

Source Code Management

Submitted by:
Amarbir Singh
2110990159
G8

Submitted to:
Dr . Monit Kapoor

1	Version control with Git	1-13
2	Project	13-17
3	Task 1.2	18-24

Step 1 - Installing Git:

Before we start using Git, we have to make it available on your computer. Even if it's already

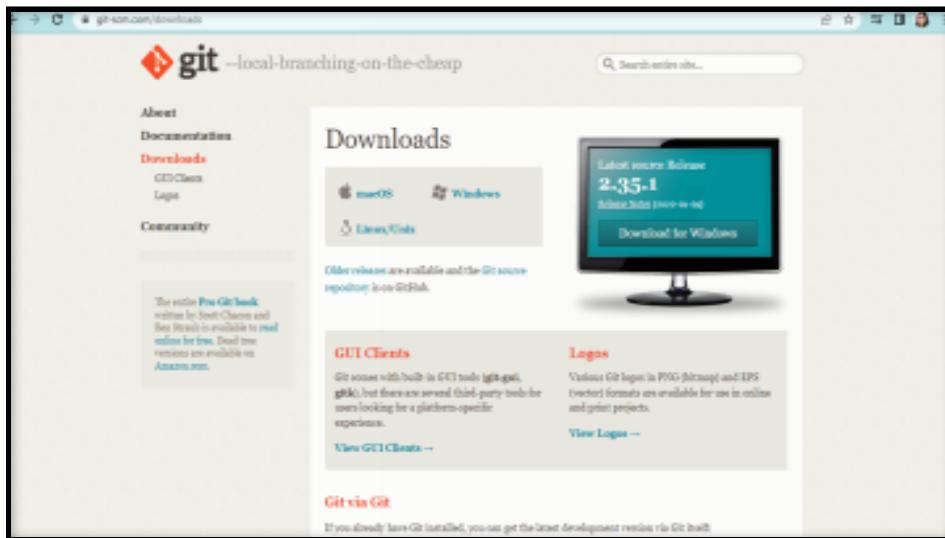
Installed , it's probably a good idea to update to the latest version. We can either install it as a package or via another installer, or download the source code and compile it yourself.

Step 2 - Installing on windows:

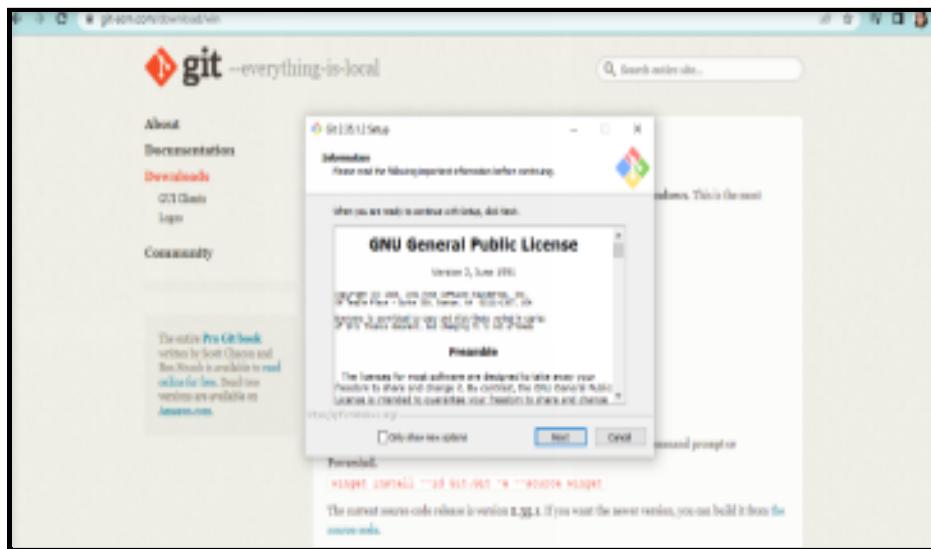
There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to <https://git-scm.com/download/win> and the download will start automatically.

Note that this is a project called Git for Windows, which is separate from Git itself; for more information on it, go to <https://gitforwindows.org>.

