extension (.txt) on a file you create from File Explorer - and then it (by default) HIDES the file extension. To avoid this, create the file from within a useful editor (e.g. Notepad++ or UltraEdit) and save the file without a file extension).

Edit this file and add the lines below (just copy/paste them from this screen); these are patterns for files to be ignored (taken from examples provided at https://github.com/github/gitignore.)

```
  #Lines (like this one) that begin with # are comments; all other lines are rules


  # common build products to be ignored at MSOE
  *.o
  *.obj
  *.class *.exe


  # common IDE-generated files and folders to
  ignore workspace.xml bin / out
  /
   .classpath
  # uncomment following for courses in which Eclipse .project files are not
checked in # .project
  #ignore automatically generated files created by some common applications, operating
  systems
  *.bak
  *.log
  *.ldb
  ~*
   .DS_Store*
   ._*
  Thumbs.d
  b


  # Any files you do not want to ignore must be specified starting with ! # For example,
  if you didn't want to ignore .classpath, you'd uncomment the following rule: #
  !.classpath
```

Note: You can always edit this file and add additional patterns for other types of files you might want to ignore. Note that you can also have a

.gitignore files in any folder naming additional files to ignore. This is useful for projectspecific build products.

Once Git is installed, there is some remaining custom configuration you must do. Follow the steps below:

a. From within File Explorer, right-click on any folder. A context menu appears containing the commands " Git Bash here" and "Git GUI here". These commands permit you to launch either Git client. For now, select Git Bash here.

b. Enter the command `git config --global user.Email "name@msoe.edu"`

This links your Git activity to your email address. Without this, your commits will often show up as "unknown login". Replace name with your own MSOE email name.

c. Enter the command `git config --global user.name "Your Name"`

Git uses this to log your activity. Replace "Your Name" by your actual first and last name.

d. Enter the command `git config --global push.default simple`

This ensures that all pushes go back to the branch from which they were pulled. Otherwise pushes will go to the master branch, forcing a merge.

Experiment No. 02

Aim

Setting up GitHub Account

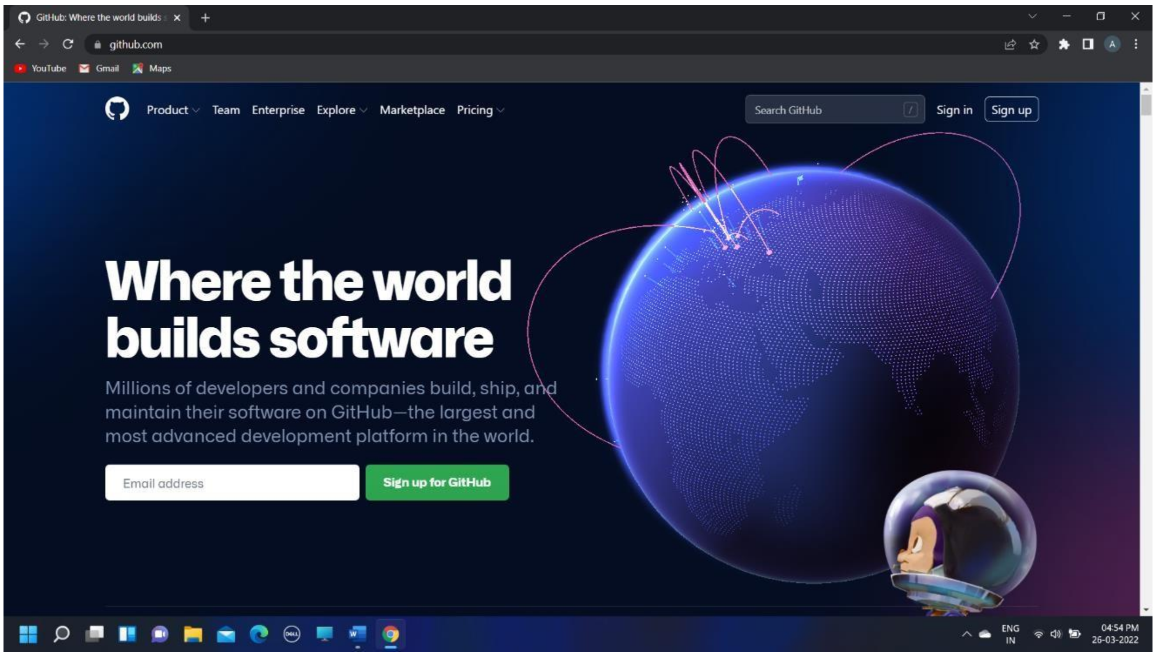
The first steps in starting with GitHub are to create an account, choose a product that fits your needs best, verify your email, set up two-factor authentication, and view your profile.

There are several types of accounts on GitHub. Every person who uses GitHub has their own user account, which can be part of multiple organisations 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".

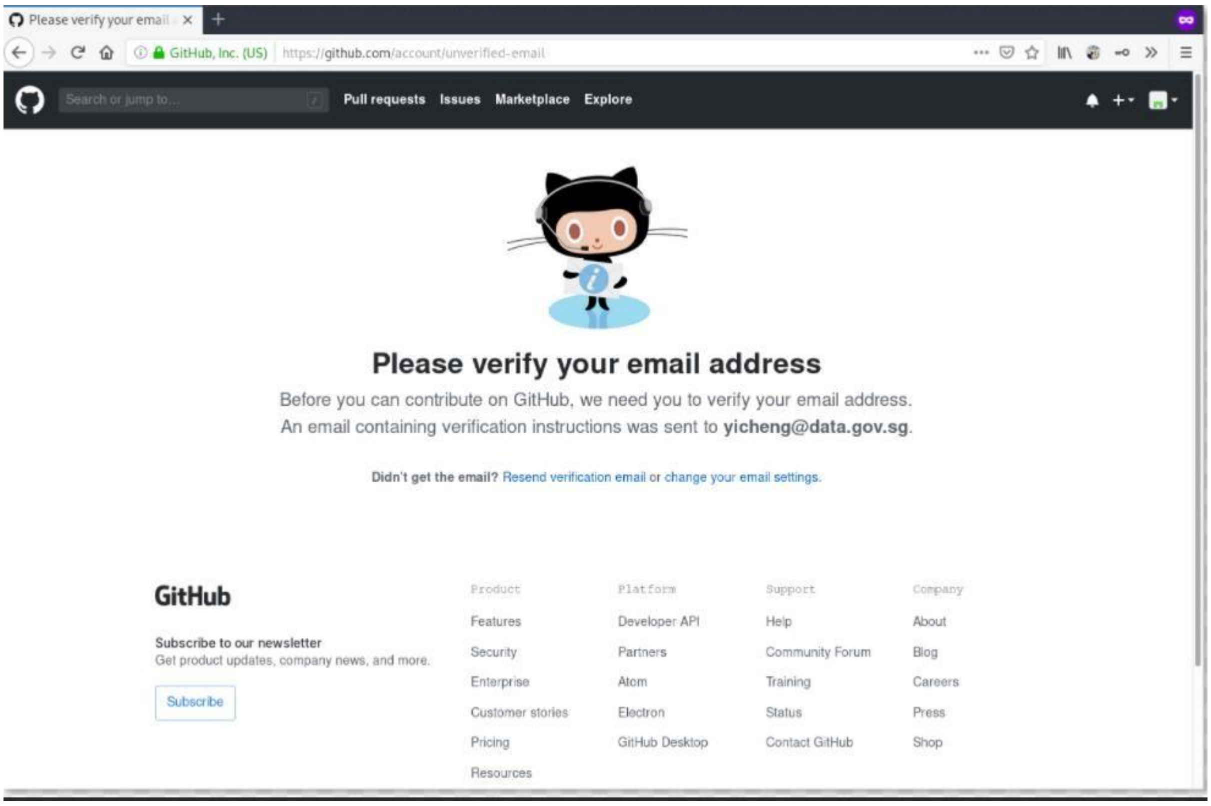


github.com
Product
Team
Enterprise
Explore
Marketplace
Pricing
Search GitHub
Sign in
Sign up
Where the world builds software
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.
Email address
Sign up for GitHub

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 GitHub's plans, see "GitHub's products".

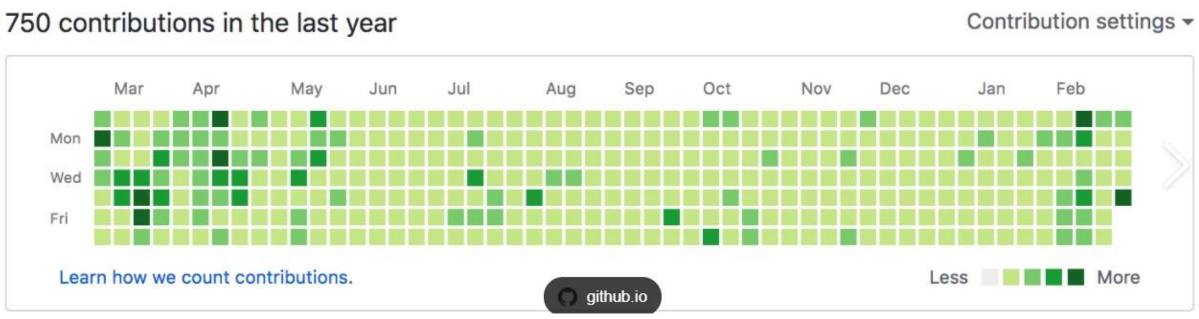
3. **Verifying your email address:** To ensure you can use all the features in your GitHub plan, verify your email address after signing up for a new account. For more information, see "Verifying your email address".


Please verify your email address

Before you can contribute on GitHub, we need you to verify your email address.
An email containing verification instructions was sent to **yicheng@data.gov.sg**.

Didn't get the email? Resend verification email or change your email settings.

4. **Viewing your GitHub profile and contribution graph:** Your GitHub profile tells people the story of your work through the repositories and gists you've pinned, the organisation 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."



Experiment No. 03

Aim: Program to generate logs

Basic Git init

$ git init command creates a new Git repository. It can be used to convert an existing, undersigned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialize repository, so this is usually the first command you'll run in a new project.

Basic Git status

$ git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does not show you any information regarding the committed project history.

