DevOps - GIT & GIT HUB

Monika Morla - 5250

Contents

DevOps

What is DevOps?

- It is a new culture/process.
- It is the process of continuous development, continuous build, continuous testing, and continuous release of the software.
- It is a combination of Development & Operations.
- The main objective of DevOps is to implement collaboration between Development & Operations teams.
- The DevOps cycle is an Infinite loop where everything is continuous.
- The beauty of DevOps is everything is Automated.

Tools:

Version Control System:

- The version control system is also known as Software Configuration Management (SCM) or Source Code Management (SCM).
- It is to maintain all versions & track all changes of software code, we required.
- Version Control Systems are software tools that help software teams manage changes to source code over time.

Benefits of Version Control System:

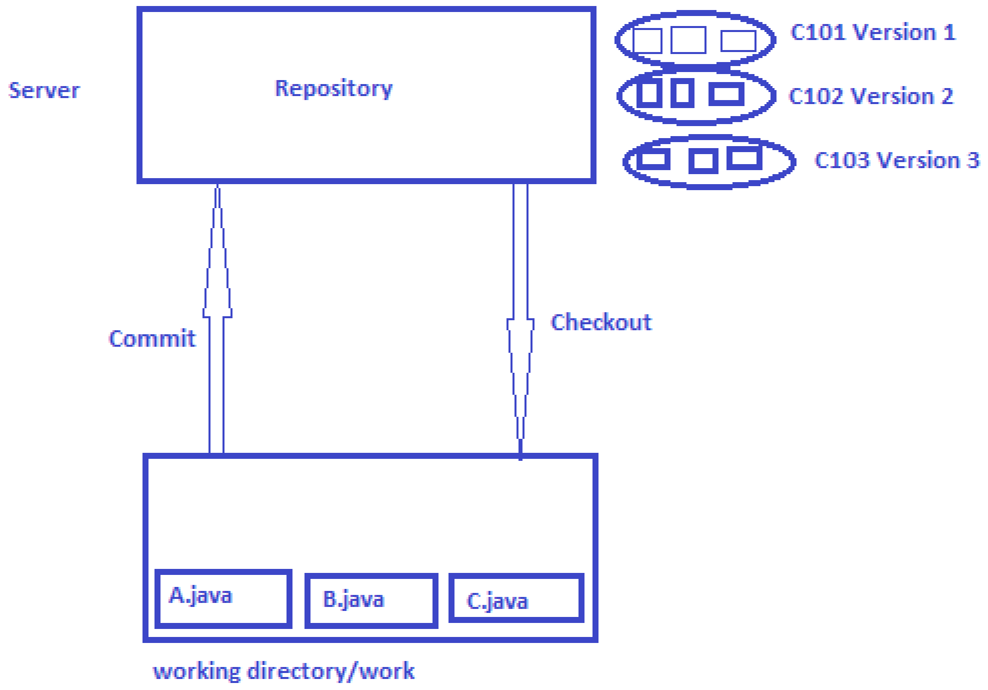
- We can maintain different versions and we can choose any version based on client requirements.
- With every version, we can maintain metadata (extra data) like the commit message, who did the changes, when did he the change, and what changes he did.
- Developers can share their code with peer developers in an easy way

- Multiple developers can work in a collaboratively way.
- Parallel Development.
- We control to provide access control like,
    1. who can read code?
    2. who can modify the code?
- Every change should be tracked.

(We should not overwrite code, we have to maintain every version)

How Version Control System Works ?:

- Version control enables multiple people simultaneously work on a single project. Each person edits his or her copy of the files & chooses when to share those changes with the rest of the team.
- The version Control system is always applicable for files that contain source code.



- Assume, we have one directory where we will develop all files which contain source code first file is A.java, the second is B.java & third is C.java. For these files, version control is not applicable, If we want a Version Control system applicable we required Repository. So we send these files to the repository for that we required program.

  Working Directory:
  Where developers are required to create/modify files. (here version control is not applicable.)

  Repository:
  In this, the data will be stored in the form of versions. where we have to store files & metadata. (here version control is applicable.)

  Commit:
  The process of sending files from the working directory to the repository. Every commit is associated with a unique ID.

  Checkout:
  The process of sending files from the repository to the working directory.

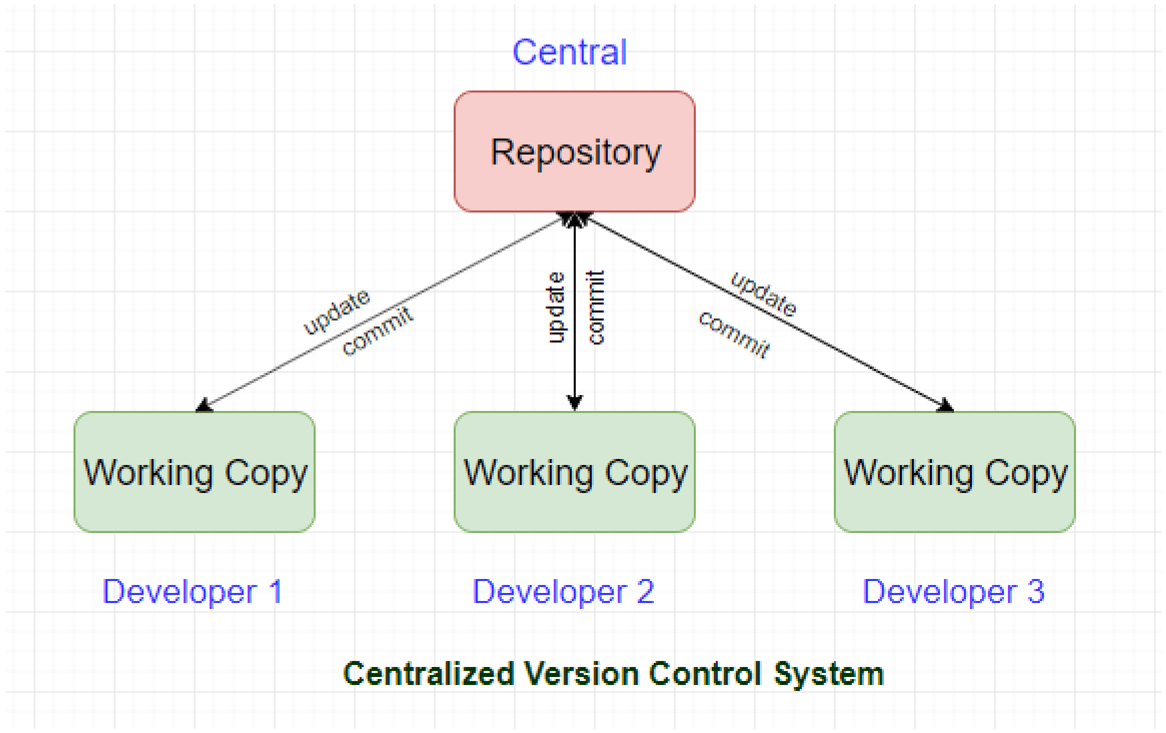
Types of Version Control Systems:

