- **WHAT IS GIT?**

  Git is a distributed version control system which is on remote server. It is software that should be installed.

- **WHAT IS REPOSITORY?**

  It is used to store the data or files to share commonly.

- **WHAT IS VERSION CONTROL?**

  When we create a project we will make multiple updates if we want to go back to the previous version and use it then it is called as version control. To achieve this we need to use some software that is called version control system.

- **WHAT ARE THE TYPES OF VERSION CONTROL SYSTEMS?**

  1. LOCAL VERSION CONTROL SYSTEM:

     Creating multiple folders or versions in our own local machine

     DISADVANTGES OF LCS:

     a) If we have a team and all want to work on the same project it is not possible
     b) If machine crashes and all data will be lost.

  2. CENTRALIZIED VERSION CONTROL SYSTEM:

     It contains just one repository globally and every user needs to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.

  3. DISTURBTED VERSION CONTROL SYSTEM:

     It contains multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository.

- **ARCHITECUTRE OF GIT?**

  1. WORKING SPACE:

     The present folder or project you're working on it where we can edit, modify project related files. It is the single checkout of one version of the project. All the files in workspace are visible to all the directories.

  2. STAGING AREA:

     It is a file contained in your git directory that stores all the information about what will go to the next commit. Git Add files are moved from work space to staging area where changes are saved.

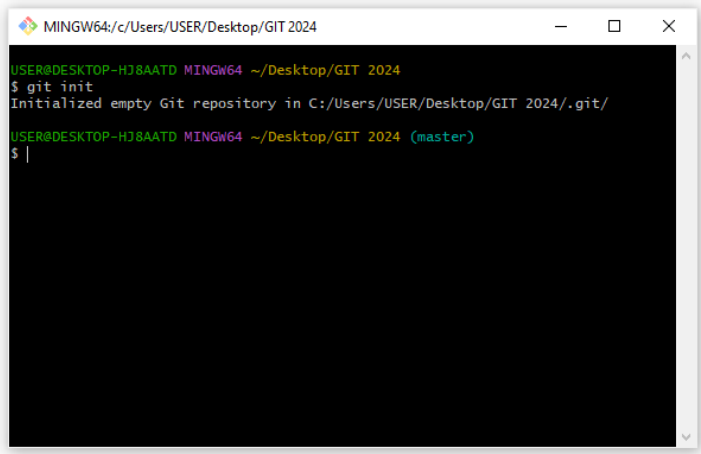
3. LOCAL REPO / GIT REPO :

It is where git stores the data and object database for your project. GIt Commit, files will be added to local/git repo & then we track the file versions. Commit Id are created here.

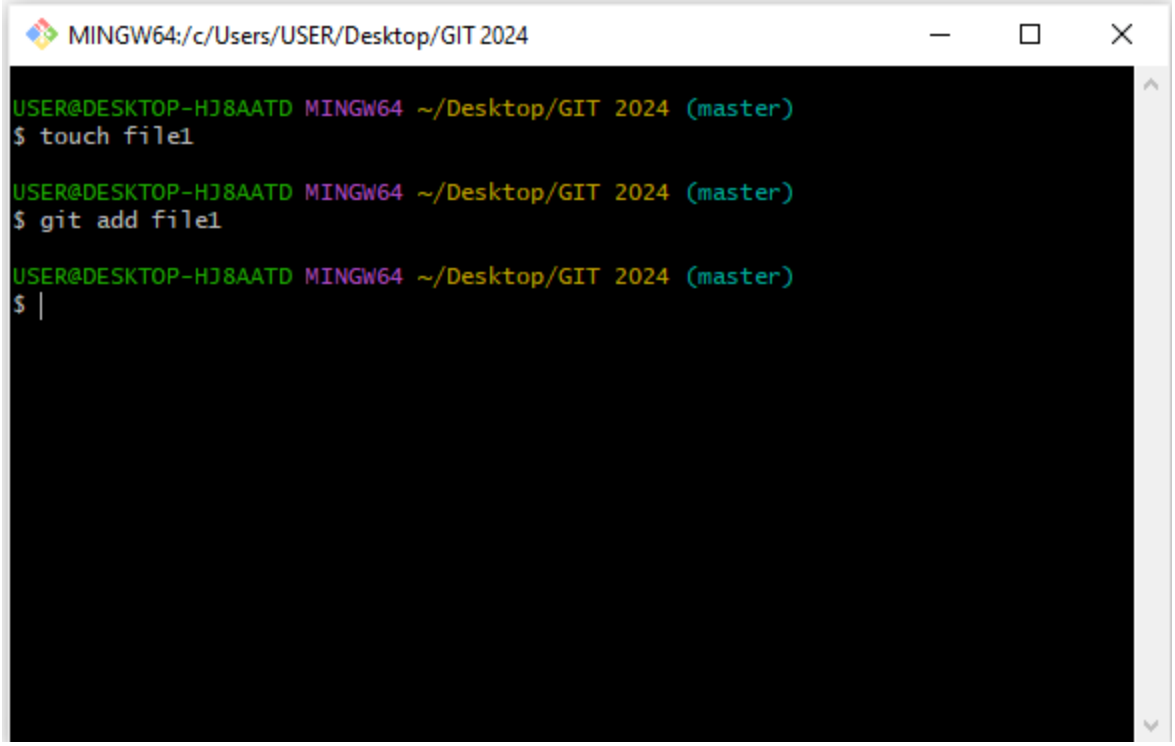
- **<u>TO INSTALL GIT :</u>**
  1. sudo yum install git
  2. mkdir git
  3. git init
  4. ls -a

- **git init:**

  This command is used to initialize the respective directory as git repository

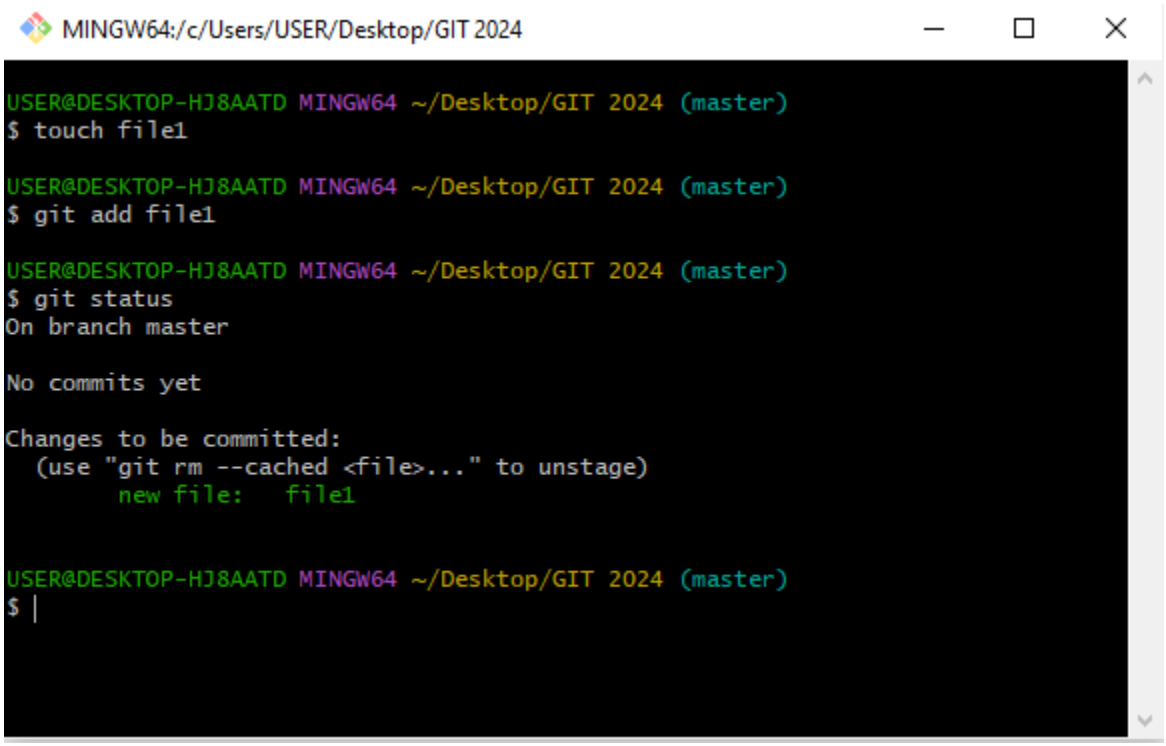


- **<u>TO CREATE AND ADD FILES:</u>**
  a) touch file1
  b) git add file1



- **git status:**

  It will show whether files are in workspace or staging are on in git repo



- **git commit :**

  This command commits the staged changes to the local repository.

  git commit –m “adding file1”

```
MINGW64:/c/Users/USER/Desktop/GIT 2024
$ touch file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ git add file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1


USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ git commit -m "adding file1"
[master (root-commit) b5b0fc9] adding file1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ |
```

- **git log:**

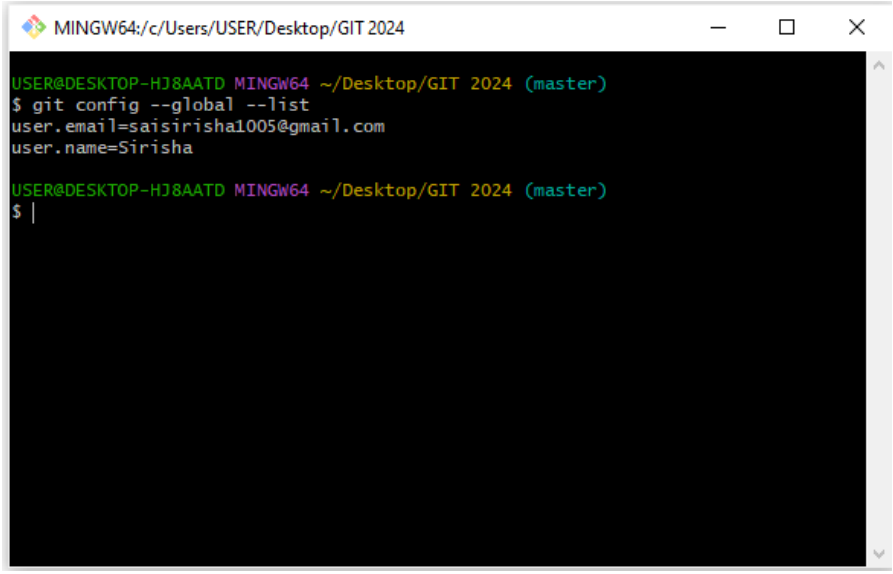
  It gives history of file or repo that we have committed.

```
MINGW64:/c/Users/USER/Desktop/GIT 2024

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1


USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ git commit -m "adding file1"
[master (root-commit) b5b0fc9] adding file1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ git log
commit b5b0fc9328d65b6800325525041d684ab4e01dbb (HEAD -> master)
Author: Sirisha <saisirisha1005@gmail.com>
Date:   Mon Jul 29 13:23:44 2024 +0530

    adding file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ |
```

git log

- **TO CHECK WHETHERE NAME AND EMAIL ID CONFIG**

  git config --global –list



- **BRANCHING:**

  Branching is a parallel development; teams can work on same piece of code on different branches parallel and later inherit by merging.

- **WHY WE NEED BRANCHING?**

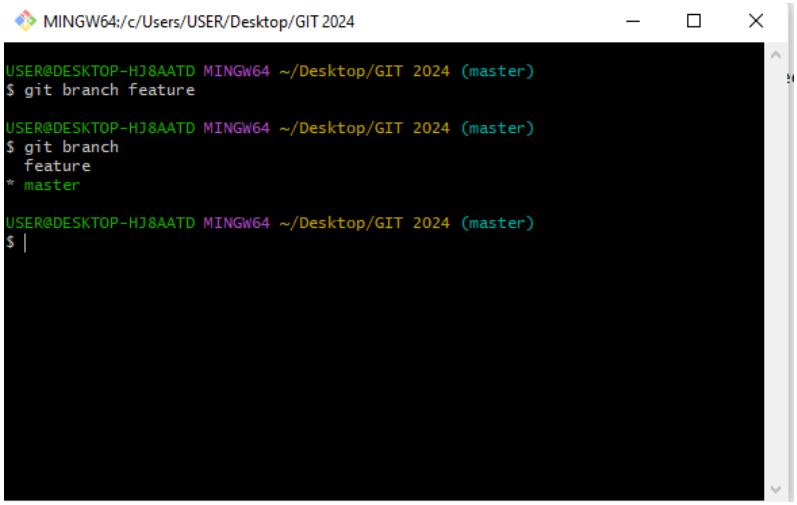
  To develop new features we go for branching.

- **HOW TO CREATE A BRANCH?**

  git branch <branch name>

- **HOW TO CHECK THE LIST OF BRANCHES?**

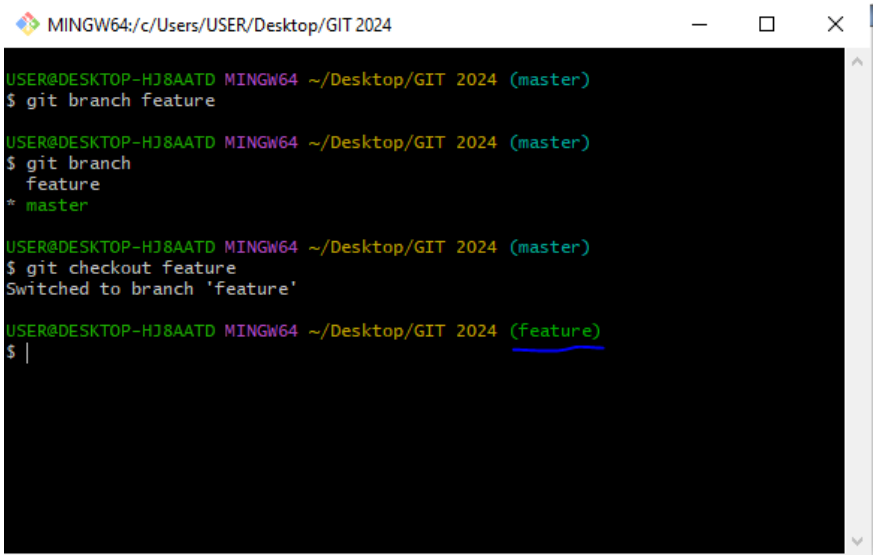
  git branch



"*" indicates the current branch.