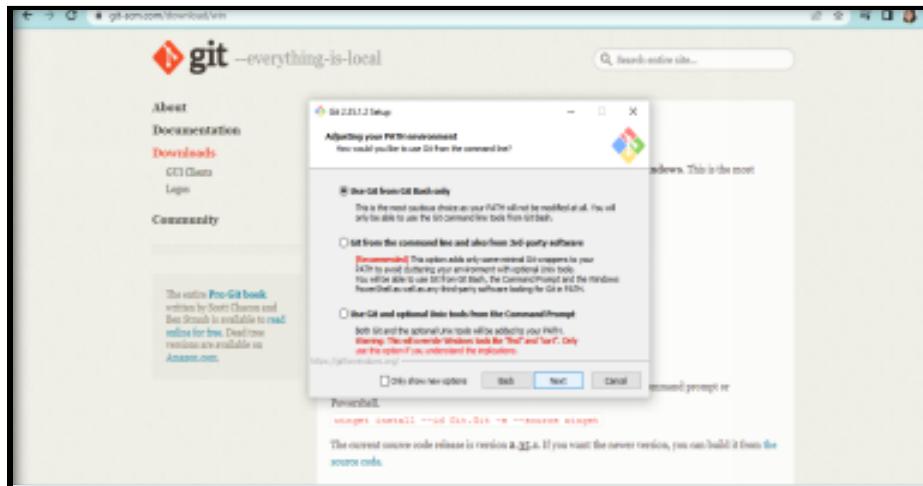
1.



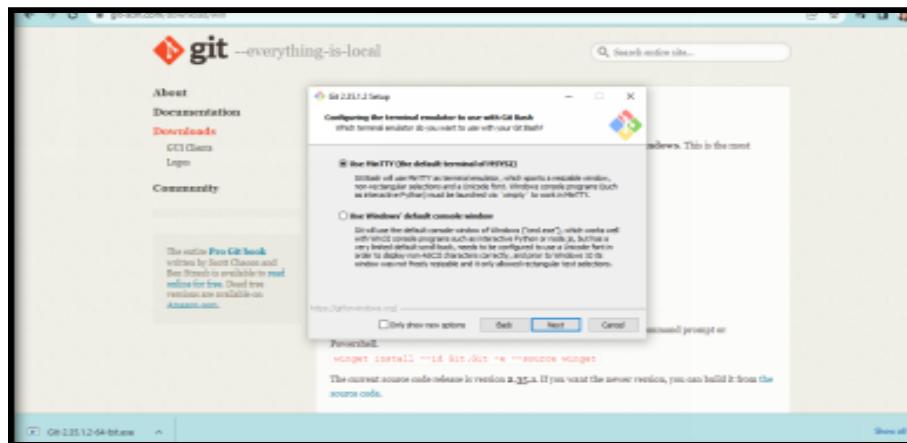
2.



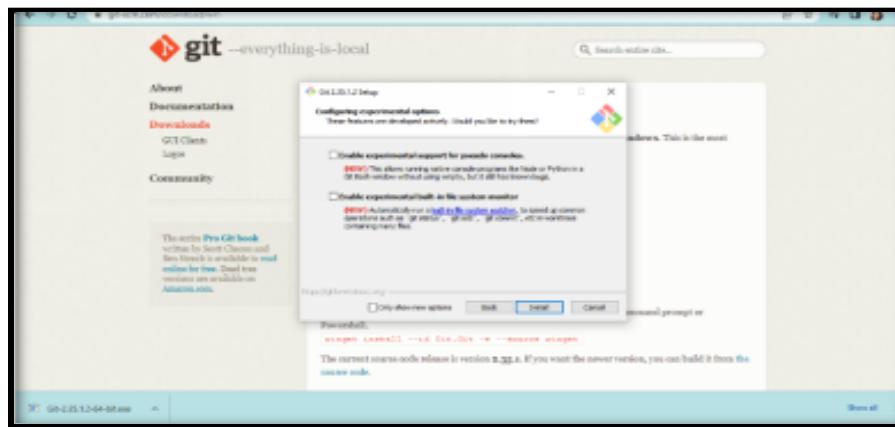
3.



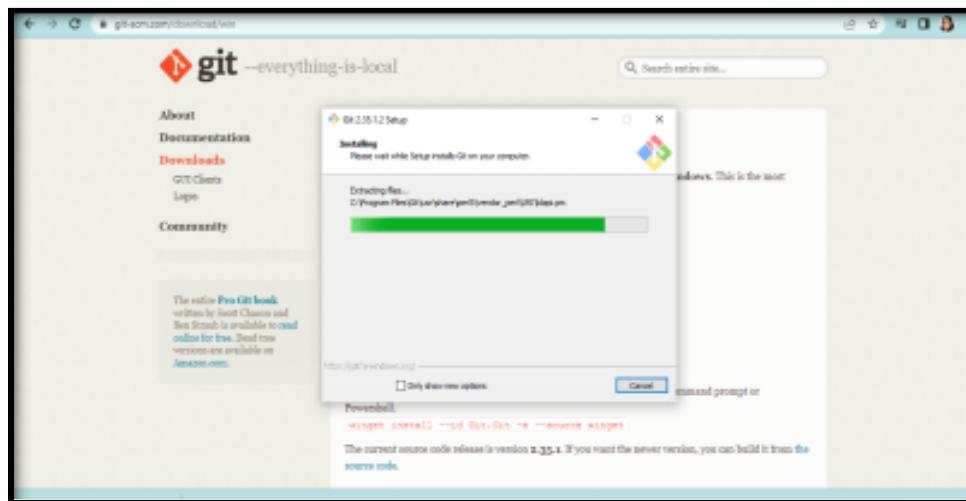
4.