Basic Git commit

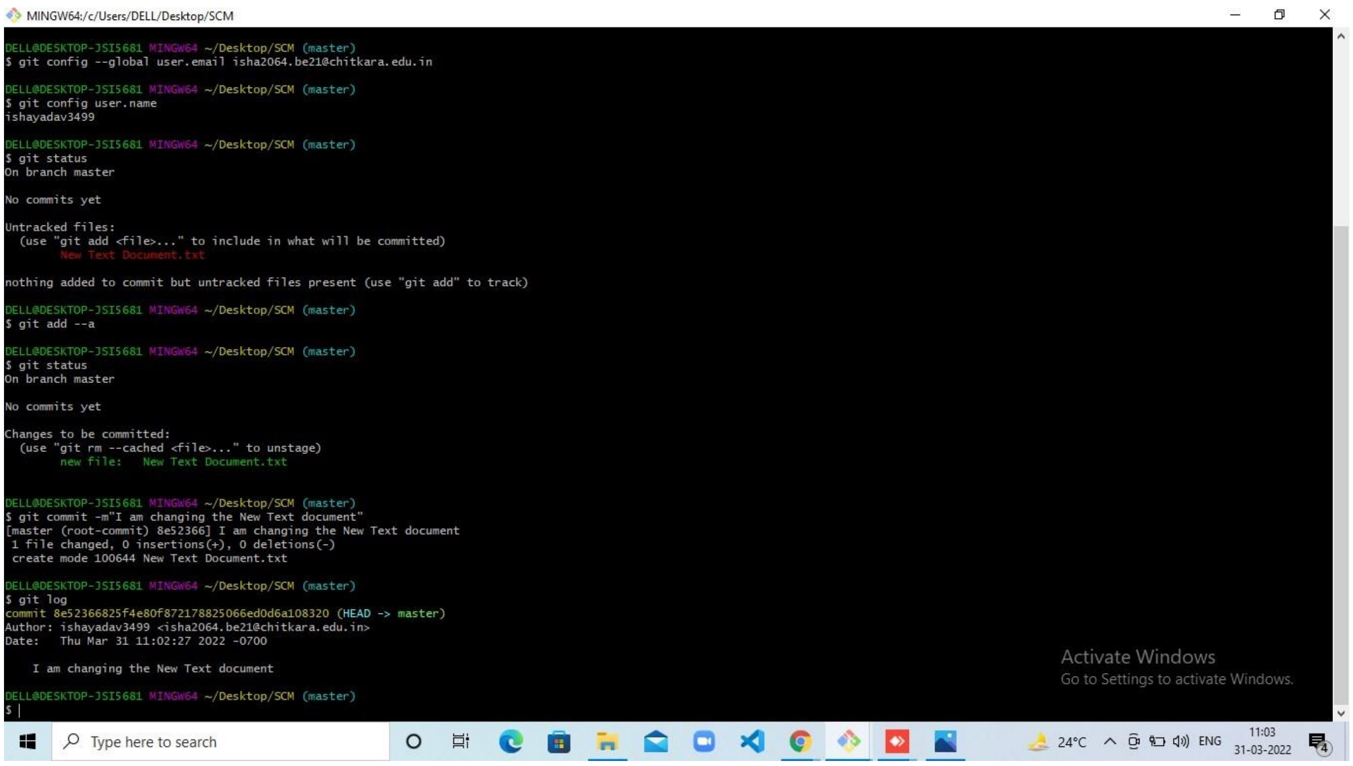
$ git commit command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as "safe" versions of a project—Git will never change them unless you explicitly ask it to. Prior to the execution of git commit, The git add command is used to promote or 'stage' changes to the project that will be stored in a commit. These two commands git commit and git add are two of the most frequently used

Basic Git add command

$ git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit

Basic Git log

$ git log command is one of the most usual commands of git. It is the most useful command for Git. Every time you need to check the history, you have to use the git log command. The basic git log command will display the most recent commits and the status of the head. It will use as:


MINGW64:/c/Users/DELL/Desktop/SCM

```
DELL@DESKTOP-JSI5681 MINGW64 ~/Desktop/SCM (master)
$ git config --global user.email isha2064.be21@chitkara.edu.in

DELL@DESKTOP-JSI5681 MINGW64 ~/Desktop/SCM (master)
$ git config user.name
ishayadav3499

DELL@DESKTOP-JSI5681 MINGW64 ~/Desktop/SCM (master)
$ git status
On branch master

No commits yet

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

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

DELL@DESKTOP-JSI5681 MINGW64 ~/Desktop/SCM (master)
$ git add --a

DELL@DESKTOP-JSI5681 MINGW64 ~/Desktop/SCM (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   New Text Document.txt


DELL@DESKTOP-JSI5681 MINGW64 ~/Desktop/SCM (master)
$ git commit -m"I am changing the New Text document"
[master (root-commit) 8e52366] I am changing the New Text document
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 New Text Document.txt

DELL@DESKTOP-JSI5681 MINGW64 ~/Desktop/SCM (master)
$ git log
commit 8e52366825f4e80f872178825066ed0d6a108320 (HEAD -> master)
Author: ishayadav3499 <isha2064.be21@chitkara.edu.in>
Date:   Thu Mar 31 11:02:27 2022 -0700

    I am changing the New Text document

DELL@DESKTOP-JSI5681 MINGW64 ~/Desktop/SCM (master)
$ |
```

Experiment No. 04

Aim: Create and visualize branches in Git

How to create branches?

The main branch in git is called 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 files which are created in branch are not shown in master branch. We also can merge both the parent(master) and child (other branches).

1. For creating a new branch: git branch “name of branch”
2. To check how many branches we have : git branch
3. To change the present working branch: git checkout “name of the branch”

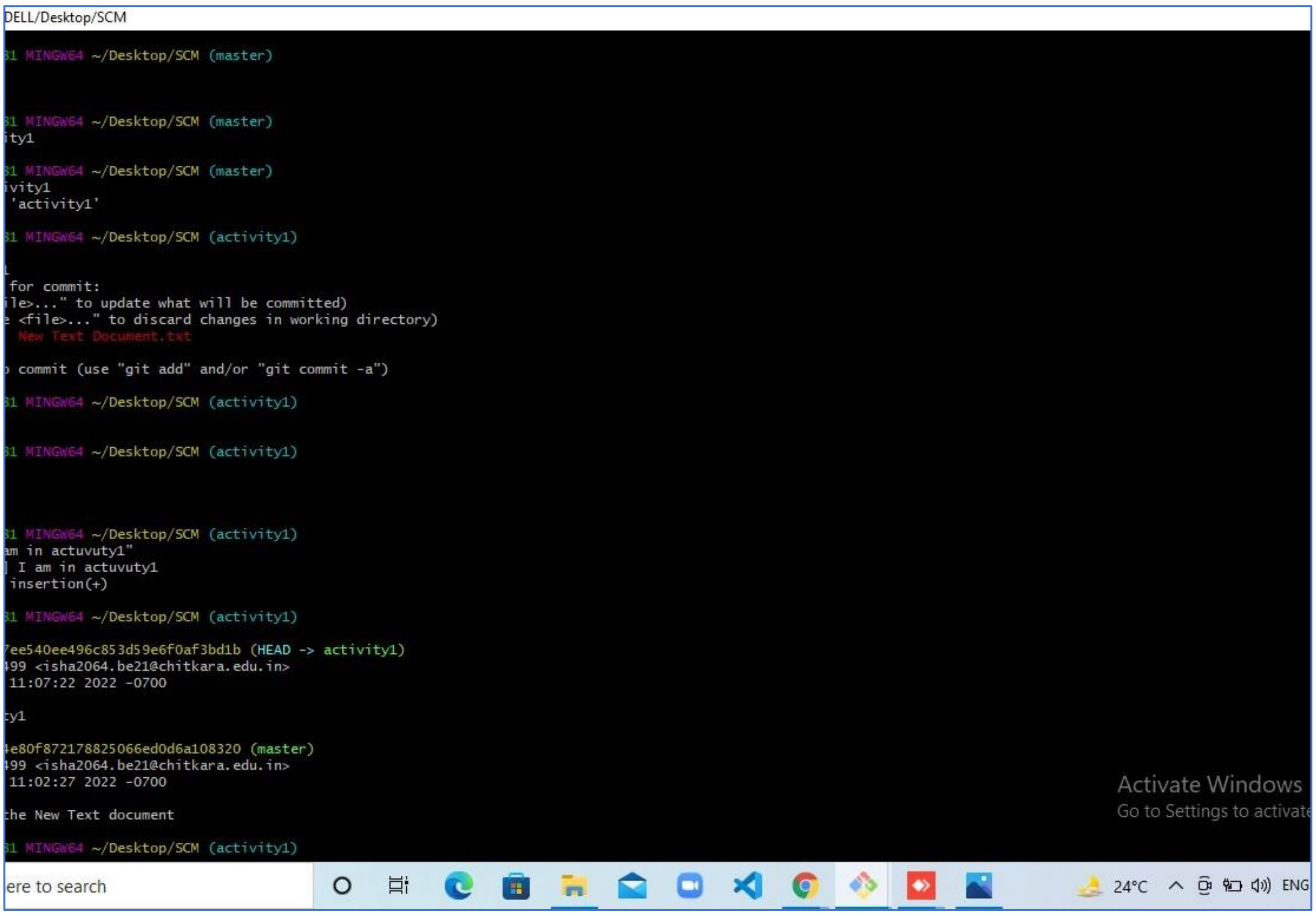
Visualizing Branches:

To visualize, we have to create a new file in the new branch “activity1” instead of the master branch. After this we have to do three step architecture i.e. working directory, staging area and git repository.

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

After this we will change the branch from activity1 to master, but when we switch back to master branch the file we created i.e “hello” will not be there. Hence the new file will not be shown in the master branch. In this way we can create and change different branches. We can also merge the branches by using the git merge command.

In this way we can create and change different branches. We can also merge the branches by using git merge command.