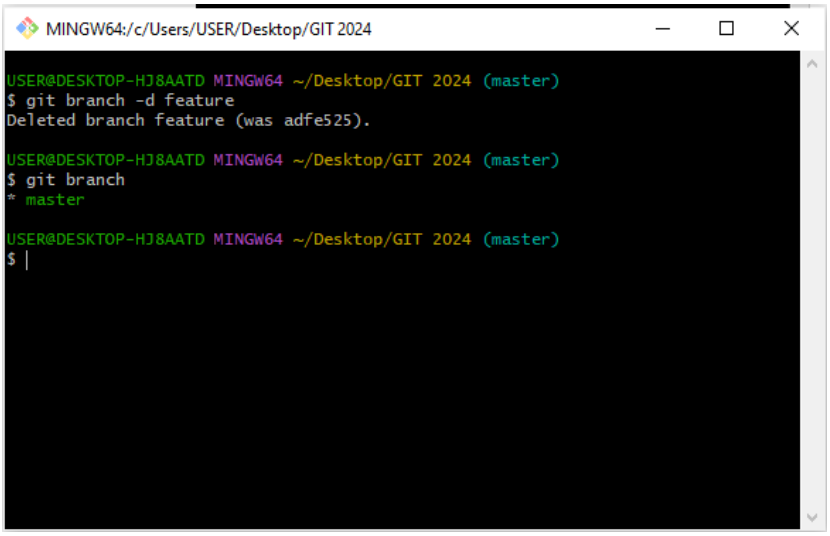
- **HOW TO ENTER INTO A BRANCH?**

  git checkout <branch name>



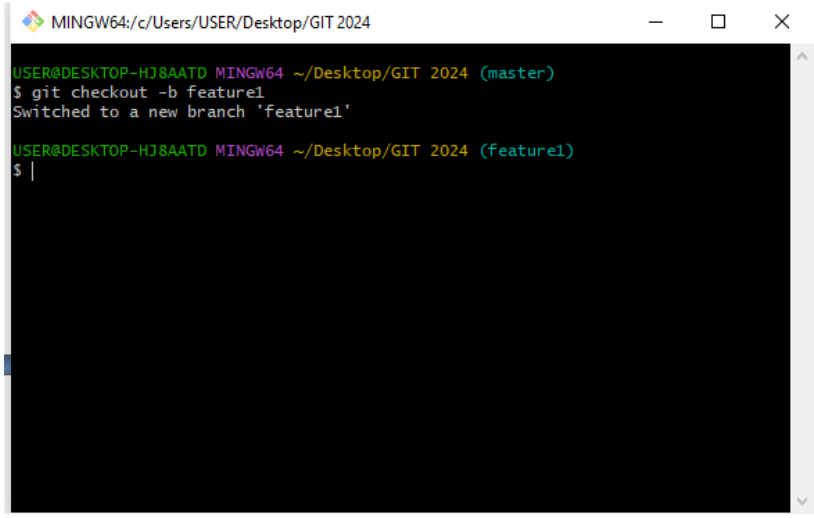
- **HOW TO DELETE A BRANCH?**

  git branch –d <branch name>



- **HOW TO CREATE A BRANCH AND ENTER INTO THE BRANCH?**

  git checkout –b <branch name>

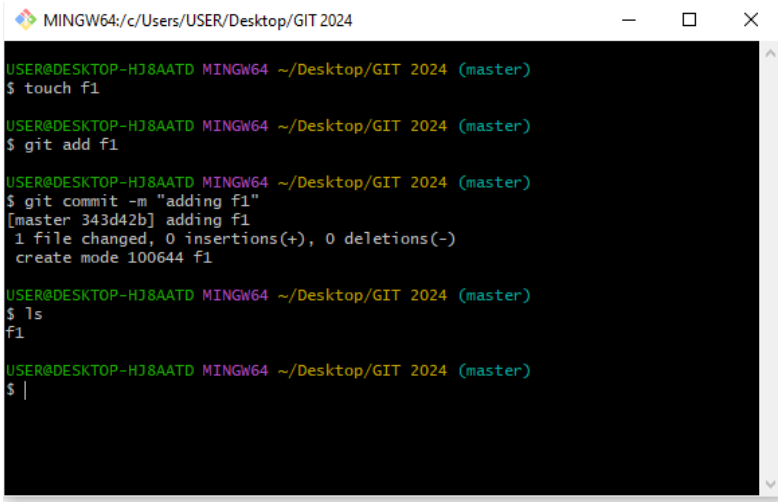


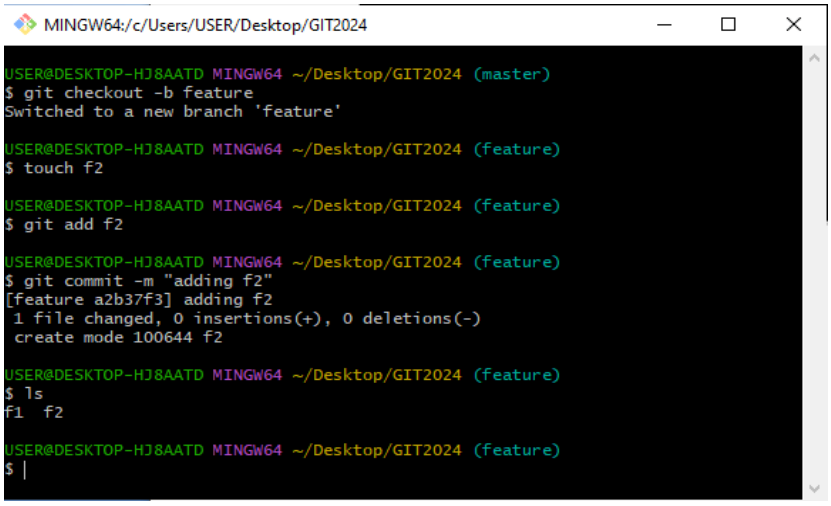
- **MERGING:**

It is used to merge specified branch to checkout branch.

git merge <branch name>

Step 1:





Step 3:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git checkout master
Switched to branch 'master'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ ls
f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git merge feature
Updating 5be3c16..a2b37f3
Fast-forward
 f2 | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ ls
f1  f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

- There are two types of merges.
    a) **Fast-forward merge**: If we merge the branch into the master and no commits are made in master branch then it is called fast-forward merging.

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git checkout master
Switched to branch 'master'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ ls
f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git merge feature
Updating 5be3c16..a2b37f3
Fast-forward
 f2 | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ ls
f1  f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

b) **RECURSIVE STRATEGY:** If we made changes in the master branch as well as feature branch if we try to merge the feature branch into master branch these two changes has to be merge into another new commit.

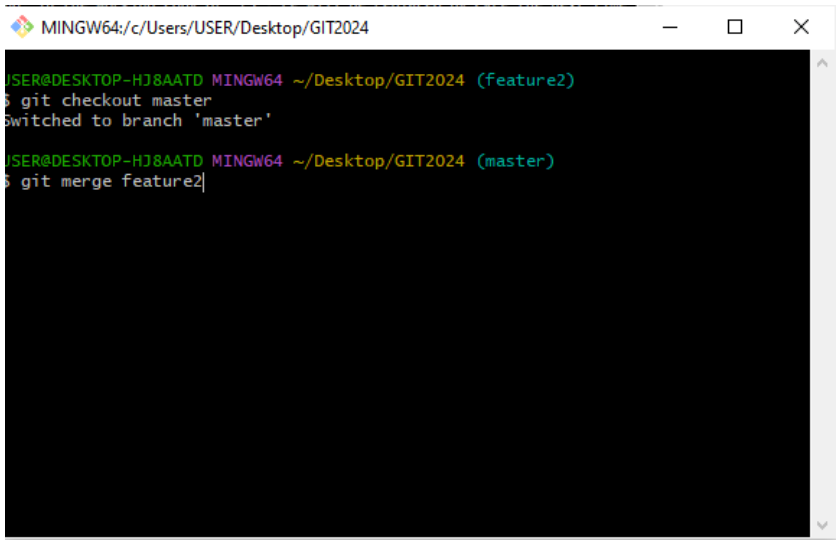
Step 1:

```
MINGW64:/c/Users/USER/Desktop/GIT2024
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git branch feature2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add f1
warning: in the working copy of 'f1', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "updated f1"
[master 7da2c98] updated f1
 1 file changed, 1 insertion(+)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
7da2c98 (HEAD -> master) updated f1
a2b37f3 (feature2, feature) adding f2
5be3c16 adding f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

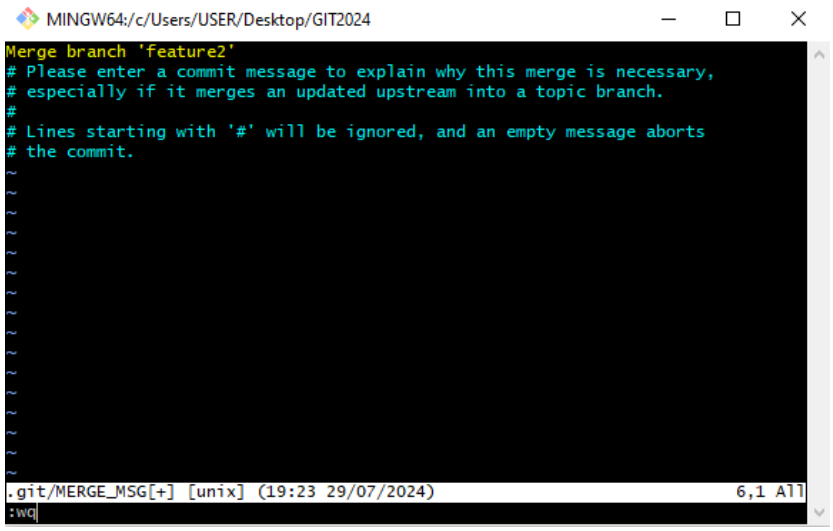
Step 2:

```
MINGW64:/c/Users/USER/Desktop/GIT2024
Switched to branch 'feature2'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature2)
$ vi f3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature2)
$ git add f3
warning: in the working copy of 'f3', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature2)
$ git commit -m "adding f3"
[feature2 62e8858] adding f3
 1 file changed, 1 insertion(+)
 create mode 100644 f3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature2)
$ git log --oneline
62e8858 (HEAD -> feature2) adding f3
a2b37f3 (feature) adding f2
5be3c16 adding f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature2)
$ |
```

Step 3:



Step 4:



- In this case a new commit will be added when we try to merge the feature branch with the main branch.
-

Step 5:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature2)
$ git checkout master
Switched to branch 'master'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git merge feature2
Merge made by the 'ort' strategy.
 f3 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 f3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
3a2fe8a (HEAD -> master) Merge branch 'feature2'
62e8858 (feature2) adding f3
7da2c98 updated f1
a2b37f3 (feature) adding f2
5be3c16 adding f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

- **MERGE CONFLICT:** It occurs when same piece of code is been worked on different branches and when we try to merge it the conflict occurs.
  - There are two types of merge conflicts
    1. Two developers modify the same line of code in different branches.
    2. When a file is deleted in one branch and modifies it in another branch
    3. When multiple branches are being merged with the changes scattered across various files and lines.

Step 1:

```
MINGW64:/c/Users/USER/Desktop/GIT 2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024
$ git init
Initialized empty Git repository in C:/Users/USER/Desktop/GIT 2024/.git/

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ touch f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ vi f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ git add f1
warning: in the working copy of 'f1', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ git commit -m "adding f1"
[master (root-commit) a6fd0ee] adding f1
 1 file changed, 1 insertion(+)
 create mode 100644 f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ |
```

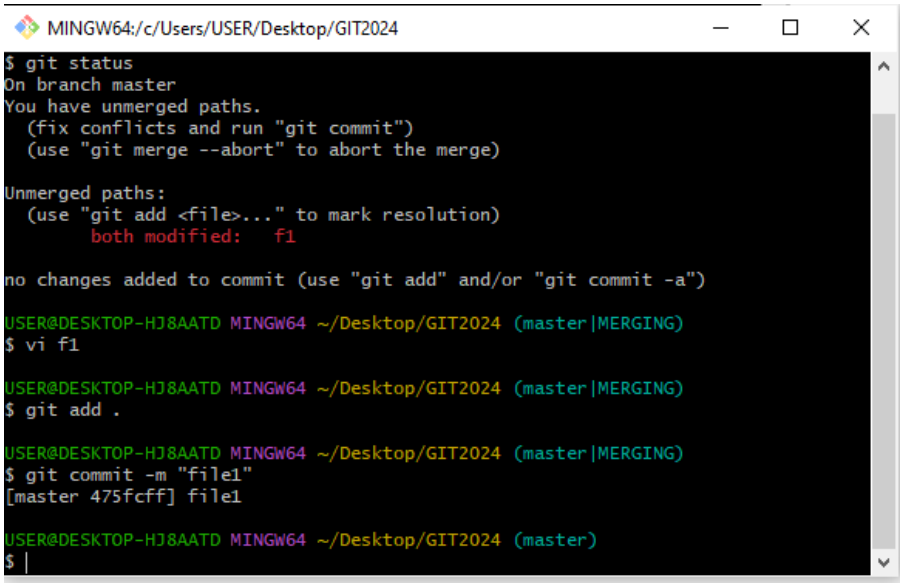
Step 2:

```
MINGW64:/c/Users/USER/Desktop/GIT 2024
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (master)
$ git checkout feature
Switched to branch 'feature'
M       f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (feature)
$ ls
f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (feature)
$ vi f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (feature)
$ git add f1
warning: in the working copy of 'f1', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (feature)
$ git commit -m "updated f1"
[feature 88ddced] updated f1
 1 file changed, 2 insertions(+), 1 deletion(-)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT 2024 (feature)
$ |
```

Step 3:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git checkout master
Switched to branch 'master'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "changes done in master"
[master 8a5dd22] changes done in master
 1 file changed, 1 insertion(+)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git merge feature
Auto-merging f1
CONFLICT (content): Merge conflict in f1
Automatic merge failed; fix conflicts and then commit the result.

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master|MERGING)
$ |
```

- **STEPS TO RESOLVE MERGING CONFLICTS:**
  1) git status
  2) open the file which we have modified(vi f1)
  3) delete the extra lines what are not required
  4) then add the file in staging area(git add f1)
  5) commit the file after making changes(git commit –m “”)



- In any case if we don't want to merge the particular file in master branch after getting merge conflict we can abort the merge.

  git merge --abort

- **<u>Tagging:</u>** It is a name given to the set of versions of files and directories, it is easy to remember the tag names and it also indicates the milestone of the project.
  1. To create a tag

     git tag <tag name>
  2. Switch into the tag

     git checkout <tag name>
  3. To list all tags

     git tag
  4. To delete the tag

     git tag –d <tag name>

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git tag v1.1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git tag
v1.0
v1.1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git checkout v1.0
Note: switching to 'v1.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 56c0eaa adding f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 ((v1.0))
$ git tag -d v1.1
Deleted tag 'v1.1' (was 56c0eaa)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 ((v1.0))
$ git tag
v1.0