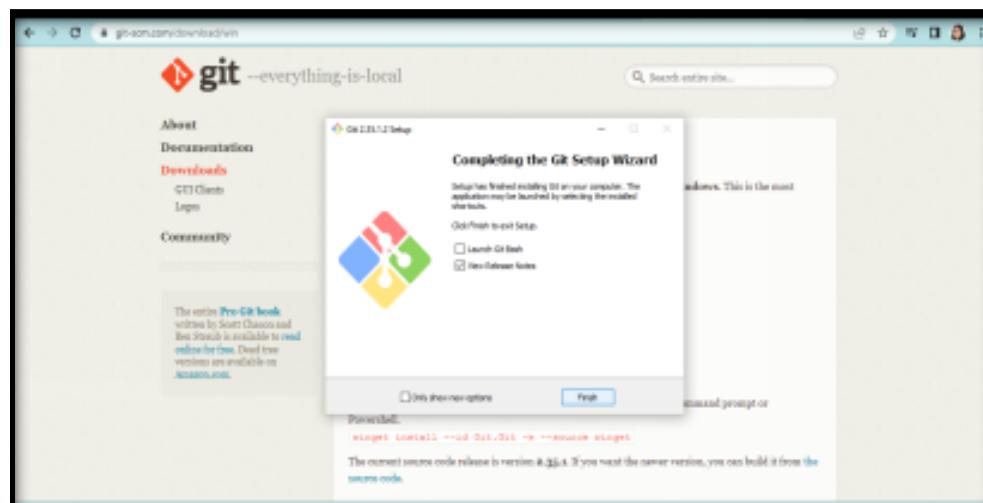
5.



6.

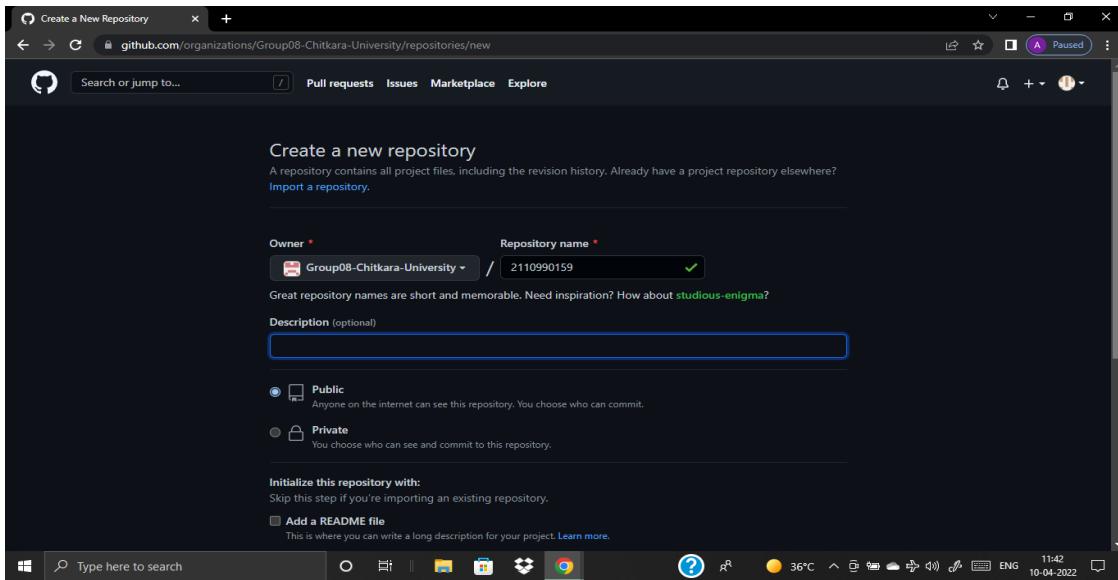


7.

