* **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:

1. If we have a team and all want to work on the same project it is not possible
2. If machine crashes and all data will be lost.
3. 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.

1. 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.

1. 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.

1. LOCAL REPO / GIT REPO :

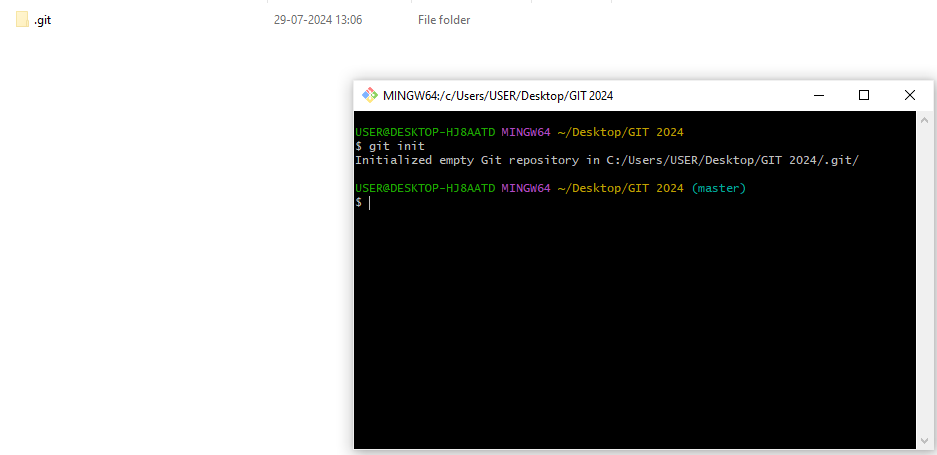
It is where git stores the data and object database for your project. GIt Commit, files will be added to local/git repo & then we track the file versions. Commit Id are created here.

* **TO INSTALL GIT :**

1. sudo yum install git
2. mkdir git
3. git init
4. ls -a

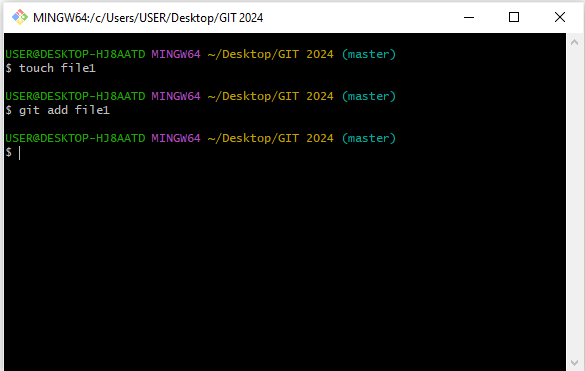
* **git init:**

Thiscommand is used to initialize the respective directory as git repository



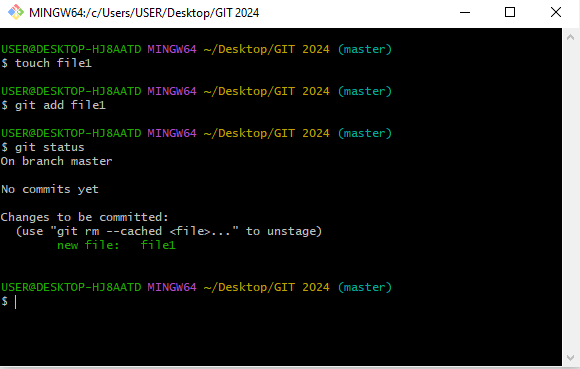
* **TO CREATE AND ADD FILES:**

1. touch file1
2. git add file1



* **git status:**

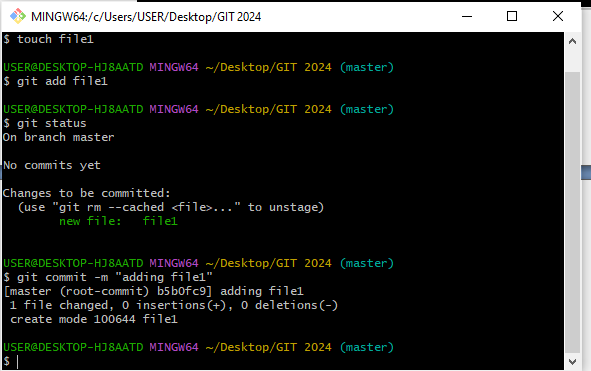
It will show whether files are in workspace or staging are on in git repo



* **git commit :**

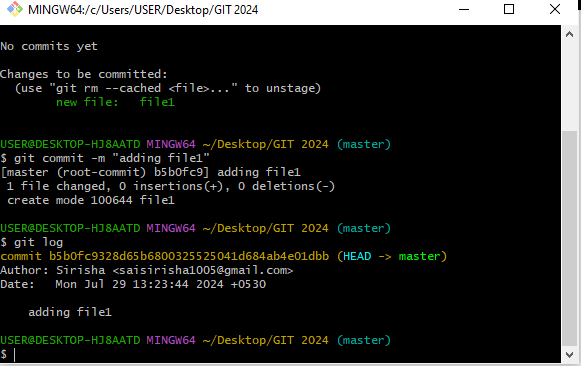
This command commits the staged changes to the local repository.

git commit –m “adding file1”



* **git log:**

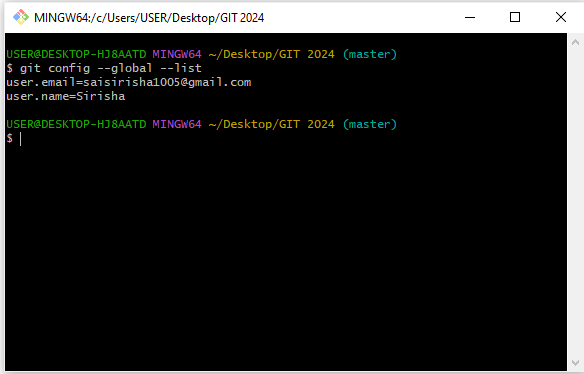
It gives history of file or repo that we have committed.



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?**

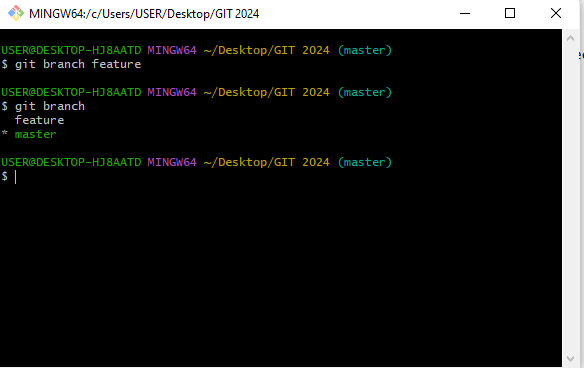
To develop new features we go for branching.

* **HOW TO CREATE A BRANCH?**

git branch <branch name>

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

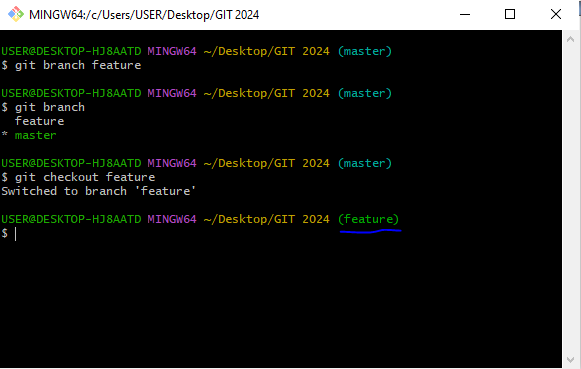
git branch



**“\*” indicates the current branch.**

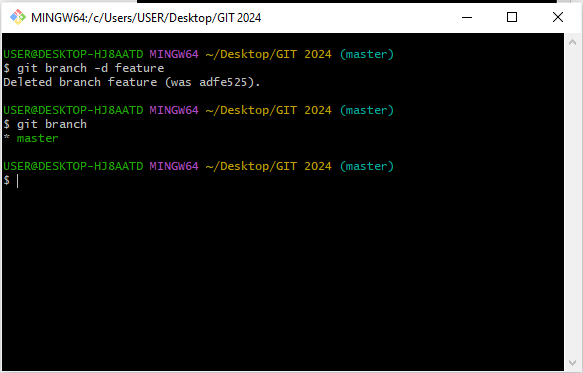
* **HOW TO ENTER INTO A BRANCH?**

git checkout <branch name>



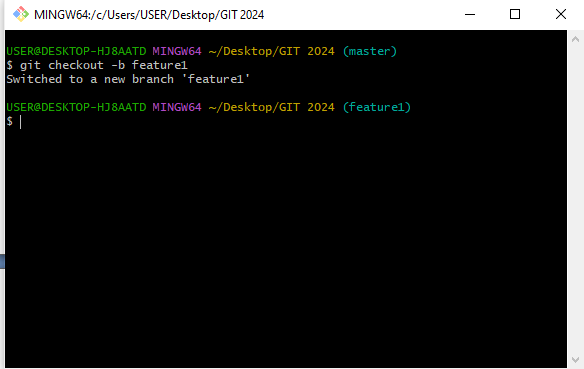
* **HOW TO DELETE A BRANCH?**

git branch –d <branch name>



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

git checkout –b <branch name>

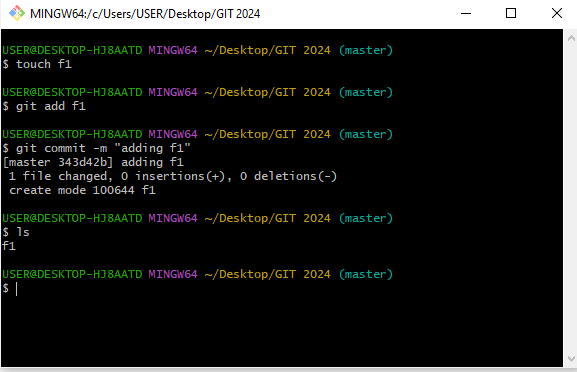


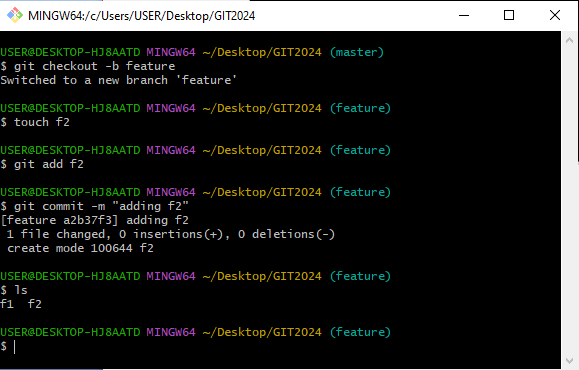
* **MERGING:**

It is used to merge specified branch to checkout branch.