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 ((v1.0))
$ |
```

- **<u>Difference b/w the tag and branch:</u>**
  - tag is a name given to a set of files.
  - Branching is for parallel development.

- **<u>REBASE:</u>**
  - It is similar to merge and rewrites the commits history
  - It is used to clean up our local history
  - It is the advanced command used rarely
- **<u>COMMON PLACES WHERE WE CAN USE REBASE</u>**
  - Cleaning up your commits before sharing your branch
  - Pulling changes from another branch without merge
- **<u>WHERE WE SHOULDN'T USE REBASE</u>**
  - When the branch is public when it is shared to all the developers
  - Most of them prefer merge rather than rebase.

Step 1:

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ touch file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 'file1', LF will be replaced by CRLF the next ti
me Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "file1 added in master"
[master (root-commit) f0104e1] file1 added in master
 1 file changed, 2 insertions(+)
 create mode 100644 file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
f0104e1 (HEAD -> master) file1 added in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

Step 2:

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git branch feature

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 'f2', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "file2 added in master"
[master 1c30116] file2 added in master
 1 file changed, 1 insertion(+)
 create mode 100644 f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi f3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
        add

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 'f3', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "file3 added in master"
[master c95b9b7] file3 added in master
 1 file changed, 1 insertion(+)
 create mode 100644 f3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
c95b9b7 (HEAD -> master) file3 added in master
1c30116 file2 added in master
f0104e1 (feature) file1 added in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

Step 3:

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git checkout feature
Switched to branch 'feature'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ vi f4

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git add .
warning: in the working copy of 'f4', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git commit -m "file4 added in feature"
[feature f4cd520] file4 added in feature
 1 file changed, 1 insertion(+)
 create mode 100644 f4

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ vi f5

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git add .
warning: in the working copy of 'f5', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git commit -m "file5 added in feature"
[feature e5fcd9b] file5 added in feature
 1 file changed, 1 insertion(+)
 create mode 100644 f5

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git log --oneline
e5fcd9b (HEAD -> feature) file5 added in feature
f4cd520 file4 added in feature
f0104e1 file1 added in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ |
```

Step 4:

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git checkout master
Switched to branch 'master'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
c95b9b7 (HEAD -> master) file3 added in master
1c30116 file2 added in master
f0104e1 file1 added in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git rebase feature
Successfully rebased and updated refs/heads/master.

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
9291aa4 (HEAD -> master) file3 added in master
f53bb0d file2 added in master
e5fcd9b (feature) file5 added in feature
f4cd520 file4 added in feature
f0104e1 file1 added in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

- In the above example when we do rebase to main branch from feature branch the commit ids has been deleted and new ids has been generated. Rebase clears the history.

- **<u>IN THE SAME WAY WE WILL DO FOR MERGE:</u>**

Step 1:

```
MINGW64:/c/Users/USER/Desktop/GIT2024
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024
$ git init
Initialized empty Git repository in C:/Users/USER/Desktop/GIT2024/.git/

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 'f1', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "adding f1 in master "
[master (root-commit) a24e21b] adding f1 in master
 1 file changed, 1 insertion(+)
 create mode 100644 f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
a24e21b (HEAD -> master) adding f1 in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

Step 2:

```
MINGW64:/c/Users/USER/Desktop/GIT2024
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git branch feature

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 'f2', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "f2 added in master"
[master 91942a3] f2 added in master
 1 file changed, 2 insertions(+)
 create mode 100644 f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi f3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 'f3', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "f3 created in master"
[master b3763d8] f3 created in master
 1 file changed, 2 insertions(+)
 create mode 100644 f3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
b3763d8 (HEAD -> master) f3 created in master
91942a3 f2 added in master
a24e21b (feature) adding f1 in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

Step 3:

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git checkout feature
Switched to branch 'feature'
A       f4

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git commit -m "f4 is added in feature"
[feature 0cd959c] f4 is added in feature
 1 file changed, 2 insertions(+)
 create mode 100644 f4

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ vi f5

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git add .
warning: in the working copy of 'f5', LF will be replaced by CRLF the next time Gi
t touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git commit -m "f5 is added in feature"
[feature 08f0fcd] f5 is added in feature
 1 file changed, 1 insertion(+)
 create mode 100644 f5

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git log --oneline
08f0fcd (HEAD -> feature) f5 is added in feature
0cd959c f4 is added in feature
a24e21b adding f1 in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ |
```

Step 4:

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git checkout master
Switched to branch 'master'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
b3763d8 (HEAD -> master) f3 created in master
91942a3 f2 added in master
a24e21b adding f1 in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git merge feature
Merge made by the 'ort' strategy.
 f4 | 2 ++
 f5 | 1 +
 2 files changed, 3 insertions(+)
 create mode 100644 f4
 create mode 100644 f5

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
a3fa600 (HEAD -> master) Merge branch 'feature' k# the commit.
08f0fcd (feature) f5 is added in feature
0cd959c f4 is added in feature
b3763d8 f3 created in master
91942a3 f2 added in master
a24e21b adding f1 in master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

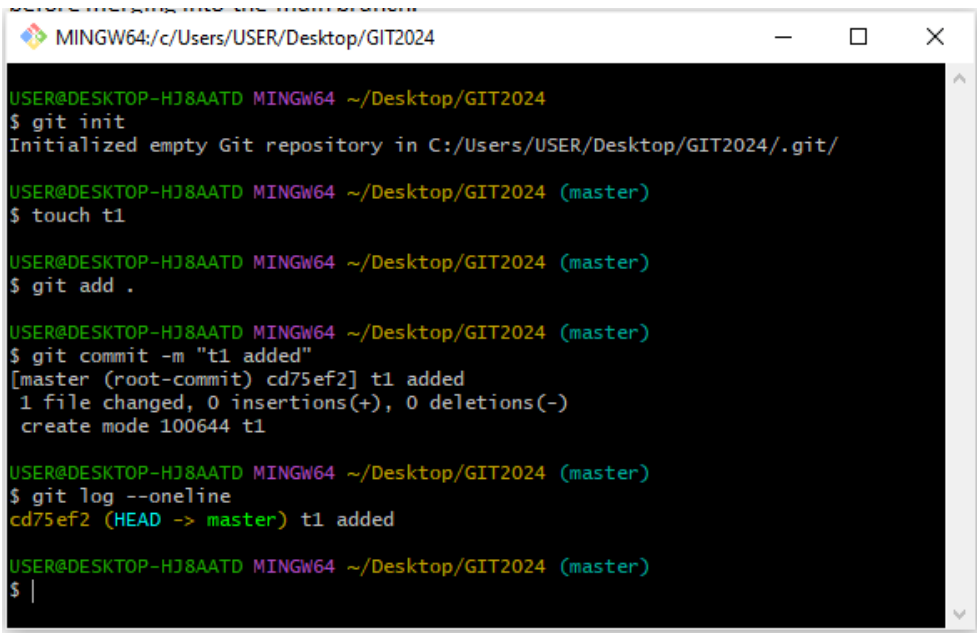
- Here when we do merge extra commit will be added at a top and remaining commit ids will remain same. It preserves the history.

- **<u>DIFFERENCE B/W MERGE AND REBASE:</u>** Both merge and rebase perform the same operation of integrating branches, but the difference is how they perform

  - It creates a new commit id indicating the merge. Merge conflicts are easily handled as the commit ids are reachable
  - In rebase it rewrites the history by creating new commits for each commit in source branch since commit history is rewritten, it will be difficult to understand the conflict in some cases as commits are no longer reachable.

- **<u>INTERACTIVE REBASING</u>**: By using this we can combine the multiple commits into single commit before merging into the main branch.

Step 1:



MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git checkout -b feature
Switched to a new branch 'feature'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ vi t2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git add .
warning: in the working copy of 't2', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git commit -m "started working in new code"
[feature 579c7e3] started working in new code
 1 file changed, 1 insertion(+)
 create mode 100644 t2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ vi t2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git add .
warning: in the working copy of 't2', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git commit -m "WIP:still working on code"
[feature 30ed50e] WIP:still working on code
 1 file changed, 1 insertion(+)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ |
```

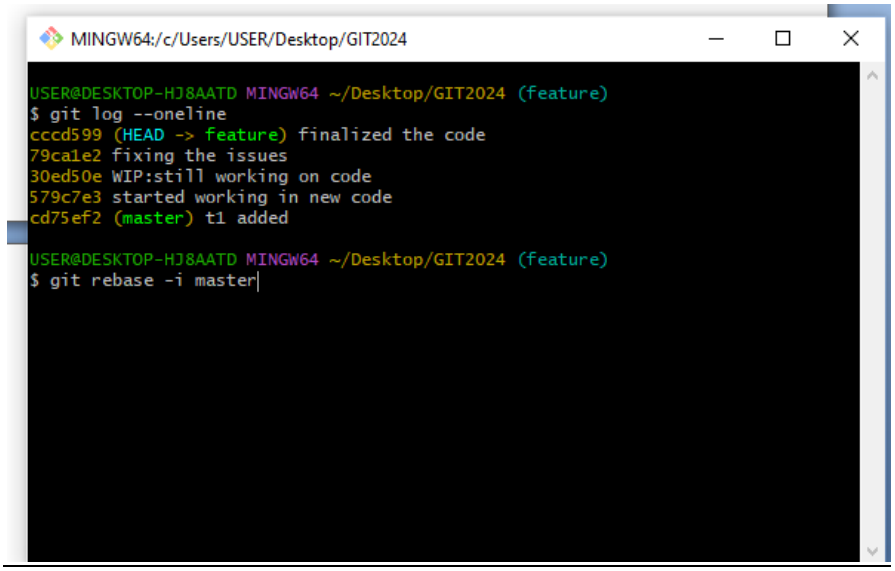
Step 2:

MINGW64:/c/Users/USER/Desktop/GIT2024

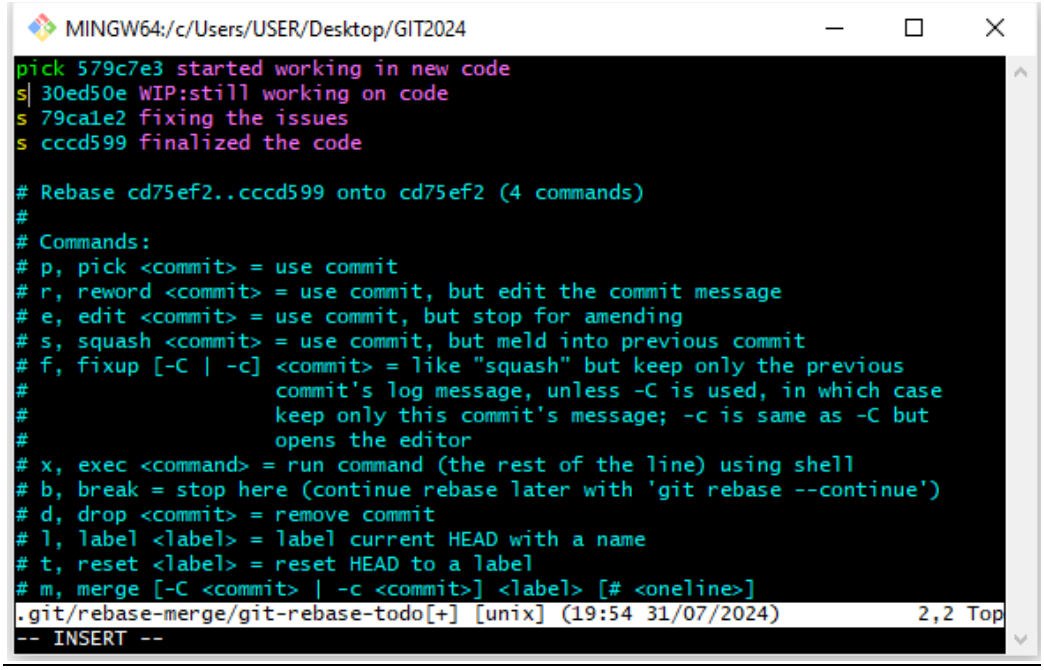
```
$ git add t2
warning: in the working copy of 't2', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git commit -m "fixing the issues"
[feature 79ca1e2] fixing the issues
 1 file changed, 1 insertion(+)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ vi t2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git add t2
warning: in the working copy of 't2', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git commit -m "finalized the code"
[feature cccd599] finalized the code
 1 file changed, 1 insertion(+)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ |
```

Step 3:

Step 4:



Step 5:



- Here we will use squash command to melt the commit into the previous commit.
- **SQUASH**: it is a technique to condense large number of commits to make into small number of meaningful commits so that we can make git history clear.

  git rebase –i master

Step 6:

```
MINGW64:/c/Users/USER/Desktop/GIT2024
# This is a combination of 4 commits.
# This is the 1st commit message:



# This is the commit message #2:



# This is the commit message #3:



# This is the commit message #4:

Implemented new code

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed Jul 31 19:41:20 2024 +0530
#
.git/COMMIT_EDITMSG[+] [unix] (20:03 31/07/2024)                16,21 Top
-- INSERT --
```

Step 7:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git log --oneline
cccd599 (HEAD -> feature) finalized the code
79ca1e2 fixing the issues
30ed50e WIP:still working on code
579c7e3 started working in new code
cd75ef2 (master) t1 added

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git rebase -i master
[detached HEAD a8a90fd] Implemented new code
 Date: Wed Jul 31 19:41:20 2024 +0530
 1 file changed, 4 insertions(+)
 create mode 100644 t2
Successfully rebased and updated refs/heads/feature.

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git log --oneline
a8a90fd (HEAD -> feature) Implemented new code
cd75ef2 (master) t1 added

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$
```

- Here only one commit is present and the new commit id has been added after combining all the previous commits into single commit. In this case we can use rebase command to give clean implementation.

- **<u>AMEND:</u>** If we want to modify a file and doesn't want a separate commit for a file change then we can use this amend command. It will modify in the previous commit itself.

  git commit -- amend

- We can use this amend command when the file is in our local repo only

Step 1:

```
MINGW64:/c/Users/USER/Desktop/GIT2024
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024
$ git init
Initialized empty Git repository in C:/Users/USER/Desktop/GIT2024/.git/

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi t1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 't1', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "adding file1"
[master (root-commit) 4f0c01b] adding file1
 1 file changed, 1 insertion(+)
 create mode 100644 t1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
4f0c01b (HEAD -> master) adding file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

- Here after the file is committed and we changed the file again new commit will not be added when we use this amend command it has been be modified in the previous commit itself. We can check it by using **git log –oneline** command
- **<u>git show:</u>** It is used to check the status of the commit which we want to know in detail on git objects such as blobs,trees,tags and commits

  git show

  - git show is similar to git log but it shows which line and what has been modified

Step 2:

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi t1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 't1', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit --amend
[master 90caaa2] adding file1
 Date: Wed Jul 31 20:25:40 2024 +0530
 1 file changed, 1 insertion(+)
 create mode 100644 t1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
90caaa2 (HEAD -> master) adding file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git show 90caaa2
commit 90caaa2f53f999b0851433c2b559dcb15c5b4f78 (HEAD -> master)
Author: Sirisha <saisirisha1005@gmail.com>
Date:   Wed Jul 31 20:25:40 2024 +0530

    adding file1

diff --git a/t1 b/t1
new file mode 100644
index 0000000..ce01362
--- /dev/null
+++ b/t1
@@ -0,0 +1 @@
+hello

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

- **<u>GIT CHERRY-PICK:</u>** It is used we want to apply the particular commit from one branch to another branch.
  - It is mainly used if we don't want to merge the whole branch but you want some of the commits then we can use this cherry-pick command
  - It is mainly used for bug fixes where we want to place the bug fix commit in all the version branches
  - If we accidently made a commit in the wrong branch and we want that commit in another branch we can use this cherry-pick command.
  - It causes duplicates.

