GIT CHEAT SHEET

#**config**
git config --global user.name "<name>"
git config --global user.email "<email>"
git config --global --edit

#**commands**
git init : to initialize a directory as a repo
git status : all changes etc done in the repo

git diff -u <filename> : to see changes(unstaged) done in the file(-u is optional, for viewing properly/more detailed)
git diff -u file1 file2 : show changes between file1 and file2
git diff -u oldfile newfile > change.diff : store the diff result in change.diff file(patch file)
git patch oldfile < change.diff : to apply the patch/changes in changed.diff to old file
git diff --staged : Compare previous commit to recent staging area
git diff : compare staging area with working directory (modified files)
git diff master..branch1 : multiline difference between branches
git diff --color-words branch1..branch2 : singleline difference between branches

git add <filename or .(for all files)> : starts tracking the file /staging area[working directory -->(git add) -->staging area →(git commit) --> repository]
git add -p <filename or .(for all files)> : shows the changes before adding to stating area

git stash : (After staging, If you don't want to commit but also don't want to lose the changes), remove all from the staging area but save it so that it can be brought back to the staging area. Repo goes to its state before those changes.
git stash pop : brings back the saved stash to the staging area.
git stash clean : to delete the saved stash.

git commit -m "<message of commit>" : commits the file
git commit -a -m "<message of commit>" : stages and commits (both) all tracked files directly
git commit --amend : to commit to previous commit without creating a new commit(but changes the commit id)
git rebase -i <hash/branch name> : gives a list of all commits above given hash. Can squash multiple commits into one. To squash, use **s or squash** instead of **pick**, merges all **s** commits with **pick** commit. After it, **Esc** -> **:x** -> Enter -> write message for the commit -> **Esc** -> **:x**

git revert <HEAD or commit id> : creates a new commit that contains inverse of all the changes made in the given(bad) commit id

git reset HEAD <filename> : resetting file state to the commit where HEAD is pointing (used to remove from staging area)
git reset --hard HEAD^ : to undo/delete latest commit and also removes the code changes
git reset --soft <hash code of the previous commit> : to delete the latest commit but not the modification/changes

git restore <filename> : to restore file to initial state
git restore -- staged <filename> : to unstage and restore file to initial state

git log : see commits done previously
git log -p <branch name(optional)> -5 : to see last 5 commits along with differences/patches applied in each commit(-p) of that branch
git log --graph --oneline --all : to better visualize like a graph and show commits in one line of the commits and branches
git log --stat : to see commits along with a short summary of the changes
git log --pretty=oneline : shows one commit per line
git log --pretty=<short or full> : shows commits with short/full details
git log --since=2.days : shows last 2 days commit (similarly we can use weeks,months etc)
git log --pretty=format:"%h -- %an" : shows all commit's abbreviated hash -- author name only (ae- for author email)
check documentation for other formats

git show <hash code of the commit> : to see changes in that particular commit(log message and diff output)

git branch <branch name> : to create a new branch
git branch -a : list all remote branches
git branch -d <branch name> : deletes the branch (gives error if branch is not merged, use -D for force delete)
git branch -v : gives all branch name and their last commit hash and message
git branch --merged : shows already merged branches -> can be used to see already merged branches and delete them
git branch --no-merged : shows branches not merged yet
git branch -m <original branch name> <new branch name> : renaming a branch

git checkout <commit hash code/branch name> : goes back to the commit/branch
git checkout <filename> : reverts changes to modified/unstaged files before they are staged (Use -p flag to be showed and asked before each change) (If file is already staged, first unstage using git reset..)
git checkout -f : goes back to previous commit
git checkout -b <branch name> : creates a new branch and also check out at that branch
git checkout -b <b1> <b> : create a new branch b1 from the branch b

git merge <branch name> : to merge the given branch with the present branch
git merge --abort : to abort the current merge in case of a merge conflict

rm -rf .git : makes the folder no more a git repo/removes the .git file
git rm <filename> : deletes the file (don't forget to commit the file again with a message why deleted)
git rm --cached <filename> : to untrack a file [used when we put a tracked file in .gitignore]
git mv <original filename> <rename filename> : renames the file

#**HEAD**
-> points to currently checked out snapshot of the project

#**git alias**
Eg : 1) to use, git st instead of git status -> first use -> git config --global alias.st status
2) to unstage a file, git config --global alias.unstage 'restore --staged --' ->now we can simply use, git unstage <filename>

.gitignore
->To ignore all files with a particular extension, use *.<extension> in the .gitignore file
Eg : to ignore all log files ;
*.log
->To ignore all folders outside/inside : <folder name>/
->To ignore only outer folders : /<folder name>/
->git ignores blank folder by default

GITHUB

***create a new repository on the command line** :-
echo "# Learning-git" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/aniketkumarpaul/Learning-git.git
git push -u origin main

***push an existing repository from the command line** :-
git remote add origin <origin link>
git remote add upstream <upstream link>
git branch -M main
git push -u origin main : -u <repo name> <branch to push>
git push origin branch1:newbranch -> pushes the branch branch1 in remote repo as name newbranch

git remote -v : shows URLs of the remote repositories(for fetch and push)
git remote show origin : same as git remote -v, but gives more details like branches in local and remote repo
git remote update : work with new branches in remote repository locally without merging with the local repo
(simply checkout to that branch after this to work in that branch)
git branch -r : shows all branches in remote repository

git clone <repo url> <folder name of your wish(optional)>: clones the repo in github to our local repo
git pull : to sync the remote and local repo / update local repo from remote repo
git push -d origin <branchname> : to delete a branch in the remote repo
git push -u origin <branchname> -f : force push, used when remote repo and local repo commits are not synched

***To be in sync with the main branch of the upstream always** :-
-> [can click on **Fetch upstream** in github]
Or,
In git,
>git checkout main
>git fetch --all --prune
>git reset --hard upstream/main
>git push origin main

Or,
>git pull upstream main
>git push origin main

***To make changes to a remote repository/branch**
-> Pull the remote branch, merge it with the local branch, then push it back to its origin.

***git fetch VS git pull**
-> git fetch fetches remote updates but doesn't merge; git pull fetches remote updates and merges.

>git fetch : fetches all remote repo update to the remote branch but doesn’t merge to the local branch
>git log origin/master : shows all commits in the master branch of the remote repository
>git merge origin/master : merges the master branch of remote repository to our local master branch

->Above 3 steps could be directly done using git pull

***When there's a divergence in branches, i.e a commit is made in the master branch after branching, then while merging, a three-way merge takes place instead of fast-forward merge,**
To keep our history linear, we will rebase the branch against the master branch's latest commit, thus simply fast forward merging later instead of three way.
-> Rebasing instead of merging rewrites history and maintains linearity, making for cleaner code.
-> You shouldn't rebase changes that have been pushed to remote repositories

>git checkout <branchname>
>git rebase master

>git rebase <branchname> : Move the current branch on top of the <branchname>

-> Keep the latest version of the project in the master branch, and the stable version on a separate branch.

***If you are working in an issue in a repository, then after doing necessary changes to solve the issue, Include "Closes #<issue id>" in the commit message. This will automatically close the issue when we merge it to the original repository.**