**GIT and GITHUB**

GIT is a free and Open-source version control system. Unlike other centralized version control system such as SVN (Apache Subversion) and CVS (Concurrent versions System), Git is distributed i.e., every developer has the full history of their code repository locally. This makes the initial clone of the repository slower, but subsequent operation such as commit, blame, diff, merge, and log faster.

GITHUB: GitHUb is a collaboration platform that uses Git for versioning. GitHub is popular place to share and contribute to open-source software.

HOW GIT WORKS

* Create a “repository” (project) with a git hosting tool (like Bitbucket)
* Copy (or clone) the repository to your local machine
* Add a file to your local repo and “commit” (save) the changes
* “push” your changes to main branch
* Make a change to your file with git hosting tool and commit the changes
* “pull” the changes to your local machine
* Create a “branch” (version), make a change, commit the change
* Open a “pull request” (propose) changes to the main branch)
* “merge” your branch to the main branch

GIT Command

* Configure user: define author name to be used for all commits in current repo. Devs commonly use --global flag to congif option for current user.

$git config user.name <name>

* Create Folder

*$ mkdir foldername*



* Change directory

*$ cd path*

change folder



Change disk



* List all the item in the folder

*$ ls*



**.git FOLDER**

This folder contains all information that is necessary for the project and all information relating commits, remote repository address, etc. it also contains a log that stores the commit history. This log can help us to roll back to the desired version of the code.

What is repository

A directory or storage space where your projects can live. It ca be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, image files, you name it, inside a repository.

There are two type of repositories:

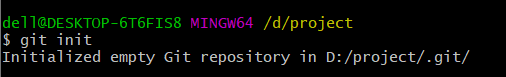
1. Central repository
2. Local repository

Central repository: typically located on remote server. exclusively consist of “.git” repository folder. Meant for team to share and exchange data.

Local repository: typically located on local machine. resides as a .git folder inside your projects root. Only admin of the machine can work with this repo.

* Creating empty git repository

*$ git init*

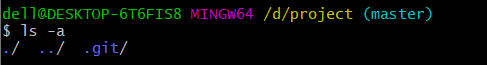


Now we will not be able to see the .git folder as it is hidden so the simple *ls* command will not work