Step 1 :

```
MINGW64:/c/Users/USER/Desktop/GIT2024
Initialized empty Git repository in C:/Users/USER/Desktop/GIT2024/.git/

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ touch f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "initial commit"
[master (root-commit) d77235c] initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
d77235c (HEAD -> master) initial commit

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ ls
f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

Step 2:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git branch 1.0

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git branch 2.0

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git branch 3.0

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git branch
  1.0
  2.0
  3.0
* master

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git checkout 3.0
Switched to branch '3.0'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ |
```

Step 3:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ touch f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ git add .

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ git commit -m "working on feature"
[3.0 20e8b90] working on feature
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ git log --oneline
20e8b90 (HEAD -> 3.0) working on feature
d77235c (master, 2.0, 1.0) initial commit

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ |
```

- Now we found a bug in the current feature and we make to know that this bug is present all other features in this case we use this cherry-pick command.

Step 4:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ vi bugfix

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ git add .
warning: in the working copy of 'bugfix', LF will be replaced by CRLF the next t
ime Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ git commit -m "fixed the bug"
[3.0 96de3e5] fixed the bug
 1 file changed, 1 insertion(+)
 create mode 100644 bugfix

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ git log --oneline
96de3e5 (HEAD -> 3.0) fixed the bug
20e8b90 working on feature
d77235c (master, 2.0, 1.0) initial commit

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ |
```

- In this case I have added a file in 3.0 which contains the bug fixes if I want to add this bug fix file in remaining branches without merging the current branch bcz we are still working in 3.0 branch then we will use this cherry-pick command.
- By using the commit id we can merge into another branch(currently checkout branch)

git cherry-pick <commit id>

Step 5:

```
MINGW64:/c/Users/USER/Desktop/GIT2024
20e8b90 working on feature
d77235c (master, 2.0, 1.0) initial commit

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (3.0)
$ git checkout 2.0
Switched to branch '2.0'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (2.0)
$ ls
f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (2.0)
$ git cherry-pick 96de3e5
[2.0 55897f5] fixed the bug
 Date: Thu Aug 1 12:41:08 2024 +0530
 1 file changed, 1 insertion(+)
 create mode 100644 bugfix

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (2.0)
$ ls
bugfix  f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (2.0)
$ |
```

- **<u>GIT RESET :</u>**It is used to move the branch from one commit to another commit. We can also go back to the workspace but commit history will be removed.
- Reset moves the current branch and optionally copies the data from the repo to the working or staging area.
- Reset has three different options
  - --hard: move the files both to the working area and staging area.
  - --mixed: moves the files only to staging area (default option)
  - --soft: does not move the files
- We can used this reset command to undo the changes

<u>Step 1:</u>

```
MINGW64:/c/Users/USER/Desktop/GIT2024
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024
$ git init
Initialized empty Git repository in C:/Users/USER/Desktop/GIT2024/.git/

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ touch t1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "adding f1"
[master (root-commit) cdec39b] adding f1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 t1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
cdec39b (HEAD -> master) adding f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

Step 2:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ touch t2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "adding t2 file"
[master 76590da] adding t2 file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 t2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
76590da (HEAD -> master) adding t2 file
cdec39b adding f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

- Now again I want to work on the t2 file after committing a new commit will be added and the head will be referred to newly added commit.

  Step 3:

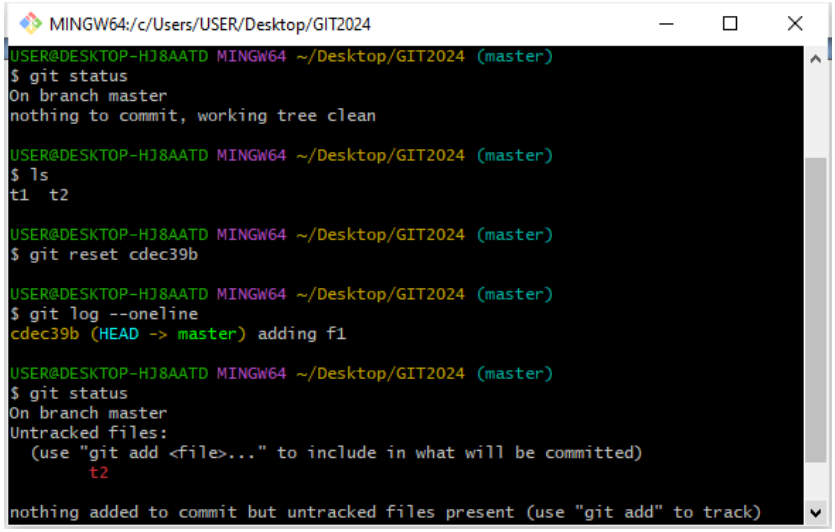
```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi t2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 't2', LF will be replaced by CRLF the next time
Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "updating t2 file"
[master 1b0b50e] updating t2 file
 1 file changed, 1 insertion(+)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git log --oneline
1b0b50e (HEAD -> master) updating t2 file
76590da adding t2 file
cdec39b adding f1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ ls
t1  t2
```

- If we want to move the master or head to the 1st commit where we have added a file in master we can use these reset command

  git reset <commit id >

- The file what we have added is deleted by using this reset command but it is still present in staging area

step 4:



- To delete a file completely in staging area we can use this command

  git reset <commit id > --hard

- **<u>Git Stash:</u>** If you want to switch a branch but you are working on an incomplete part of a current branch and you want to go back to other branch but you don't want to commit your half done work this git stash allows us to do.
    - This command enables us to switch the branches without committing the current branch
    - Stash means "store something safely in a hidden place"

Step 1:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024
$ git init
Initialized empty Git repository in C:/Users/USER/Desktop/GIT2024/.git/

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ touch file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "adding file1"
[master (root-commit) efdc5da] adding file1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

Step 2:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git checkout -b feature
Switched to a new branch 'feature'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ vi file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git status
On branch feature
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2

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

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git checkout master
Switched to branch 'master'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ vi file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git add .
warning: in the working copy of 'file1', LF will be replaced by CRLF the next ti
me Git touches it
warning: in the working copy of 'file2', LF will be replaced by CRLF the next ti
me Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git commit -m "updated file1"
[master 8c29957] updated file1
 2 files changed, 2 insertions(+)
 create mode 100644 file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

Step 3:

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ git checkout feature
Switched to branch 'feature'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git log --oneline
efdc5da (HEAD -> feature) adding file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ vi file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git status
On branch feature
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:   file1

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

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git checkout master
error: Your local changes to the following files would be overwritten by checkou
t:
        file1
Please commit your changes or stash them before you switch branches.
Aborting

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ |
```

Step 4:

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash
warning: in the working copy of 'file1', LF will be replaced by CRLF the next ti
me Git touches it
Saved working directory and index state WIP on feature: efdc5da adding file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash list
stash@{0}: WIP on feature: efdc5da adding file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git checkout master
Switched to branch 'master'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (master)
$ |
```

- **git stash list** command is used to list the stash that are saved.
- **git stash pop** command is used to take the recently pushed stash and delete it from the stash and give it to the branch
- In this way we can stash the file and unstash it

Step 5:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash pop
On branch feature
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:   file1

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (f05d008c851daa31f9ad411e978e8618c2d3083b)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash list

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ |
```

- If we want to add a custom name in stash we can use

  git stash save “ ”

Step 6:

```
MINGW64:/c/Users/USER/Desktop/GIT2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash save "modified file"
Saved working directory and index state On feature: modified file

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash list
stash@{0}: On feature: modified file

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ |
```

- If we don’t want to delete the stash and want to retrieve it to the feature branch

  git stash apply

MINGW64:/c/Users/USER/Desktop/GIT2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash pop
On branch feature
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:   file1

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (480275320c3d0ca83fe1bc33107a8e3b8dddf15a)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash list

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash save "modified stash"
Saved working directory and index state On feature: modified stash

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash list
stash@{0}: On feature: modified stash

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash apply
On branch feature
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:   file1

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

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$ git stash list
stash@{0}: On feature: modified stash

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/GIT2024 (feature)
$
```

- **<u>git checkout:</u>**
  - It helps to move from one branch to other

    git checkout <branch name>
  - It creates a new branch if not existed and moves the head to that branch
  - To create a branch and enter into the branch

    git checkout –b <branch name>
  - To move the head position to the first commit

    git checkout –
  - To move the head position to any other commit

    git checkout Head~1

- **<u>git switch command</u>**:
  - git switch command: It also moves from one branch to another branch
  - It is alternative to git checkout command

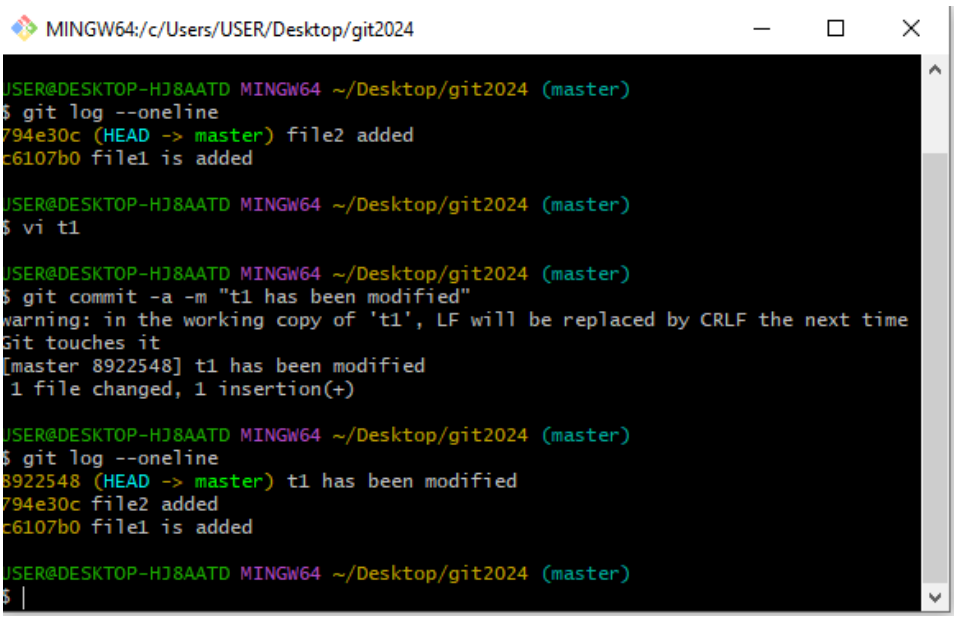
    git switch <branch name>
  - To create a branch and enter into the branch

    git switch –c <branch name>

- Switch command doesn't move to the particular commit hash but git checkout command go back to the particular commit hash

- **<u>git revert command</u>:** when we want to revert back the changes of a particular file or a commit we can use this git revert command.
  - **Difference between git reset and git revert command:**
    1. In revert we can go back to the workspace after committing, but commit history will be stored
    2. In reset we can go back to the workspace after committing, but commit history is removed

Step 1:



Step 2:

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ vi t2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git commit -a -m "t2 has been modified"
warning: in the working copy of 't2', LF will be replaced by CRLF the next time
Git touches it
[master 5c47c82] t2 has been modified
 1 file changed, 1 insertion(+)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git log --oneline
5c47c82 (HEAD -> master) t2 has been modified
8922548 t1 has been modified
794e30c file2 added
c6107b0 file1 is added

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ vi t1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git commit -a -m "t1 has been modified2"
warning: in the working copy of 't1', LF will be replaced by CRLF the next time
Git touches it
[master e839e8a] t1 has been modified2
 1 file changed, 1 insertion(+)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ vi t2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git commit -a -m "t2 has been modified2"
warning: in the working copy of 't2', LF will be replaced by CRLF the next time
Git touches it
[master 5835be5] t2 has been modified2
 1 file changed, 1 insertion(+)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ |
```

Step 3:

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git log --oneline
5835be5 (HEAD -> master) t2 has been modified2
e839e8a t1 has been modified2
5c47c82 t2 has been modified
8922548 t1 has been modified
794e30c file2 added
c6107b0 file1 is added

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git reset --hard Head~1
HEAD is now at e839e8a t1 has been modified2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git log --oneline
e839e8a (HEAD -> master) t1 has been modified2
5c47c82 t2 has been modified
8922548 t1 has been modified
794e30c file2 added
c6107b0 file1 is added

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ |
```

- This is how we do by using the reset command now we will do by using revert command

Step 1:

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git log --oneline
f2a976a (HEAD -> master) file2 modified
e8c22f1 file1 modified
a1a6534 file2 added
9d2d87c file1 added

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git revert f2a976a
```

Step 2:

```
MINGW64:/c/Users/USER/Desktop/git2024
Revert "file2 modified"

This reverts commit f2a976a5adc2982aae8a499067be801f2947afd9.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       modified:   file2
#
~
~
~
~
~
~
~
~
~
~
~
.git/COMMIT_EDITMSG [unix] (21:18 04/08/2024)
"~/Desktop/git2024/.git/COMMIT_EDITMSG" [unix] 11L, 293B
```

Step 3:

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git log --oneline
f2a976a (HEAD -> master) file2 modified
e8c22f1 file1 modified
a1a6534 file2 added
9d2d87c file1 added

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git revert f2a976a
[master 523a143] Revert "file2 modified"
 1 file changed, 1 deletion(-)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git log --oneline
523a143 (HEAD -> master) Revert "file2 modified"
f2a976a file2 modified
e8c22f1 file1 modified
a1a6534 file2 added
9d2d87c file1 added