1. Centralized Version Control System
2. De-Centralized Version Control System

1. Centralized Version Control System:

- The centralized version control system contains only one central repository, Every developer is required to connect with that to continue their work.
- Here, Version control is happening in only one place which is called Central Repository.

- For example, CVS, SVN, perforce, ClearCase, TFS, etc, all these tools are based on Centralized Version control only.



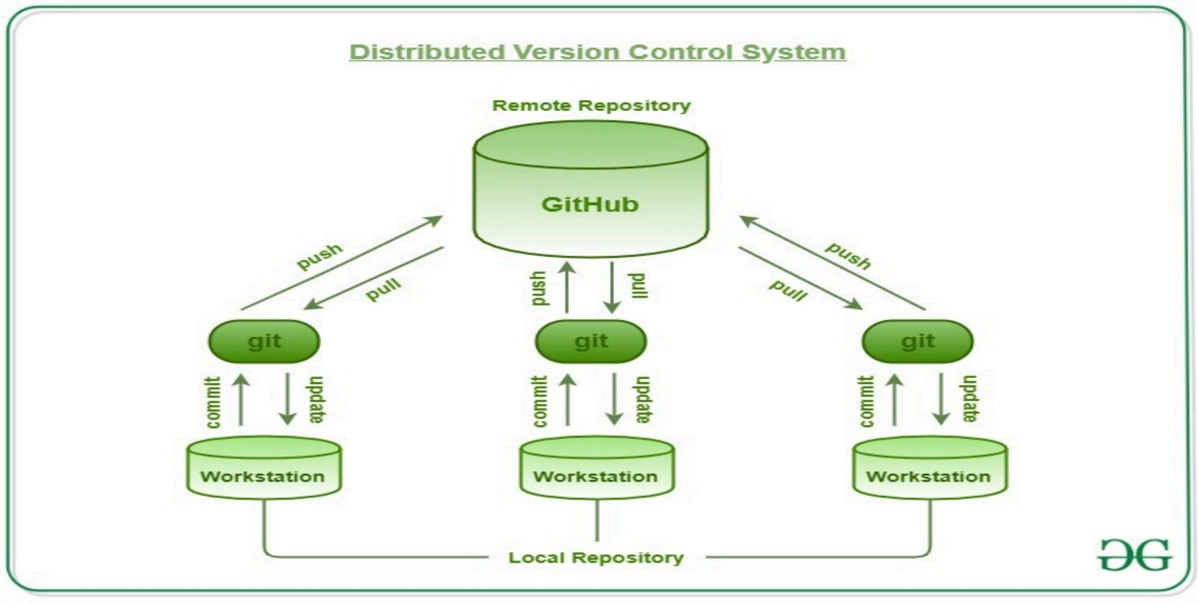
2. De-Centralized / Distributed Version Control System (With Remote Repository):

- Here, the Repository is distributed, If four developers are there, then 4 local repositories will be there on their machine.

Advantages:

- All commit and checkout operations will be performed locally and hence speed is more.

- If a developer (1) wants to share code with the developer (2) we can perform a push operation.
- commit and checkout operations will be performed locally between the working directory and the repository, To perform these **operations network is not required.**
- push and pull will be performed between two repositories. These are remote operations, the network must be there.
- The main purpose of the Remote repository is to share our code to peer developers.
- The final code is delivered in Remote Repository but we won't develop any code in the remote repository.



Push :
The process of sending files from our repository to others' repositories.

Pull:
The process of getting files from other repositories to the local repository.

For example, GIT, and Mercurial..all these tools are available based on the Distributed version control system.

Examples of Remote Repository Server: Github, Gitlab, Bit Bucket, etc..,

GIT

GIT:

- GIT is a set of repositories.
- Git is a free & open source, It is developed based on Distributed Version Control System tool.
- It is designed to handle everything from small to very large projects with speed & efficiency.
- Linus Torvalds also develops GIT.

Features of GIT (Why GIT):

1. It is based on the Distributed Model
2. **Staging Area:**
    - It is also known as Index Area
    - In GIT, Commit is 2 steps. First, we have to add files to the staging area & then we have to commit from that staging area.
    - The staging area is virtual, It is a virtual layer between the working directory & local repository.
    - The advantage of a staging area is we can cross-check or double-check our changes before committing. If everything is fine then we can commit.

3. **Branching and Merging:**

- Branching: We can create and work on multiple branches simultaneously and all these branches are isolated (independent) from each other. It enables multiple workflows. (For example: If you are working on one project and you got recruited for another project in that case, you can create a branch & you can work on it.

- Merging: We can merge multiple branches in a single branch.