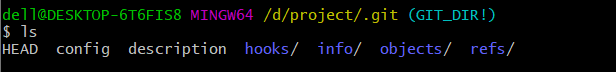


So, we use

*$ ls -a*

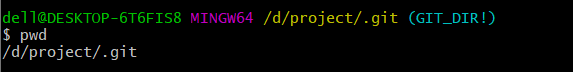
**

Now inside .git folder we get all the file



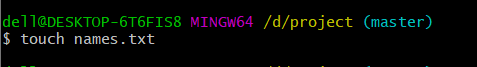
* To get the current file location

$ *pwd*



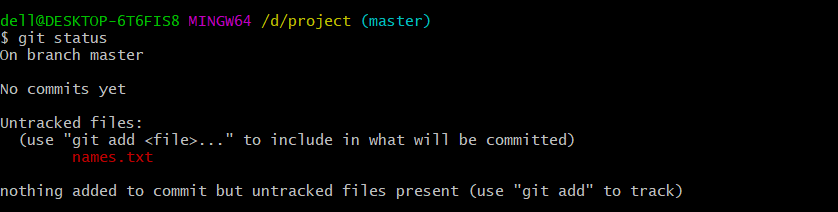
* To create new file

$ touch filename

**

* To get the status of the added/removed file

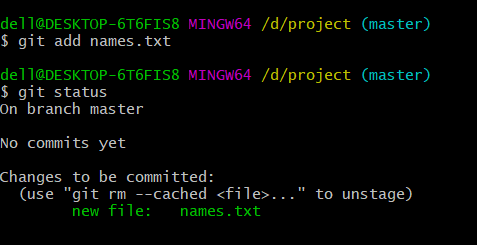
*$ git status*



Right now, no one knows that the file is added

* To add file

*$ git add filename*



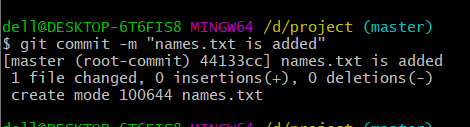
* b to add all the file at once

*$ git add .*

* to commit changes

*$ git commit -m “names.txt file is added”*

-m is used for message

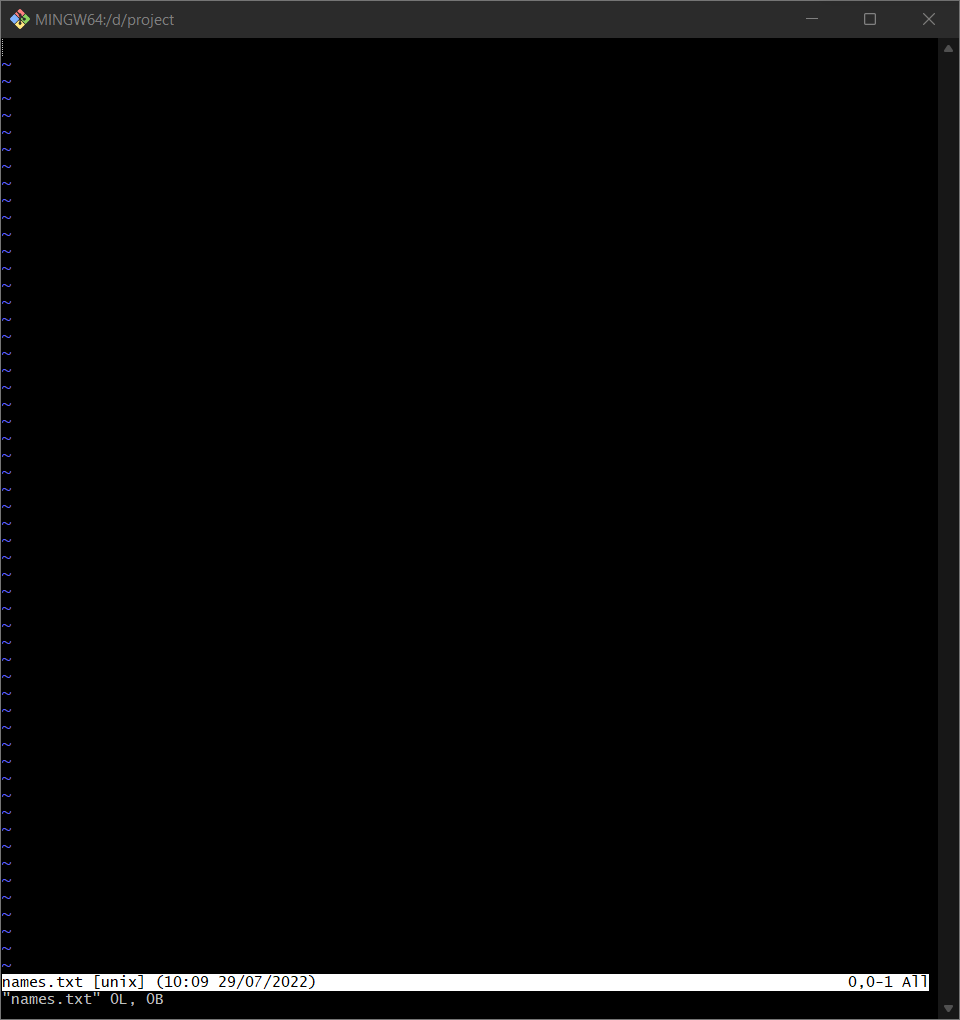


Here names.txt is added and in file there are 0 insertions, 0 deletion.