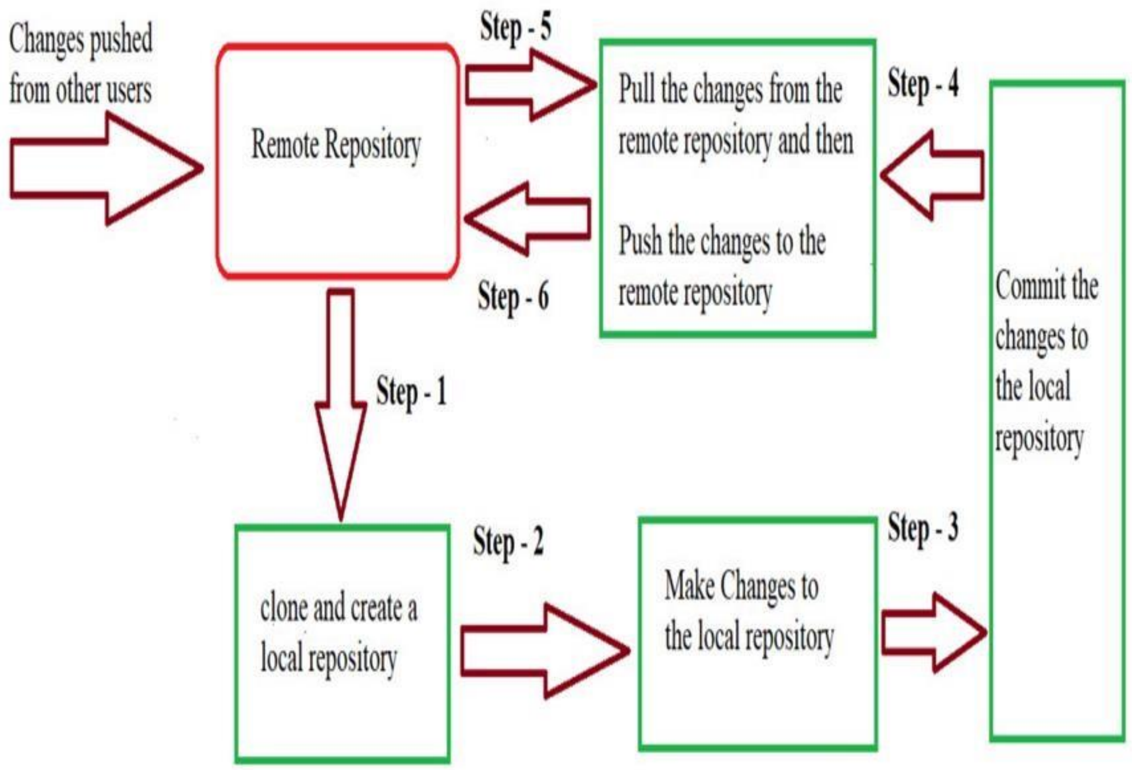
DELL/Desktop/SCM

```
31 MINGW64 ~/Desktop/SCM (master)


31 MINGW64 ~/Desktop/SCM (master)
ity1

31 MINGW64 ~/Desktop/SCM (master)
ivity1
 'activity1'

31 MINGW64 ~/Desktop/SCM (activity1)

l
 for commit:
ile>..." to update what will be committed)
e <file>..." to discard changes in working directory)
  New Text Document.txt

o commit (use "git add" and/or "git commit -a")

31 MINGW64 ~/Desktop/SCM (activity1)


31 MINGW64 ~/Desktop/SCM (activity1)



31 MINGW64 ~/Desktop/SCM (activity1)
am in actuvuty1"
] I am in actuvuty1
 insertion(+)

31 MINGW64 ~/Desktop/SCM (activity1)

7ee540ee496c853d59e6f0af3bd1b (HEAD -> activity1)
499 <isha2064.be21@chitkara.edu.in>
 11:07:22 2022 -0700

ty1

4e80f872178825066ed0d6a108320 (master)
499 <isha2064.be21@chitkara.edu.in>
 11:02:27 2022 -0700

the New Text document

31 MINGW64 ~/Desktop/SCM (activity1)
```

Experiment No. 05

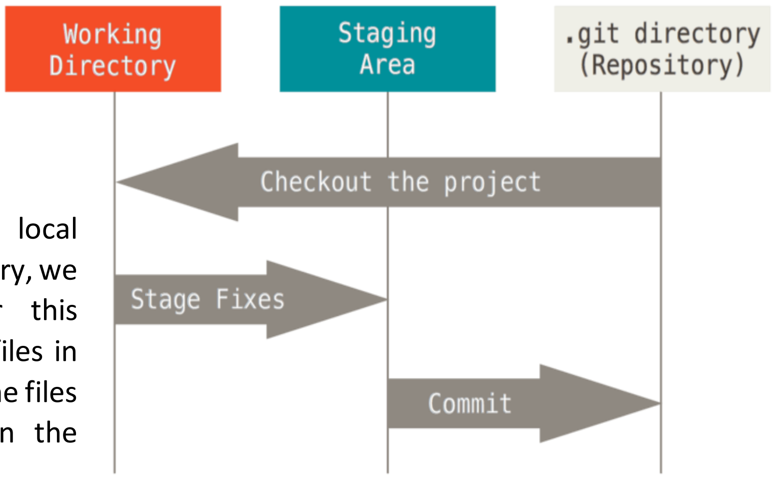
Aim: Git lifecycle description

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 Lifecycle description that git has and understand more about its life cycle. Let us see some of the basic steps that we have to 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 - 4
Step - 6
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

- **Step 1-** We first clone any of the code residing in the remote repository to make our won local repository.
- **Step 2-** We edit the files that we have cloned in our local repository and make the necessary changes in it.
- **Step 3-** We commit our changes by first adding them to our staging area and committing them with a commit message.
- **Step 4 and Step 5-** We first check whether there are any of the changes done in the remote repository by some other users and we first pull that changes.
- **Step 6-** If there are no changes we push our changes to 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-



1. **Working Directory**

   Whenever we want to initialize aur local project directory to make a Git repository, we use the git init command. After this command, git becomes aware of the files in the project although it does not track the files yet. The files are further tracked in the staging area.

2. **Staging Area**

   Now, to track files 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, 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.
   git add<filename>

   git add.

3. **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 aur 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 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 <Message>

Experiment No. 06

Aim: Add collaborators on GitHub Repo

In GitHub, We can invite other GitHub users to become collaborators to our private repositories(which expires after 7 days if not accepted, restoring any unclaimed licenses). Being a collaborator, of a personal repository you can pull (read) the contents of the repository and push (write) changes to the repository. You can add unlimited collaborators on public and private repositories(with some per day limit restrictions). But, in a private repository, the owner of the repo can only grant write-access to the collaborators, and they can't have the read-only access.

GitHub also restricts the number of collaborators we can invite within a period of 24 hours. If we exceed the limit, then either we have to wait for 24-hours or we can also create an organization to collaborate with more people.

Actions that can be Performed By Collaborators

Collaborators can perform a number of actions into someone else's personal repositories, they have gained access to Some of them are,

- Create, merge, and close pull requests in the repository
- Publish, view, install the packages
- Fork the repositories
- Make the changes on the repositories as suggested by the Pull requests.
- Mark issues or pull requests as duplicate
- Create, edit, and delete any comments on commits, pull requests, and issues in the repository
- Removing themselves as collaborators on the repositories.
- Manage releases in the repositories.

Now, let's see how can we invite collaborators to our repositories.

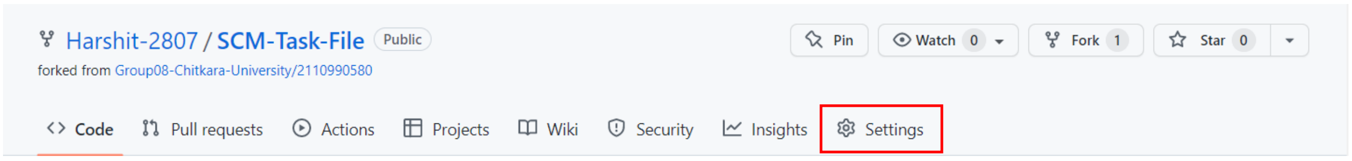
Inviting Collaborators to your personal repositories

Follow the steps below to invite collaborators to your own repository(public or private).

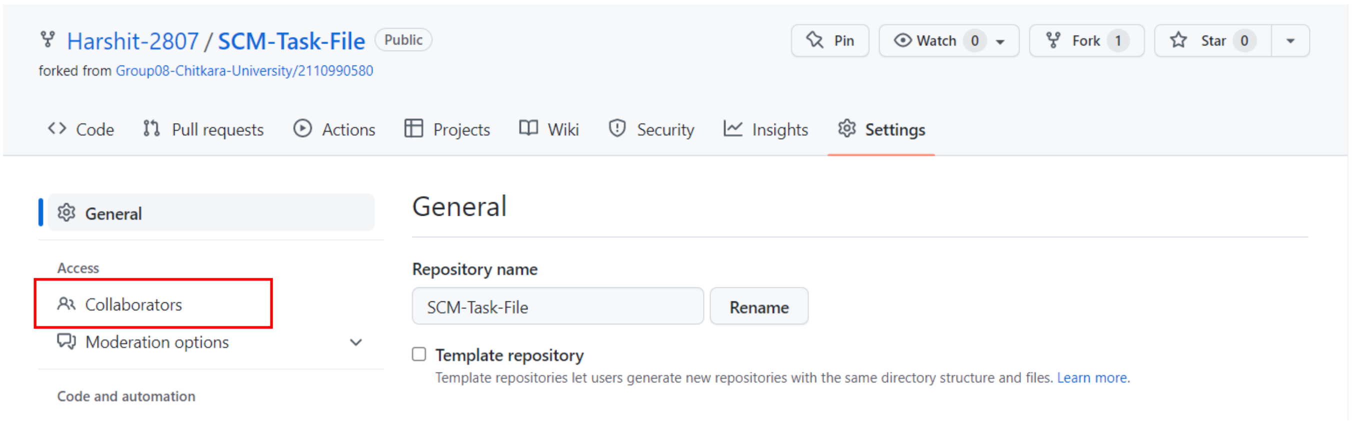
Step 1: Get the usernames of the GitHub users you will be adding as collaborators. In case, they are not on GitHub, ask them to sign in to GitHub.

Step 2: Go to your repository(intended to add collaborators)

Step 3: Click into the Settings.