4. It is freeware & Open source.
5. It provides support for multiple platforms (Linux, windows, mac).

The architecture of GIT:

GIT has two types of Repositories:

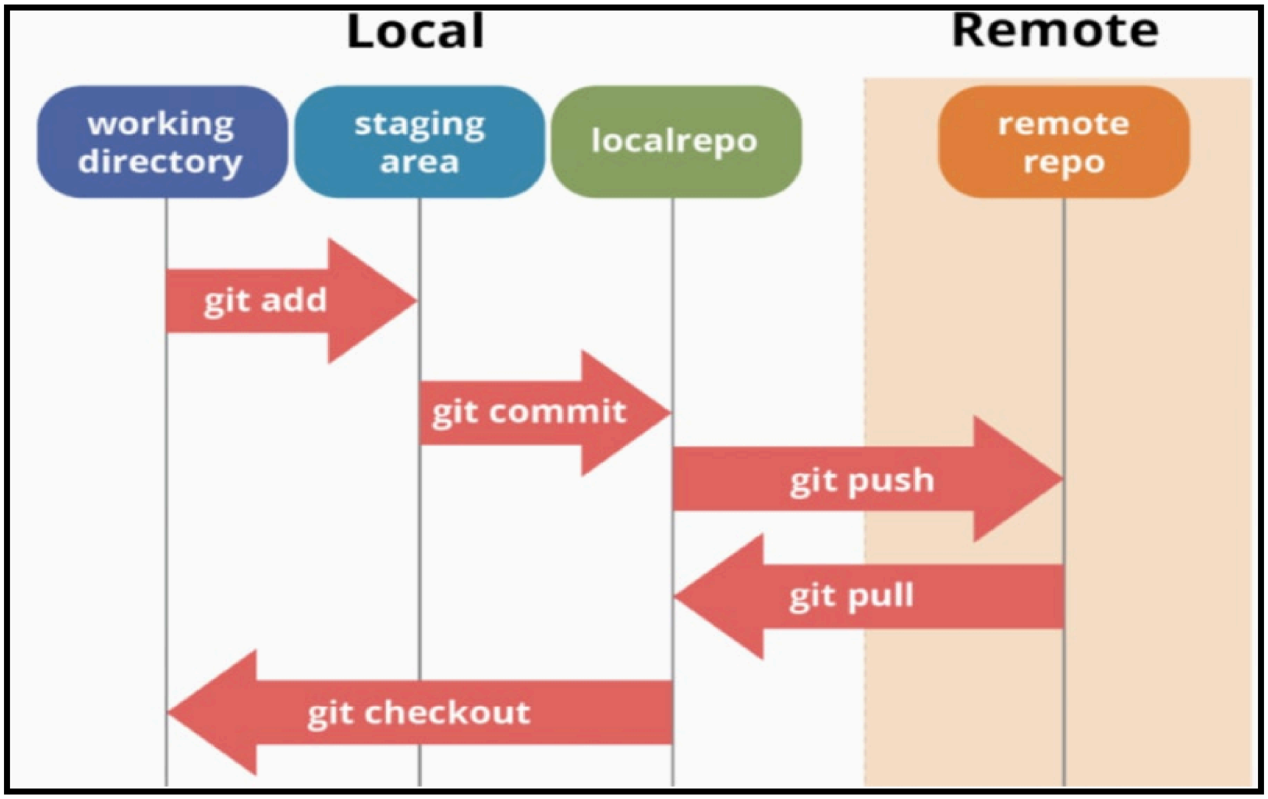
1. **Local Repository:** It is stored on your computer.

2. **Remote Repository:** It is stored on some remote computer.

GIT HUB:

GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code.
It is helpful when more than just one person is working on a project. For example, a software development team wants to build a website and everyone has to update their codes simultaneously while working on the project. In this case, GitHub helps them to build a centralized repository where everyone can upload, edit and manage the code files.

GIT WORKFLOW:



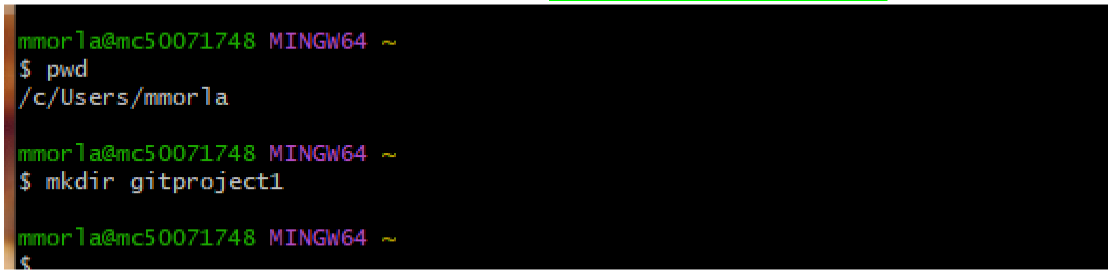
GIT Basic Commands:

- git init >> To create an empty git repository
- git status >> It shows the current status of all files in each area, like untracked files, modified files, staged files., etc
- git add >> To add files from the working directory to the staging area.
- git add. >> To add all files in the current working directory.
- git commit -m “commit message” >> To add files from the staging area to the Local repo.
- git push >> To add files from the Local repo. to Remote repo.

- git clone >> To create a new local repository from the remote repository (exact copy).
- git pull >> To get updated files from the remote repository to the local repo.

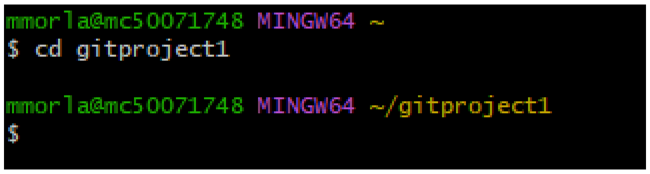
Uploading files on GitHub by Git bash

Step1: (Create working directory) **mkdir <foldername>**



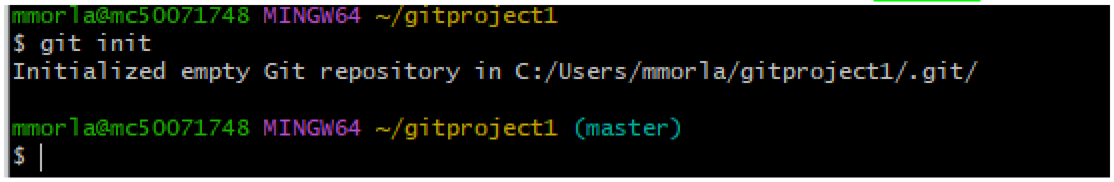
```
mmorla@mc50071748 MINGW64 ~
$ pwd
/c/Users/mmorla

mmorla@mc50071748 MINGW64 ~
$ mkdir gitproject1

mmorla@mc50071748 MINGW64 ~
$
```