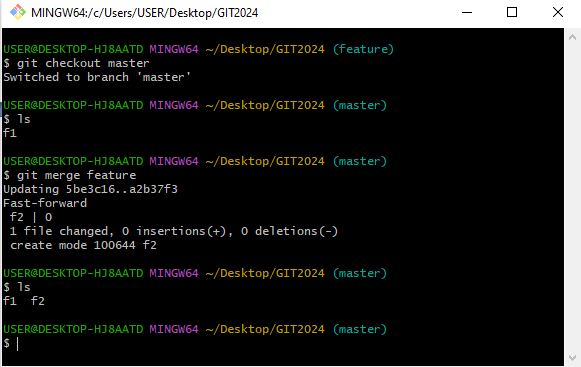
git merge <branch name>

Step 1:



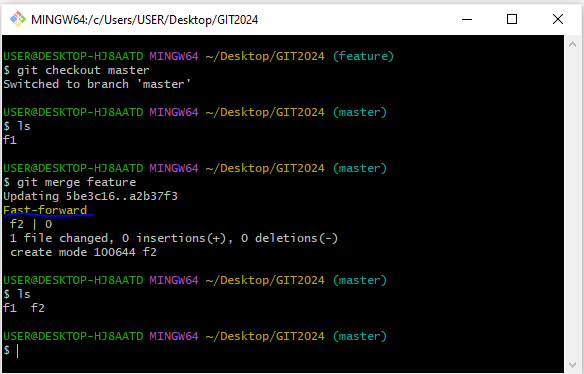


Step 3:

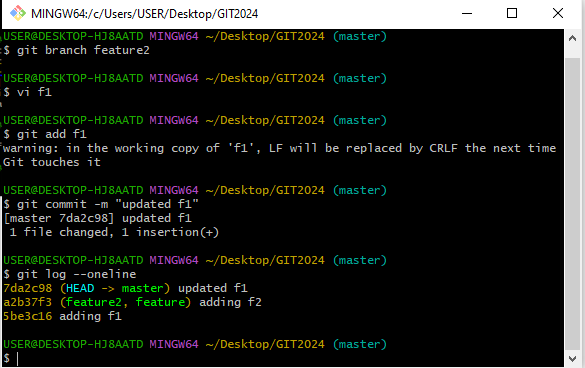


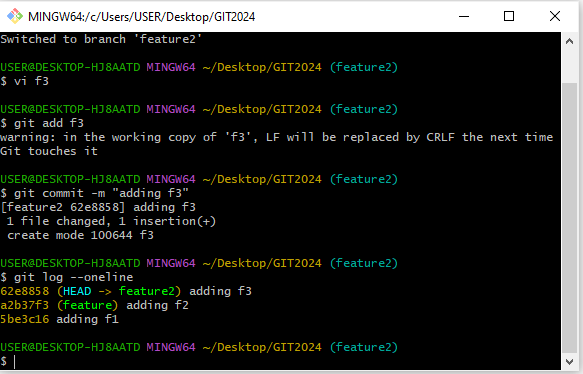
* There are two types of merges.

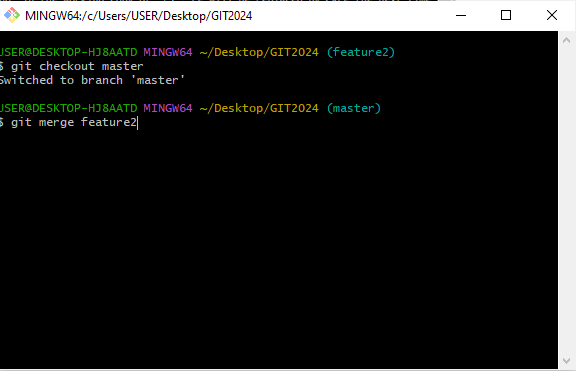
1. **Fast-forward merge**: If we merge the branch into the master and no commits are made in master branch then it is called fast-forward merging.

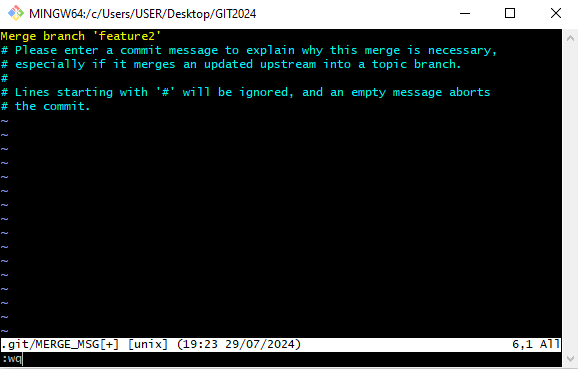


1. **RECURSIVE STRATEGY:** If we made changes in the master branch as well as feature branch if we try to merge the feature branch into master branch these two changes has to be merge into another new commit.

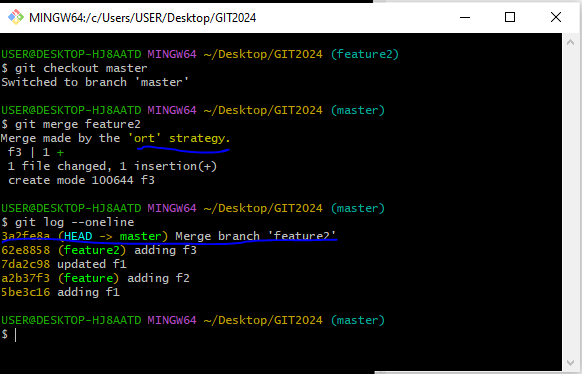
Step 1: 

Step 2: 

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: 

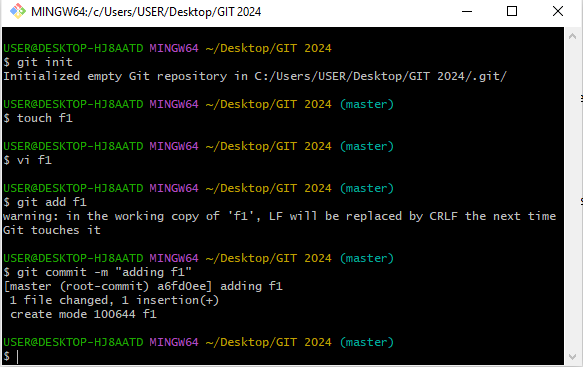
* **MERGE CONFLICT:** It occurs when same piece of code is been worked on different branches and when we try to merge it the conflict occurs.
* There are two types of merge conflicts

1. Two developers modify the same line of code in different branches.

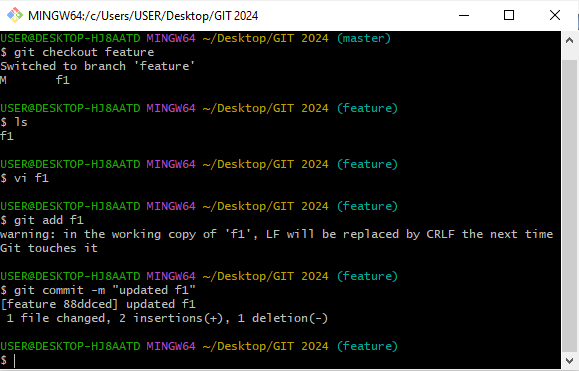
2. When a file is deleted in one branch and modifies it in another branch

3. When multiple branches are being merged with the changes scattered across various files and lines.

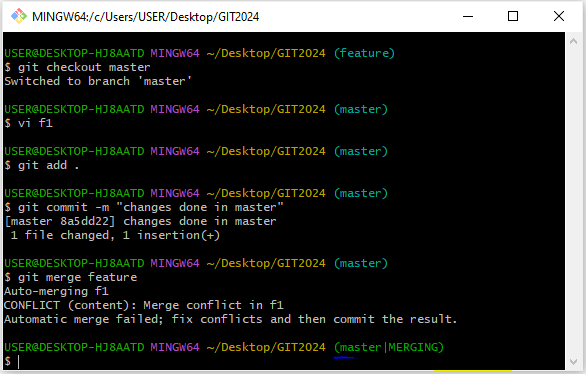
Step 1:



Step 2:

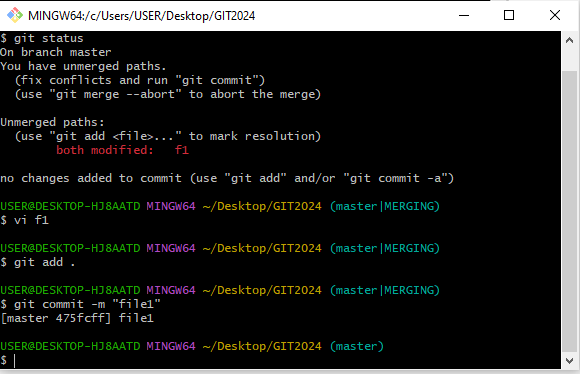


Step 3:



* **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

* **Tagging:** 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>

1. Switch into the tag

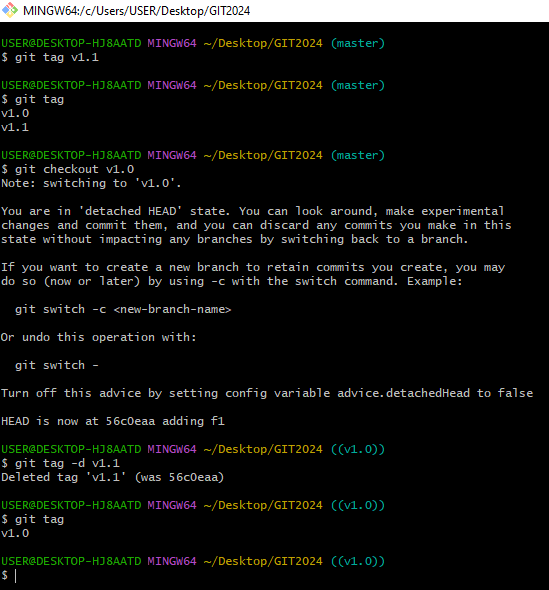
git checkout <tag name>

1. To list all tags

git tag

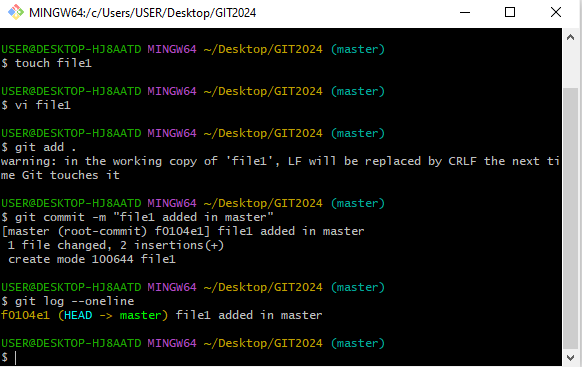
1. To delete the tag

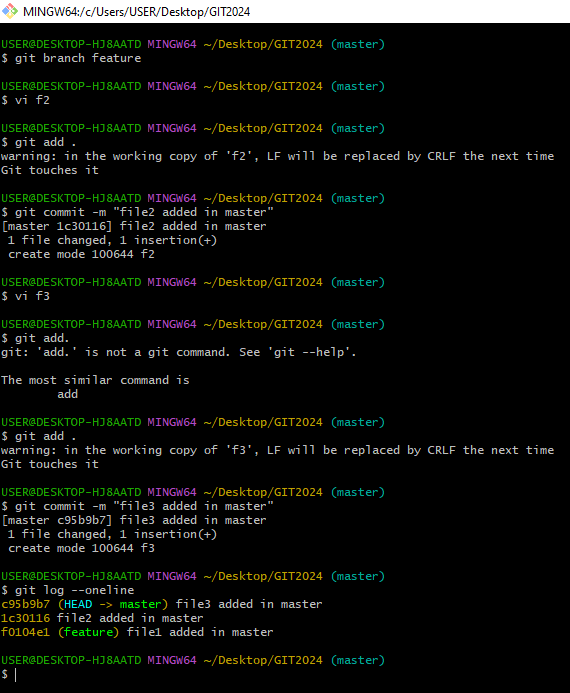
git tag –d <tag name>



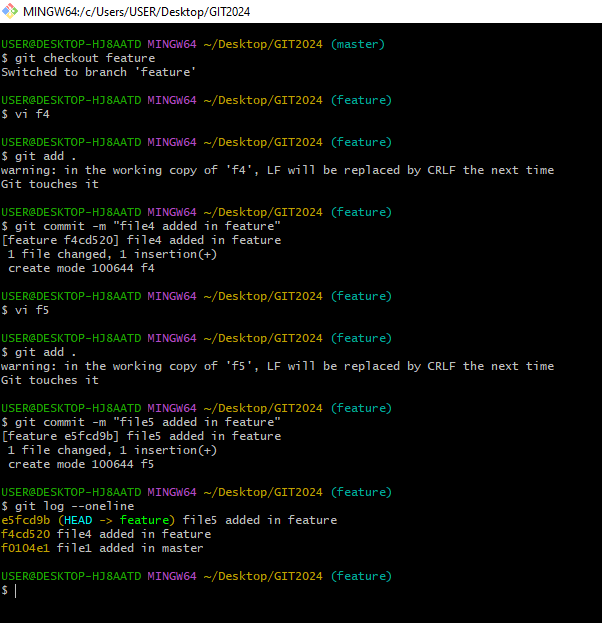
* **Difference b/w the tag and branch:**
* tag is a name given to a set of files.
* Branching is for parallel development.
* **REBASE:**
* It is similar to merge and rewrites the commits history
* It is used to clean up our local history
* It is the advanced command used rarely
* **COMMON PLACES WHERE WE CAN USE REBASE**
* Cleaning up your commits before sharing your branch
* Pulling changes from another branch without merge
* **WHERE WE SHOULDN’T USE REBASE**
* When the branch is public when it is shared to all the developers
* Most of them prefer merge rather than rebase.