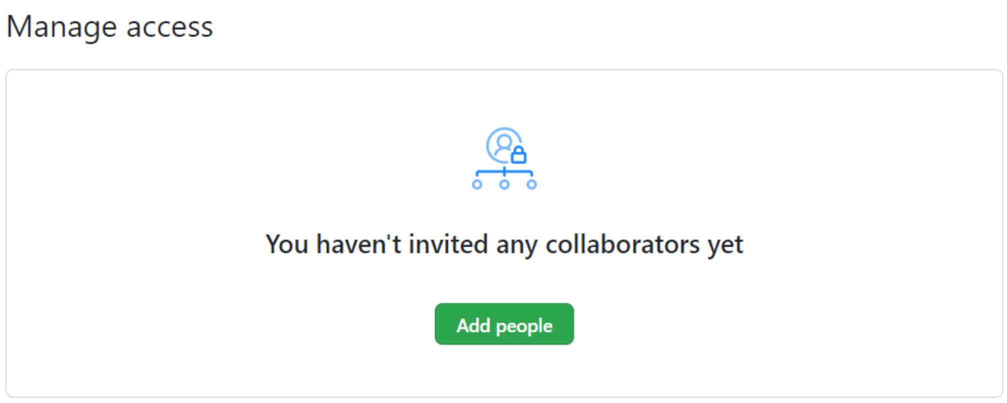


Step 4: A settings page will appear. Here, into the left-sidebar click into the Collaborators.

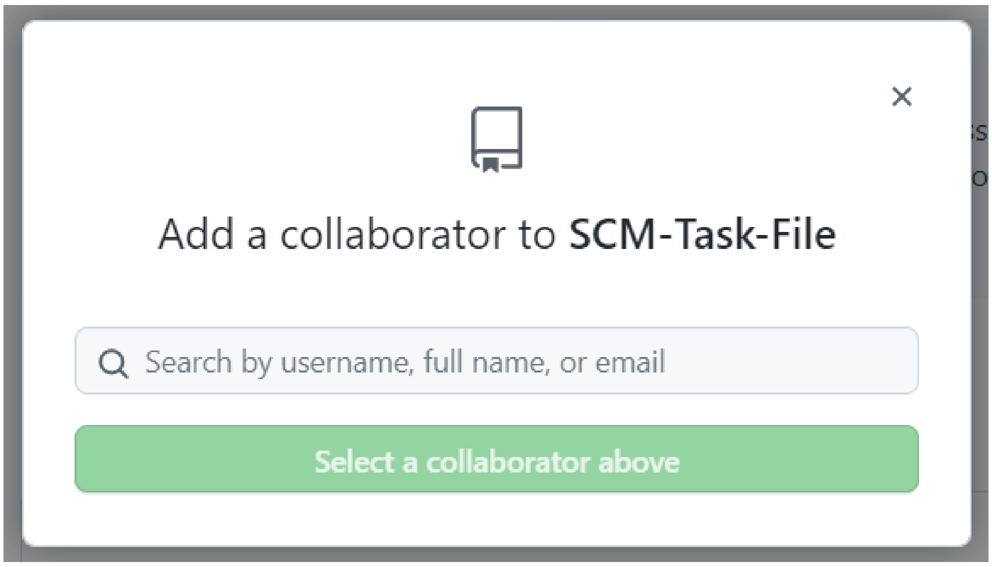


Step 5: Then a confirm password page may appear, enter your password for the confirmation.

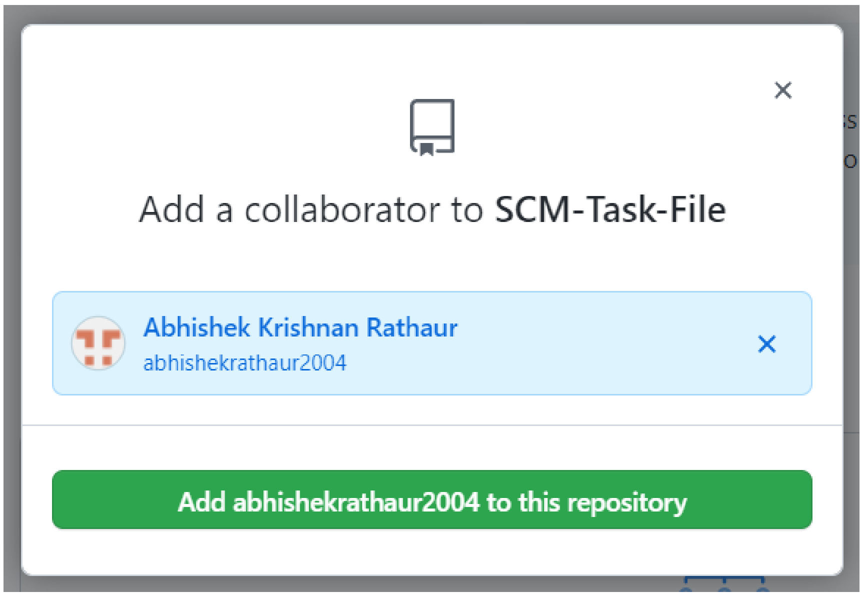
Step 6: Next, click into Add People.



Step 7: Then a search field will appear, where you can enter the username of the ones you want to add as collaborator.


Add a collaborator to SCM-Task-File
Search by username, full name, or email
Select a collaborator above

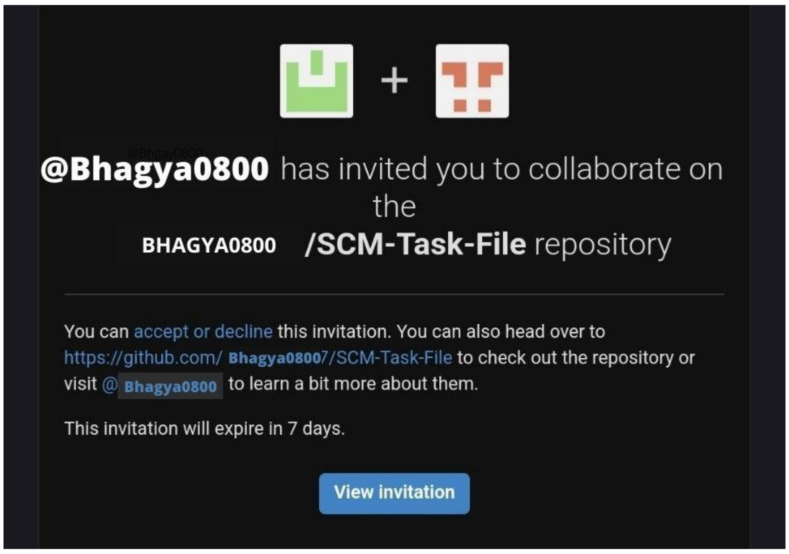
Step 8: After selecting the people, add them as collaborator.


Add a collaborator to **SCM-Task-File**

Abhishek Krishnan Rathaur
abhishekrathaur2004

Add abhishekrathaur2004 to this repository

Step 9: After sending the request for collaboration to that person whom we wanted as our collaborator for our project will get a email from the Team leader (by GitHub). He/she has to open its Email to view the invitation.

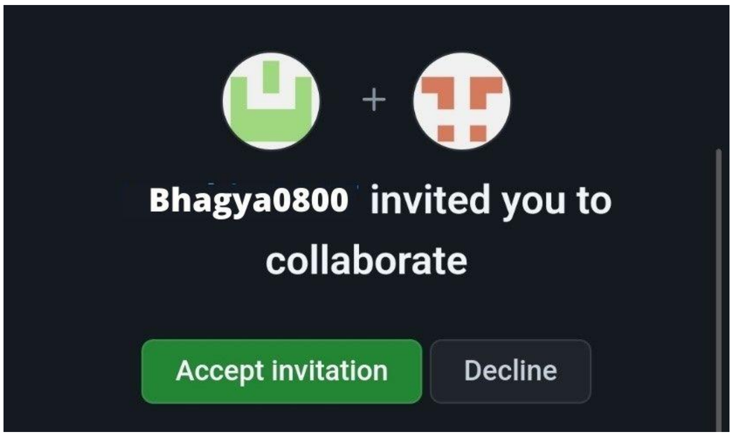

@Bhagya0800 has invited you to collaborate on the
BHAGYA0800 **/SCM-Task-File** repository

You can accept or decline this invitation. You can also head over to https://github.com/ Bhagya0800/SCM-Task-File to check out the repository or visit @ Bhagya0800 to learn a bit more about them.

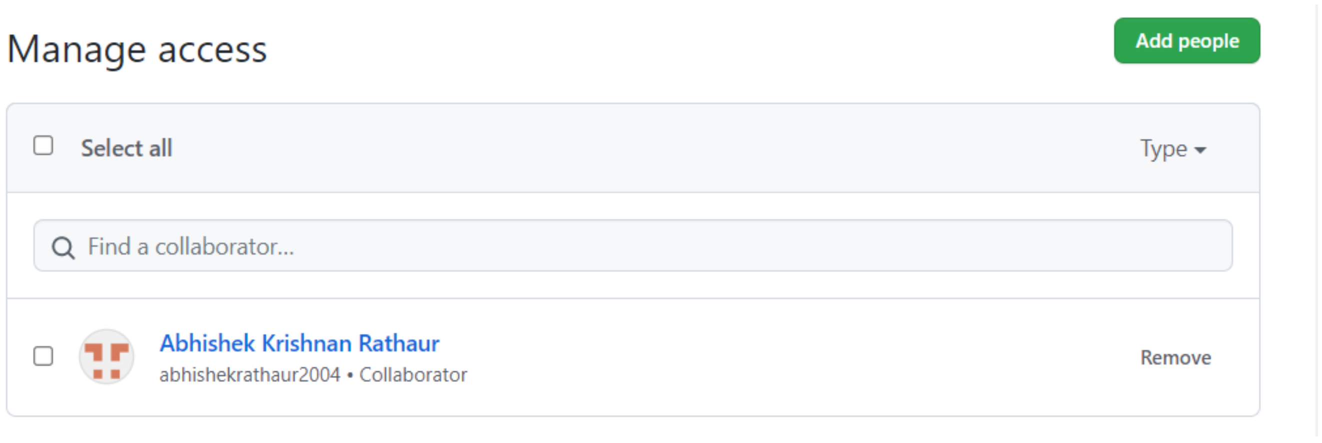
This invitation will expire in 7 days.

View invitation

Step 10: After Clicking the View invitation, He/she will be redirected to the GitHub Page for accepting the invitation sent by the team leader.



Step 11: We are done adding a single collaborator. Now, they will get a mail regarding invitation to your repository. Once they accept, they will have collaborator access to your repository. Till then it will be in pending invitation state. You can also add more collaborator and delete the existing one as depicted below.



Experiment No. 07

Aim: Fork and Commit

Forking a repository means creating a copy of the repo. When you fork a repo, you create your own copy of the repository on your GitHub account. This is done for the following reasons:

1. You have your own copy of the project on which you may test your own changes without changing the original project.
2. This helps the maintainer of the project to better check the changes you made to the project and has the power to either accept, reject or suggest something.
3. When you clone an Open Source project, which isn't yours, you don't have the right to push code directly into the project.

