**Git :**

* It is a version controlling system used to track the files, changes in the files.
* We can use github or bitbukket to store our git repositories.
* Git is open source but guithub, bitbucket, gitlab are not opensource and not free.
* We can go to git website directly to download the git.

**There are 3 stages of GIT**

* 1 > Working Directory when we have added some changes to the code but we have not sent it to the staged area. To move from working directory to the staging area we use git add .
* 2> Staging Area when we have added the changes and not we are ready to do the commit, to move from staging to repository area we use git commit command
* 3>git directory when we have commited all the changes and we are ready to push the code to the repository we use git push command

**Commands**

**git status**

* It return the status of the repository ,
* If it is not a repository it returns a erorr saying that it is not a git repository.

**git init:**

* It initializes the current folder as the git repository.
* Internally it creates a hidden folder with the name .git, where all our changes tracking information is saved
* Git bash works as a linux terminal, so most of the syntax in the git bash as linux types.

**git add:**

* This command is used to move from the working directory to the staging area,
* git add . -> it will add all the files to the staging area
* git add <file name> it will add only the given file to the staging area.
* All these files will be shown in the tracking area
* git add –A stages all the files
* git add . stages new and modified, without deleted
* gid add –u stages modified and deleted, without adding the newly created filedonly tracks the already exting filed before this command is run.

**git commit:**

* git commit –m “<Message>” this command is used to commit all the tracked changes to the repository
* every commit has a hash code which identifies that change done during that commit uniquely.

**git log:**

* sometimes we need to know that who has commited the changes in that case we can use this command to see the changes done by different users/developrs

**git config –global user.name:**

* to set the user name globally means not only for this directory but anywhere on the current computer if we make any changes in the git repository it will be considered as changes done by me.
* If we do not use the –global then it will take the config defined for the current repository only not for all the other repositories in the computer.

**git config –global user.email:**

* to set the email globally for the same above mentioned reason.

**How to set Remote Origin?**

**git remote add origin <URL of the host>**

This command is used to add the origin or the remote server where we want to host our code or where our code will be saved, we can also pull from the existing code.

When we are using the git bash we can use the ssh URL rather than HTTPS

**git push –u origin master:**

when we use this command for the first time it will give us warning asking for the key if we have not added

Enter passphrase for key '/c/Users/DELL/.ssh/id\_rsa':

fatal: Could not read from remote repository.

<https://help.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

x

**git pull origin master:**

to pull the repository from the git

**git diff <File name>:**

display the difference in the previous file and the changes applied in the current working directory.

**git diff --staged <File name>:**

display the difference in the previous commit and currently added or staged files.

Here it will diplay the diference between previous commit and currently staged files by the command git add .

**How to go back to the previous status, or to remove the file from the stagin area?**

**git resat <name of the file>**

it will remove the given file from the staging area to the untracking or working directory

this is just used to remove the file from the staging area

if we want to move back the files from the git add to normal

git reset . 🡪 remove back all the added filed from staging area to the working directory or untracking area.

**git checkout <Name of the file>** :

now it will take the given file back from the changed state to the initial state when we had done the previous commit or the previous state.

If we want to move back the changed in the file to the previous commit we use this command

Git checkout . to reset all the files to previous commit

**Cloning**

1. create a new folder.
2. Open new git-bash in that folder
3. git clone <URL of the project> it clones that project to the current folder with a new head folder
4. git clone <URL of the project> . it clones the project files inside the main folder of the folder which we are in.

**Ignoring Certain Files in the git repository?**

we don’t want to track changes in certain files.

1. Touch .gitignore 🡪 it creates a blank file with the name .gitignore inside which we will writethe names of the files which we don’t want to track
2. notepad .gitignore 🡪 to open the file in notepad to add rules
3. git rm –cached <Name of the file> 🡪 if we were tracking a file but now we are adding it to gitignore but still its history will be saved, so to delete that history we use this command.
4. Whatever file names we are adding inside .gitignore will be ignored
5. Example if we want to ignore certain extension of file we can use \*.<extension >
6. Like \*.pyc, \*.dll, \*.javac all files with that extention will be ignored.
7. We already have certain .gitignore files we can use those files we can get that from google.
8. ‘we can search for the .gitignore files for the specific language and we can det that files and add it , and also go through it and get an idea about it.

**Branching In GIT**

**git branch 🡪**  to know in which branch we are currently in.(the branch with color green is the branch we are currently in) and the branch in white are other branches

**git branch <Name of a branch> 🡪**  it creates a new branch with the given name .

**git checkout <Name of the branch> 🡪**  it takes us to the given branch name or switches the branch and diplays the message switched to branch <branch name>

Now we can do all the adding and commiting of the changes in the file and now we have to push it to the given branch only

**git push –u origin <Name of branch we are in>:**

it will push the changes to the remote repository with the given new branch if it does not exist it will create a new branch and push the changes.

**To merge the branch to the master branch 🡪**

**Step1 -> go into the master branch by the command** git checkout master

**Step2 -> use the command and merge it** git merge <Name of the branch>

Not it will merge both the branches or the given branch or changes to the master branch but the changes are still in the local only now we have to use the push command to push the changes to the remote repository git push –u origin master

Steps ->

Git checkout master

Git merge <name of branch>

git push –u origin master

**How to delete the branch we have created to add a functionality and now we have merged it and now we don’t need that specific branch?**

The need to delete is it takes space,

**To delete from local system**

**git branch –d <Name of the branch> 🡪** it deleted the given branch from the local system.

**To delete from remote system/repo**

**git push origin –del <Name of the branch> 🡪**  it deletes the given branch from the remote repository= and we will not be able to see this branch hence after.