Now we will use vi editor

* command to invoke vi editor

*$vi filename*



Now we can edit our text file

After editing file

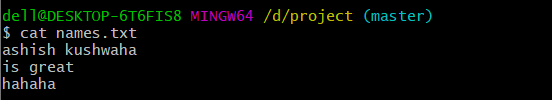
Click esc

The :wq

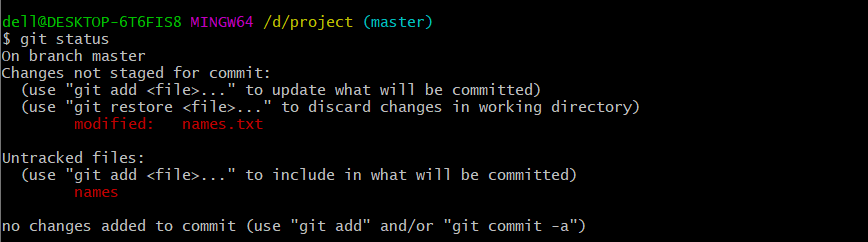
File will be saved and exit from vi editor

* to see the all the content of file

*$ cat filename*

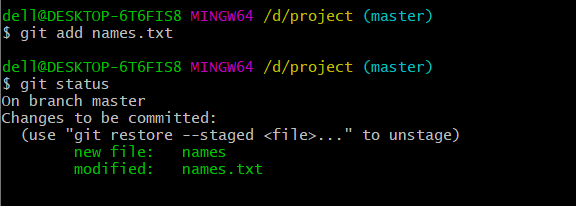


Now we check the changes



This says names.txt is modified

Now we add the file

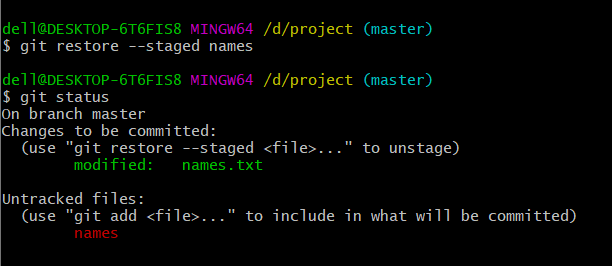


The files are on the stage

From here also if we don’t want to add files, we do that. For example, I have created one extra file names

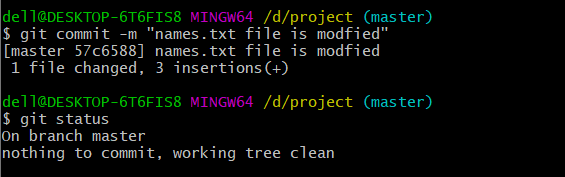
* command to unstage the file

*$ git restore --staged names*



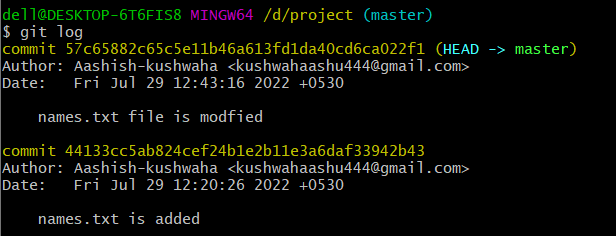
Now we can see ‘names’ is out of stage

Now we commit names.txt



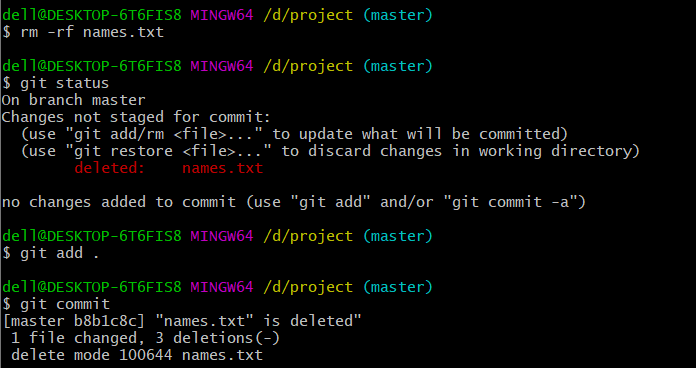
* command to check all the history

*$ git log*



* command to delete the file

$ rm -rf names.txt

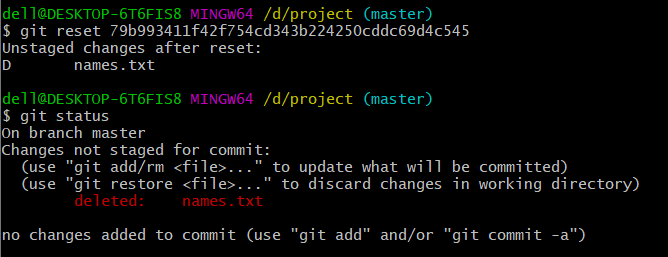


We check using log



Now if we want go back any of the commits (the stage of project which we want to see again) we use the HASHKEY associated with that commit