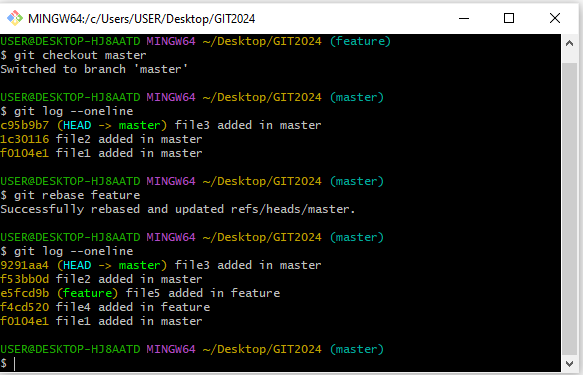
Step 1:

****

Step 2: 

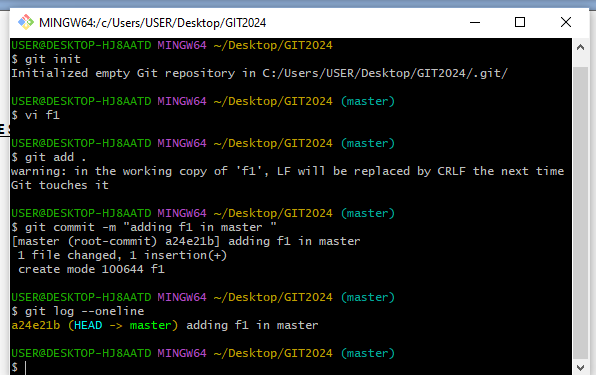
Step 3:



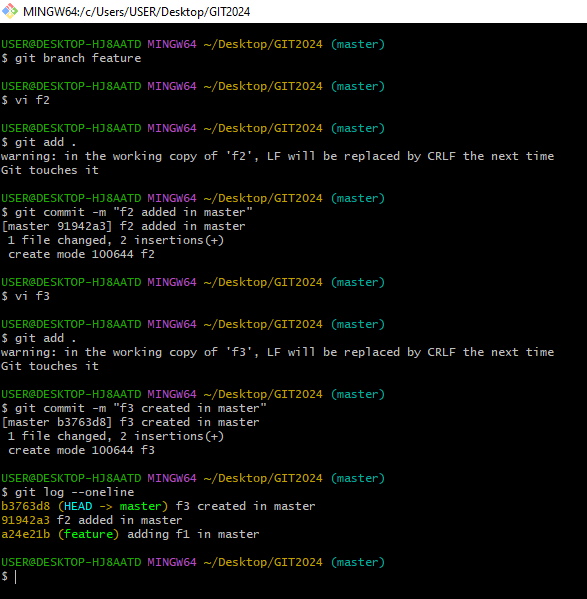
Step 4:

* 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.
* **IN THE SAME WAY WE WILL DO FOR MERGE:**

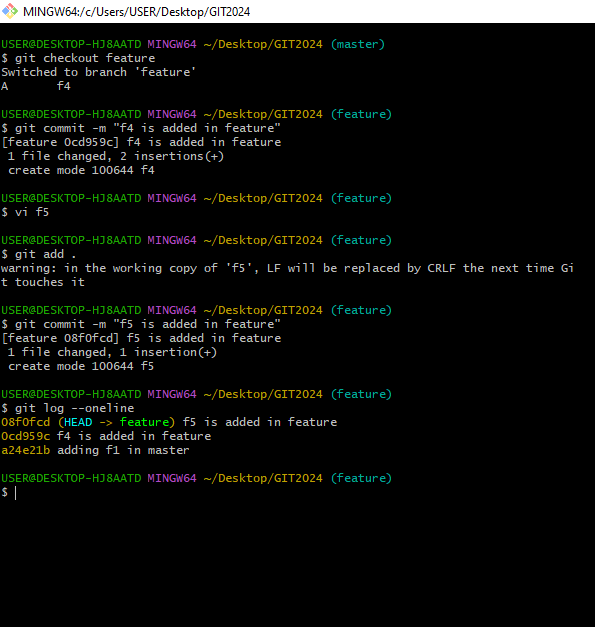
Step 1:

****

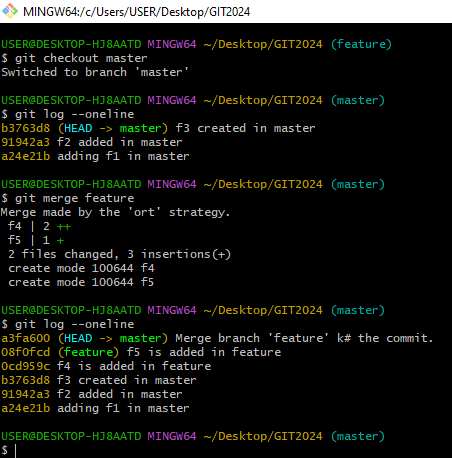
Step 2:

****

Step 3:

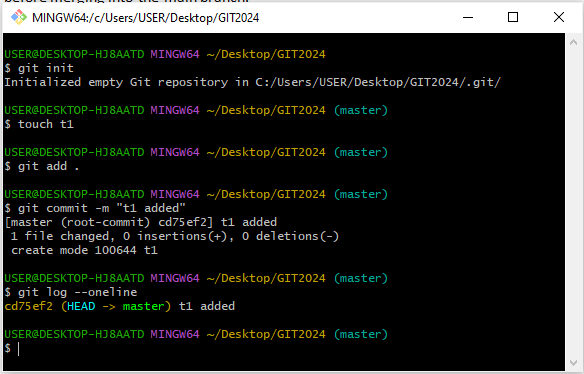
****

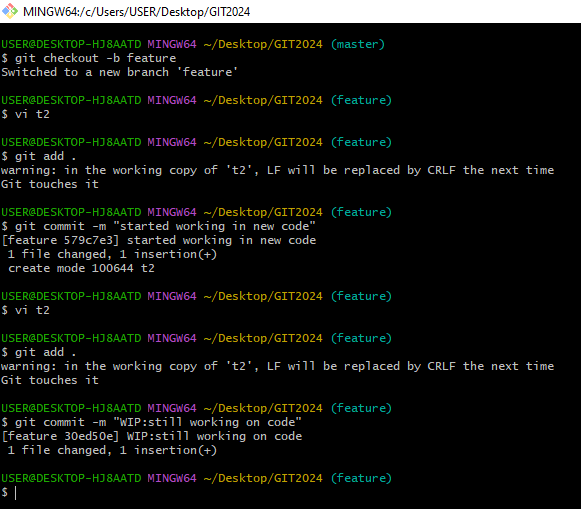
Step 4:

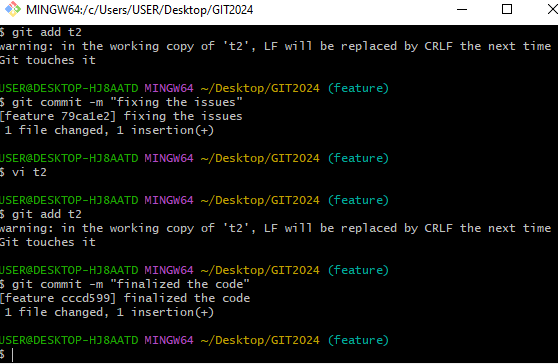
****

* Here when we do merge extra commit will be added at a top and remaining commit ids will remain same. It preserves the history.
* **DIFFERENCE B/W MERGE AND REBASE:** 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.
* **INTERACTIVE REBASING**: By using this we can combine the multiple commits into single commit before merging into the main branch.

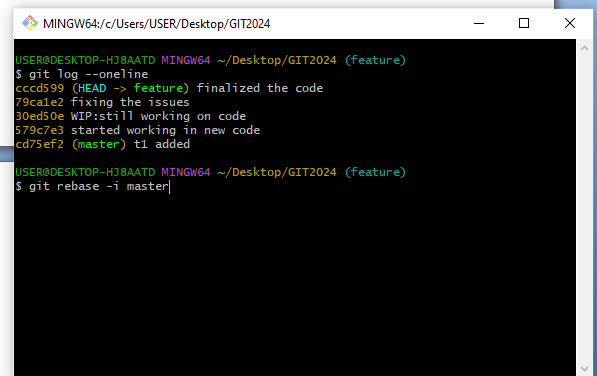
Step 1:



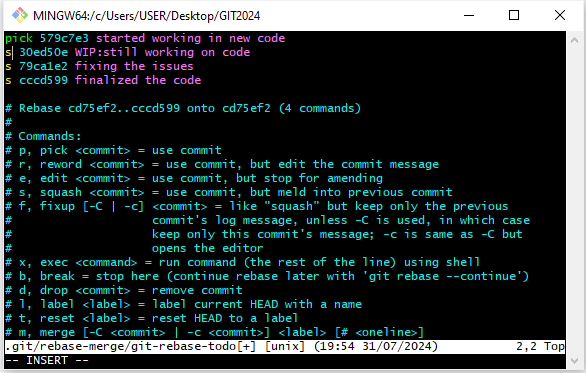
Step 2: 

Step 3: 

Step 4:

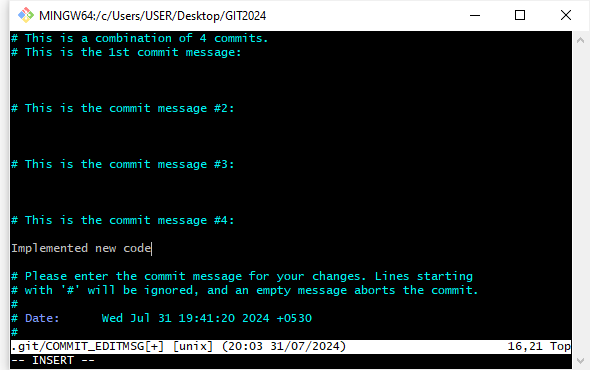


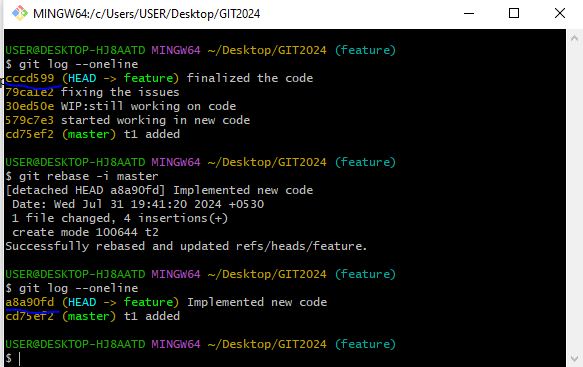
Step 5:



* 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

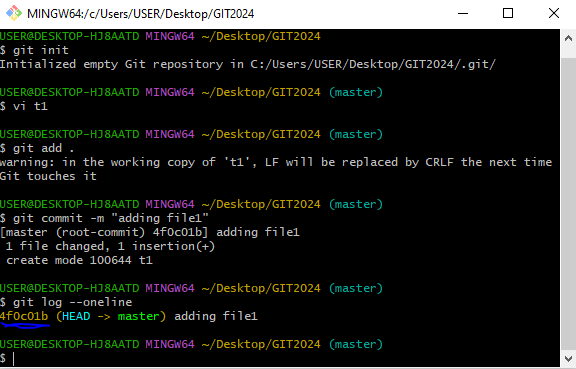
Step 6:

Step 7:

* 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.
* **AMEND:**  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