Step2: (Go to the working directory) cd <foldername>



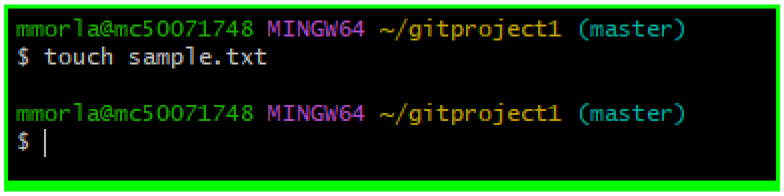
```
mmorla@mc50071748 MINGW64 ~
$ cd gitproject1

mmorla@mc50071748 MINGW64 ~/gitproject1
$
```

Step 3: (Create a Local repo. in the working directory) **git init**



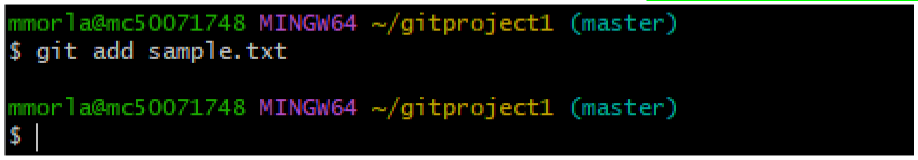
```
mmorla@mc50071748 MINGW64 ~/gitproject1
$ git init
Initialized empty Git repository in C:/Users/mmorla/gitproject1/.git/

mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ |
```

Step 4: (Create a file) **touch sample.txt**



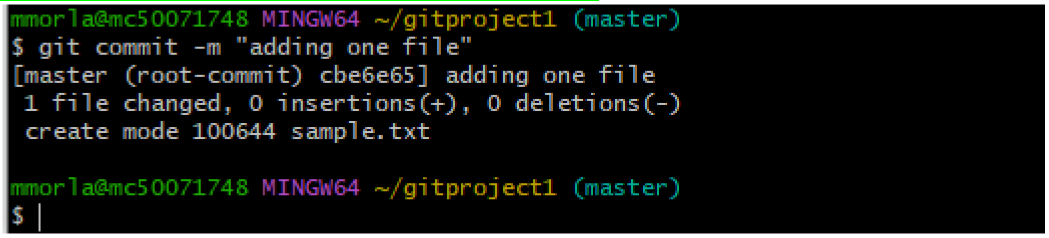
```
mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ touch sample.txt

mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ |
```

Step 5: (Add the files to staging area) **git add <filename>**



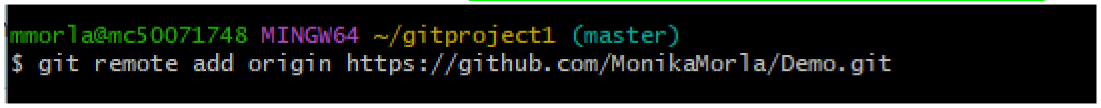
```
mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ git add sample.txt

mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ |
```

Step 6: (Move the files from the staging area to the local repo.)
git commit -m “commit message”



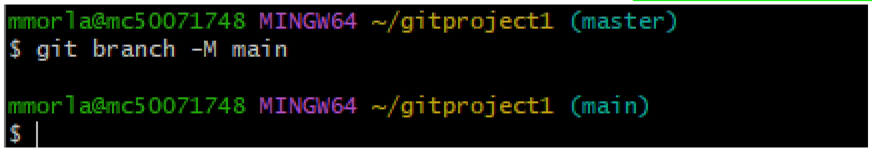
```
mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ git commit -m "adding one file"
[master (root-commit) cbe6e65] adding one file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 sample.txt

mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ |
```

Step 7: (Copy the URL link) **git remote add origin “copy link”**



```
mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ git remote add origin https://github.com/MonikaMorla/Demo.git
```

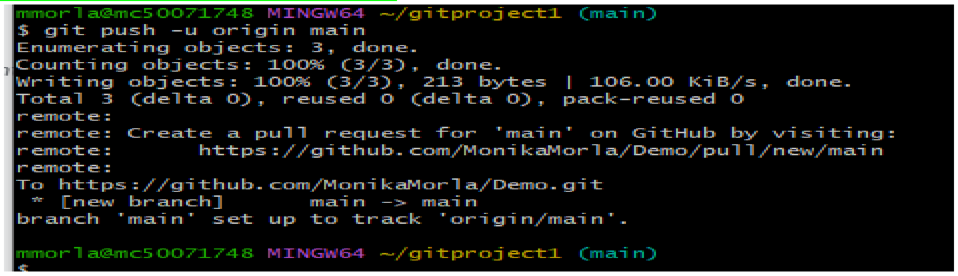
Step 8: (Change the branch to main) **git branch -M main**



```
mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ git branch -M main

mmorla@mc50071748 MINGW64 ~/gitproject1 (main)
$ |
```