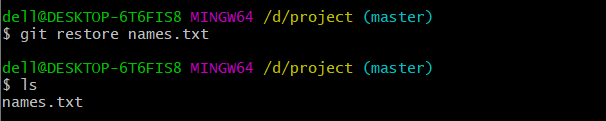
* $ git reset hashkey



All the files are now on the unstage area

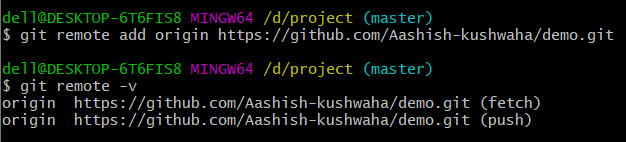
* To restore the file from unstage area

$ git restore filename



* Connecting remote repository to local repository

$ git remote add origin <repository link>



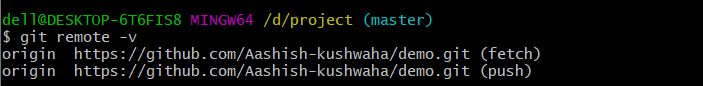
Remote means we are working with URLs (of remote repository)

add means that we just added an URL

origin just mean you name of the URL you want to add

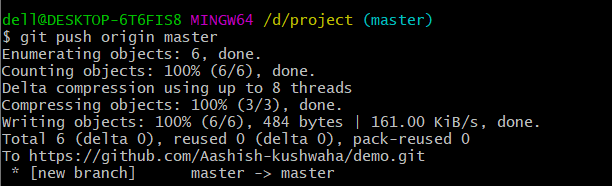
* to see all the URLs attached

$ git remote -v



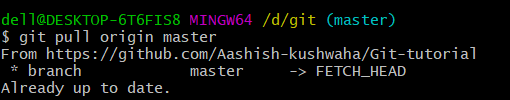
* to push the changes to the BRANCH

$ git push origin <branch name>



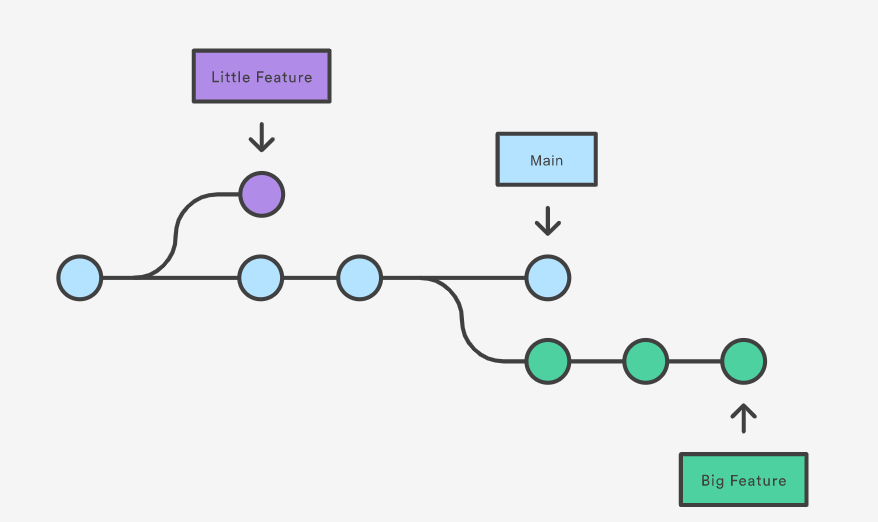
* To pull the changes from remote repo to local

$git pull origin <branch name>



BRANCH

A branch is a parallel version of our repository. By default, our repo has one branch name “main” and it is definitive branch. A branch represent an independent line of development. Git branches are effectively a pointer to a snapshot of our changes.