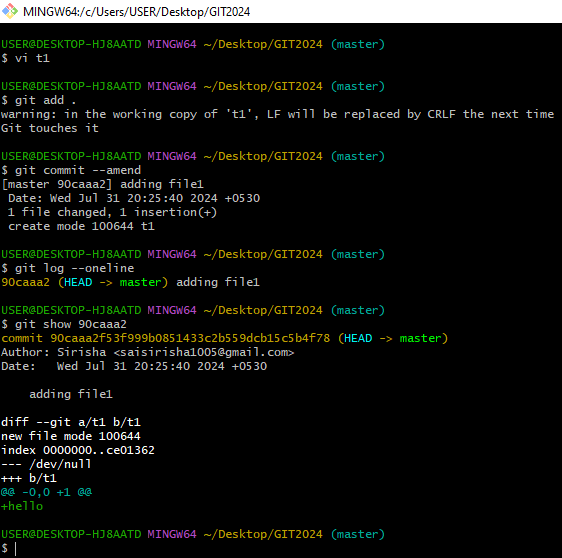
* We can use this amend command when the file is in our local repo only

Step 1: 

* 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
* **git show:** 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

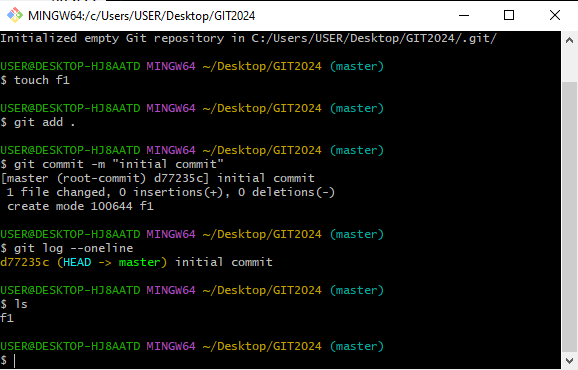
git show

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

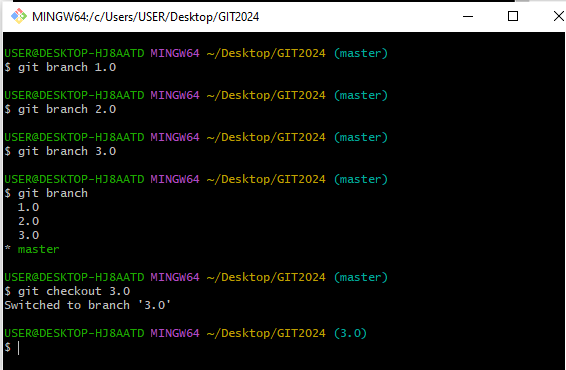
Step 2: 

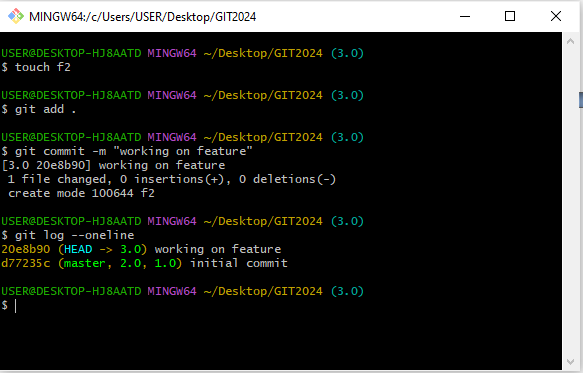
* **GIT CHERRY-PICK:** It is used we want to apply the particular commit from one branch to another branch.
* It is mainly used if we don’t want to merge the whole branch but you want some of the commits then we can use this cherry-pick command
* It is mainly used for bug fixes where we want to place 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 :

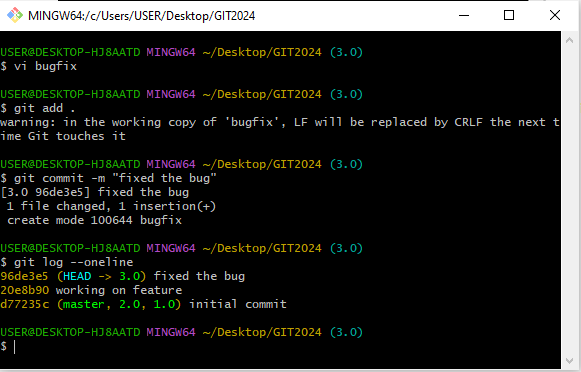


Step 2:



Step 3: 

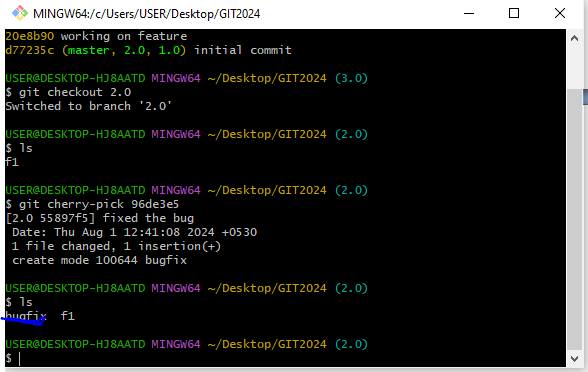
* Now we found a bug in the current feature and we make to know that this bug is present all other features in this case we use this cherry-pick command.

Step 4: 

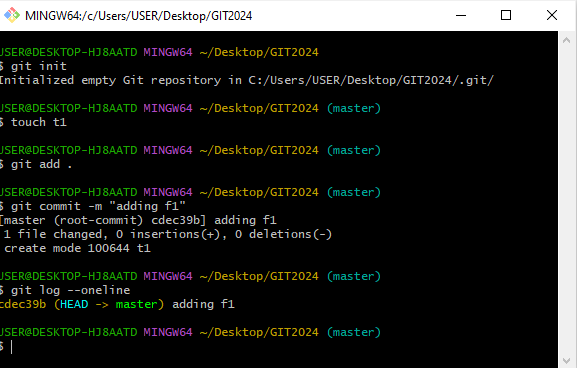
* In this case I have added a file in 3.0 which contains the bug fixes if I want to add this bug fix file in remaining branches without merging the current branch bcz we are still working in 3.0 branch then we will use this cherry-pick command.
* By using the commit id we can merge into another branch(currently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

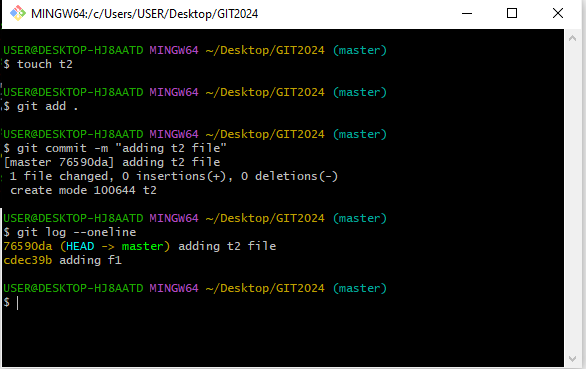
**git cherry-pick <commit id>**

Step 5:



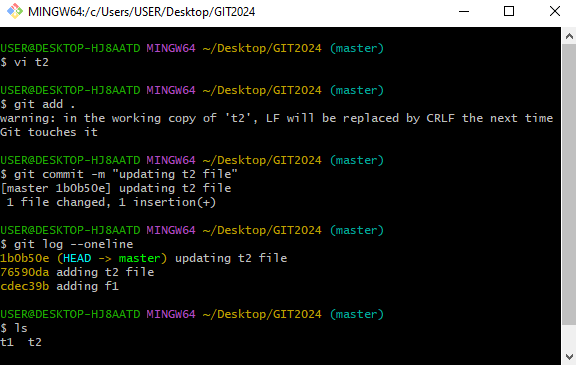
* **GIT RESET :**It is used to move the branch from one commit to another commit. We can also go back to the workspace but commit history will be removed.
* Reset moves the current branch and optionally copies the data from the repo to the working or staging area.
* Reset has three different options
* --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

**Step 1:** 

Step 2: ****

* 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:

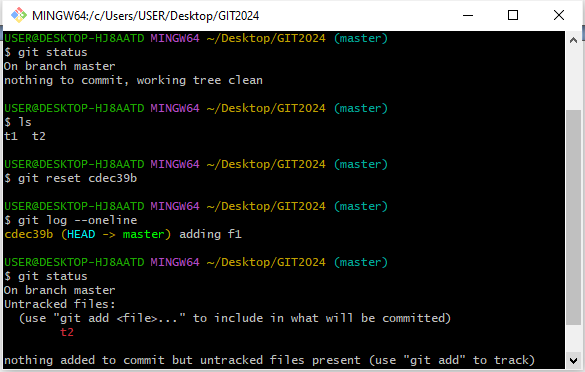
****

* 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

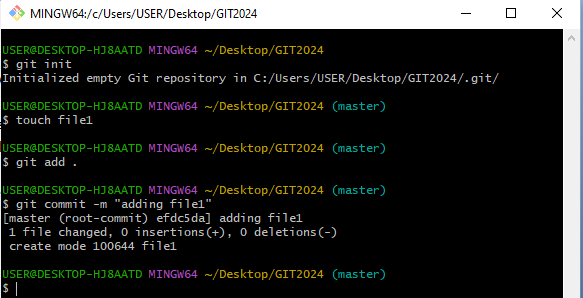
step 4:

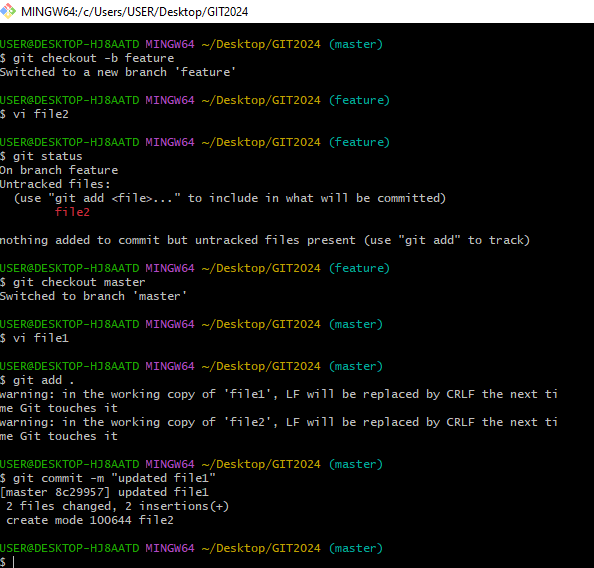


* To delete a file completely in staging area we can use this command

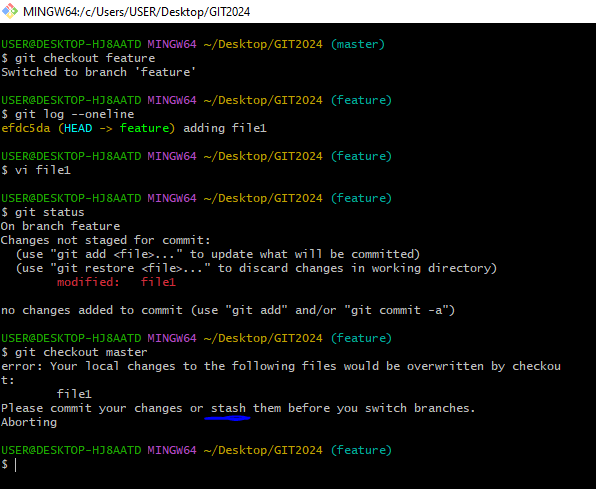
git reset <commit id > --hard

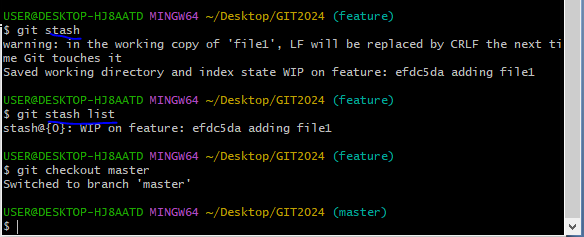
* **Git Stash:** If you want to switch a branch but you are working on an incomplete part of a current branch and you want to go back to other branch but you don’t want to commit your half done work this git stash allows us to do.
* This command enables us to switch the branches without committing the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

Step 1:

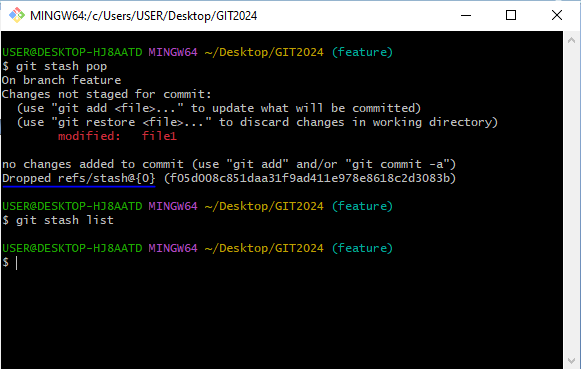
Step 2:

**Step 3:**

****

Step 4:

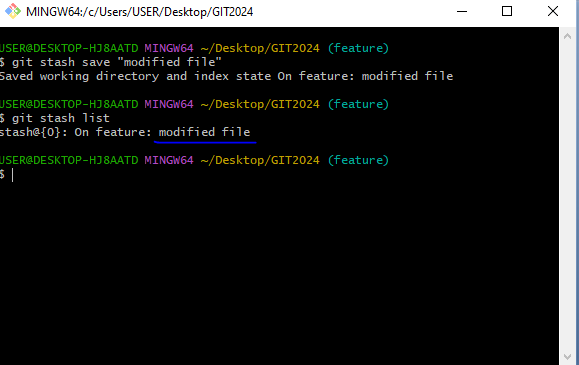
* **git stash list** command is used to list the stash that are saved.
* **git stash pop** command is used to take the recently pushed stash and delete it from the stash and give it to the branch
* In this way we can stash the file and unstash it

Step 5: 

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

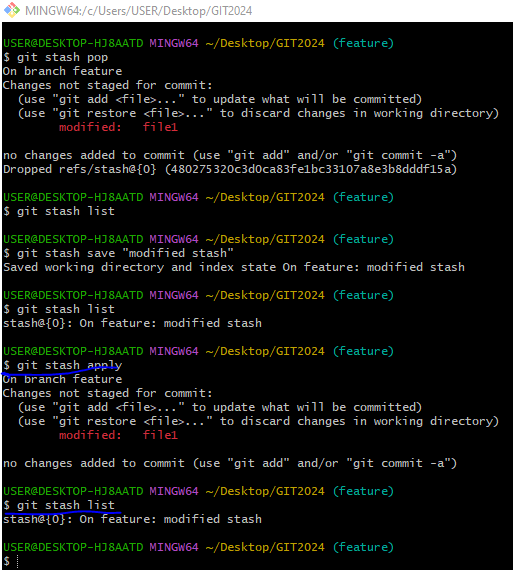
git stash save “ ”

Step 6:



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

git stash apply



* **git checkout:**
* 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

* **git switch command**:
* git switch command: It also moves from one branch to another branch
* It is alternative to git checkout command

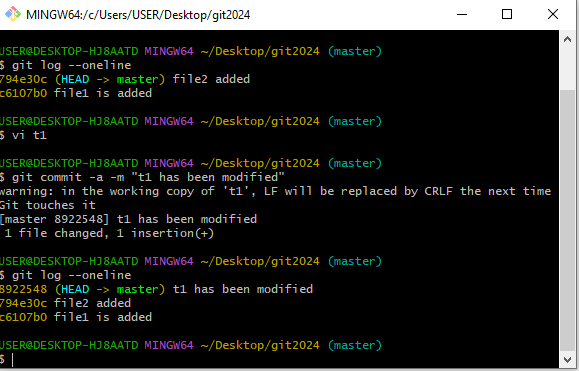
git switch <branch name>

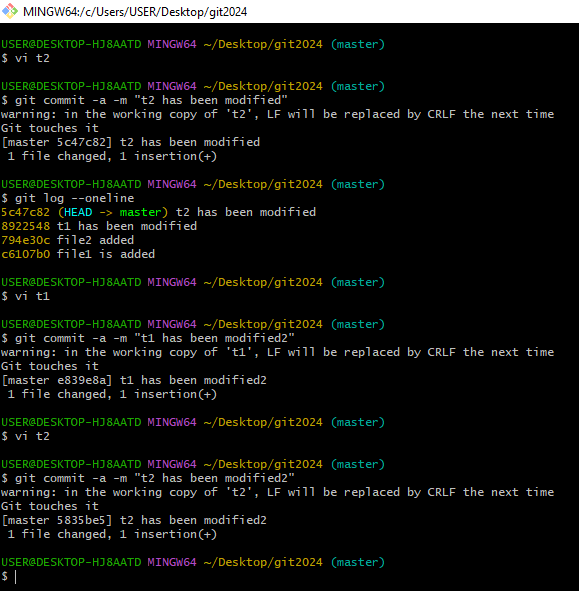
* To create a branch and enter into the branch

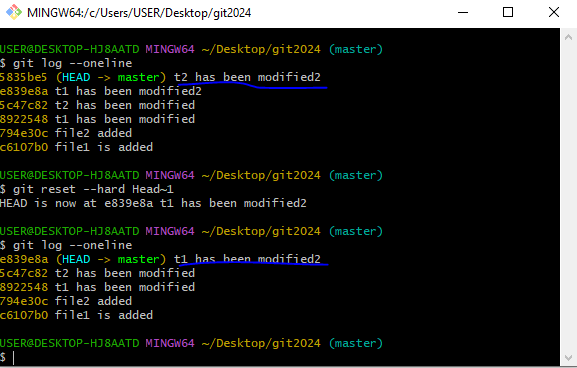
git switch –c <branch name>

* Switch command doesn’t move to the particular commit hash but git checkout command go back to the particular commit hash
* **git revert command:** 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:**

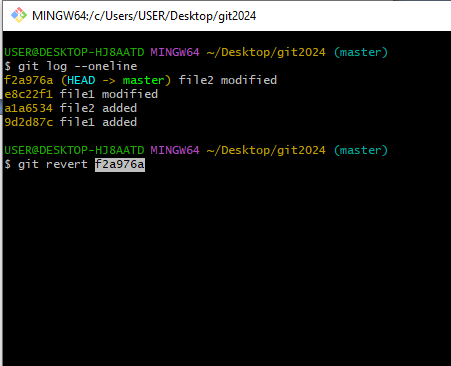
1. In revert we can go back to the workspace after committing, but commit history will be stored
2. In reset we can go back to the workspace after committing, but commit history is removed

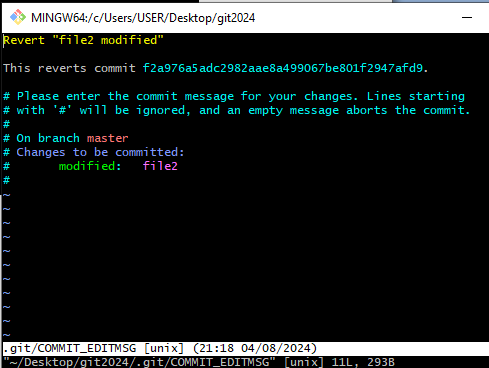
Step 1: 

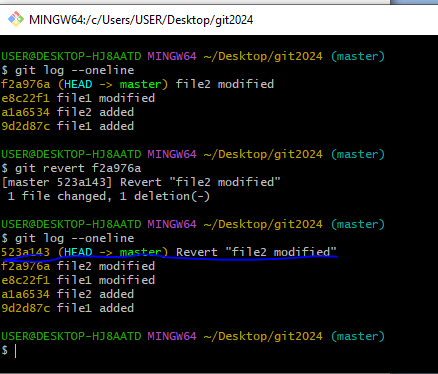
Step 2:

Step 3: 

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

Step 1: 

Step 2: ****

Step 3: ****

* **When to use revert and reset command:**
* Reset is used only when your commits are in local
* Revert command is used when your commits are went into remote.
* **When u want to revert the particular file :**

git revert <commit id >

* **git diff command:** 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

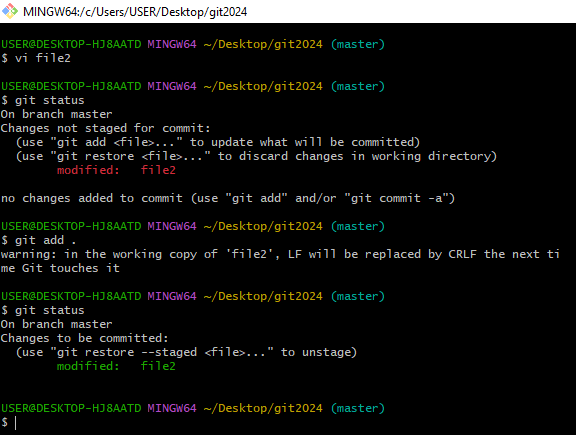
git diff

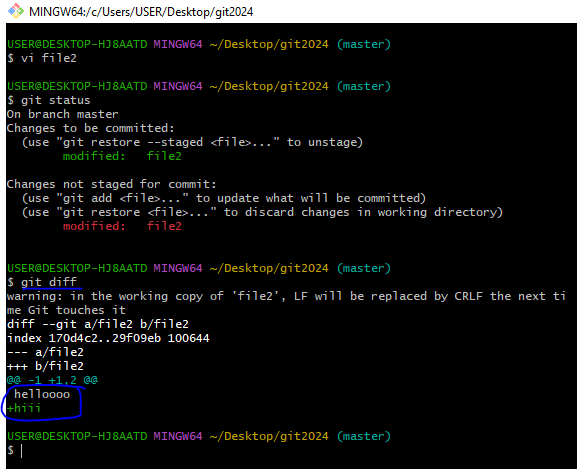
* To check the changes between staging area and repo area

git diff -- staged

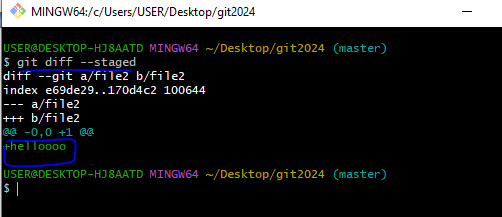
* To check the changes between repo area and working directory

git diff -- head

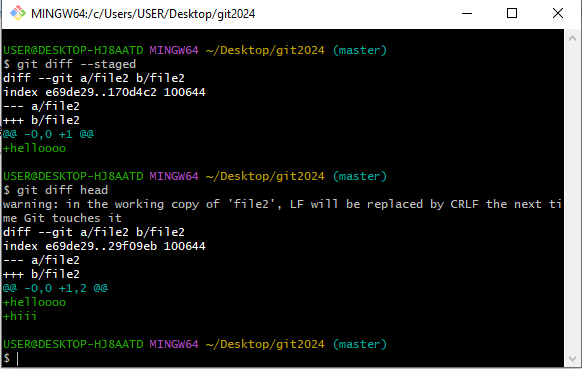
Step 1:****

Step 2:

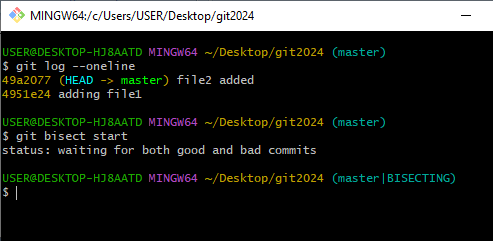
* here the file is in staging area and the same file is in working directory by using this command we can find the diff between the same file what the data has been added.