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ |
```

- **<u>When to use revert and reset command:</u>**
  - Reset is used only when your commits are in local
  - Revert command is used when your commits are went into remote.

- **<u>When u want to revert the particular file :</u>**

  git revert <commit id >

- **<u>git diff command:</u>** The git diff command shows the differences between the files in two commits or between your current repository and a previous commit.
  - To check the changes between working area and staging area

    git diff
  - To check the changes between staging area and repo area

    git diff -- staged
  - To check the changes between repo area and working directory

    git diff -- head

Step 1:

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ vi file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git status
On branch master
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:   file2

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

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git add .
warning: in the working copy of 'file2', LF will be replaced by CRLF the next ti
me Git touches it

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file2


USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ |
```

Step 2:

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ vi file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file2

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:   file2


USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git diff
warning: in the working copy of 'file2', LF will be replaced by CRLF the next ti
me Git touches it
diff --git a/file2 b/file2
index 170d4c2..29f09eb 100644
--- a/file2
+++ b/file2
@@ -1 +1,2 @@
 helloooo
+hiii

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ |
```

- here the file is in staging area and the same file is in working directory by using this command we can find the diff between the same file what the data has been added.

Step 3:

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git diff --staged
diff --git a/file2 b/file2
index e69de29..170d4c2 100644
--- a/file2
+++ b/file2
@@ -0,0 +1 @@
+helloooo

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ |
```

- here the file is still in the staging area and nothing is present in the repo area so this gives the command between them

Step 4:

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git diff --staged
diff --git a/file2 b/file2
index e69de29..170d4c2 100644
--- a/file2
+++ b/file2
@@ -0,0 +1 @@
+helloooo

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git diff head
warning: in the working copy of 'file2', LF will be replaced by CRLF the next ti
me Git touches it
diff --git a/file2 b/file2
index e69de29..29f09eb 100644
--- a/file2
+++ b/file2
@@ -0,0 +1,2 @@
+helloooo
+hiii

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ |
```

- to check the difference between the repo and working area we can use this command

- **<u>git bisect:</u>** The git bisect command is used to discover the commit that has introduced a bug in the code. It helps to track down the commit where the code works and the commit where it does not, hence, tracking down the commit that introduced the bug into the code.
    - When we have multiple commits and our latest commit has the bug then we need to find out in which commit the bug is present in this kind of cases we can use this bisect command.

Step 1:

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git log --oneline
49a2077 (HEAD -> master) file2 added
4951e24 adding file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git bisect start
status: waiting for both good and bad commits

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master|BISECTING)
$ |
```

- it is used to find the good and bad commits

Step 2 :

```
MINGW64:/c/Users/USER/Desktop/git2024

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git log --oneline
49a2077 (HEAD -> master) file2 added
4951e24 adding file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ git bisect start
status: waiting for both good and bad commits

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master|BISECTING)
$ git bisect bad
status: waiting for good commit(s), bad commit known

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master|BISECTING)
$ git bisect good
49a2077d82c3a21854a36729bc64aa9482807020 was both good and bad

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master|BISECTING)
$ |
```

- In this way we can find the good and bad commits.

- **<u>git insta web</u>**: Instaweb is a script used to set up a temporary instance of GitWeb on a web server for browsing local repositories**<u>.</u>**
- **<u>git drop:</u>** If we want to delete the commit we can use this git drop
- **<u>Amazon EMR:</u>** Amazon EMR (previously called Amazon Elastic MapReduce) is a managed cluster platform that simplifies running big data frameworks, such as Apache Hadoop and Apache Spark , on AWS to process and analyze vast amounts of data.
- **<u>TYPES OF AMI:</u>**
  - EBS: backend instance
  - Instance store : backend instance

- **<u>What is difference between git and other repository:</u>**
  - ➢ git is a distributed version control system, that means whole repository will be present in the local workspace.
  - ➢ If you want to go to previous version of the code, it will be available in the local workspace.
  - ➢ In git we can work offline (local workspace).
  - ➢ git has many advanced features like fetch, revert, rebase..etc
  - ▪ **<u>other repo:</u>**
    - ➢ Centralized version control systems, only the latest version of code will be there in the local workspace.
    - ➢ If you want the previous version of the code, it needs to be checked out from the central repo.
    - ➢ We need to interact with central repo frequently.
    - ➢ We don't have direct commands to all these features.

- **<u>git hook:</u>** Git hook allows us to run the customized scripts whenever important event occurs in git life-cycle such as committing, merging, pushing. It runs locally

MINGW64:/c/Users/USER/Desktop/git2024/.git/hooks

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/.git (GIT_DIR!|BISECTING)
$ ls -la
total 25
drwxr-xr-x 1 USER 197121   0 Aug  4 22:35 ./
drwxr-xr-x 1 USER 197121   0 Aug  4 22:17 ../
-rw-r--r-- 1 USER 197121   0 Aug  4 22:35 BISECT_ANCESTORS_OK
-rw-r--r-- 1 USER 197121 358 Aug  4 22:35 BISECT_LOG
-rw-r--r-- 1 USER 197121   1 Aug  4 22:33 BISECT_NAMES
-rw-r--r-- 1 USER 197121   7 Aug  4 22:33 BISECT_START
-rw-r--r-- 1 USER 197121   9 Aug  4 22:35 BISECT_TERMS
-rw-r--r-- 1 USER 197121  12 Aug  4 22:33 COMMIT_EDITMSG
-rw-r--r-- 1 USER 197121  23 Aug  4 22:14 HEAD
-rw-r--r-- 1 USER 197121 130 Aug  4 22:14 config
-rw-r--r-- 1 USER 197121  73 Aug  4 22:14 description
drwxr-xr-x 1 USER 197121   0 Aug  4 22:14 hooks/
-rw-r--r-- 1 USER 197121 209 Aug  4 22:33 index
drwxr-xr-x 1 USER 197121   0 Aug  4 22:14 info/
drwxr-xr-x 1 USER 197121   0 Aug  4 22:15 logs/
drwxr-xr-x 1 USER 197121   0 Aug  4 22:33 objects/
drwxr-xr-x 1 USER 197121   0 Aug  4 22:35 refs/

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/.git (GIT_DIR!|BISECTING)
$ cd hooks/

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/.git/hooks (GIT_DIR!|BISECTING)
$ ll
total 49
-rwxr-xr-x 1 USER 197121  478 Aug  4 22:14 applypatch-msg.sample*
-rwxr-xr-x 1 USER 197121  896 Aug  4 22:14 commit-msg.sample*
-rwxr-xr-x 1 USER 197121 4726 Aug  4 22:14 fsmonitor-watchman.sample*
-rwxr-xr-x 1 USER 197121  189 Aug  4 22:14 post-update.sample*
-rwxr-xr-x 1 USER 197121  424 Aug  4 22:14 pre-applypatch.sample*
-rwxr-xr-x 1 USER 197121 1649 Aug  4 22:14 pre-commit.sample*
-rwxr-xr-x 1 USER 197121  416 Aug  4 22:14 pre-merge-commit.sample*
-rwxr-xr-x 1 USER 197121 1374 Aug  4 22:14 pre-push.sample*
-rwxr-xr-x 1 USER 197121 4898 Aug  4 22:14 pre-rebase.sample*
-rwxr-xr-x 1 USER 197121  544 Aug  4 22:14 pre-receive.sample*
-rwxr-xr-x 1 USER 197121 1492 Aug  4 22:14 prepare-commit-msg.sample*
-rwxr-xr-x 1 USER 197121 2783 Aug  4 22:14 push-to-checkout.sample*
-rwxr-xr-x 1 USER 197121 2308 Aug  4 22:14 sendemail-validate.sample*
-rwxr-xr-x 1 USER 197121 3650 Aug  4 22:14 update.sample*

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/.git/hooks (GIT_DIR!|BISECTING)
$ |
```

- **pre-commit:** checks the commit message for spelling mistakes
- **pre-receive**: enforce project coding standards
- **post-commit:** Email/SMS team members of a new commit
- **post-receive**: push the code to production

- **git blame:** This command is used to show the code of each line who has modified it.

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master|BISECTING)
$ ls
file1  file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master|BISECTING)
$ git blame file1
^4951e24 (Sirisha 2024-08-04 22:15:07 +0530 1) hai

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master|BISECTING)
$
```

- **What language is used in git:**

  GIT is fast, and 'C' language makes this possible by reducing the overhead of runtimes associated with higher languages.
- **How will you know in GIT if a branch has been already merged into master?**
  - Git branch—merged lists the branches that have been merged into the current branch
  - Git branch—-no merged lists the branches that have not been merged
- **What is the function of 'git config'?**
  - The 'git config' command is a convenient way to set configuration options for your Git installation.
  - Behavior of a repository, user info, preferences etc. can be defined through this command.
- **What does commit object contain?**

  a) A set of files, representing the state of a project at a given point of time

  b) Reference to parent commits objects

  c) A SHAI name, a 40-character string that uniquely identifies the commit object.
- **Git Remote?**

  The git remote command lets you create, view, and delete connections to other repositories
- **What is the function of 'git rm'?**

  To remove the file from the staging area and also off your disk 'git rm' is used.

- **Branching Strategy :**

  Branches can be created for multiple reasons, here we create branches for releases, and development will be going

On the dev branch. Once the code is ready on the dev branch for the first release we create release 1 branch and we make a release from the release 1 branch and this branch acts as a maintenance branch for the 1st release that means whatever the issues related to 1st release will be fixed on release 1 branch. And parallel development will be going on the dev branch for the 2nd release once the code is ready for the 2nd release on the dev branch before we create release 2 branch we merge release 1 branch to dev branch then we create branch for 2nd release from the dev branch. Whatever the issues that we have seen in previous release should not be visible in the next release.

- There are five stages in the branches strategy and will diff from company to company
  1. Dev branch
  2. QA
  3. Staging area
  4. Performance
  5. Production
  6. Hot fix

- **<u>GIT Fork?</u>**

  A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

- **<u>Central Repo:</u>**
  - git clone --> will bring central repo to local work space for the first time
  - git pull --> it will compare if there are any changes, it will bring changes from central repo and merges to local repo automatically
  - git push --> it moves local changes from local repo to central repo.

- if you want consider current directory as central repository
- git init --bare --> acts a central repo , we can only push and clone/pull changes to repository
- git init -> act as local repo (non bare repository)

- **<u>We have two types of repositories</u>**

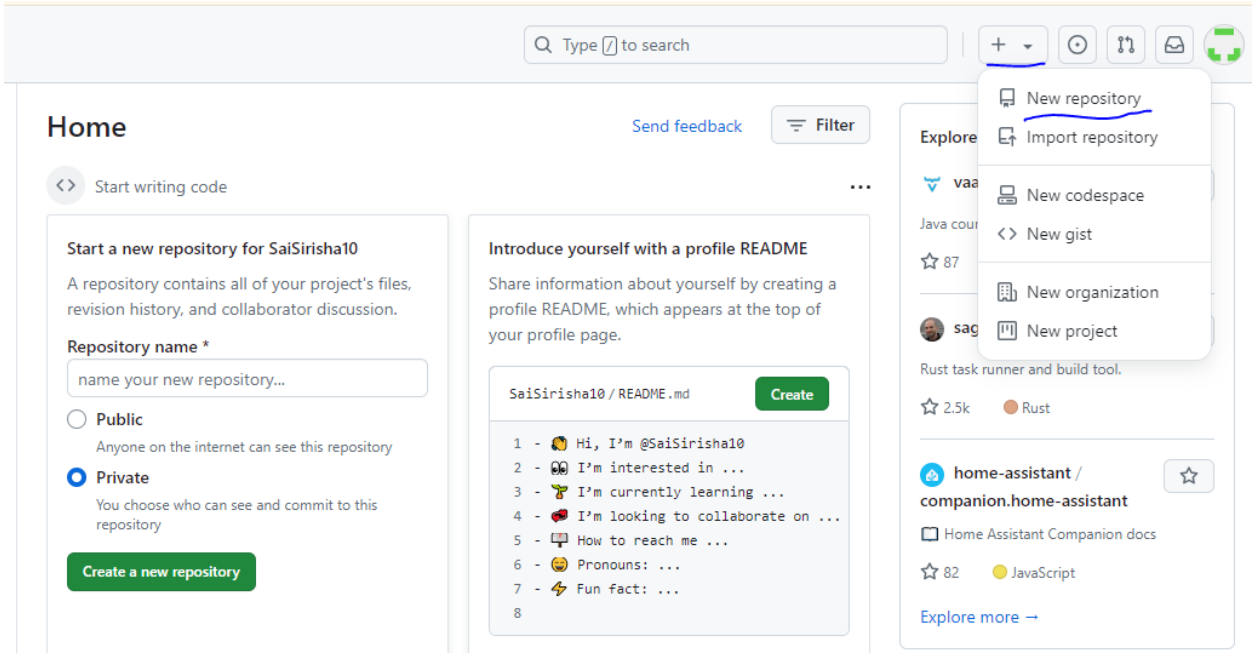
1. Bare repository - only we can pull and push the files, git operations cannot be performed

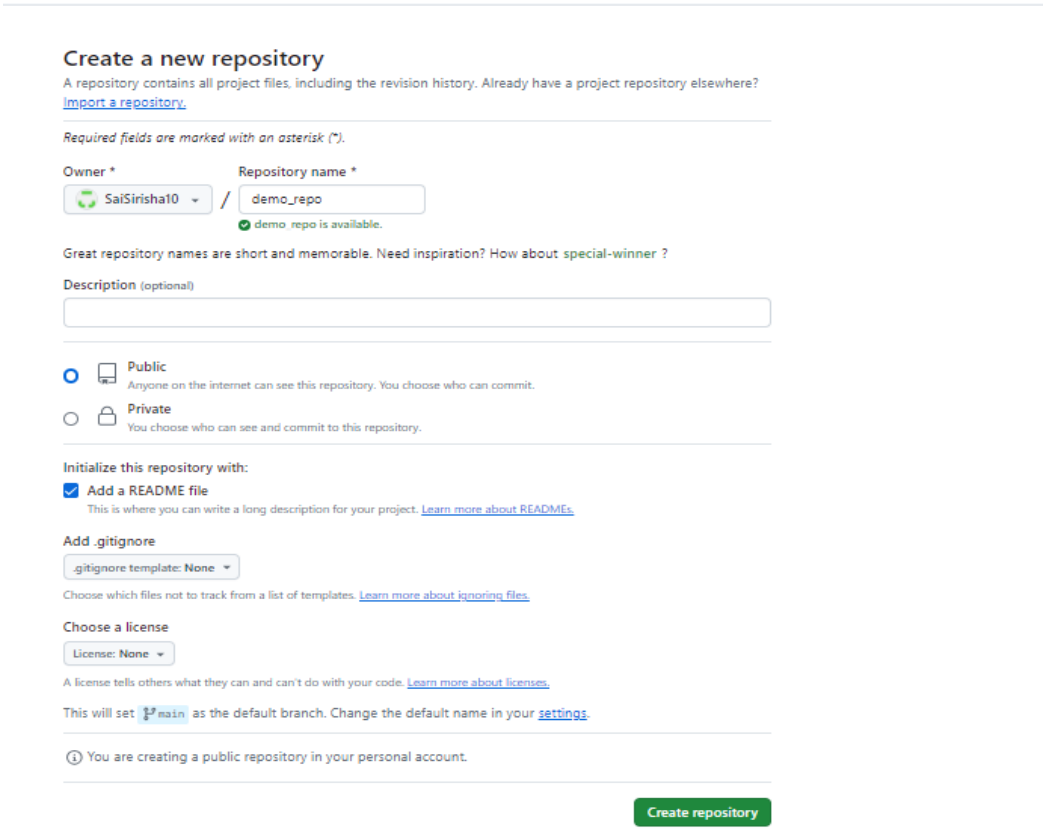
2. Non bare repository - all the git operations are performed here,we can modify files push to central ,run all git commands.