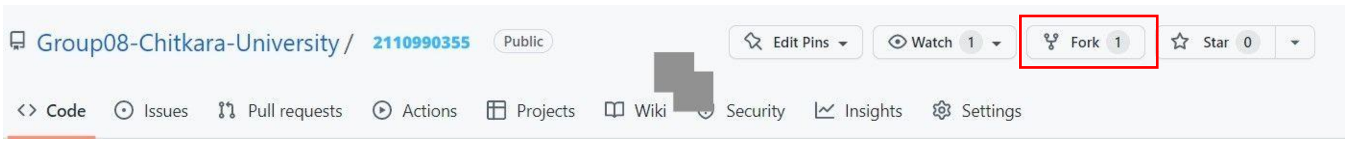
For these reasons, you are always suggested to FORK. Let's have a screenshot walkthrough of the whole process. When getting started with a contribution to Open Source Project, you have been advised to first FORK the repository(repo). But what is a fork?

You must have seen this icon on every repository in the top right corner. Now, this button is used to Fork the repo. But again, what is a fork or forking a repository in GitHub as shown in the below media as follows:

Procedure:

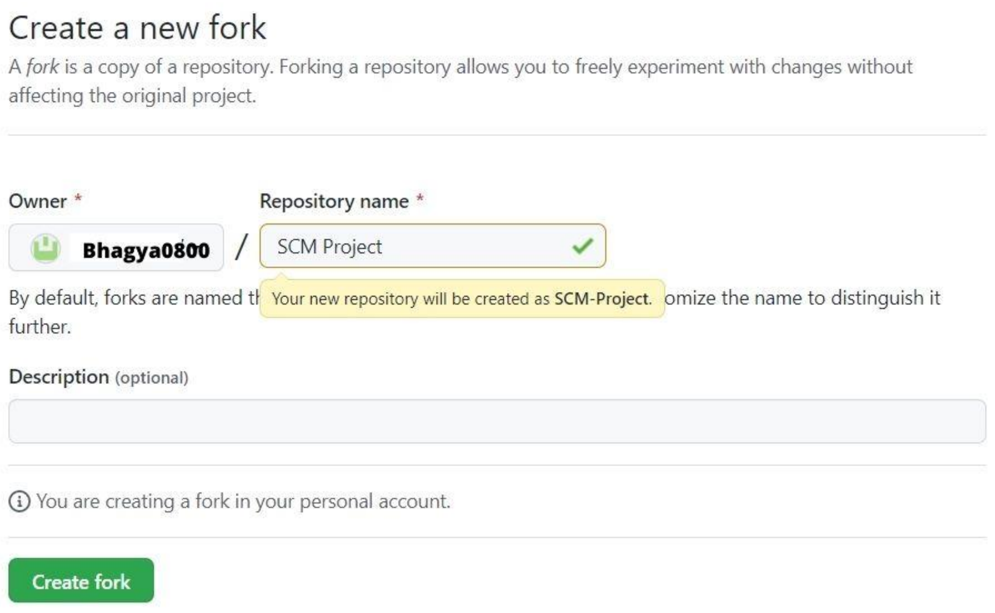
Step 1: Go to **Project SCM** official repository.

Step 2: Find the Fork button on the top right corner.



Step 3: Click on **Fork**.

You will see this screen.


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.

Owner * Bhagya0800 / Repository name * SCM Project

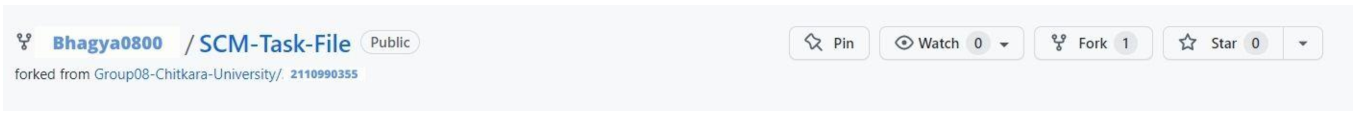
Your new repository will be created as SCM-Project.

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

Description (optional)

You are creating a fork in your personal account.

After this, we will see how to work in the forked repositories which were forked by the collaborators. If the collaborator tries to change in his forked repository, it will not affect the main repository.


Bhagya0800 / SCM-Task-File Public

forked from Group08-Chitkara-University/ 2110990355

What is COMMIT in GitHub?

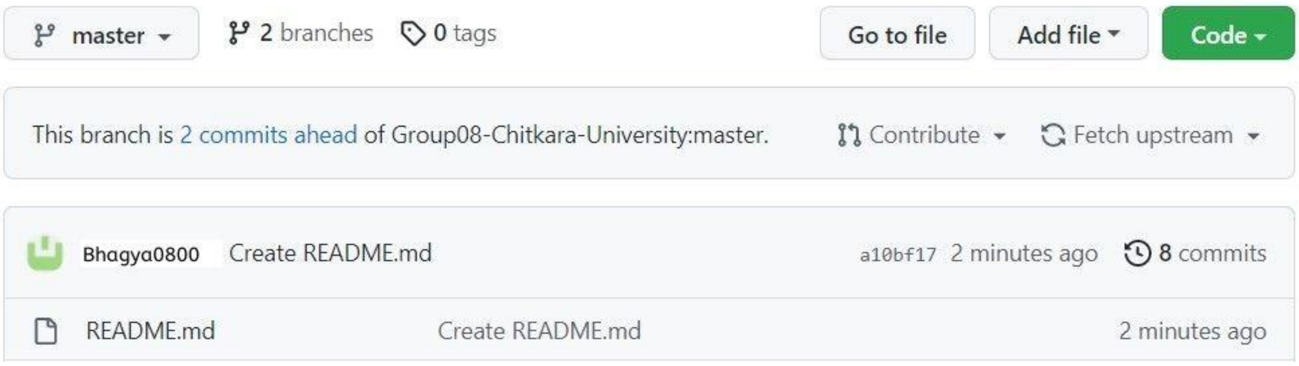
Commit is like a snapshot of your repository. These commits are snapshots of your entire repository at specific times. You should make new commits often, based around logical units of change.
Over time, commits should tell a story of the history of your repository and how it came to be the way that it currently is.
Commits include lots of metadata in addition to the contents and message, like the author, timestamp and more.

It is similar to saving a file that's been edited, a commit records changes to one or more files in your branch. Git assigns each commit a unique ID, called a SHA or hash, that identifies:

- The Specific Changes
- When the Changes were made
- Who created the changes

When you make a commit, you must include a commit message that briefly describes the changes, you can also add a co-author on any commits you collaborate on.

Now, we will see the main repository content of the main Project-SCM Repository. We can see that there is a **README.md** file



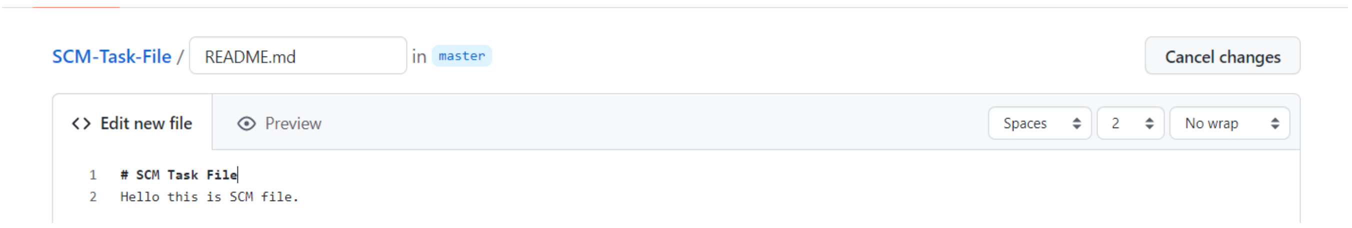
This branch is 2 commits ahead of Group08-Chitkara-University:master.

Bhagya0800 Create README.md a10bf17 2 minutes ago 8 commits

README.md Create README.md 2 minutes ago

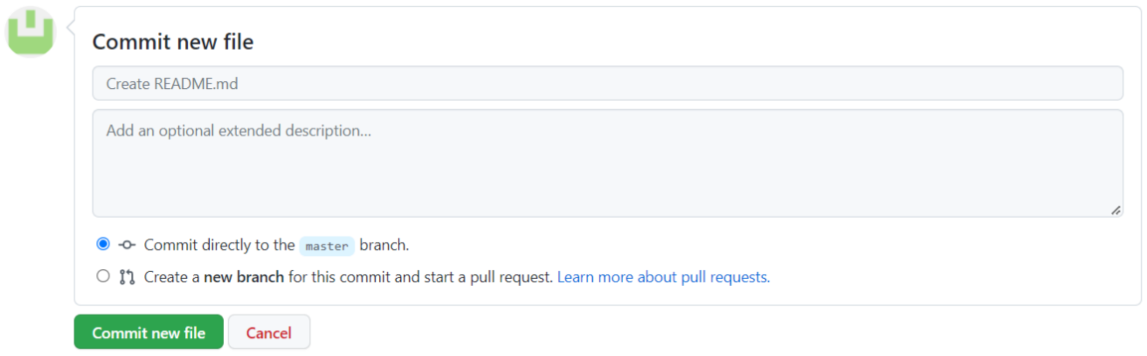
The content in the **README.md** file in the main **Mascots** repository, is given as below:

- Now, when Bhagya0800 (collaborator) tries to change the content of the **README.md** file in his forked repository i.e.,

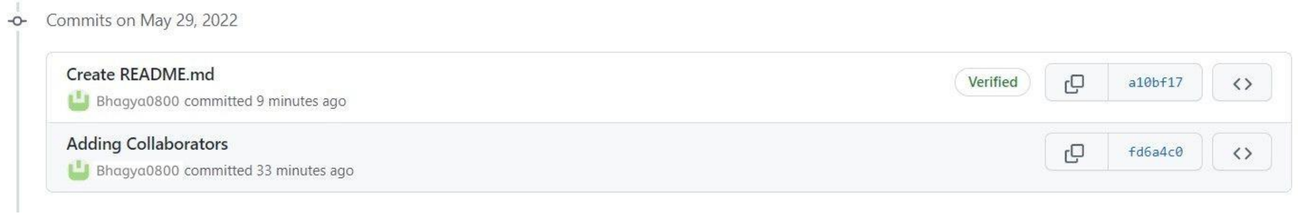


```
# SCM Task File
Hello this is SCM file.
```

- We can see that the **README.md** file has been edited and now it will be committed in this forked repository.