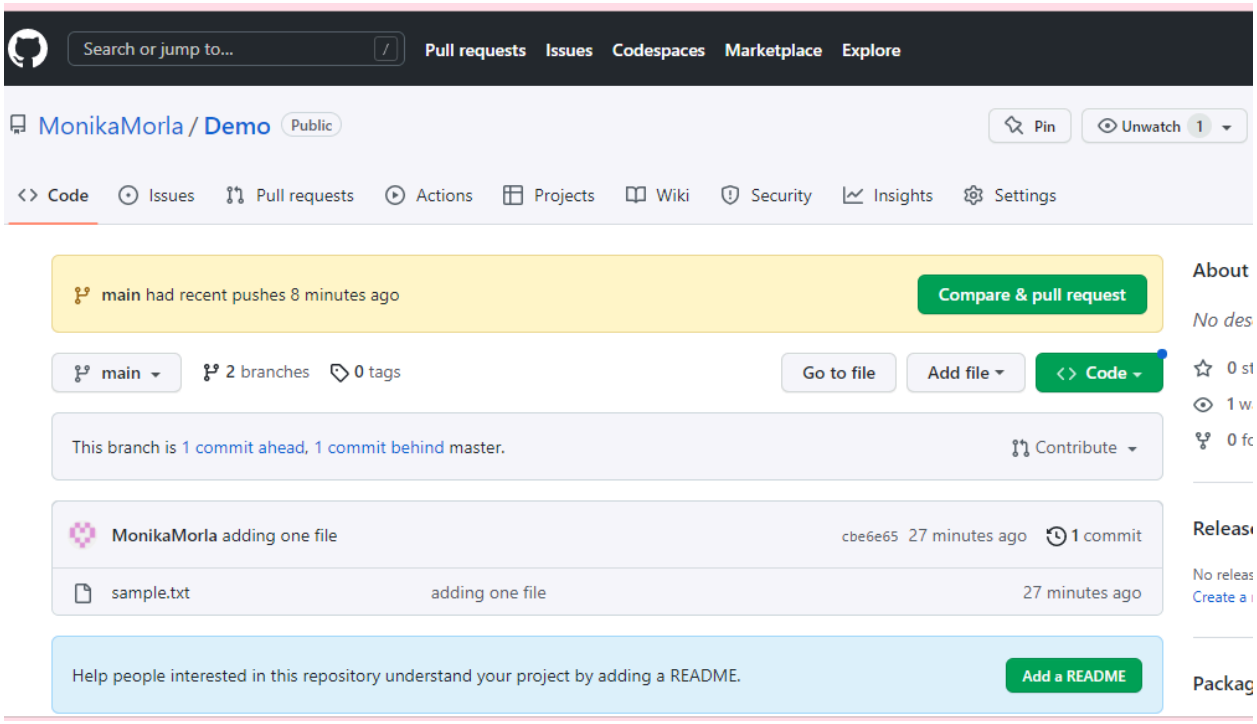
Step 9: (Move the files from the Local repo. to the Remote repo.)

git push -u origin main



```
mmorla@mc50071748 MINGW64 ~/gitproject1 (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 213 bytes | 106.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'main' on GitHub by visiting:
remote:      https://github.com/MonikaMorla/Demo/pull/new/main
remote:
To https://github.com/MonikaMorla/Demo.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

mmorla@mc50071748 MINGW64 ~/gitproject1 (main)
$
```

Step10: Now we can check in GIT HUB



MonikaMorla / Demo Public

main had recent pushes 8 minutes ago

This branch is 1 commit ahead, 1 commit behind master.

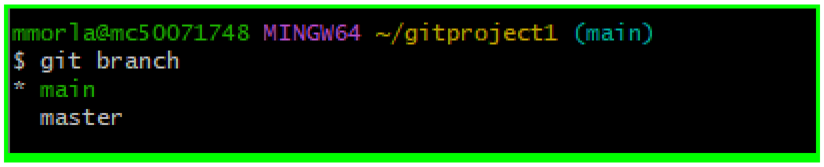
MonikaMorla adding one file cbe6e65 27 minutes ago 1 commit

sample.txt adding one file 27 minutes ago

Help people interested in this repository understand your project by adding a README.

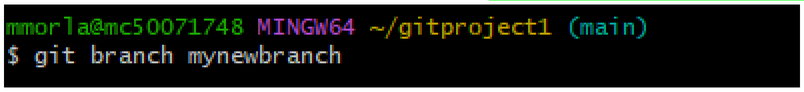
Creating Branches and Merging:

Step1: Check branches >> git branch

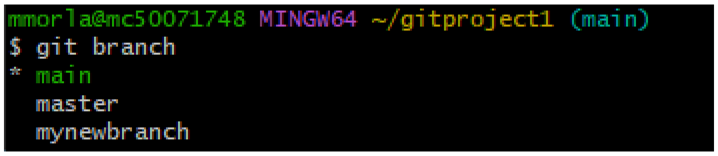


```
mmorla@mc50071748 MINGW64 ~/gitproject1 (main)
$ git branch
* main
  master
```

Step2: Create New branch >> git branch filename

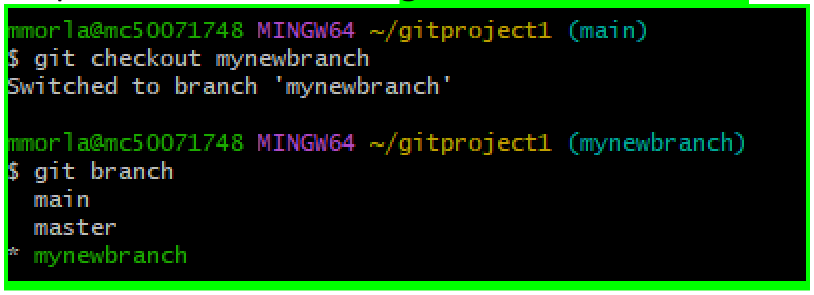


```
mmorla@mc50071748 MINGW64 ~/gitproject1 (main)
$ git branch mynewbranch
```



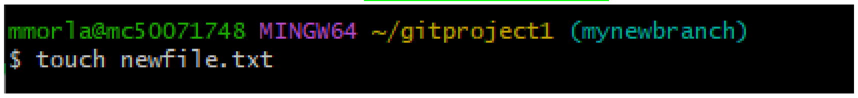
```
mmorla@mc50071748 MINGW64 ~/gitproject1 (main)
$ git branch
* main
  master
  mynewbranch
```