- **<u>git fetch command</u>**: It bring changes from central repo to separate branch (under FETCH_HEAD) without merging.

- **<u>To create a repo in git hub:</u>**

Step 1:



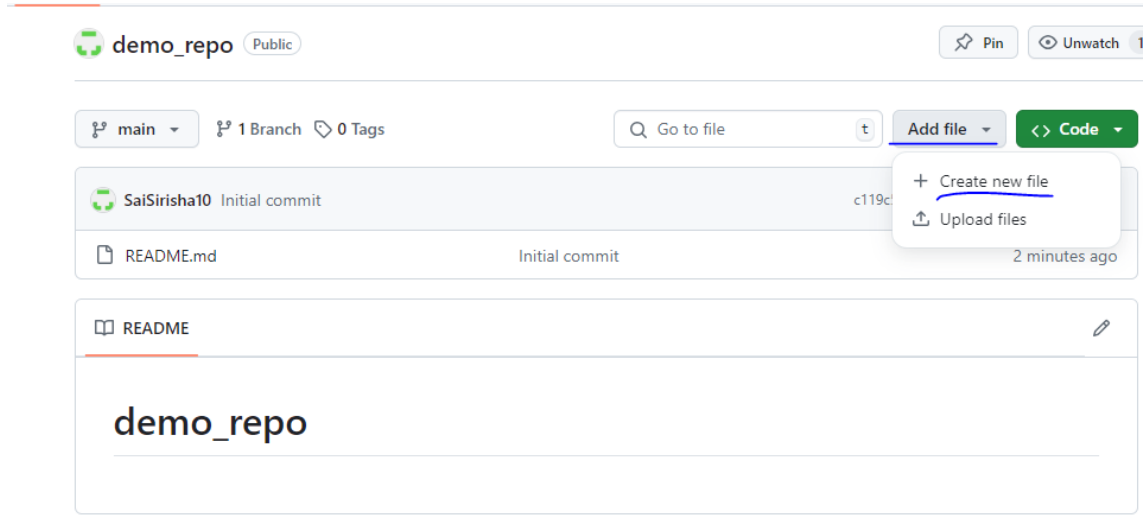


Step 2:

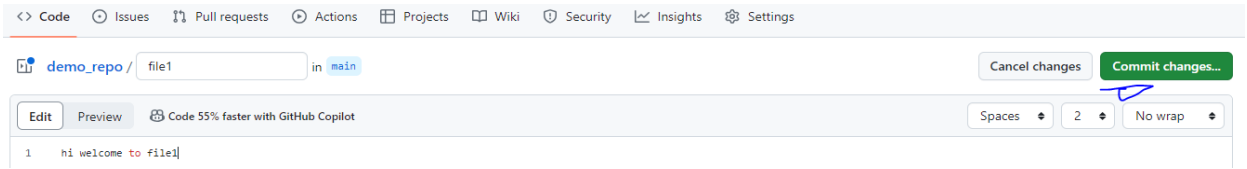
- ReadMe file is important in every project because it helps the project how to run.

- **<u>To add a file in git hub:</u>**

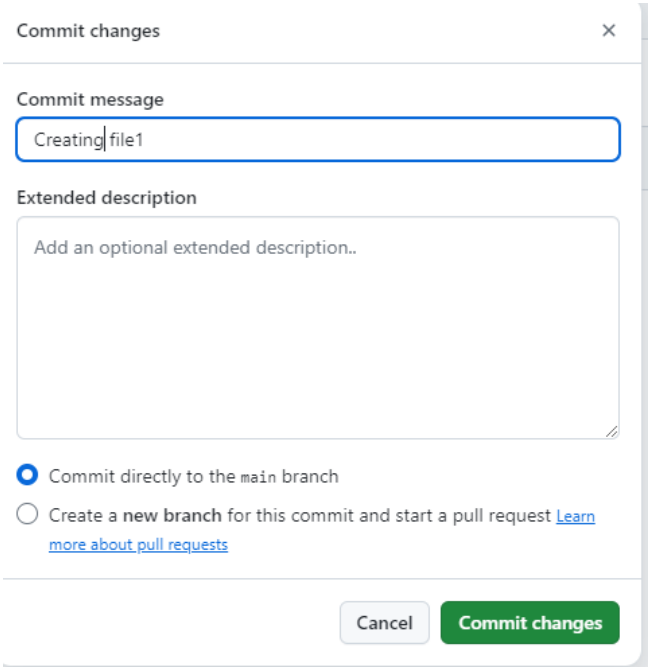
Step 1:



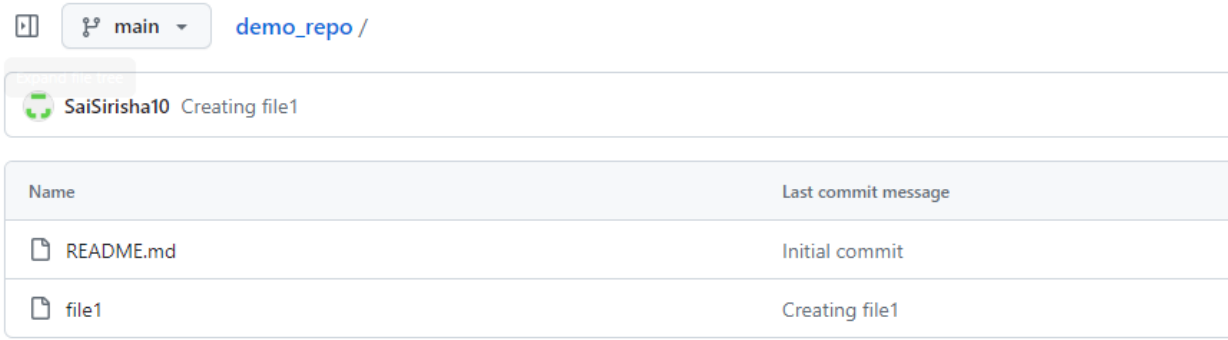
Step 2: add the file and click on commit changes



Step 3: To commit a file

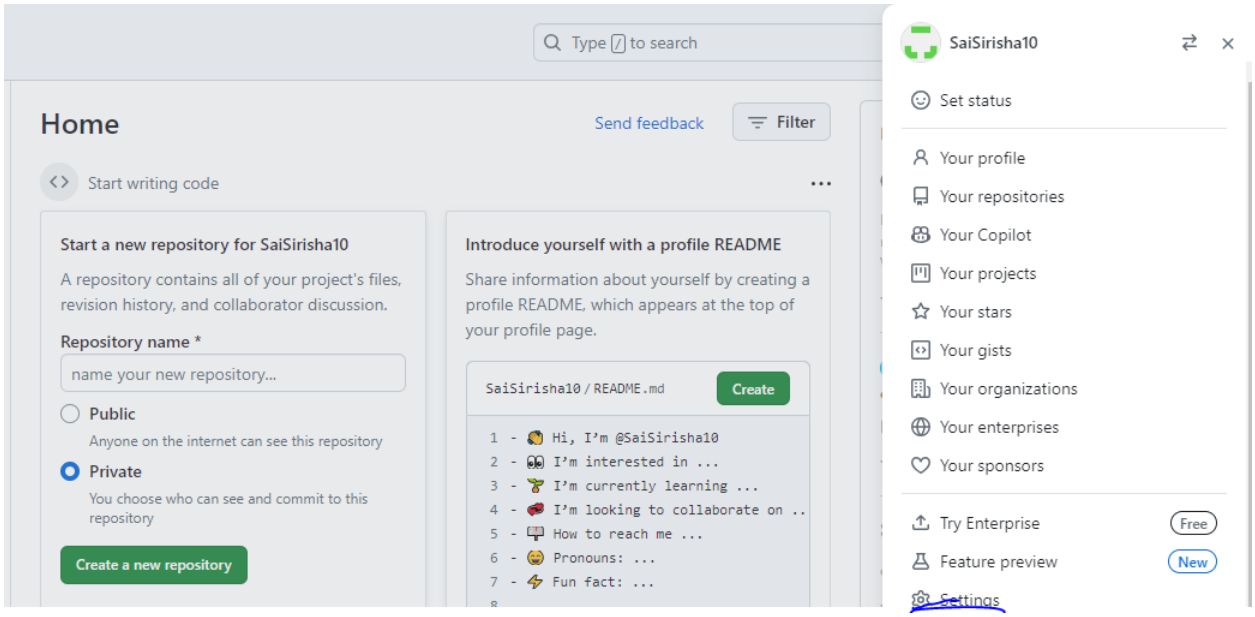


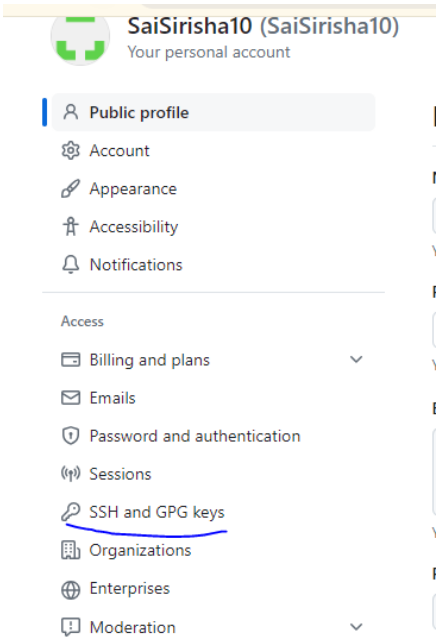
Step 3 : file has been created.



- **<u>To generate a SSH key in git hub:</u>**

<u>Step 1:</u>



Step 2: 

Step 3:

- Now we need to generate ssh key in the terminal using the command ssh-keygen –t rsa

- **<u>Step 4:</u>** after generating the key we need to paste in git hub

MINGW64:/c/Users/USER/Desktop/git2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/USER/.ssh/id_rsa):
/c/Users/USER/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/USER/.ssh/id_rsa
Your public key has been saved in /c/Users/USER/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:arwKMXSFLeaHYXRX1XdMPIY8iAoUhbxIHjLMGAS83KM USER@DESKTOP-HJ8AATD
The key's randomart image is:
+---[RSA 3072]----+
|Xo .+=".,.o.+..+.|
|.* o=".. . ..+.o+|
|. O+++o ..    .o..|
| + "o...         |
|  + ..  S        |
| E o . .         |
|  .   +          |
|   . . .         |
|    ...          |
+----[SHA256]-----+

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$ cat  /c/Users/USER/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDMsEShCOyTD+uthTGO8kWbtt0lk4wPygFEXcmdhSs+E5OqQfcFR6aTpNKYMiblMJWq9UlIkcLCwJDn84GNCciXmtFFzPOqOHMAw1sLZvXkhUlnb9nqAOfbRl18+oOLiJhiSlhjR/ATkadqWA4ljOdO
On614HMCkJzLhdxyQV9kbrUHph3FXflKLqs1ioxXxB8WVJyTIvOyrzosiOukj7HZU3hQ5aRqnNSFJ+MxrMESGQYNrPv2OnZ79KcDl+jZdUf/Nyj+HKL/OOvVKtlZxMZHMxrgnodWtXJxJWCDnEJI9OVTTkMJCqB42Dwm2b7uJU5MGg5/TUsWonJkcNlU
zTeLYnPbcwzjTR+XoZdKyqNjCPM8QoBa8/FHfKR/P2ruhieHn/1FkSF/ETg+crmFYNT4VqQzx3/fDNR5tHpSwIsqJFW7tOb2RHuw3tBeNRBsZlImRiH7TPkuwwn1cOnN6+GluZqc2jQhj3HWnxm9rxUe5q3b8ah/12KEXVO= USER@DESKTOP-HJ8AATD