Step 3:

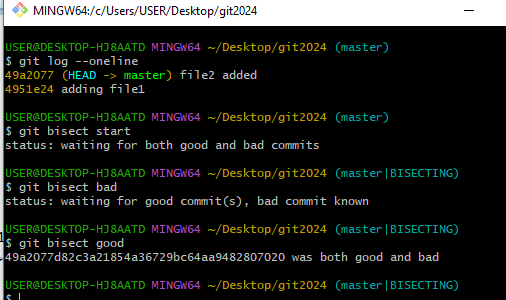
* here the file is still in the staging area and nothing is present in the repo area so this gives the command between them

Step 4:

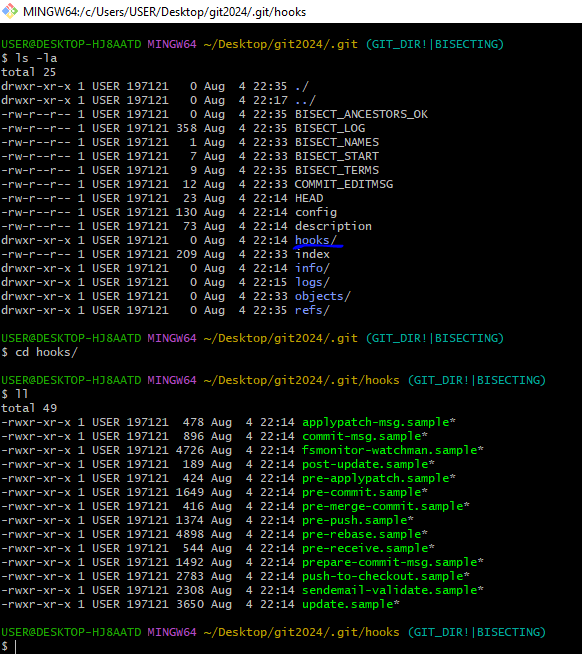
* to check the difference between the repo and working area we can use this command
* **git bisect:** The git bisect command is used to discover the commit that has introduced a bug in the code. It helps to track down the commit where the code works and the commit where it does not, hence, tracking down 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: ****

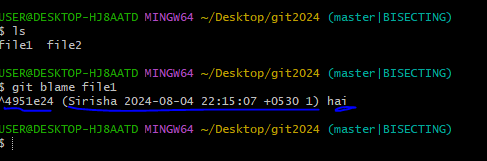
* it is used to find the good and bad commits

Step 2 : ****

* In this way we can find the good and bad commits.
* **git insta web**: Instaweb is a script used to set up a temporary instance of GitWeb on a web server for browsing local repositories**.**
* **git drop:** If we want to delete the commit we can use this git drop
* **Amazon EMR:** 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.
* **TYPES OF AMI:**
* EBS: backend instance
* Instance store : backend instance
* **What is difference between git and other repository:**
* 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
* **other repo:**
* 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.
* **git hook:** 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

****

* **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.

****

* **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

* **GIT Fork?**

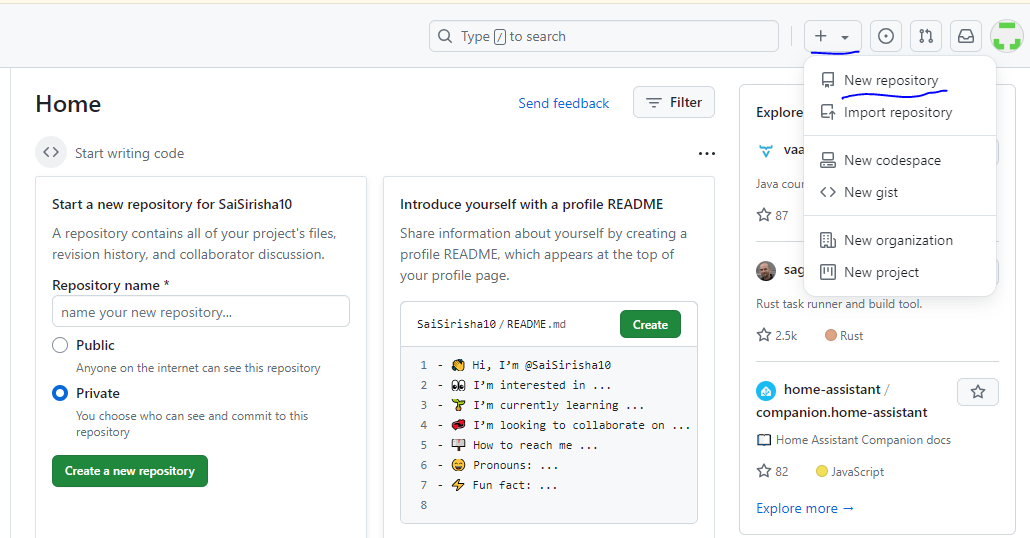
A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

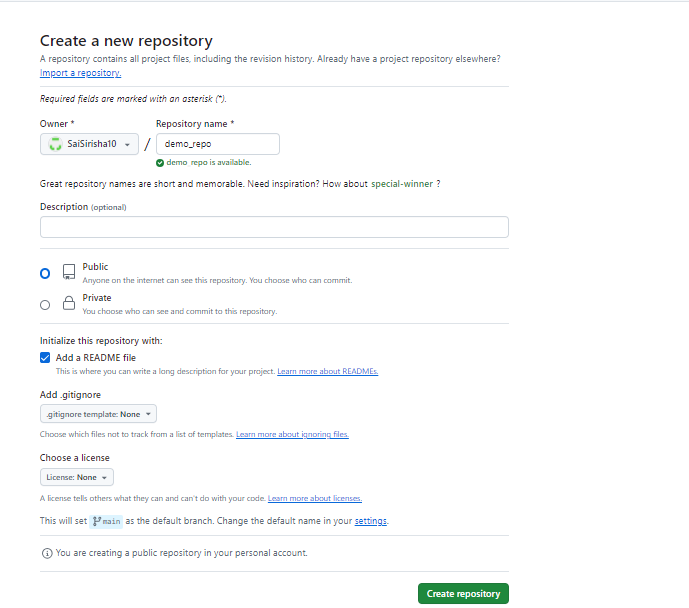
* **Central Repo:**
* git clone --> will bring central repo to local work space for the first time
* git pull --> it will compare if there are any changes, it will bring changes from central repo and merges to local repo automatically
* git push --> it moves local changes from local repo to central repo.
* 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)
* **We have two types of repositories**

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

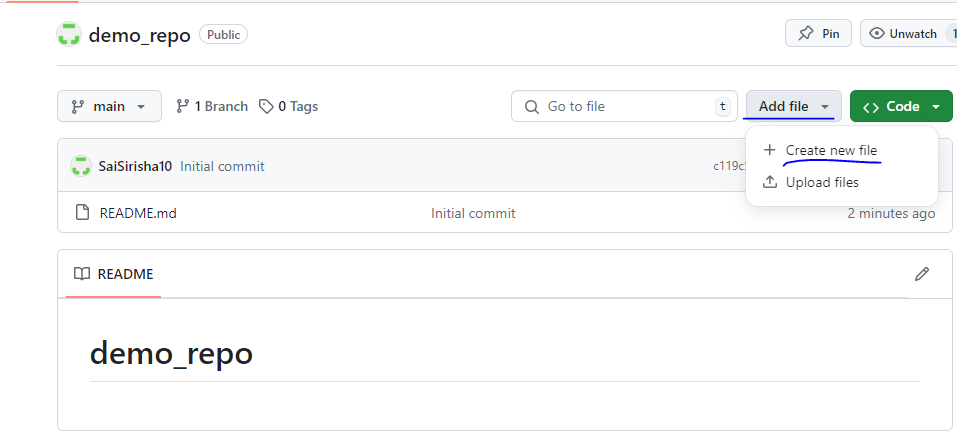
2. Non bare repository - all the git operations are performed here,we can modify files push to central ,run all git commands.

* **git fetch command**: It bring changes from central repo to separate branch (under FETCH\_HEAD) without merging.
* **To create a repo in git hub:**

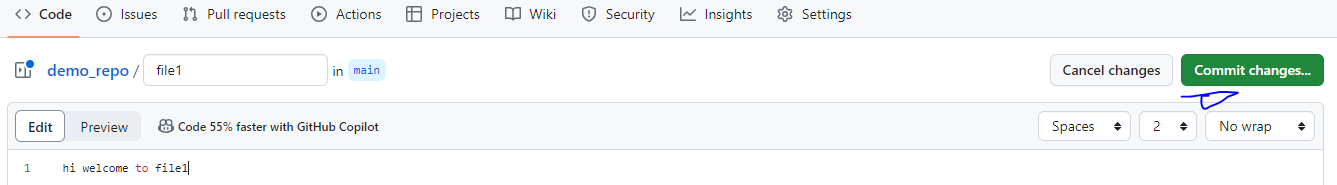
Step 1:

Step 2: 

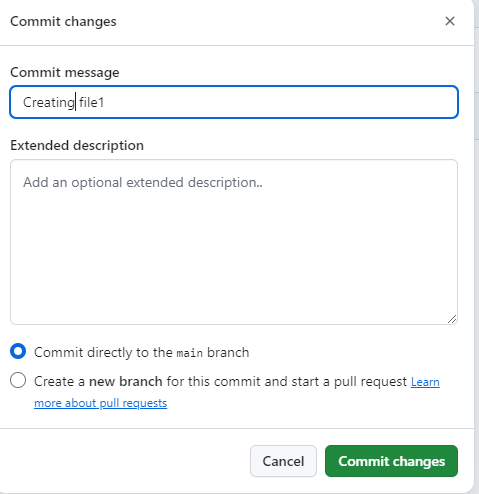
* ReadMe file is important in every project because it helps the project how to run.
* **To add a file in git hub:**

Step 1: ****

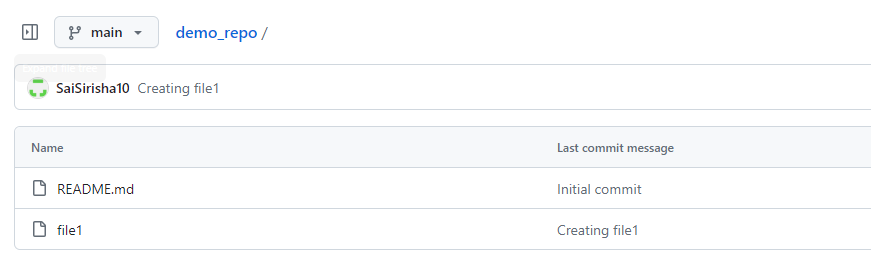
Step 2: add the file and click on commit changes



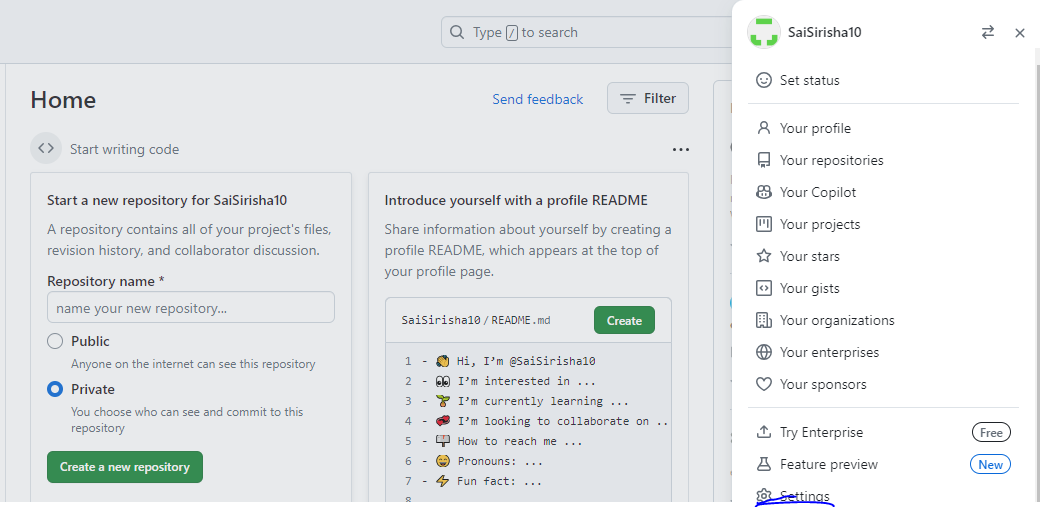
Step 3: To commit a file

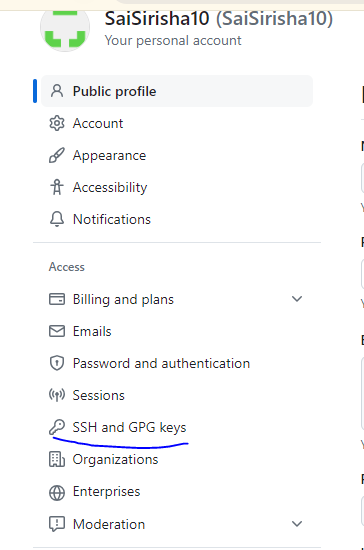


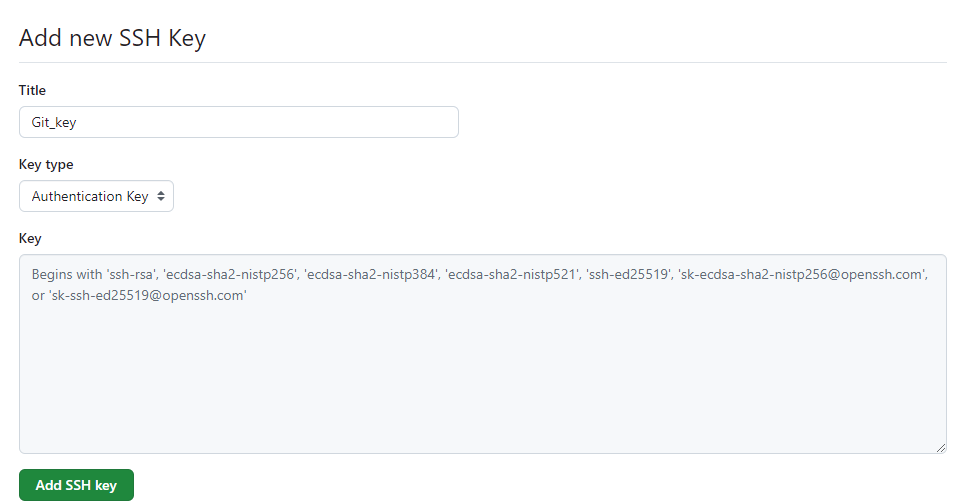
Step 3 : file has been created.