Step3: switch branch >> git checkout filename



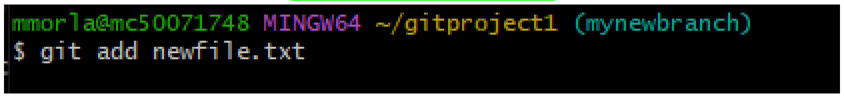
```
mmorla@mc50071748 MINGW64 ~/gitproject1 (main)
$ git checkout mynewbranch
Switched to branch 'mynewbranch'

mmorla@mc50071748 MINGW64 ~/gitproject1 (mynewbranch)
$ git branch
  main
  master
* mynewbranch
```

Step4: Create a file >> touch filename



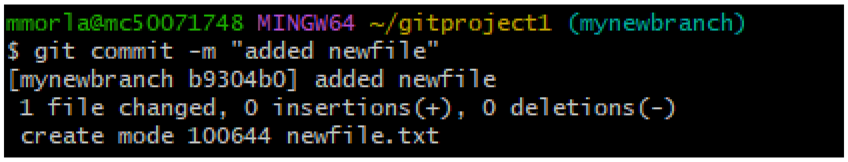
```
mmorla@mc50071748 MINGW64 ~/gitproject1 (mynewbranch)
$ touch newfile.txt
```

Step5: add file >> git add filename



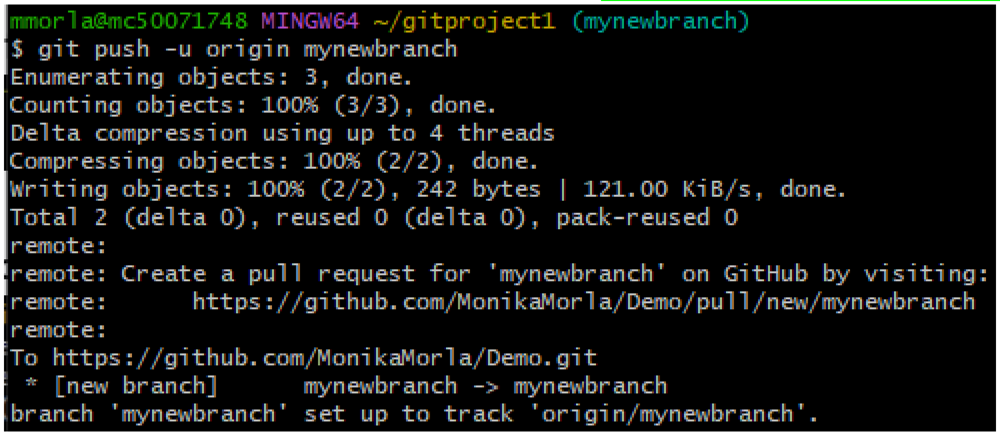
```
mmorla@mc50071748 MINGW64 ~/gitproject1 (mynewbranch)
$ git add newfile.txt
```

Step5: commit file >> git commit -m “commit message”



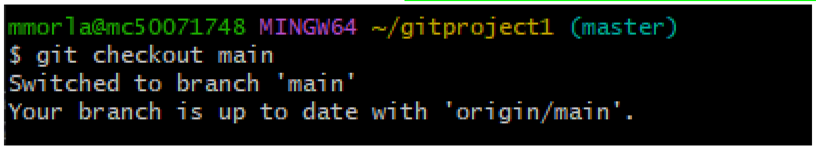
```
mmorla@mc50071748 MINGW64 ~/gitproject1 (mynewbranch)
$ git commit -m "added newfile"
[mynewbranch b9304b0] added newfile
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newfile.txt
```

Step 6: Move to the remote repo. >> git push -u origin branch name



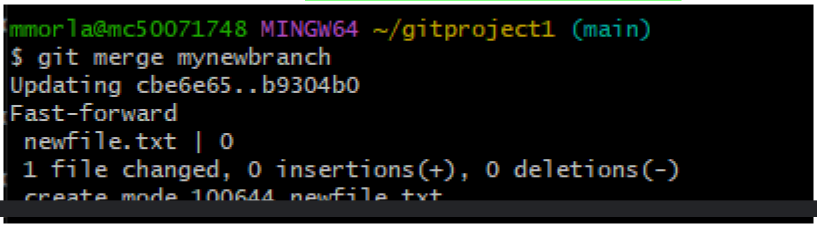
```
mmorla@mc50071748 MINGW64 ~/gitproject1 (mynewbranch)
$ git push -u origin mynewbranch
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 242 bytes | 121.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'mynewbranch' on GitHub by visiting:
remote:      https://github.com/MonikaMorla/Demo/pull/new/mynewbranch
remote:
To https://github.com/MonikaMorla/Demo.git
 * [new branch]      mynewbranch -> mynewbranch
branch 'mynewbranch' set up to track 'origin/mynewbranch'.
```

Step 7: switch branch >> git checkout branch-name

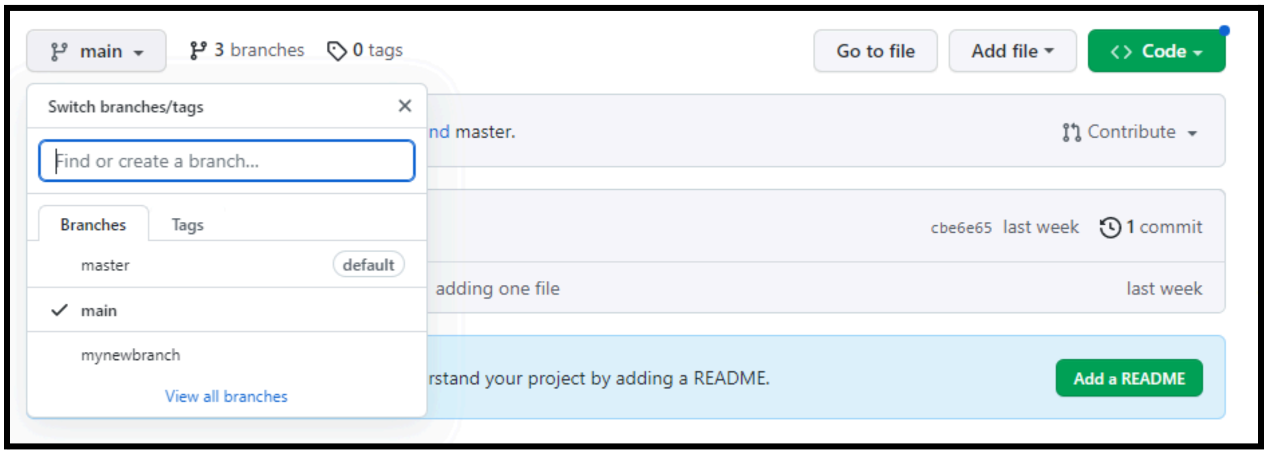


```
mmorla@mc50071748 MINGW64 ~/gitproject1 (master)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

Step 8: Merging >> git merge branchname



