**GIT**- It is a Source code management tool ,used to track the changes made in your code

***Basic Configuration-***

**Username-** $ git config --global user.name "Monali Zade"

**Usermail-** $ git config --global user.email [monalib@maveric-systems.com](mailto:monalib@maveric-systems.com)

Mkdir simple-calc-batch7

Cd simple-calc-batch7

**Editor-** git config --global core.editor gedit

***Version Control system-***

you can keep track of changes that you are making in project

***Difference bet Central n Distributed version Control System***

**Central Distributed**

|  |  |
| --- | --- |
| 1.It requires internet connectivity | 1. it does not requires internet connectivity |
| 2.it have single server and other clients can access it |  |
| 3. if it fails down nobody can pull/push changes so risk of loosing data is high | 3. if server fails then u have the data on your local system, you can restore it back to central server |

***Advantage of GIT over other DCS:***

1. Speed is fast & Free tool(others are paid).

2. GIT ensures that nothing goes untracked

3. In GIT, everything is a snapshot ,not differences

4. Every operation is local, means you need not to connect internet every time

5. Git has integrity means you can not change any files content withing GIT knowing it

**Difference between GIT & GITHUB**

**GITHUB-** is a server which is used for hosting git repository Eg- GIThub, Gitlab, Bigbucket

**GIT-** it is a source code management tool to manage your source code Eg- Mercirial, GIT,SVN

***3 stages of GIT:***

* **Working area-** will reflect the changes we are making in directory // it contains uncommited changes
* **Staging Area-** contains which you want to commit
* **Git repository**

***3 stages where git config maintained-***

1. **Sys config>>/etc/gitconfig—**sys level config will maintain—1 sys can have multiple user accounts configured—any configuration which we will make at sys level are applicable for all users on that system
2. **Global config>>**it is applicable at individual user accounts>>username,email
3. **Local config**>> it is with respect to particular git project

**Priority>>local>>global>>system**

***How to make simple folder into git repository:***

1. **Git clone-** if there is already ongoing project repository is present on server, we need to clone it
2. **Git init-** To create fresh repository

After this , .git folder will get generated, now your repository is GIT Repository.

***How to get help about particular command***

**Git - - l command name**

**Git command name - - l**

**Man git -command name**

***How to see sys configuration***

**Views config - - command**

***For additional files***

**-- show – origin**

***Command to get git repository***

**Git clone**

**Git init**

***Difference between clone and init?***

**Git init-** firstly initialize the repository>>normal project folder you want to convert into git repository

**Git clone-** there is already git repository present n used by your teammates , u want to get that in your machine

**\*\*\*\* While cloning any existing repository to our local sys. We can change its name. Ex. “Simple calculator” is the repository present in server and I want to clone it on my desktop with name”My Calsi” we can do it –**

$ git clone <https://github.com/ShreeharshaJois/NewSimpleCalculator.git> ~/desktop/MyCalsi

***Significance of .gitingnore file?***

Consist of some of the content which we want to ignore, liabrary files,credentials etc.

**\*To check which files are in which state:**

**Git status >>**Used to see what files has been modified/added/deleted

**Git status -s>>** to get short status

**Git help add>> Man git add>>**to get help

**\*Ignoring files:**

**vi.gitignore**>>>> creating gitignore file

**Cat.gitignore**>>> see the content of .gitignore file

**\*Viewing Changed File contents:**

**Git diff filename(or commit id)>>**to see actual changes in the file>>it compares git repo file & working directory file

**Git diff –chached>>** compare staging area file with working directory

**Git diff –staged>>**difference between staging area and git repo

**\*Add commands(Adding the content to the STAGING AREA):**

**Git add filename>>**To move content from working dire to staging area

**Git add – all>>** to add all files which are present In Working Dire

**Git add \* >>**

**Git add .** >> to add files are in particular directory

**Git reset/restore>>**To revert back from staging to working dire.

**\*Commiting Changes:**

**Git commit**

**Git commit -a**>> add+commit all modified files not newly added files

**Git commit -a -m”msg”**

**Git commit -m “msg”>>**To move content from staging area to git repo

**\*Removing Files:**

***Normal remove* command Rm filename** >> removed file will be In workig directory.Deleted:filename

**Git rm filename>>** file ot only ge deleted from workig diredctory but also it will get marked deleted in staging area. Deleted:filename

**Git rm -cached**

**Git checkout>>**To clear changes in working directory

**\*Viewing Commit Logs:**

**GIT Log**- to know commit history>>commit id, author name, date & time of commit, mgs

**Git log - - oneline// Git log - - oneline -2 -p**>>> to get comments in one line

**HEAD>>** told which branch you are currently in

**Git log -p -2**>>>>last 2 commits

p>>patch>>to get changes made between 2 commits>>same as git diff

**git log – since=2.weeks/month/year**

**git log –help**

**git log –branch name>>** to see logs of other branch

***adding missing changes to last commit:***

**git commit -m “initial commit”>>**

**git add forgotten file>>**

**git commit amend>>**

**git commit –amend -a -m>>** amend is used to modify the last or topmost commits

amend should not be done with commits that has been already pushed to the server because whenever you commit it,it will change the commit id.

**GIT RESET: - to revert back to previous commited state**

1. **Git reset –soft -HEAD~1 >>** revert back the changes but it will keep the changes in the staging area
2. **Git reset –hard -HEAD~1>>** permanently revert back the changes

***\*Tagging:-***

**Why tag is IMP:**whenever we make a release ,we want to keep a note on ,from which commit we made the production release, for this purpose we create a TAG.