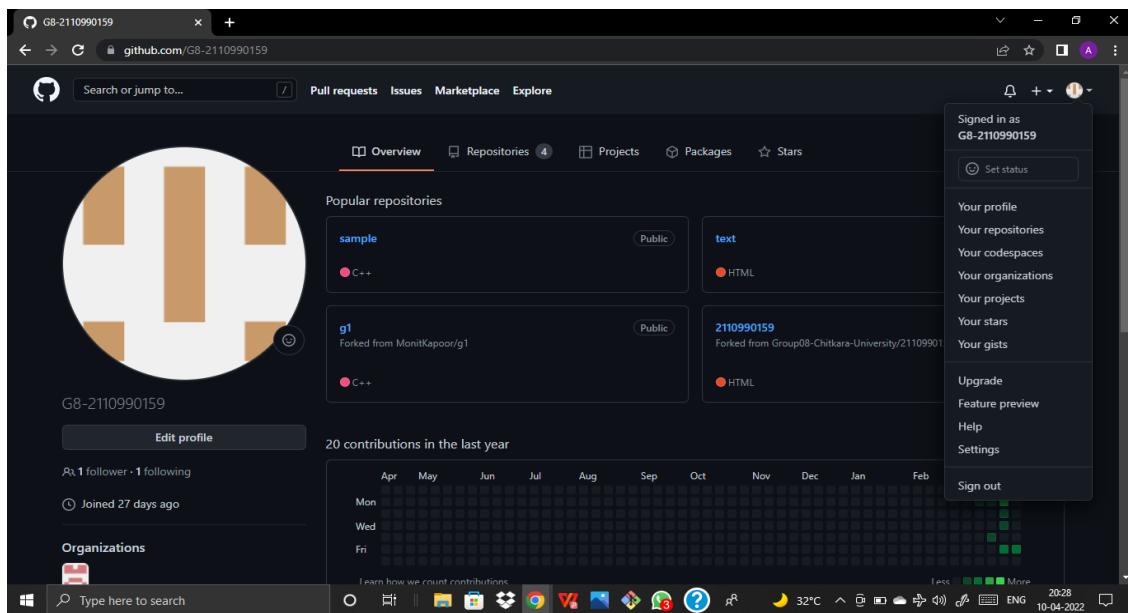


Step 3 – Setting up github account:

1.



2.



3. Configuration of Git:

Step1)

You can configure your Git by typing: -

1. Set your username: git config --global user.name "Your Name"
2. Set your email address: git config --global user.email "Your Email"

The page looks like as: -



A screenshot of a terminal window titled "MINGW64:/c/Users/amar/Desktop". The window shows three lines of command-line input:

```
amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$ git config user.name
G8-2110990159

amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$ git config --global user.email
amarbir0159.be21@chitkara.edu.in

amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$
```

Step2)

You can check configuration of Git by typing -

1. git config --list

The page looks like as: -

Program to Generate logs:

Advantage of version control systems like git is that it can record changes. ‘Git log’ command is used to display all these changes that were made by the user in the repository. Log is a record of all the previous commits.

To understand Logs we need to get familiar with all the commands that are used in making changes to a repository.

- 1)Repository: A repository is a directory that contains all the project-related data.
- 2)Git init: The git init command is used to create a new blank repository.
- 3)Git status: We can list all the untracked, modified and deleted files using the git status command.
- 4)Git add: Adds all the untracked and modified files to the staging area.
- 5)Git commit: Git commit finalizes the changes in the repository. Every commit is saved in the branch you are working in and can be used to revert back to older versions of the project.