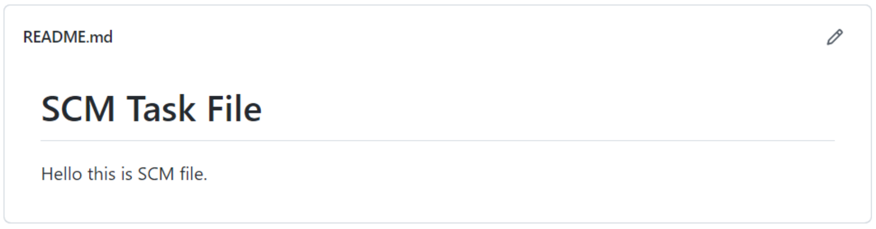

Commit new file
Create README.md
Add an optional extended description...
Commit directly to the master branch.
Create a new branch for this commit and start a pull request. Learn more about pull requests.
Commit new file
Cancel

- We can see the Commit history of the Harshit-2807/**SCM-Task-File** Repository.


Commits on May 29, 2022

Create README.md
Bhagya0800 committed 9 minutes ago
Verified a10bf17

Adding Collaborators
Bhagya0800 committed 33 minutes ago
fd6a4c0

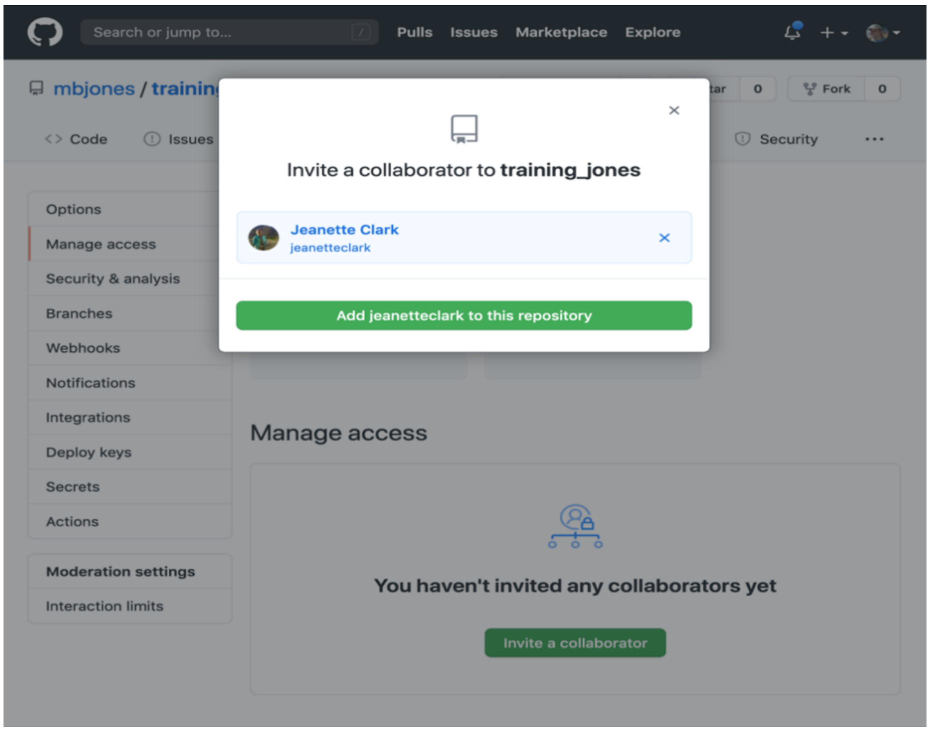

README.md

SCM Task File

Hello this is SCM file.

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

5.3.1 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).

5.3.2 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.

5.3.3 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.

5.3.4 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!

5.3.5 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.

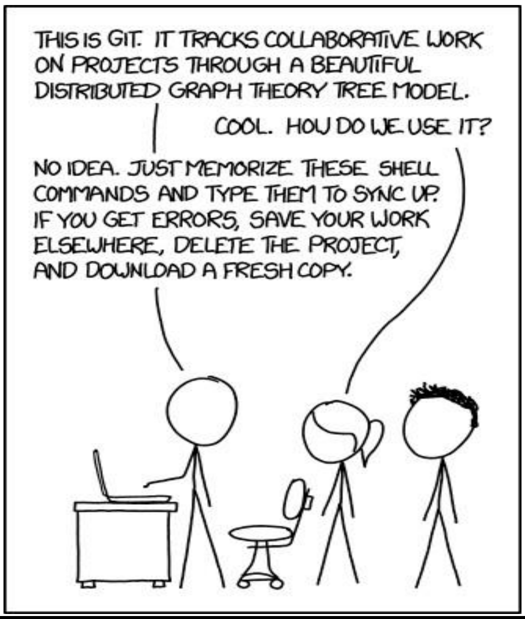
5.3.6 Step 6: Collaborator pull

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:

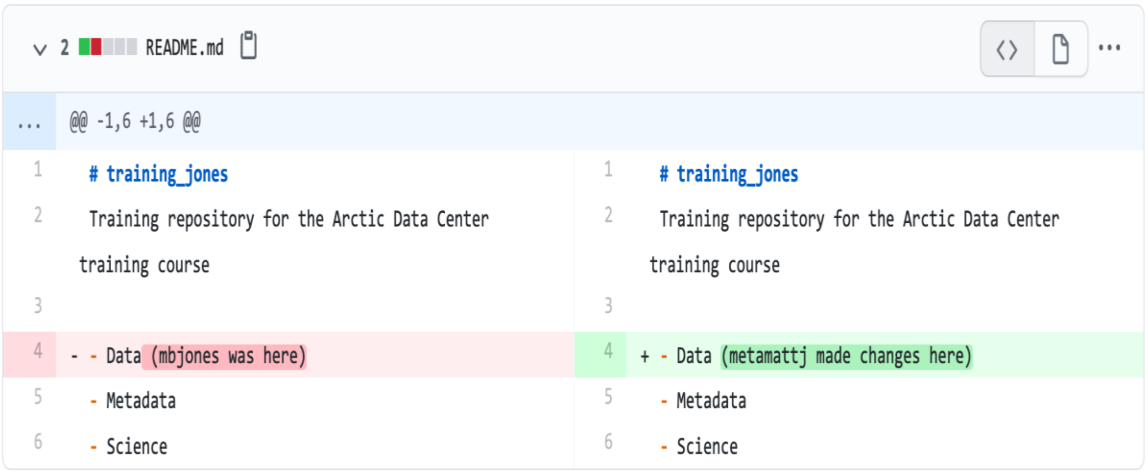
- Step 0: Setup permissions for your collaborator
- Step 1: Collaborator clones the Owner repository
- Step 2: Collaborator Edits the README file
- Step 3: Collaborator commits and pushes the file to GitHub
- Step 4: Owner pulls the changes that the Collaborator made
- Step 5: Owner edits, commits, and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


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

5.4 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.



```
2 README.md
@@ -1,6 +1,6 @@
# training_jones                                    # training_jones
Training repository for the Arctic Data Center      Training repository for the Arctic Data Center
training course                                     training course

- - Data (mbjones was here)                         + - Data (metamattj made changes here)
  - Metadata                                          - Metadata
  - Science                                           - Science
```

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.

5.5 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 still 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

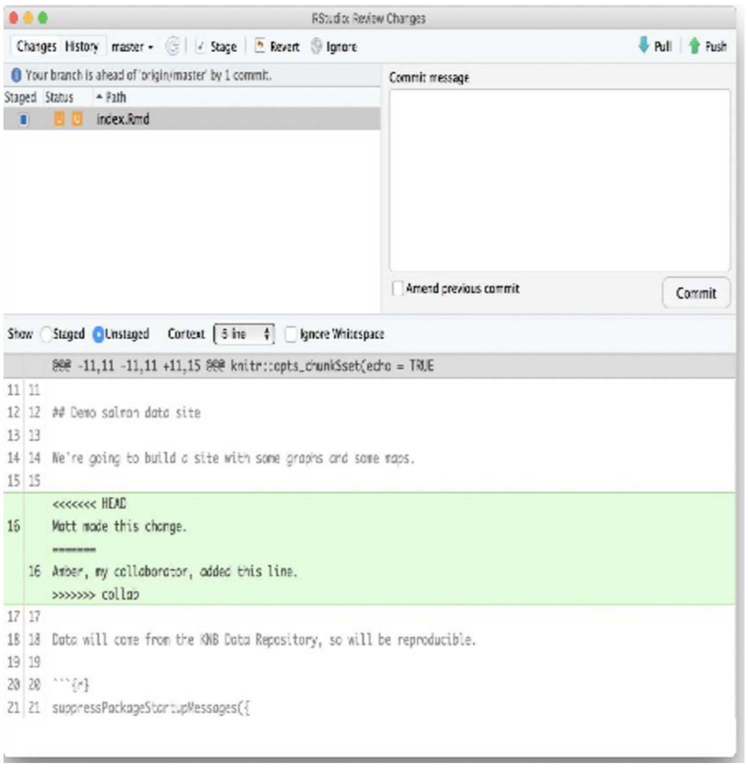
The simplest way to resolve a conflict, given that you know whose version of the file you want to keep, is to use the commandline git program to tell git to use either *your* changes (the person doing the merge), or *their* changes (the other collaborator).

- 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 command line. 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:



~~~
@@ -11,11 -11,11 +11,15 @@ knitr::opts_chunk$set(echo = TRUE

## Demo salmon data site

We're going to build a site with some graphs and some maps.

<<<<<<< HEAD
Matt made this change.
=======
Amber, my collaborator, added this line.
>>>>>>> collab

Data will come from the KNB Data Repository, so will be reproducible.