**Types:**

1. **Annotated tags-** contains much more info(tag msg, commit id it is pointing to, author of tag,)>>-a
2. **Lightweight tags->>** tagger info & msg is missing

**Git tag>>** to get list all tags

**Git tag - - help**

**Git tag –“v1.8.5”>>** it will search tag matching with particular pattern

**Annotated tags- git tag -a v1.4 -m “my version 1.4”>>** to create Annotated tag

**git show v1.4>>** to see tag data along with the commit that was tagged

**Lightweight tags- git tag v1.4-lw**

**git show v1.4-lw**

***Diff bet Annotated & Lightweight Tags*>>>>** In lightweight tag, Tagger info(Author,Date,time of tag,tagging msg) is missing

**git tag -a<tag-name><commit id>** >> tagging later

**git push<remote-name><tag-name>** >> sharing tag

**git push origin –tags**(push multiple tags)

**Git tag -d v1.0.0>>** to delete Tag

**Git checkout -b version v2.0.0** >> Create new branch and switch to new branch

**Git commit>>** to move contents from working dire to git repository

\**Aliases:* Means creating new commands in shorter format

**Git config - - global aliase.co checkout**

**Git config - - global aliase.br branch**

**Git config - - global aliase.ci commit**

**Git config - - global aliase.st status**

This means co=>checkout

Git config – global alias.unstage “reset HEAD --”

Git unstage fileA

***\*GIT Branching***

**Git branch**>> gives list of all local branches

**Git branch -r**>> gives list of all branches on server.

**Git branch –merged**>>gives list of all merged branches

**Git branch branch-name**>> it will create a new branch

**Git log –oneline—decorate**>> where the pointers are pointing

**Switching branches:**

**Git checkout -b name of branch to switch**>> it creates branch first and switch to newly created branch

**Git checkout branch name**>> to switch to another branch

**Merging of branches:**

**Git merge branch-name>>** will merge two branches

1. types of merge:

**1.Fast Forwarding** -

the latest commit of the branch to which your are merging the changes can be reached from the latest commit of the target branch. It simply updates the current branch pointer to point to the latest commit in the target branch, it is c/s “Fast Forwarding”

**2. Recursive Merging -**

**Divergent History-** it means there are some changes in one branch but not present is another branch. For ex. If you created “Topic ” branch from “Master ” branch. There are some changes in “Master” branch , so me other changes in “Topic ” branch. “Master” branch changes are not in “Topic ” branch is c/s “Divergent History”.

To Merge such branches, git does three Way Merge. It takes the tip of commits of two branches and common ancestor and it creates merge commit , it is c/s “three Way Merge”.

**Merge Conflict Error**- causes When same section of file get modified by two developers..

**\*\*\*when conflict arises\*\*\***

**We have to open code file in any editor and need to resolve conflict>>>save the file>>go to gui>>>excecute *git add* command>>>it will move file to working dire to staging area**

***How Merge Commit is different from other commit???***

A merge commit has two parents , it points to two parent commits.

Git fetch>> updates the remote tracking branch to point to what is there in the server.it has nothing to do with local branch.

Git pull>>combination of git fetch+git merge

***\*Deleting Branch:***

**Git branch -d branch name>>** to delete the branch

***\*Rename Branch:***

**Git branch –move branch branchA**>>> Rename branch to branchA

***\*Tracking Branches:***

**Git branch -vv>>** gives info about remote branches . it tells how many commits your local branch is ahead/behind of a remote branch

**Git branch r**>> gives list of remote branches

**Git branch**>> gives list of local branches

**Git branch –all**>> gives list of all branches

**Git stash save”msg”>>**alternate option for commit>>temporarily saving your changes

**Git stash list**

**Git stash pop**>> removes the topmost stash

**Git stash apply stash{0}>>stash id**>>>it is still in the stash whereas changes will be applied

**Git checkout –f target branch name**>> to switch to target branch(-f forcefully checking out to another branch)

**\*\*\* when you commit below objects will get created\*\*\***

**Commit>> creating a snapshot**

**>>Commit object**(contain multiple info. like Commit id, commit message)

**>> tree object**(represents a directory. It contains blob info. Which blob is referring to which file)

**>>blob object**(checksum for the file which is to maintain to identify and integrity of the file. If any file modification happens the blob value will get change)

git init - command to initialise a new git repo inside a non git folder  
git clone - create or download a already existing git repo by using url  
git add - move the changes from working directory to staging already  
git commit - commit the changes in staging area to git repo  
git status - check the status of working directory for any modified, deleted or added files  
git log - to see the commit history  
git diff - to see the contents of the changed files  
git branch <branch-name> - to create a new branch  
git branch - to view the list of branches in local repo  
git branch -r - to see the list of remote branches  
git branch -d <branch-name> - to delete a already merged branch  
git branch -D <branch-name> - to forcefully delete a unmerged branch  
git checkout -b <branch-name> - to create a new branch and checkout to the newly created branch  
git checkout --track -b <branch-name> origin/<branch-name> - to create a new branch tracking the remote origin branch and checkout into newly created branch  
git reset HEAD \* - unstage all the staged changes  
git restore --staged \* - unstage all the staged changes  
git checkout -- <file> - to revert back any changes to the file  
git push - push your changes from local to remote repo  
git pull - update you local current branch with the latest changes in the reemote tracking branch  
git push origin --delete branch-name - to delete a remote branch  
git merge <source-branch> - to merge changes from source branch to current branch with merge commit  
git rebase <source-branch> - to rebase current branch changes on top of source branch commits(creates a new commit id for the feature branch commits)  
git tag -a tagname -m <msg> - to create a annotated tag pointing to current branch head  
git remote add <remote name> <remote url> - to add a new remote  
git remote rm <remote name> - to remove remote  
git remote show <remote name> - to view more details of the remote  
.gitignore - file containing the list of all files and folders that needs to be ignored by git