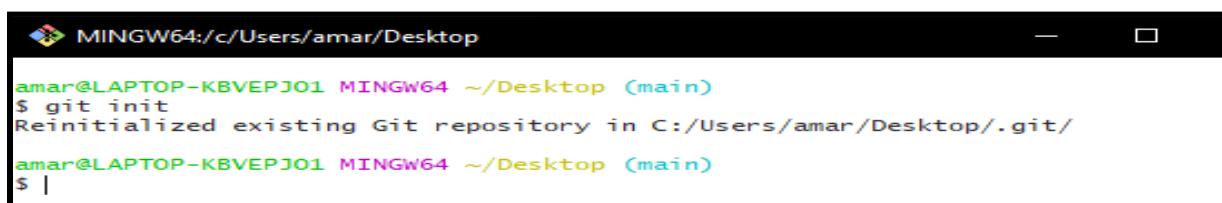
Making GIT Repository

Step1: Git init

Initializing a new repository, You Can do it by typing: -

1.git init

The page looks like as: -

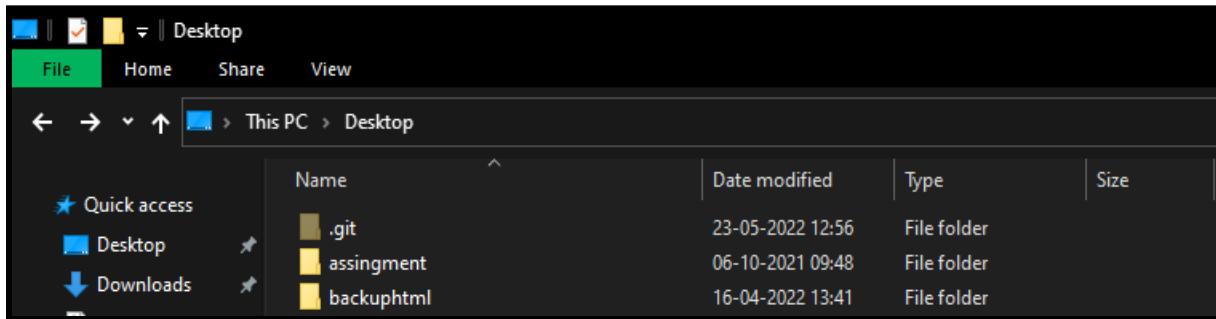


```
MINGW64:/c/Users/amar/Desktop
amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$ git init
Reinitialized existing Git repository in C:/Users/amar/Desktop/.git/
amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$ |
```

Step2: Adding the files to the folder

Just like (Samples...)

The page looks like as: -



Step3: Git add

The git add command adds a change in the working directory to the staging area.

You Can do it by typing: -

- 1.git add
- 2.git add (current file name)

The page looks like as: -

```
amar@LAPTOP-KBVEPJ01 MINGW64 /d/g1 (master)
$ git add .

amar@LAPTOP-KBVEPJ01 MINGW64 /d/g1 (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   first.cpp
```

Step4: Git Commit

The "commit" command is used to save your changes to the local repository.

You Can do it by typing: -

1.git commit -m"any text"

The page looks like as: -

```
amar@LAPTOP-KBVEPJ01 MINGW64 /d/g1 (master)
$ git commit -m "added headings to first.cpp"
[master 6df3564] added headings to first.cpp
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Step5: Git Log

Git log will show all the commits made by the author with their time.

After every commit the checksum value (written In yellow color) of the folder changes.

Checksum is used to verify that the data in that file has not been tampered with or manipulated, possibly by a malicious entity.

You Can do it by typing: -

1.git log

The page looks like as: -

```
MINGW64:/d/g
amar@LAPTOP-KBVEPJ01 MINGW64 /d/g (main)
$ git log
commit 74b3cf92b487d1a7f48cefe99c6b5d3a029e2f23 (HEAD -> main)
Author: G8-2110990159 <amarbir0159.be21@chitkara.edu.in>
Date:   Sun Apr 10 13:40:03 2022 +0530

    second commit

commit 9ae938d4f9d112d388c17ec75c0d3e2162976407 (origin/main)
Author: G8-2110990159 <amarbir0159.be21@chitkara.edu.in>
Date:   Sun Apr 10 12:54:01 2022 +0530

    first commit
```

Step6: Create and Visualize Branches

A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is master.

Step1: CHECKING UP THE BRANCHES

You can check which branch you are working in by using the command

1.'git branch'.

The default branch is always the master branch.

The page looks like as: -

```
amar@LAPTOP-KBVEPJ01 MINGW64 /d/g (main)
$ git branch
  feature
* main

amar@LAPTOP-KBVEPJ01 MINGW64 /d/g (main)
$ git branch feature
fatal: a branch named 'feature' already exists
```

Step3: Changing Branches

To switch to the other branch

You Can do it by typing: -

1.git checkout (BRANCH NAME)

The page looks like as: -

```
MINGW64:/c/Users/amar/Desktop
git
amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$ git branch
* main

amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$ git branch activity1

amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$ git branch
activity1
* main

amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$ git checkout activity1
Switched to branch 'activity1'
M      2.html
M      2.html.bak
M      Notepad++.lnk
M      Zoom.lnk
D      about.html.bak
D      bhavik.html.bak
D      electronics/6. BJT_SLIDES_electronics
D      electronics/TUTORIAL SHEET 2(transistor) with hints.pdf
D      example.html
D      p.html.bak
D      project.html.bak

amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (activity1)
$ git branch
* activity1
main
```

Step6: GIT MERGING

Now you can merge two branches by command.

1.git merge (BRANCH NAME)

If you want to merge a new branch in master branch you need to first checkout into the master branch and then run the command.

The page looks like as: -