```{r}
suppressPackageStartupMessages({
~~~

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.

5.5.1 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.

5.5.1.1 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 RStudio. 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.

5.5.1.2 Owner makes a change and commits

From that clean slate, the Owner first modifies and commits a small change including 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 cannot yet see.

5.5.1.3 Collaborator makes a change and commits on the same line

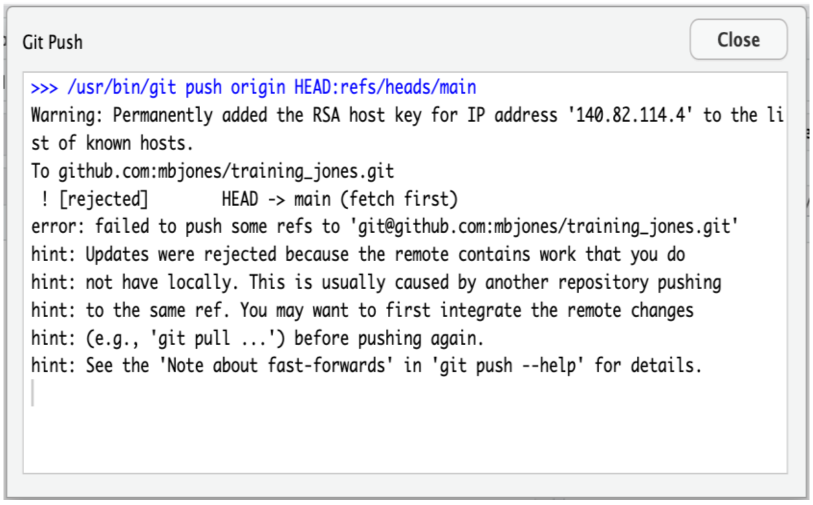
Now the collaborator also makes changes to the same (line 4) of the README.md file in their RStudio copy of the project, adding their name to the line. They then commit. At this point, both the owner and collaborator have committed changes based on their shared version of the README.md file, but neither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

5.5.1.4 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.

5.5.1.5 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).



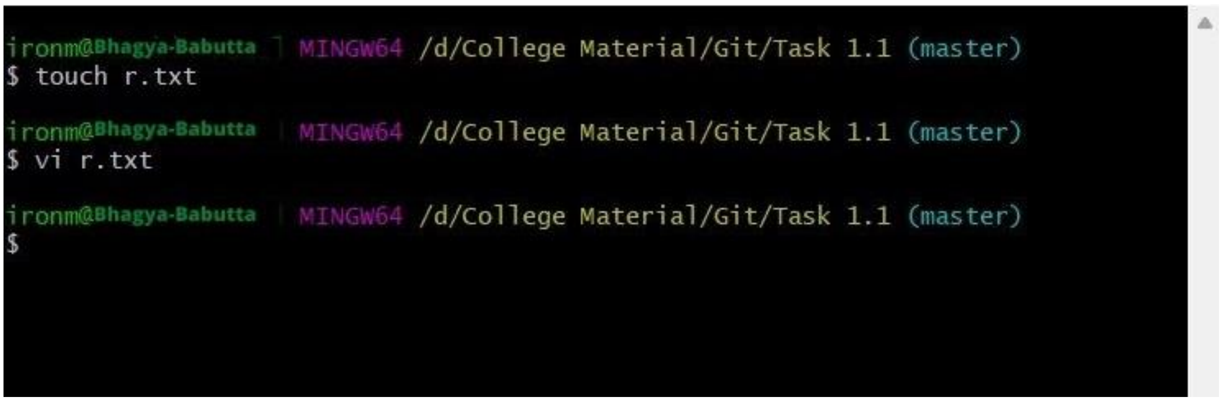
```
>>> /usr/bin/git push origin HEAD:refs/heads/main
Warning: Permanently added the RSA host key for IP address '140.82.114.4' to the li
st of known hosts.
To github.com:mbjones/training_jones.git
 ! [rejected]        HEAD -> main (fetch first)
error: failed to push some refs to 'git@github.com:mbjones/training_jones.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Experiment No. 09

Aim: Reset and Revert

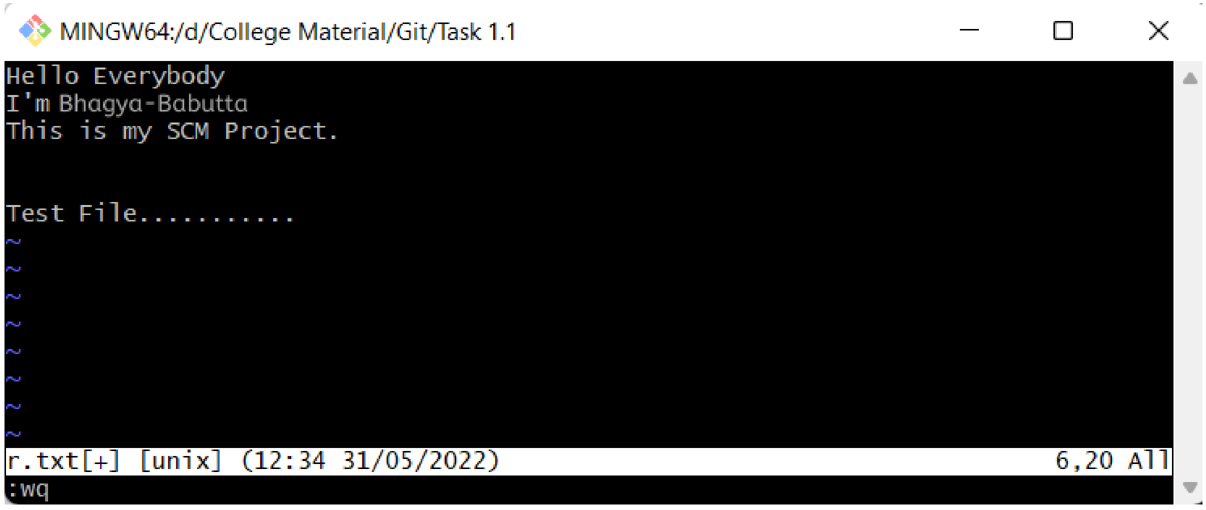
Git Reset: If we want to **revert the changes** that you just made and go back to the files that you had.This technique is called "**reset to HEAD**".

- We will create a file using touch command.
  touch r.txt
- To view content and in-order to add any content we use vi command.
  vi r.txt



```
ironm@Bhagya-Babutta MINGW64 /d/College Material/Git/Task 1.1 (master)
$ touch r.txt

ironm@Bhagya-Babutta MINGW64 /d/College Material/Git/Task 1.1 (master)
$ vi r.txt

ironm@Bhagya-Babutta MINGW64 /d/College Material/Git/Task 1.1 (master)
$
```

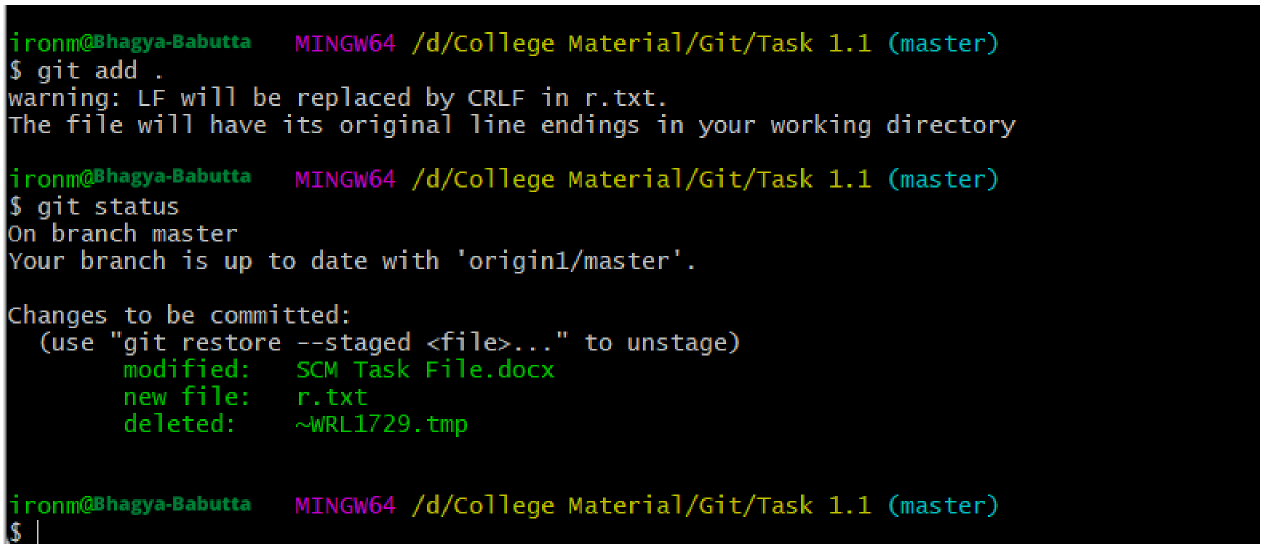
- We will add the contents as we wish to.
- To move back on the command line we will click on esc button and ":" key and we will write wq and press enter.
  Esc: wq



MINGW64:/d/College Material/Git/Task 1.1

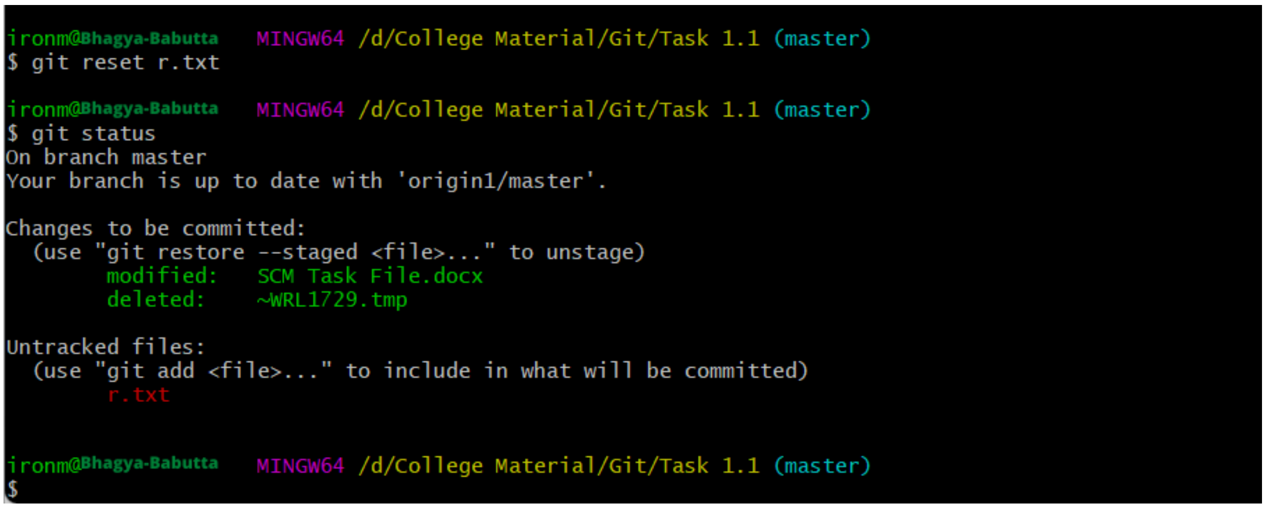
```
Hello Everybody
I'm Bhagya-Babutta
This is my SCM Project.


Test File..........
~
~
~
~
~
~
~
~
r.txt[+] [unix] (12:34 31/05/2022)                                    6,20 All
:wq
```

- We will add to the staging area using git add . command.



```
ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git add .
warning: LF will be replaced by CRLF in r.txt.
The file will have its original line endings in your working directory

ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git status
On branch master
Your branch is up to date with 'origin1/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   SCM Task File.docx
        new file:   r.txt
        deleted:    ~WRL1729.tmp


ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$ |
```

- Now we will use git reset command to revert the changes.
  git reset r.txt
  r.txt again became untracked.