* **To generate a SSH key in git hub:**

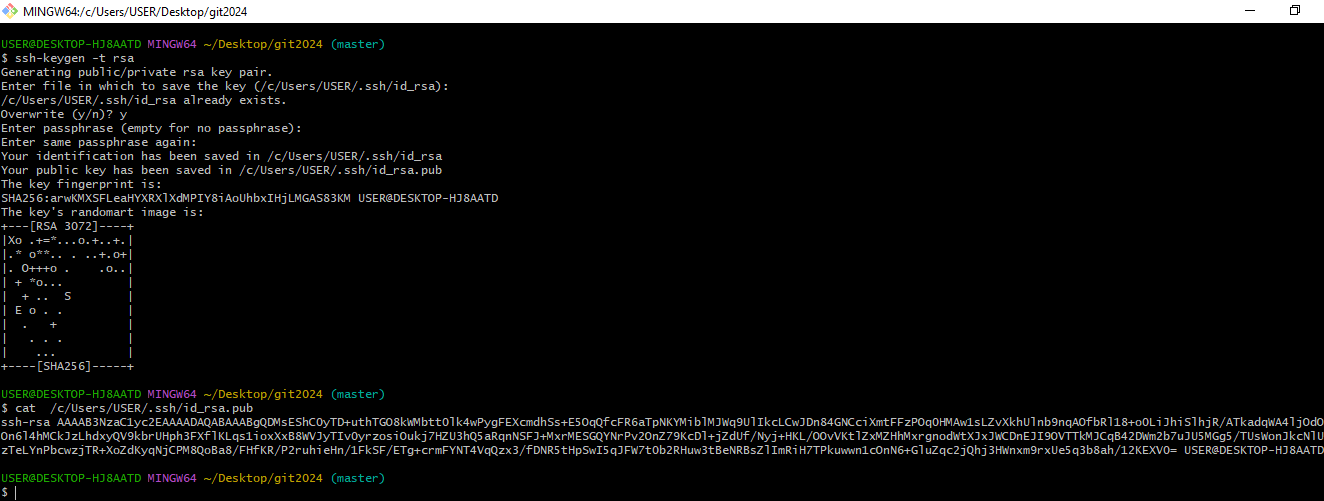
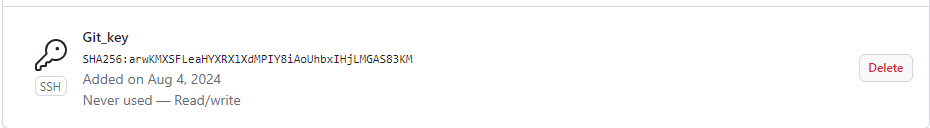
**Step 1:**

Step 2: ****

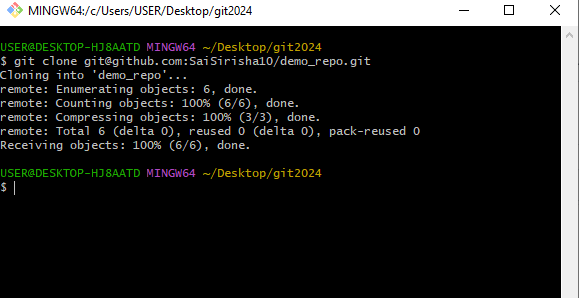
Step 3:****

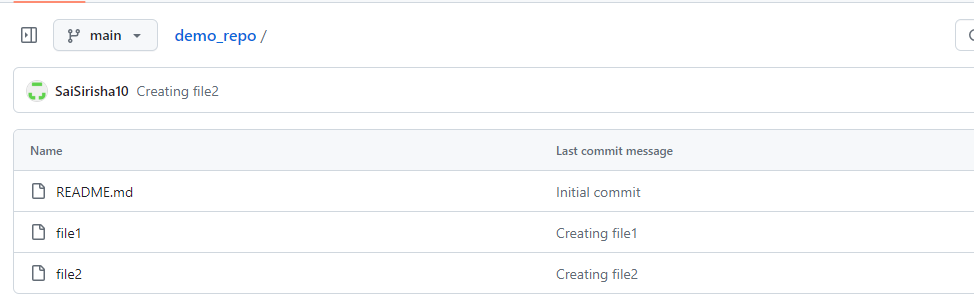
* Now we need to generate ssh key in the terminal using the command

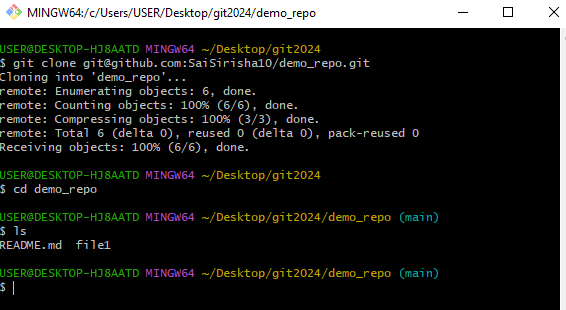
ssh-keygen –t rsa

* **Step 4:**  after generating the key we need to paste in git hub ****
* Step 5: new ssh key has been generated****
* **To clone the repo from central to local repo:**

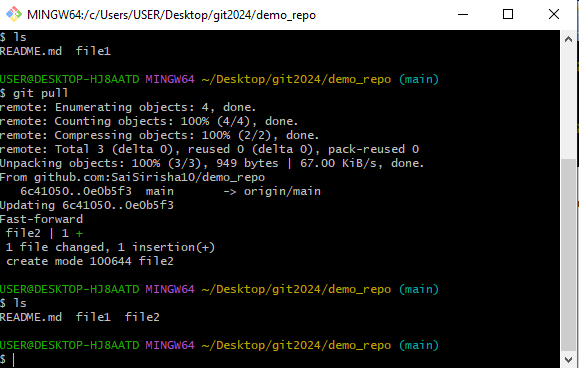
git clone

step 1:

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

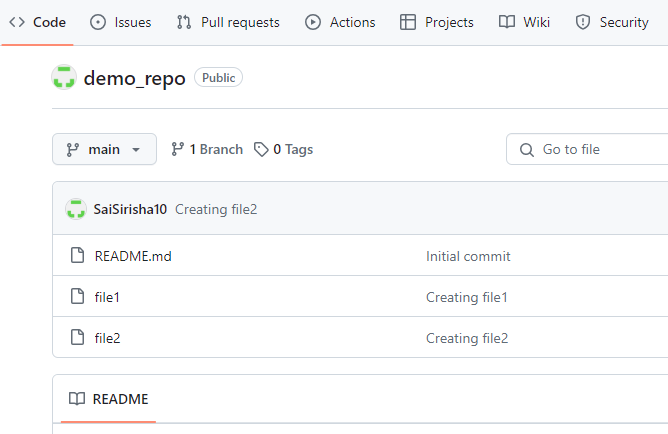
step 3:

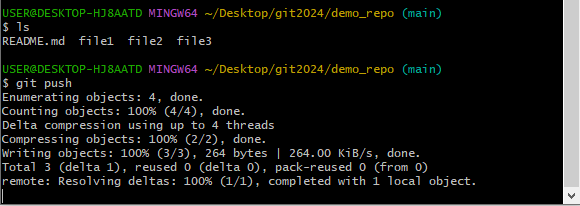
* In this use we need to use **git pull** command to reflect the changes in local repo

Step 4: 

* 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.

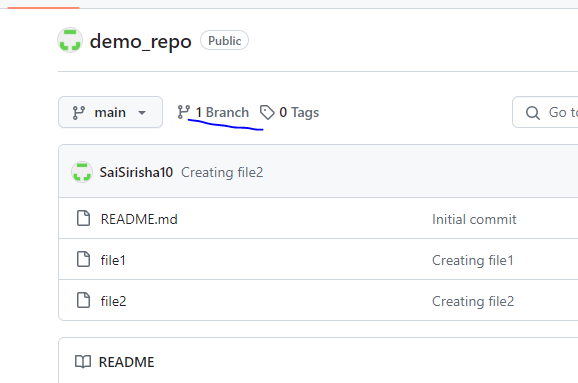
Step 5:

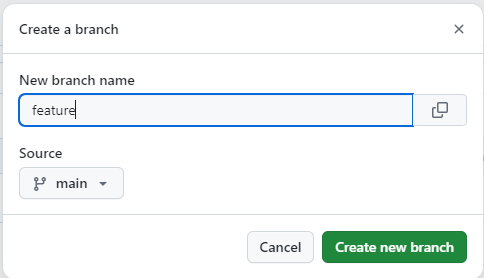
Step 6: the changes has not reflected in central repo .

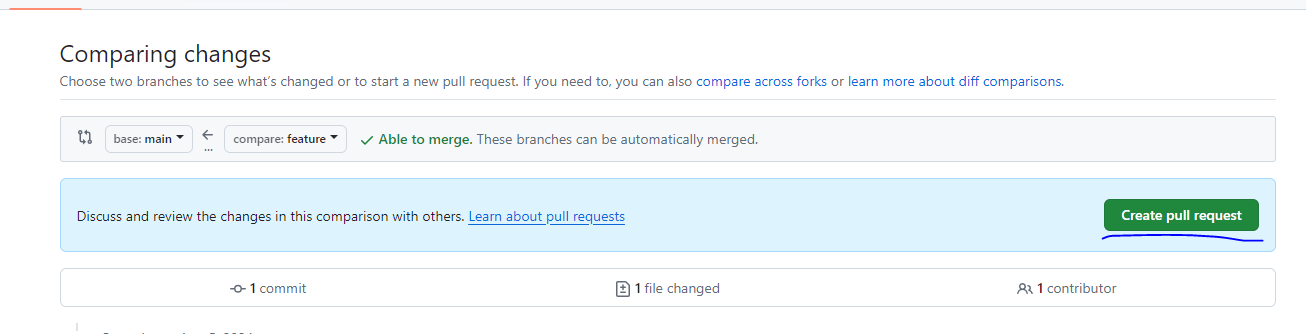
Step 7: now the changes will reflect in central repo.