```
MINGW64:/c/Users/amar/Desktop
amar@LAPTOP-KBVEPJ01 MINGW64 ~/Desktop (main)
$ git merge text.txt
merge: text.txt - not something we can merge
```

There are three stages for git lifecycle:

- 1)Working directory
- 2)Staging area
- 3)Git repository

Working Directory:

The working directory is the folder in your local computer where the project files and folders are stored.

The local directory is created by the command ‘git init’ which creates a ‘.git’ named folder which is used to track the files in the directory.

‘.git folder’ is generally hidden but can be tracked enabling hidden files.



Staging area:

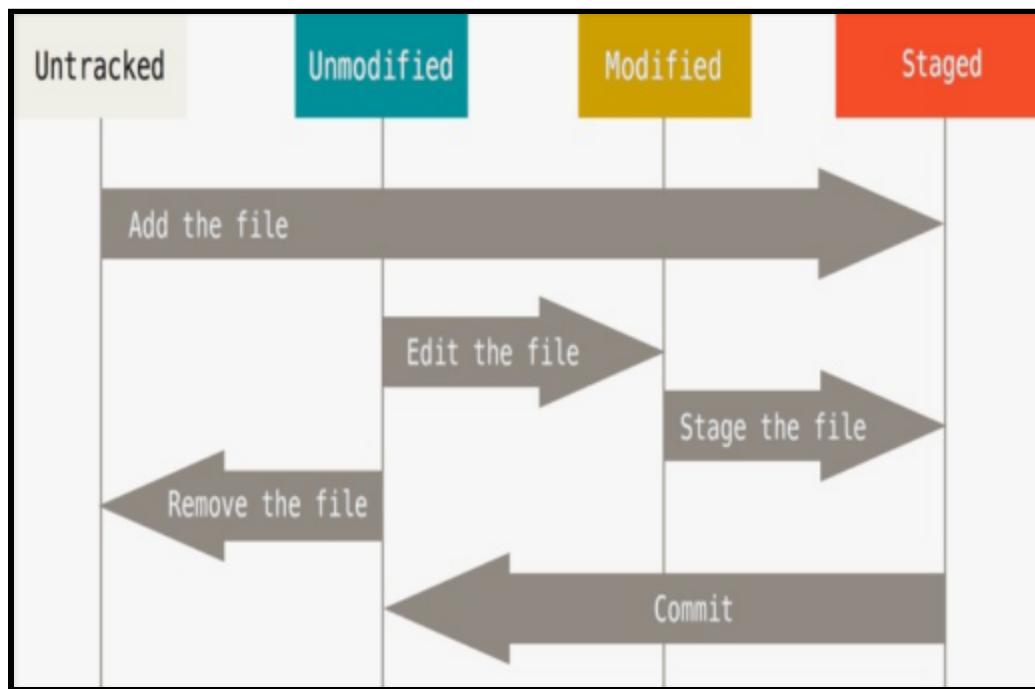
The staging area has those files which are supposed to go to the next commit. Only those files which are needed to go to the next commit stay in the staging area.

You can shift the files to the git repository by using the command ‘git add --a’.

Git repository:

Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit our files using the git commit command. Commit helps us in keeping the track of the metadata of the files in our staging area. We specify every commit with a message which tells what the commit is about.

You can commit files by using command ‘git commit -m “message”’



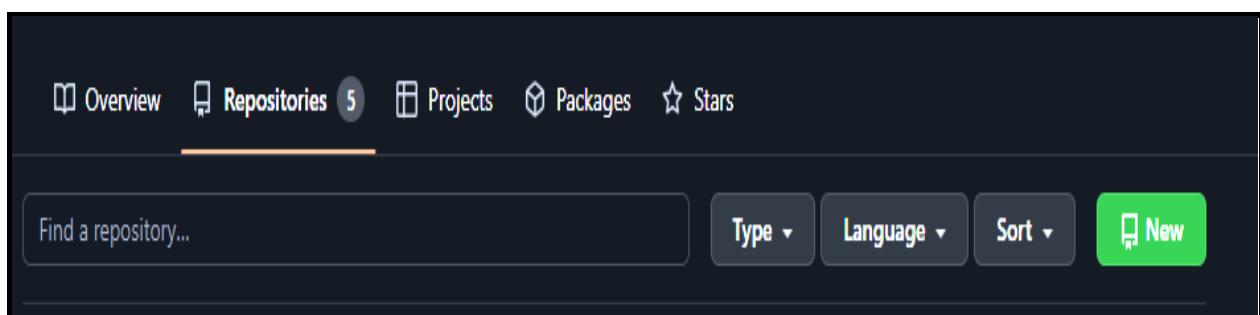
Uploading data on github

NOTE-

YOU HAVE TO MAKE A REPOSITORY IN GITHUB.

Step1) CREATING REPOSITORY IN GITHUB

The page looks like as: -



By clicking on new you are able to make a new repository.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * G8-2110990159 / **Repository name *** teams 

Great repository names are short and descriptive. teams is available. Need inspiration? How about [cautious-spork](#)?

Description (optional)

 **Public**
Anyone on the internet can see this repository. You choose who can commit.

 **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. [Learn more.](#)

Write the repository name and click on next.



Your GITHUB Repository has been created.

Step2) GIT ADDING REMOTE BRANCH

Git stores a branch as a reference to a commit, and a branch represents the tip of a series of commits.

You Can do it by typing: -

1.git branch -M main

The page looks like as: -