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024 (master)
$
```

- Step 5: new ssh key has been generated

SSH

Git_key

SHA256:arwKMXSFLeaHYXRX1XdMPIY8iAoUhbxIHjLMGAS83KM

Added on Aug 4, 2024

Never used — Read/write

Delete

- **<u>To clone the repo from central to local repo:</u>**

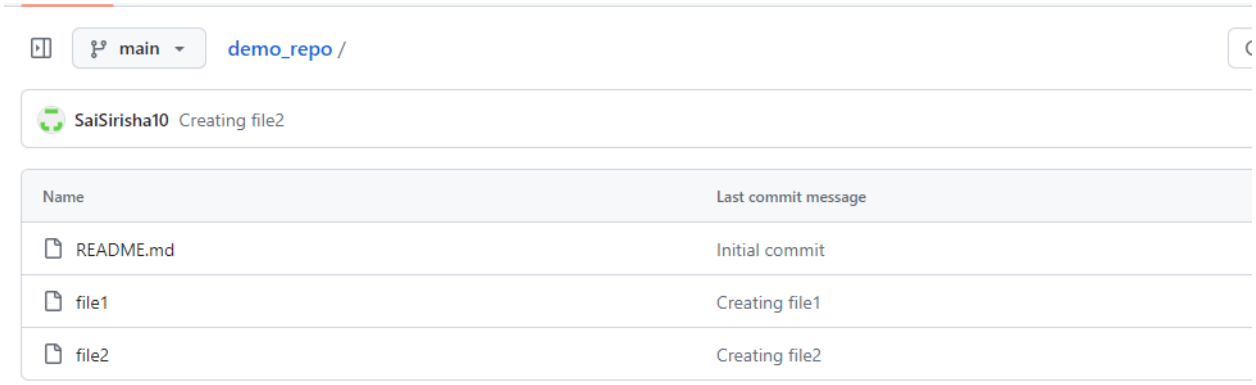
git clone

step 1:

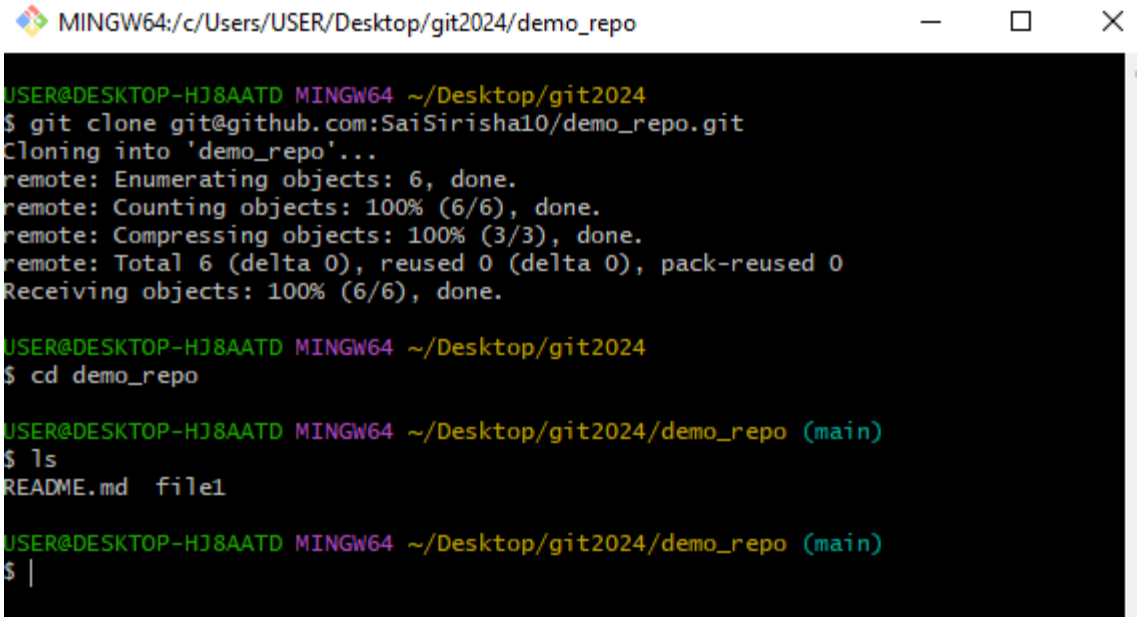
MINGW64:/c/Users/USER/Desktop/git2024

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024
$ git clone git@github.com:SaiSirisha10/demo_repo.git
Cloning into 'demo_repo'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024
$
```

step 2: I have added a new file in central repo but its not effected in my local repo



step 3:



- In this use we need to use **git pull** command to reflect the changes in local repo

Step 4:

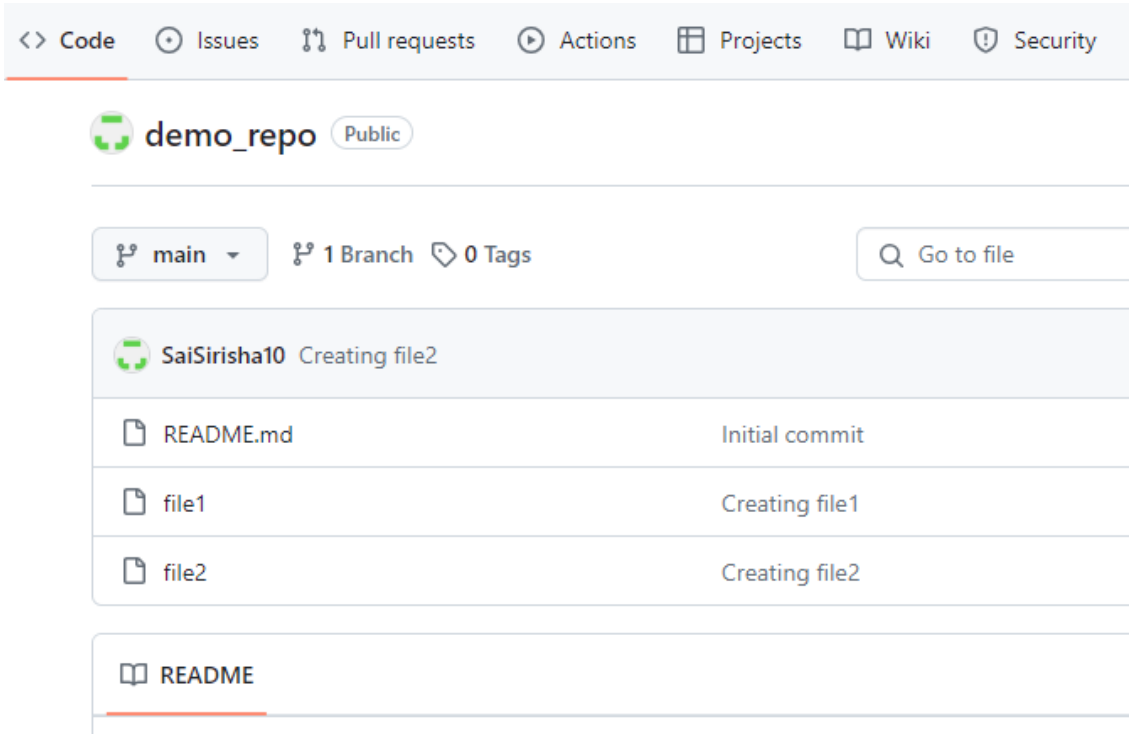
```
MINGW64:/c/Users/USER/Desktop/git2024/demo_repo
$ ls
README.md  file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 949 bytes | 67.00 KiB/s, done.
From github.com:SaiSirisha10/demo_repo
   6c41050..0e0b5f3  main       -> origin/main
Updating 6c41050..0e0b5f3
Fast-forward
 file2 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ ls
README.md  file1  file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ |
```

- Now the changes has been reflected in our local repo also
- In the same case when we create a file in our local repo it will not affect in central repo in that case we need to use **git push** command.

Step 5:

```
MINGW64:/c/Users/USER/Desktop/git2024/demo_repo

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ touch file3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ git add .

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ git commit -m "adding file3"
[main 8226710] adding file3
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ ls
README.md  file1  file2  file3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ |
```

Step 6: the changes has not reflected in central repo .

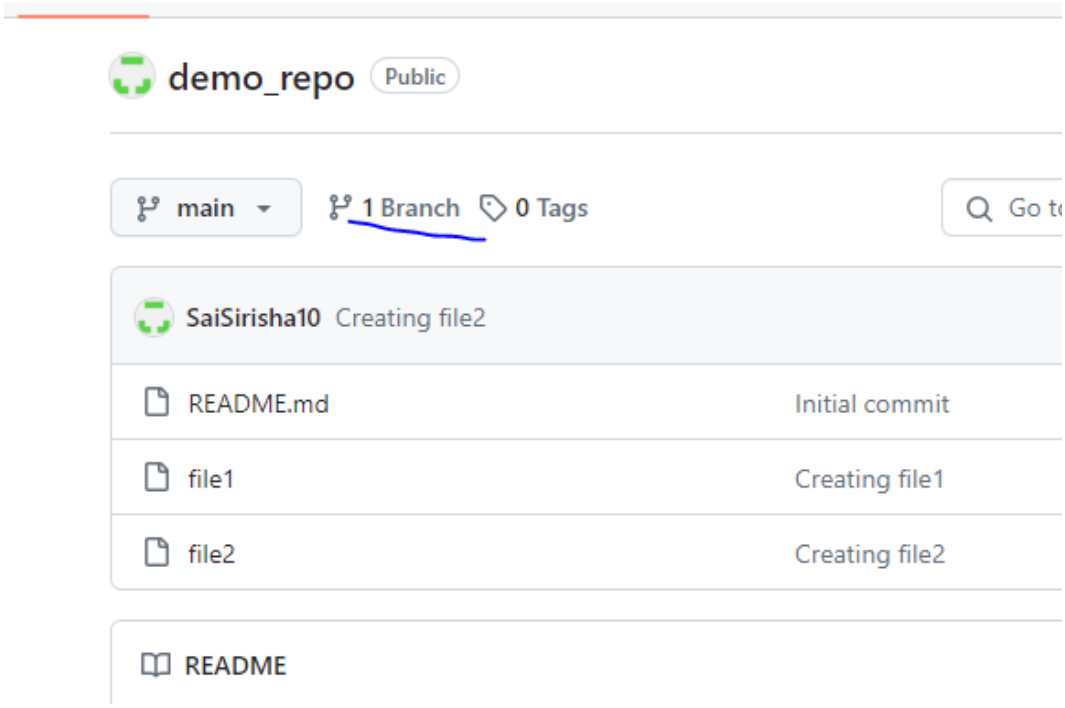


Step 7: now the changes will reflect in central repo.

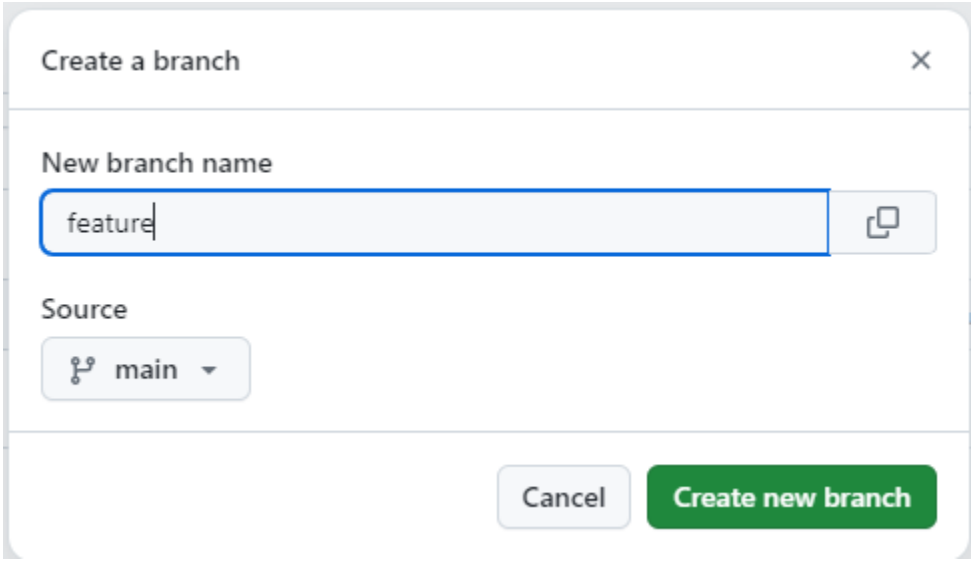
```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ ls
README.md  file1  file2  file3

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/demo_repo (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 264 bytes | 264.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
```

- **PULL Request:** Firstly we need to create a branch and a file inside a branch in git hub repo
- **To create a branch:**

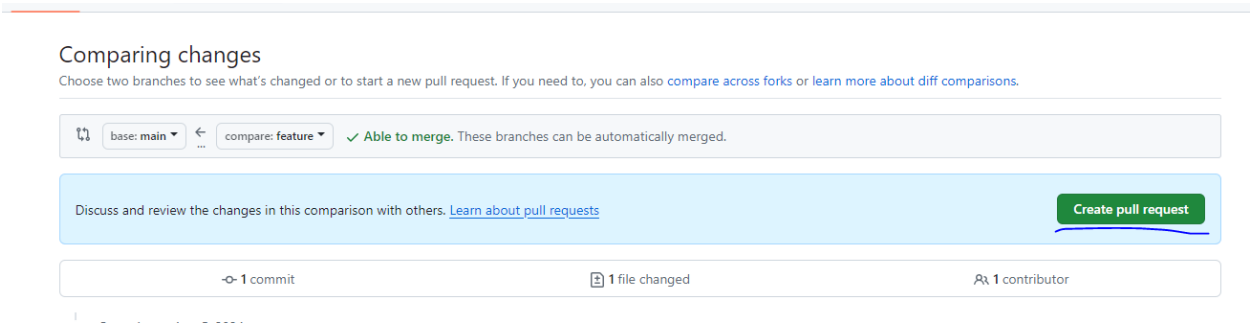
Step 1:


demo_repo
Public
main
1 Branch
0 Tags
Go t
SaiSirisha10 Creating file2
README.md
Initial commit
file1
Creating file1
file2
Creating file2
README

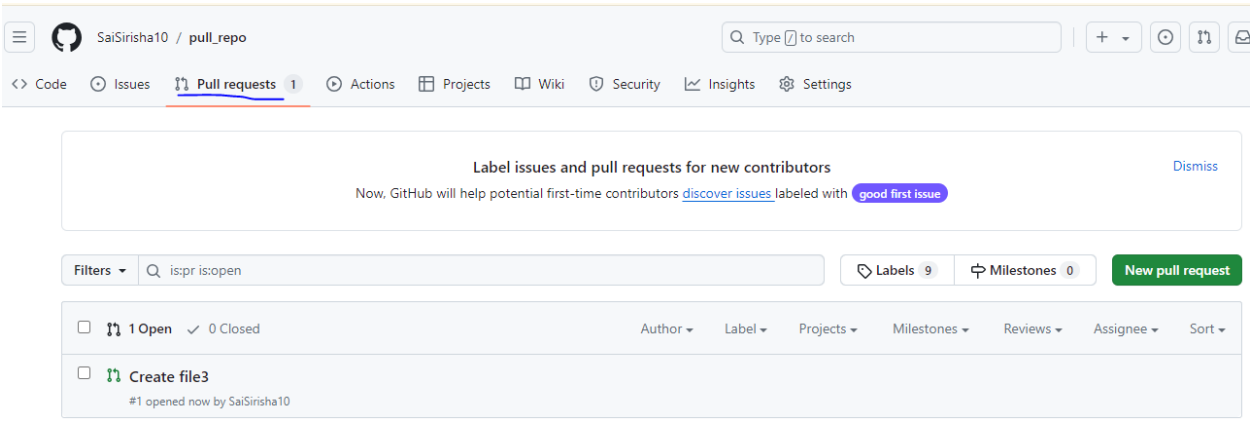
Step 2:


Create a branch
New branch name
feature
Source
main
Cancel
Create new branch

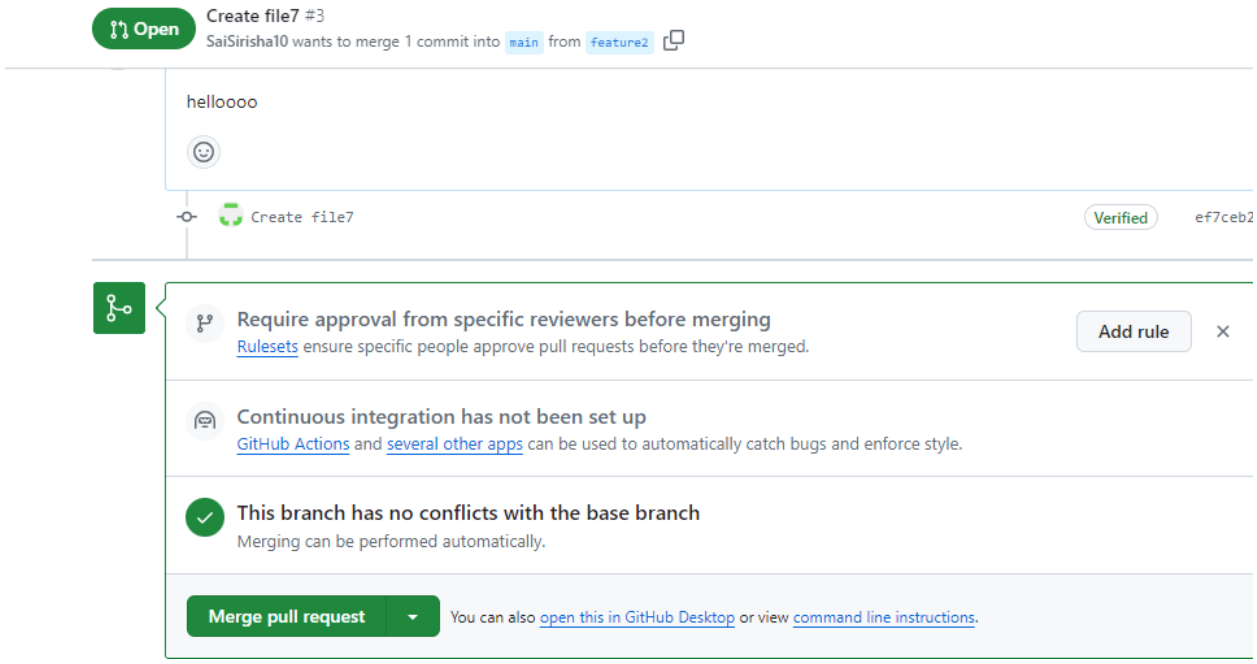
Step 3:



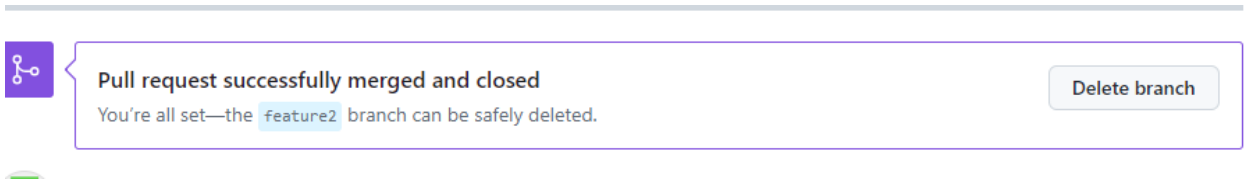
Step 4:



- git branch –r : is used to see the branches created in the central repo.



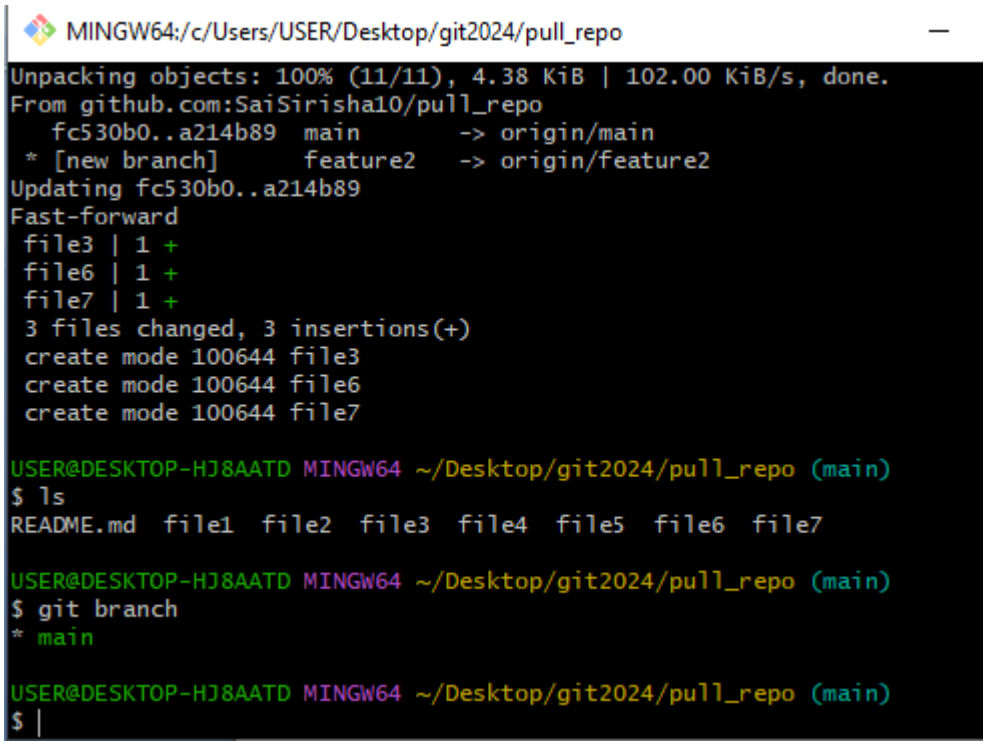
- click on the merge pull request and merge the file.



- pull request has been merged successfully
- In this way we can do pull request in git hub repo.

Git fetch command:

Step 1:



- To find the feature branch in our clone repo after pulling it from git hub we can use
  git branch -r

Step 2:

```
MINGW64:/c/Users/USER/Desktop/git2024/pull_repo
file6 | 1 +
file7 | 1 +
3 files changed, 3 insertions(+)
create mode 100644 file3
create mode 100644 file6
create mode 100644 file7

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ ls
README.md  file1  file2  file3  file4  file5  file6  file7

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ git branch
* main

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ git branch -r
  origin/HEAD -> origin/main
  origin/feature
  origin/feature2
  origin/main

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ |
```

Step 3 :

```
MINGW64:/c/Users/USER/Desktop/git2024/pull_repo
README.md  file1  file2  file3  file4  file5  file6  file7

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ git branch
* main

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ git branch -r
  origin/HEAD -> origin/main
  origin/feature
  origin/feature2
  origin/main

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ git branch -f feature2 origin/feature2
branch 'feature2' set up to track 'origin/feature2'.

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ git branch
  feature2
* main

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ |
```

- **To** get the feature branch in our local machine we can use

  git branch –f feature2 origin/feature2

step 4:

```
MINGW64:/c/Users/USER/Desktop/git2024/pull_repo

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ git fetch origin main
From github.com:SaiSirisha10/pull_repo
 * branch            main       -> FETCH_HEAD

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ git checkout origin/main -- file8

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ ls
README.md  file1  file2  file3  file4  file5  file6  file7  file8

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/pull_repo (main)
$ |
```

Git fetch :

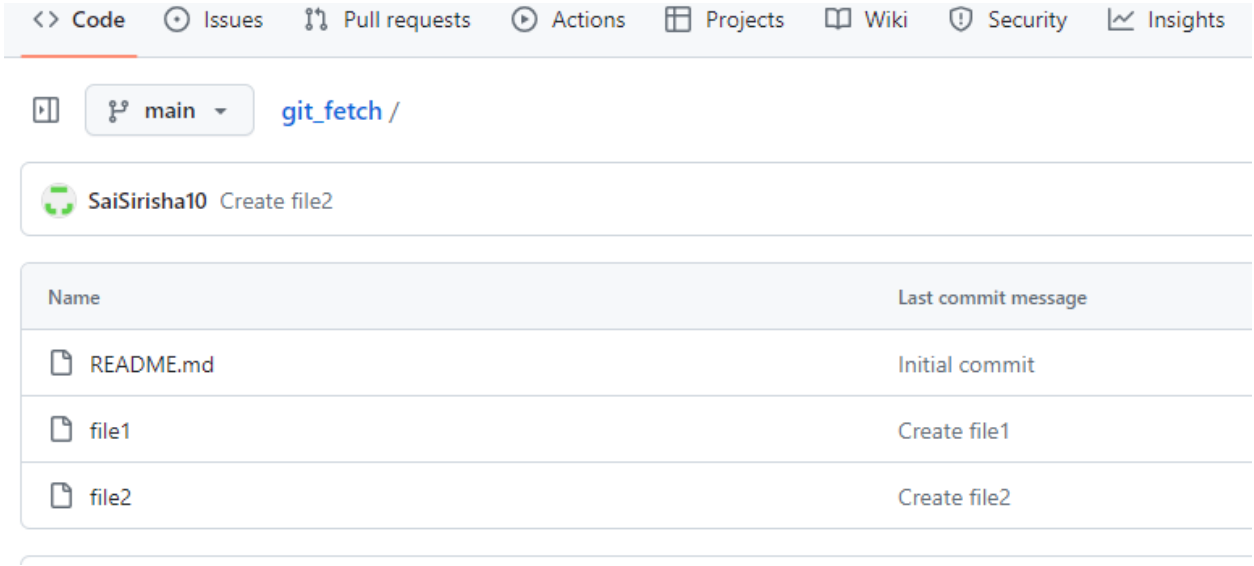
- Create a file in git hub clone and clone it to our local repo

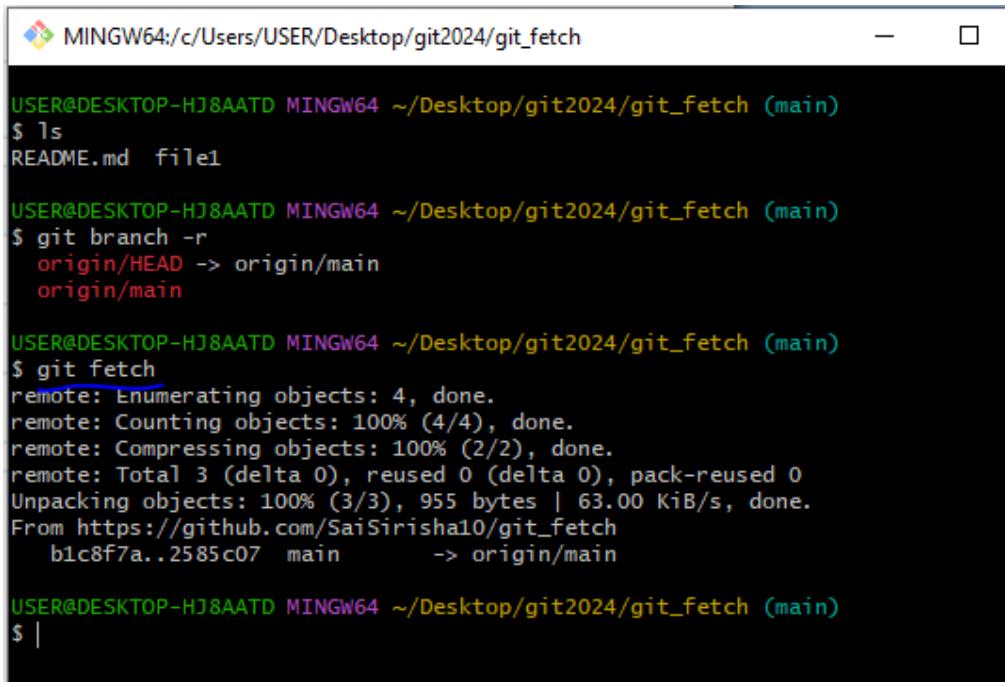
  Step 1:

```
MINGW64:/c/Users/USER/Desktop/git2024/git_fetch

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024
$ git clone https://github.com/SaiSirisha10/git_fetch.git
Cloning into 'git_fetch'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024
$ ls
git_fetch/

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024
$ cd git_fetch

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ ls
README.md  file1

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ |
```

- Now create another file in git hub repo i.e file2

Step 2:


Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
main
git_fetch /
SaiSirisha10 Create file2
Name
Last commit message
README.md
Initial commit
file1
Create file1
file2
Create file2


MINGW64:/c/Users/USER/Desktop/git2024/git_fetch
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ ls
README.md file1
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ git branch -r
origin/HEAD -> origin/main
origin/main
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ git fetch
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 955 bytes | 63.00 KiB/s, done.
From https://github.com/SaiSirisha10/git_fetch
b1c8f7a..2585c07 main -> origin/main
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$

Step 3:

MINGW64:/c/Users/USER/Desktop/git2024/git_fetch

```
$ git checkout origin/main
Note: switching to 'origin/main'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to fals

HEAD is now at 2585c07 Create file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch ((2585c07...))
$ ls
README.md  file1  file2
```

Step 4 :

Step 5:

MINGW64:/c/Users/USER/Desktop/git2024/git_fetch

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch ((2585c07...))
$ ls
README.md  file1  file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch ((2585c07...))
$ git checkout main
Previous HEAD position was 2585c07 Create file2
Switched to branch 'main'
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ git status
On branch main
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ |
```

step 6 :

```
MINGW64:/c/Users/USER/Desktop/git2024/git_fetch

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ git merge origin/main
Updating b1c8f7a..2585c07
Fast-forward
 file2 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ ls
README.md  file1  file2

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ |
```

- In the same way create two files in the git hub repo i.e file3 and file4 and come back to the local repo then fetch the data from git hub using

  git fetch

- Then enter into the origin/main branch

  git checkout origin/main

- Next check the log of the files commit in our git hub repo copy the commit id of the recent id and come back to the main branch

  git checkout main

- Now use the cherry pick command and copy the commit id that we are copied in the origin/main branch

  git cherry-pick <commit id>

MINGW64:/c/Users/USER/Desktop/git2024/git_fetch

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch ((80d2765...))
$ git checkout main
Previous HEAD position was 80d2765 Create file4
Switched to branch 'main'
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ git cherry-pick 80d2765c5f747585a6ebc3f90b0cc34c9c2a319d
[main 5aee009] Create file4
 Author: SaiSirisha10 <saisirisha1005@gmail.com>
 Date: Mon Aug 5 07:34:30 2024 +0530
 1 file changed, 1 insertion(+)
 create mode 100644 file4

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ ls
README.md  file1  file2  file4

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ |
```

step 7:

MINGW64:/c/Users/USER/Desktop/git2024/git_fetch

```
USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (main)
$ git checkout -b feature2
Switched to a new branch 'feature2'

USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (feature2)
$ git push
fatal: The current branch feature2 has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin feature2

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.


USER@DESKTOP-HJ8AATD MINGW64 ~/Desktop/git2024/git_fetch (feature2)
$ git push --set-upstream origin feature2
|
```

step 8 :

- In this way we can fetch the data from git hub to the local repo and local repo to git hub.

- **Difference between git pull and git fetch command?**

Git pull copies the changes from a remote repo directly into our working directory but git fetch does not.

----End------