```
mmorla@mc50071748 MINGW64 ~/gitproject1 (main)
$ git merge mynewbranch
Updating cbe6e65..b9304b0
Fast-forward
 newfile.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newfile.txt
```


main 3 branches 0 tags Go to file Add file Code

Switch branches/tags

Find or create a branch...

Branches Tags

master default

✓ main

mynewbranch

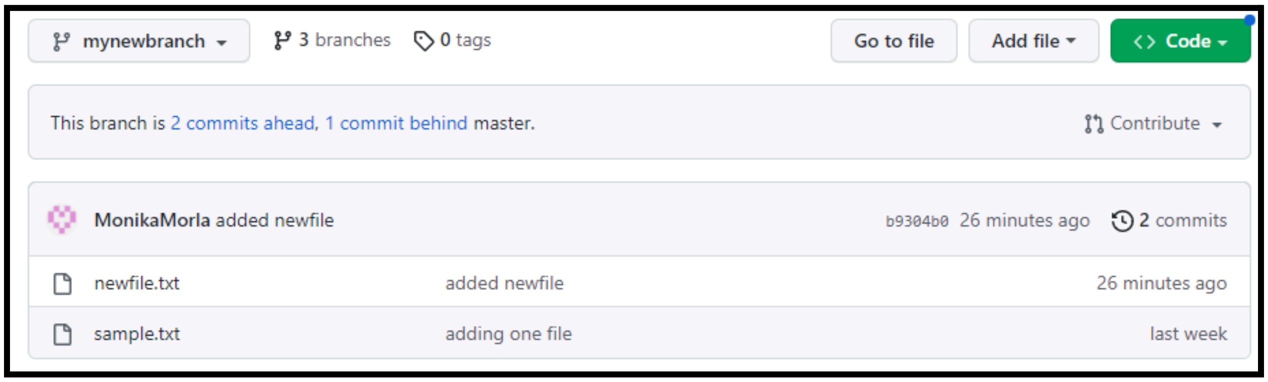
View all branches

nd master. Contribute

cbe6e65 last week 1 commit

adding one file last week

rstand your project by adding a README. Add a README


mynewbranch 3 branches 0 tags Go to file Add file Code

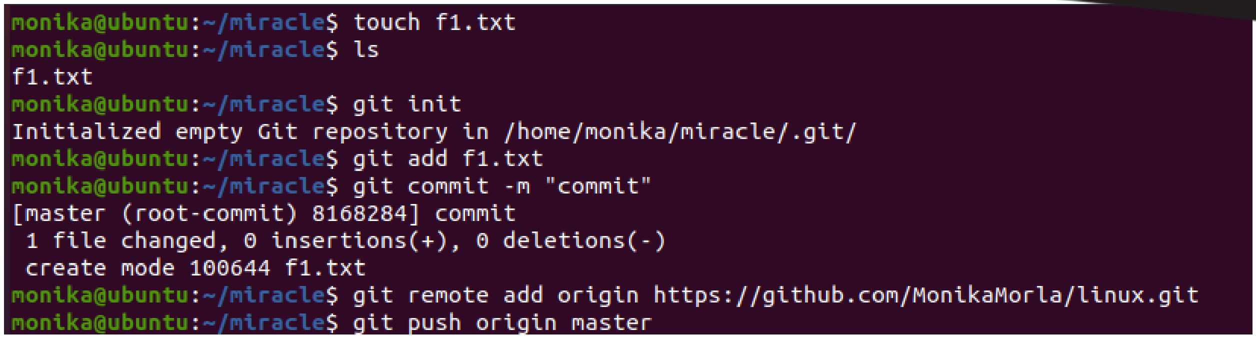
This branch is 2 commits ahead, 1 commit behind master. Contribute

MonikaMorla added newfile b9304b0 26 minutes ago 2 commits

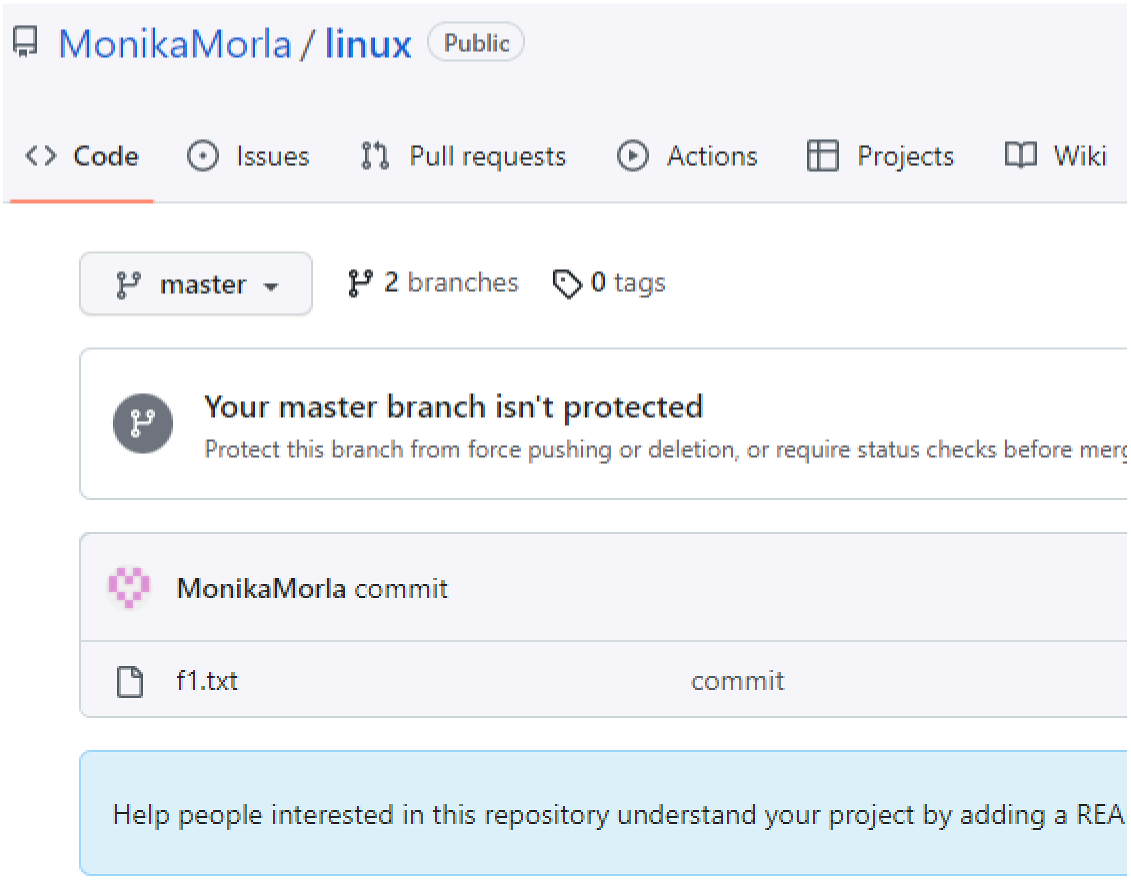
newfile.txt	added newfile	26 minutes ago
sample.txt	adding one file	last week

Uploading files in Github by using Ubuntu

- Step1: Create folder >> mkdir folder name
- Step2: cd folder name
- step3: touch file name
- step4: git init
- step5: git add filename
- step6: git commit -m “commit message”
- step7: git remote add origin <url>
- step8: git push origin master