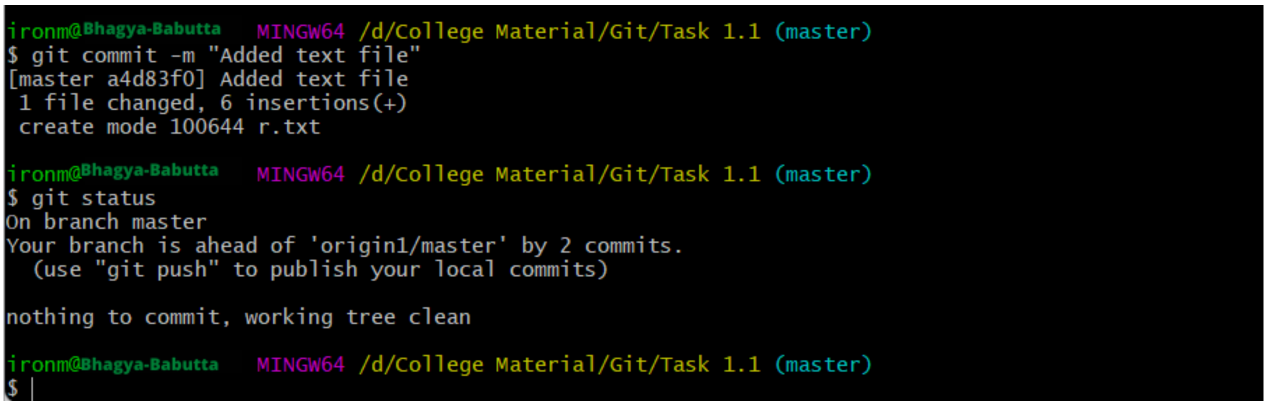


```
ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git reset r.txt

ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git status
On branch master
Your branch is up to date with 'origin1/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   SCM Task File.docx
        deleted:    ~WRL1729.tmp

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


ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$
```

There are two types of reset command →

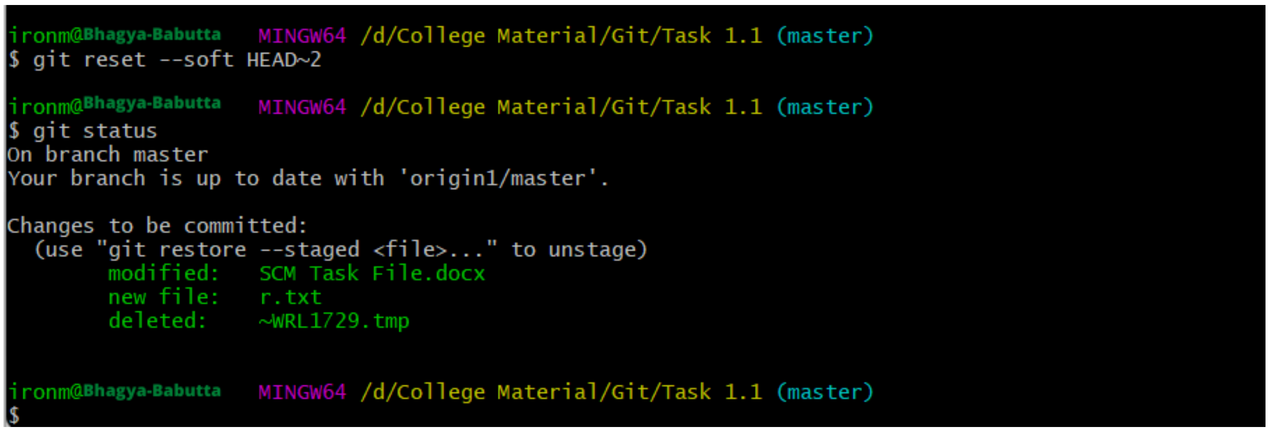
- Hard Reset : **In order to undo the last commit and discard all changes in the working directory and index, execute the "git reset" command with the "–hard" option**
  git reset --hard HEAD~1
- Soft Reset : **The easiest way to undo the last Git commit is to execute the "git reset" command with the "–soft" option that will preserve changes done to your files.**
  git reset –soft HEAD~

- **SOFT RESET-**
  Before using hard reset command.



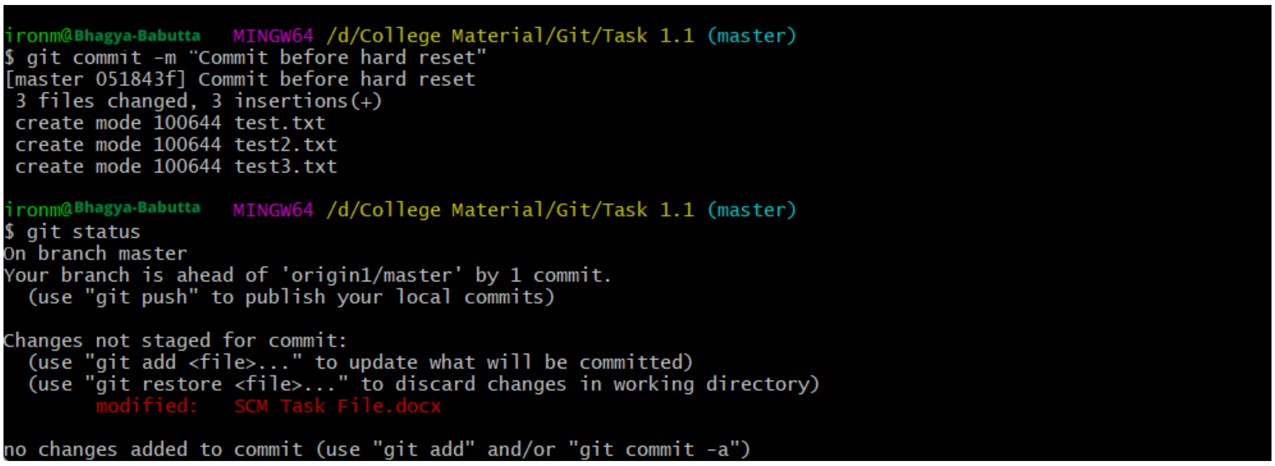
```
ironm@Bhagya-Babutta  MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git commit -m "Added text file"
[master a4d83f0] Added text file
 1 file changed, 6 insertions(+)
 create mode 100644 r.txt

ironm@Bhagya-Babutta  MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git status
On branch master
Your branch is ahead of 'origin1/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

ironm@Bhagya-Babutta  MINGW64 /d/College Material/Git/Task 1.1 (master)
$ |
```

After using soft reset command, the file again became uncommitted.



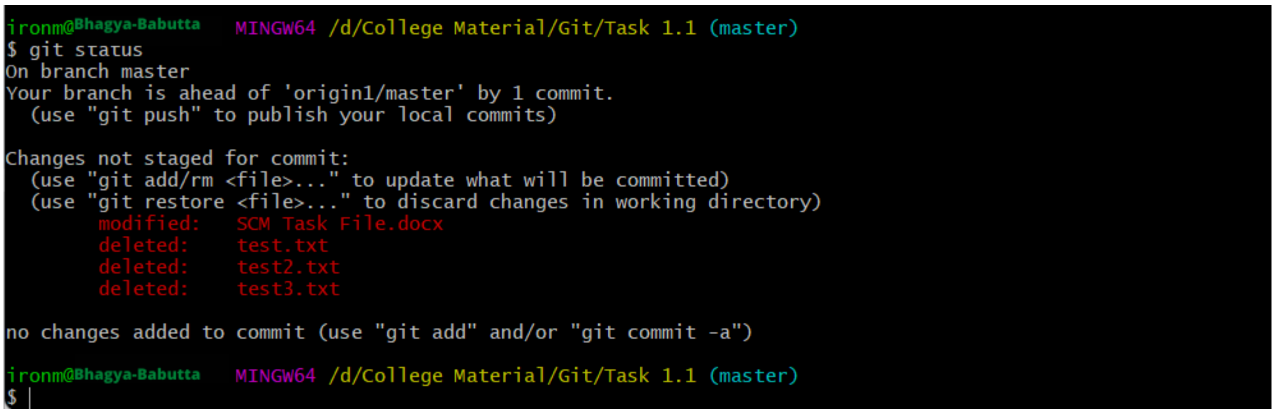
```
ironm@Bhagya-Babutta  MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git reset --soft HEAD~2

ironm@Bhagya-Babutta  MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git status
On branch master
Your branch is up to date with 'origin1/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   SCM Task File.docx
        new file:   r.txt
        deleted:    ~WRL1729.tmp


ironm@Bhagya-Babutta  MINGW64 /d/College Material/Git/Task 1.1 (master)
$
```

- **HARD RESET-**
  Before using hard reset command



```
ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git commit -m "Commit before hard reset"
[master 051843f] Commit before hard reset
 3 files changed, 3 insertions(+)
 create mode 100644 test.txt
 create mode 100644 test2.txt
 create mode 100644 test3.txt

ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git status
On branch master
Your branch is ahead of 'origin1/master' by 1 commit.
  (use "git push" to publish your local commits)

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:   SCM Task File.docx

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

After using hard reset command , all the files got untracked.



```
ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$ git status
On branch master
Your branch is ahead of 'origin1/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   SCM Task File.docx
        deleted:    test.txt
        deleted:    test2.txt
        deleted:    test3.txt

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

ironm@Bhagya-Babutta   MINGW64 /d/College Material/Git/Task 1.1 (master)
$ |
```