Git Bash Commands

(NOTE => ignore '<' and '>' signs)

- pwd -->> to see the path where you are right now.
- mkdir <folder_name> -->> to create folder.
  mkdir <folder_name> <folder_name> -->> to create more than one folder.
- touch <file_name> -->> to create file like .txt, .html
- ls -->> to see whats in your folder.
  ls <folder_name> -->> to see whats in your folder.
  ls -a -->> to see hidden files or folders also.
  ls -al -->> to see details of all files and folders (Hidden also).
- cat <file_name> -->> to see whats in your file
- clear -->> to clear terminal.
- cd <path> -->> to navigate through folders.
  e.g.,
  - step 1 -->
    cd c:/users/<user_name>/desktop -->> to navigate through many folders at one time you can use '/' to connect between them.
  - step 2 -->
    You can navigate by one folder also.
    Let say you have one folder called project on desktop and you want to navigate it then use cd project .
- cd .. -->> to go back by one folder.
- cd ../.. -->> to go back by two folders.
- cd -->> to go back to drive.
- rm -r <file/folder_name> -->> to delete file or folder.
  rm <file_name> -->> to delete file only.
  rmdir <folder_name> -->> to delete folder only.
  rmdir <folder_name> <folder_name> -->> to delete more than one folder.
- vi <file_name> -->> to edit file on terminal only.
  This will open text editor like interface on terminal to edit on terminal only
  To exit editor ->
  Press 'esc' and type ':x!' / ':wq' and hit enter button

- cd <drive_name>: -->> to change drive

Basic Git Commands

- git config --global user.name "user name"
  git config --global user.email "user's email"
  -->> to set user's details
- git config --list -->> to get user's details
  OR
  git config --global user.name
  git config --global user.email
- git config --global –edit -->> to edit user details.

- git init -->> to initialize git repository.
- git add <file name> -->> to add file to stagging area.
  git add . -->> to add all files to stagging area.
  git add -A -->> to add all files to stagging area.
- git status -->> to check is there any file untracked.
  git status -s -->> gives summarized status of your repo.
  Green color shows staged area & red color shows untracked area
  M => for modified and added in stagging area
  M => for modified and untracked
  A => for added to stagging area
  ?? => for untracked files
- git restore –staged <file name> -->> to unstage file, if you don't want this file to be tracked by git but also never want to delete it.
- git reflog -->> to get hashcodes of older commits
  OR to get commit history.
- git revert <hashcode> -->> revert is like undo. (learn more)
- git commit -m "/* comments */" -->> to commit changes
- git reset –hard <hashcode> -->> this will delete all commits done after <hashcode>.
- rm -fr .git -->> to delete git repository.
- git mv old_filename new_filename -->> used to rename file.
- git ls-files -->> to see all committed files
- git checkout <file name> -->> if your file is modified but not committed and u don't want file to be modified then u can use this command to undo the changes made. Git will get u the last committed file.(This is for only one file) & git checkout -f this will be for your all files

- git log -p -<noOfCommits> -->> used to see last commits.
  noOfCommits => number of last commits you want to see.
- git diff -->> it compares working directory file with file present in stagging area(or with same file which is staged).
  'Working directory file' => file which on you are working but not staged yet. But if u staged working directory file(after modification) then it will never show anything

- git diff --staged -->> it compares staged file with last committed file.
- git rm --cached <file name> -->> this will transfer your file to untracked files from commit (it will never delete it).
- git rm <file name> -->> this will delete your file from repo.
- git restore --staged <file name> -->> this will unstage staged file
- git diff --name-only --cached -->> this will show files present on stagging area.

❖ .gitignore -->> is the folder that includes files or folders that you want to be ignored by git which are not important for your project.
e.g., commands =>
- Secrete.txt -->> git will ignore this file wherever it present in git repository.
- /secrete.txt
  -->> secrete.txt will be ignored only from those folder where .gitignore is present bcoz we added '/' sign at starting of file name
- *.log
  -->> ignores all files having .log extension
- *.txt
  -->> ignores all files having .txt extension
- ignore/
  -->> this is how folders are ignored by marking '/' at end of folder name
- .gitignore -->> now git will ignore gitignore file also.

- git branch <branch_name> -->> used to create new branch.

git branch -a

- git branch -->> used to see list of all branches.
- git checkout <branch name/hashcode> -->>used to switch between branches. Hash code is used when you want to go on older commit.
- git checkout -b <branch name> -->> used to create new branch and navigate directly on it.
- git merge <branch name> -->> used to merge two branches.
- git branch -d <localBranchName> -->> used to delete local branch
- git push origin --delete <remoteBranchName> -->> used to delete remote branch (i.e., branch on github)

- git pull -->> to get the changes done on github (in remote repository), to local repository. (This will directly reset your code in local repository).
  Another Way =>
  git fetch -->> this will fetch all your changes made in remote repository but will never apply to local repository. To apply that changes you must use git merge command.
- git clone <URL> -->> to get repository on local device after forking it
- To go on older commit and delete rest of all commits then use git reset <hashcode_Of_Older_Commit> this will bring your main branch to older commit. Then to push these changes you must use forced push and command for that is git push origin main -f . (Note: this will reset your all commits till the hash you used to reset.)