```
monika@ubuntu:~/miracle$ touch f1.txt
monika@ubuntu:~/miracle$ ls
f1.txt
monika@ubuntu:~/miracle$ git init
Initialized empty Git repository in /home/monika/miracle/.git/
monika@ubuntu:~/miracle$ git add f1.txt
monika@ubuntu:~/miracle$ git commit -m "commit"
[master (root-commit) 8168284] commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f1.txt
monika@ubuntu:~/miracle$ git remote add origin https://github.com/MonikaMorla/linux.git
monika@ubuntu:~/miracle$ git push origin master
```



Your master branch isn't protected

Protect this branch from force pushing or deletion, or require status checks before merg

MonikaMorla commit

f1.txt commit

Help people interested in this repository understand your project by adding a REA

GIT Commands:

- git --version >> To know the version of the GIT
- git config >> It can be used for git configurations like username, mail id, etc
- git config –list >> To list out all git configurations.

- git config --global user.username <your username> >> To add a username.
- git config --global user.email <your email address> To add an email address.
-
- mkdir <folder_name> >> To create a Working Directory.
- cd <folder_name> >> To change the working path to the Working directory.
- git init >> To create a Local Repository.
- touch <filename> >> To create an empty text file
- ls >> To check if the file is created
- rm -fr <folder_name> >> To delete the working directory in Git.
- rm -fr .git >> To Delete the Local/Git Repository
- git status >> It shows the current status of all files in each area like untracked files, modified files, staged files., etc
- git status <filename> >> It shows the current status of particular files in each area like untracked files, modified files, staged files., etc
- git add . or git add * >> To add all the files in the staging area.
- git add <filename n> >> To add particular files in the staging area.
- git rm --cached <filename> >> To unstage a specific file. If the filename is not mentioned, how many files are staged; those all are unstaged when running this command.
- git restore --staged <filename> >> To unstage a specific file. If the filename is not mentioned, how many files are staged; those all are unstaged when running this command.
- git commit <filename> -m ”Your Commit Message” >> To add files from the staging area to the Local repo.
- git commit -m “Your commit message” >> To add all the files from the staging area to the Local repo.

- git log >> It shows the details of all commits like commit id, author name, mail id, timestamp, and commit message.
- git log <filename> >> It shows the details of Particular file commits like commit id, author name, mail id, timestamp, and commit message.
- git log --pretty=oneline >> It shows the details of Particular file commits like commit id, author name, mail id, timestamp, and commit message. (In a single line)
- git diff >> To find the differences in the content of a file (same file)like working directory & staging area, working directory & last commit, staging area & last commit
- git checkout -- <file name> >> To undo all recent changes in Working area for a particular file(before staging),
- git checkout – >> To undo all recent changes in your working area for all files.
- git remote add origin <Repository HTTPS URL> >> To connect to your remote repo through your git repo.
- git remote -v >> To check the Remote repositories connected to our Git repo
- git remote rename origin <Your_new_origin_name> >> To rename the origin
- git branch <branch name> >> To create a branch
- git branch -M <branch name>
- git branch -a To list all available branches.
- git branch -r To list all Remote Branches.
- git branch -D <branch_name> >> To delete the Branch Locally in your Git Repository.
- git show-branch >> To show branches with their recent commits.
- git push -u origin <branch_name> To push the code to the remote repository

- git push origin --delete <branch_name> To delete the branch Remotely or in the Central Repository. (If you run this command then only the branch will be deleted from your GitHub account.)
- git clone <Repository HTTPS URL> >> To get updated files from the remote repository to the local repo(exact copy)
- git pull origin <branch_name> >> To get updated files from the remote repository to the local repo.(updated copy)
- git merge <branch_name> To merge the code in one branch to another branch
- git revert <commit ID> >> This command reverts the commit(delete the given commit changes) and it commits the code with a new commit ID.
- git revert -n <Commit_ID> >> If you want to commit the file manually, run the below command.
- git remote remove <repo_name> or <repo_url> >> To disconnect from Remote Repository.