```
amar@LAPTOP-KBVEPJ01 MINGW64 /d/g (master)
$ git branch -M main

amar@LAPTOP-KBVEPJ01 MINGW64 /d/g (main)
```

Step3) GIT ADDING REMOTE ORIGIN

Is a Git repository that's hosted on the Internet

You Can do it by typing: -

1.git remote add origin

The page looks like as: -

```
amar@LAPTOP-KBVEPJ01 MINGW64 /d/g (master)
$ git remote add origin https://github.com/Group08-chitkara-University
/2110990159.git
```

Step4) GIT CLONE

The git clone command is used to upload local repository content to a remote repository.

You Can do it by typing: -

1.git clone url

The page looks like as: -

```
amar@LAPTOP-KBVEPJ01 MINGW64 /d/cloned
$ git clone https://github.com/G8-2110990159/g1
Cloning into 'g1'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 12 (delta 2), reused 12 (delta 2), pack-reused 0
Receiving objects: 100% (12/12), done.
Resolving deltas: 100% (2/2), done.
```

Final Result

1. Document in your system

The page looks like as: -

Name	Date modified	Type
.git	22-04-2022 13:05	File folder
cloned	01-04-2022 21:13	File folder
g	11-04-2022 12:21	File folder
g1	01-04-2022 12:00	File folder

2. Document in your GITHUB Repository

The page looks like as: -

The screenshot shows a GitHub repository page for the repository 'G8-2110990159/g1'. The repository is public and has been forked from 'MonitKapoor/g1'. The 'Code' tab is selected. The master branch has 1 branch and 0 tags. A message indicates that the branch is 1 commit ahead and 1 commit behind the MonitKapoor/master branch. One commit is listed, showing a file named 'first.cpp' was updated on April 1, 2022, by user 'G8-2110990159', with 5 commits in total.

Commit	Author	Date	Commits
first.cpp	G8-2110990159	on Apr 1	5 commits

Task 1.2

1. Add Collaborators on Github Repo :-

1. Ask for the username of the person you're inviting as a collaborator. ...

The screenshot shows the GitHub repository settings for 'G8-2110990159 / team' (Public). The 'Settings' tab is selected. In the 'Who has access' section, the 'Access' dropdown is set to 'Collaborators'. The 'PUBLIC REPOSITORY' box states: 'This repository is public and visible to anyone.' The 'DIRECT ACCESS' box states: '1 has access to this repository. 0 collaborators. 1 invitation.' A 'Manage' button is present. Below this, the 'Manage access' section includes a 'Select all' checkbox, a 'Type' dropdown, and a search bar with placeholder text 'Find a collaborator...'. A green 'Add people' button is located at the top right of this section.

This screenshot shows the same GitHub repository settings page after adding a collaborator. The 'Who has access' section remains the same. In the 'Manage access' section, a new entry for 'Chopra2212' is listed. The entry shows a circular profile picture, the username 'Chopra2212', the status 'Awaiting Chopra2212's response', and a 'Pending Invite' button. A 'Remove' button is also present. The bottom of the page shows navigation links for 'Previous' and 'Next'.

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

The screenshot shows the GitHub repository page for 'G8-2110990159'. At the top, there's a search bar and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the header, the repository name 'G8-2110990159 / team' is displayed, along with a 'Public' badge. To the right are buttons for 'Pin', 'Unwatch', 'Fork', and 'Star'. A dropdown menu shows the current branch is 'main', with 2 branches and 0 tags. The 'Code' tab is selected. A commit card for 'G8-2110990159 fifth commit' by user 'b7e6ff0' on March 28 at 5 commits ago is shown, with a file named 'hello.cpp'. A call-to-action box encourages adding a README, with a green 'Add a README' button. On the right side, sections for 'About', 'Releases', and 'Packages' are visible, each with a note that no content has been provided.

3. Under our repository name, click Settings

This screenshot is identical to the one above, showing the GitHub repository page for 'G8-2110990159'. The difference is that the 'Settings' link in the top navigation bar is highlighted with a blue underline, indicating it is the active section of the page.

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

The screenshot shows the GitHub repository settings for 'G8-2110990159 / team'. The 'Settings' tab is selected. On the left, the 'Access' section is open, showing 'Collaborators' is selected. The main area displays 'Who has access' with two sections: 'PUBLIC REPOSITORY' (This repository is public and visible to anyone) and 'DIRECT ACCESS' (1 has access to this repository. 0 collaborators, 1 invitation). Below this is the 'Manage access' section with a search bar and a green 'Add people' button.

5. Click Invite a collaborator.

The screenshot shows the 'Add a collaborator to team' modal dialog. It features a search bar with placeholder text 'Search by username, full name, or email' and a green button at the bottom labeled 'Select a collaborator above'. The background shows the 'Who has access' section of the GitHub repository settings.

The screenshot shows the 'Who has access' section of a GitHub repository settings page. On the left, there's a sidebar with options like General, Access (selected), Collaborators, Moderation options, Branches, Tags, Actions, Webhooks, Environments, Pages, Security, Code security and analysis, Deploy keys, and Secrets. The main area shows that the repository is a PUBLIC REPOSITORY and has DIRECT ACCESS. It lists two collaborators: 'Chopra2212' and 'deveshgupta9024', both of whom have pending invites. There are buttons for 'Manage' and 'Add people'.

2. Fork And Commit :- A fork is a copy of a repository that we manage. Forks let us make changes to a project without affecting the original repository. We can fetch updates from or submit changes to the original repository with pull requests.

If we need to fork a GitHub or GitLab repo, it's as simple as navigating to the landing page of the repository in your web browser and clicking on the Fork button on the repository's home page. A forked copy of that Git repository will be added to your personal GitHub or GitLab repo. That's it. That's all we have to do to fork a Git repo.

A modal dialog box titled 'Commit changes'. It contains fields for 'Update test.txt' and an optional extended description. At the bottom, there are two radio button options: one selected for 'Commit directly to the master branch.' and another for 'Create a new branch for this commit and start a pull request.' Below the radio buttons are 'Commit changes' and 'Cancel' buttons.

The screenshot shows a GitHub repository page. At the top, it displays the repository name 'G8-2110990159 / g1' (Public) and a note that it's forked from 'MonitKapoor/g1'. The header includes standard GitHub navigation links: Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the header, there's a summary bar showing 'master' branch, '1 branch', '0 tags', and buttons for 'Go to file', 'Add file', and 'Code'. A message states 'This branch is 1 commit ahead, 1 commit behind MonitKapoor/master.' To the right, there's an 'About' section with a note 'No description, website, or to', and metrics: 0 stars, 0 watching, and 1 fork. Below this is a 'Releases' section with a note 'No releases published' and a link 'Create a new release'. The main content area shows a single commit: 'GB-2110990159 added headings to first.cpp' by user 'b986493' on April 1, which has 5 commits and was made 2 months ago.

3. Merge and Resolve conflicts created due to own Activity and Collaborators activity:-

There are a few steps that could reduce the steps needed to resolve merge conflicts in Git.

1. The easiest way to resolve a conflicted file is to open it and make any necessary changes.
2. After editing the file, we can use the git add command to stage the new merged content.
3. The final step is to create a new commit with the help of the git commit command.
4. Git will create a new merge commit to finalize the merge

Let us now look into the Git commands that may play a significant role in resolving conflicts.