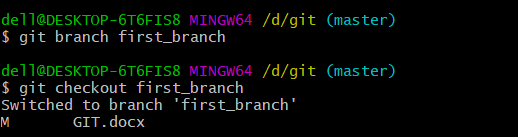


TYPES of BRANCH

1. Local branch
2. Remote tracking branch

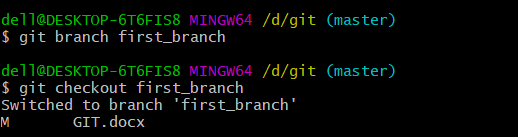
* To create branch

$git branch <branch name>



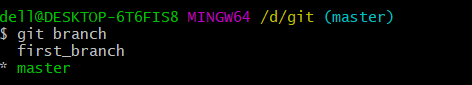
* To switch the branch

$git checkout <branch name>



* To list all the branches in the repository

$ git branch

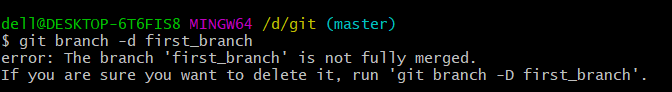


DELETE the branch

There are two commands to delete the branch. We will discuss them one by one

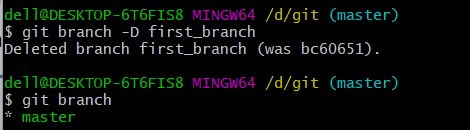
$ git branch -d <branch name>

This command prevents you from deleting the branch if it not unmerged



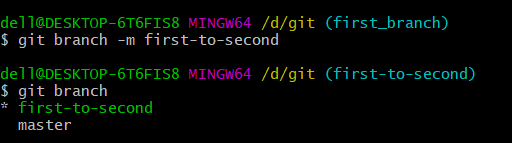
$ git branch -D <branch name>

Force delete the specified branch, even if it has unmerged changes, this is the command if we permanently delete all the commit



To change the name of the branch

$ git branch -m <new name>

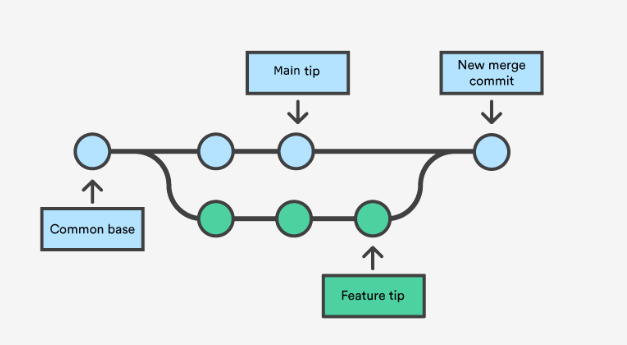


MERGE

Merging is the git’s way of putting a forked history back together again.

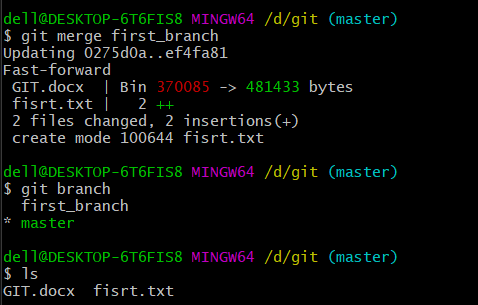
Git merge will combine multiple sequences of commits into one unified history.

Git merge takes two commit pointers, usually the branch tips, and finds the commo base between them once it find the commo base commit it will create a new “merge commit” that combines the changes of each queued merge commit sequence.



# to before merging we should be in destination branch or we can say we should checkout to destination branch

$ git merge <branchname>



After merging the first branch still exist

And changes made in the branch does not effect the master branch

GIT REBASE

Rebasing is the process of moving or combining a sequence of commits to a new base commit. Rebasing is most useful and easily visualized in the context of a feature branching workflow.

This is also a way of combining the work between branches

It can be used to make a liner sequence of commits.