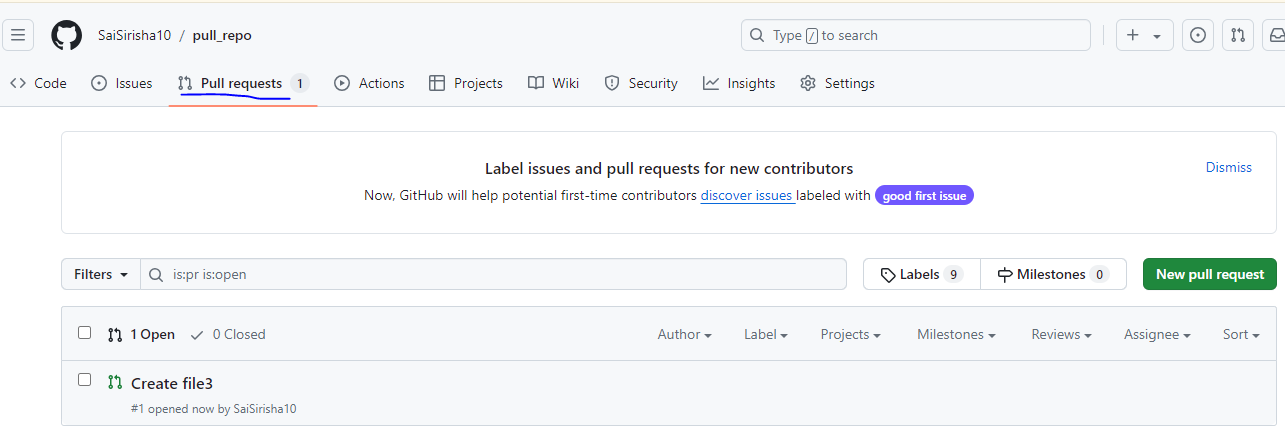
* **PULL Request:** Firstly we need to create a branch and a file inside a branch in git hub repo
* **To create a branch:**

**Step 1:**

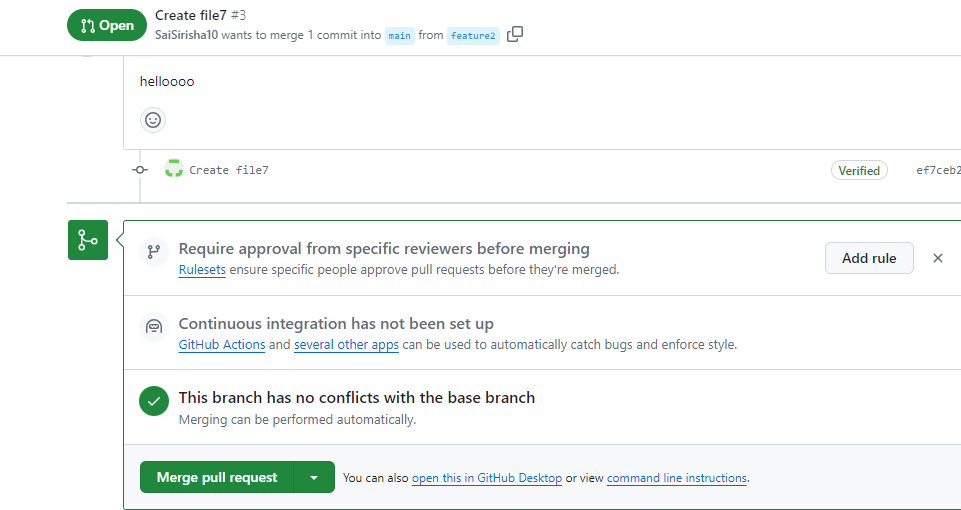
****

Step 2: 

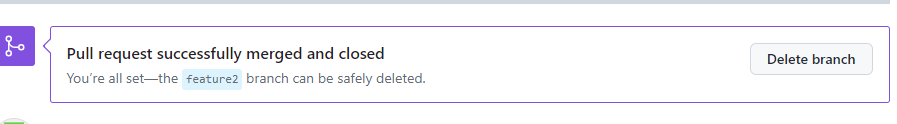
Step 3:

Step 4:

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

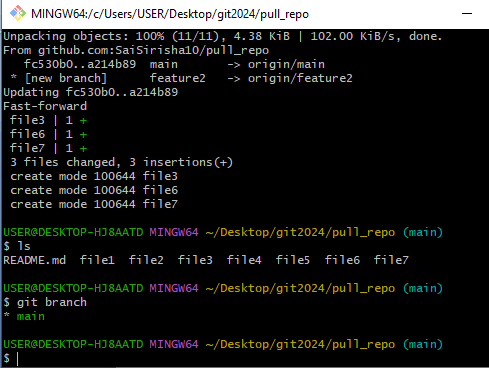


* click on the merge pull request and merge the file.



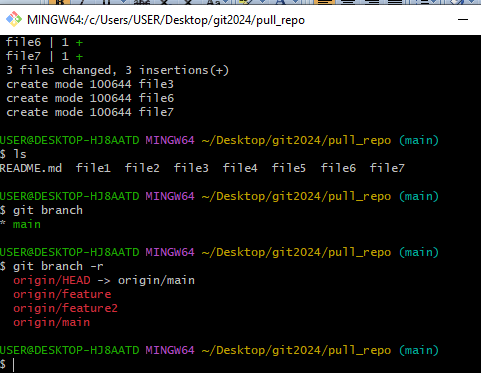
* pull request has been merged successfully
* In this way we can do pull request in git hub repo.

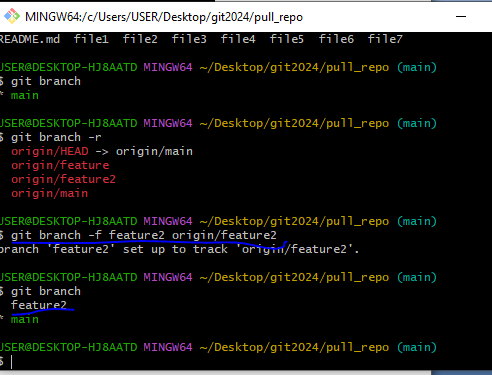
**Git fetch command:**

**Step 1: **

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

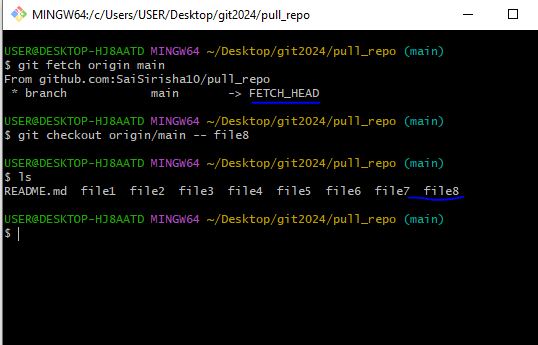
git branch -r

**Step 2:**

**Step 3 :**

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

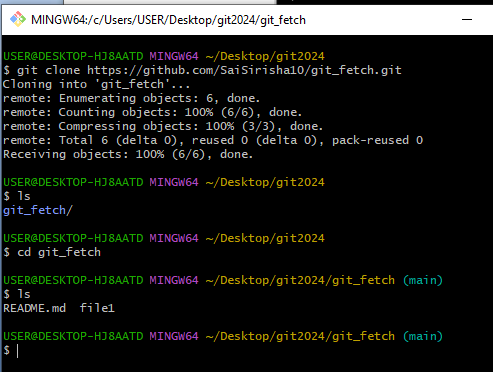
**git branch –f feature2 origin/feature2**

**step 4: **

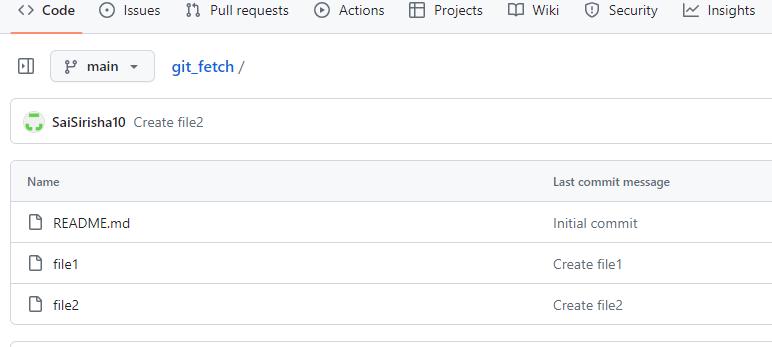
**Git fetch :**

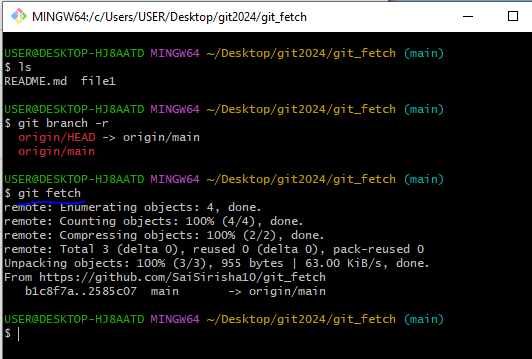
* Create a file in git hub clone and clone it to our local repo

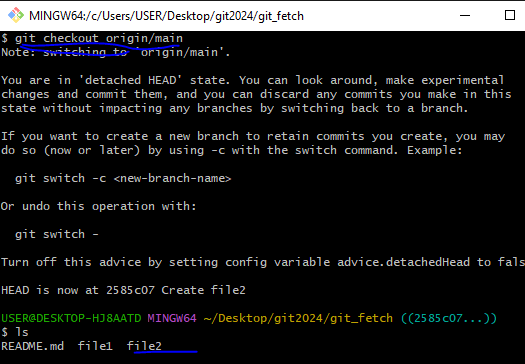
Step 1:

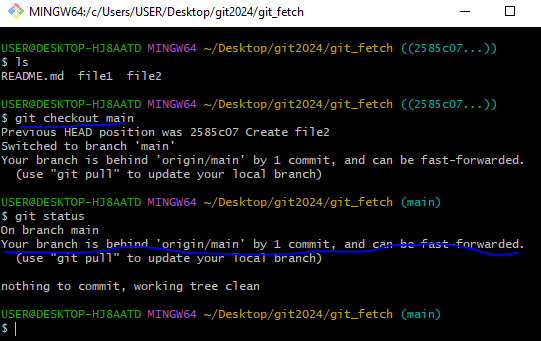


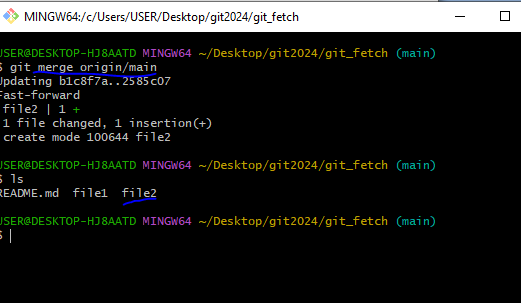
* Now create another file in git hub repo i.e file2

**Step 2:**

**Step 3:**

**Step 4 :**

**Step 5: **

**step 6 :**

* 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

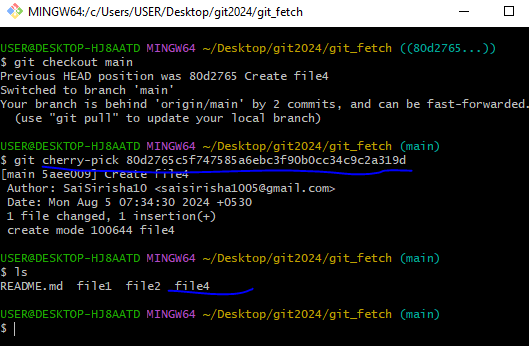
**git checkout origin/main**

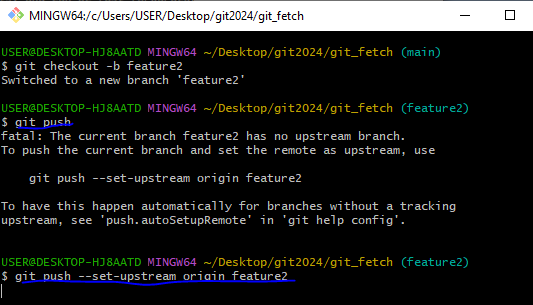
* Next check the log of the files commit in our git hub repo copy the commit id of the recent id and come back to the main branch

**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>**

**step 7:**

**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------