```

Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/hello (newhello|MERGING)
$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc codecompare smerge emerge vimdiff
Merging:
text1.txt

Normal merge conflict for 'text1.txt':
{local}: modified file
{remote}: modified file
Hit return to start merge resolution tool (vimdiff):
4 files to edit

Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/hello (newhello|MERGING)
$ git status
On branch newhello
Your branch is up to date with 'origin/newhello'.

All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Untracked files:
(use "git add <file>..." to include in what will be committed)
  text1.txt.orig

Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/hello (newhello|MERGING)
$ git add .

```

```

ii am writing in text1 in newhello
ii am writing in text1
ii am writing in text1 in master
<<<<< HEAD
ii am writing in text1 in newhello
=====
ii am writing in text1 in master
>>>>> b4ab8756d8d9a6bdcf80a54ec821303fb4b5ca81

```

4. Git Reset :- Git reset is a powerful command that is used to undo local changes to the state 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 local directory.

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git reset --soft HEAD~1
fatal: ambiguous argument 'HEAD~1': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
git <command> [<revision>...] -- [<file>...]'
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ vi text1.txt
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ vi text1
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git add .
warning: LF will be replaced by CRLF in text1.
The file will have its original line endings in your working directory
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git add .
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git commit -m "second commit"
[master a6bab34] second commit
 1 file changed, 1 insertion(+)
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git reset --soft HEAD~1
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   text1
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ notepad text1
```

```
Microsoft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git add .
```

```
msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git commit -m "second commit"
[master fc546b9] second commit
1 file changed, 1 insertion(+)

msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git reset --hard HEAD~1
HEAD is now at 7487caf first commit

msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ notepad text1

msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ touch text2.bin

msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ touch .gitignore

msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ vi .gitignore

msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ vi text2.bin

msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore

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

msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ cat text2.bin
Hello i am writing in text2

msft@DESKTOP-S0SPJP1 MINGW64 /d/text6 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore

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