Session - 10

24/03/25

GIT

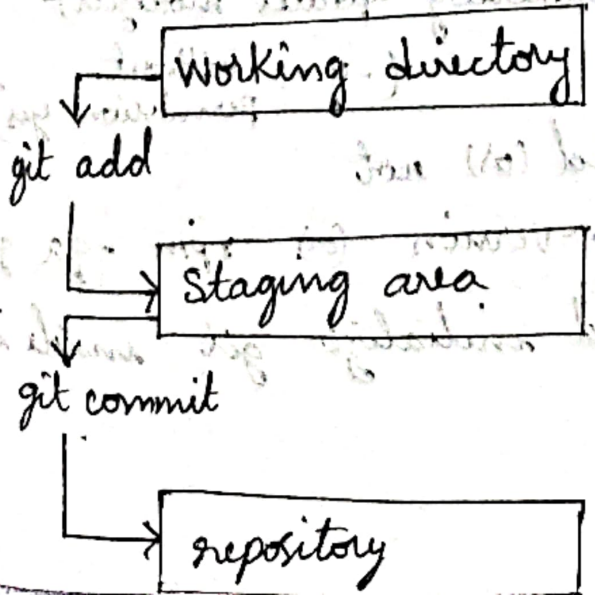
Global information tracker which is local.

→ It is a version control system (VCS) or source code mangement (SCM)
→ used to track the files (who, when, why, changes are made)
→ It will maintain multiple versions of same file.
→ It is a platform - independent
→ It is free and open-source
→ They can handle larger projects efficiently.
→ 3rd version of VCS
→ written in 'c' and came out in 2005.

Before 2005, we initially used SCCM in 1972 which is used to track single file and then RCS here multiple files are tracking but not the folders. then CVS here we can track multiple files and folder but single user and the SVN tracking multiple files, folders and users.

Git is providing an option called roll back we can able to go whatever version we wants to.

Git Stages :- 3 stages



→ git is aware of having files in your project.
→ git will not track files
→ in order to track the files, have to add git to the files.
→ All the tracking files moved to stagging area.
→ you can add / delete / modify you code here.

→ if you done any changes to code / files in staging area. then we have to commit git.

→ To save the history, done by the changes we are committing and file is moved to Repository stage.

For example, you have create a file inside a folder. then the file will be in working directory stage.

Inorder to track the file, we have to add git (git add) to the file. then the file will be in stagging area. In this stage we can do some changes.

Inorder to know the change history to the file, we have commit (git commit). then the file will be in repository stage.

Again if you do any changes for the second time, then the file will be moved to staging area. after the changes are done have to commit. then it will be repository stage.

go to the root user

→ sudo -i

To install git (yum = yellowdog updater modifia)

→ yum install git -y (-y → without permission yes)

To check git is installed (or) not

→ git -v (or) git --version (or) rpm -qa git

→ create a folder and initialize git inside it

→ mkdir folder

→ git init

we are giving git init because inorder to track that particular folder. we can see the .git folder by ll -a. through this .git folder git command will works.

Create a file → touch file
the file will be in working directory, to know this
→ git status (which will be in untracked files in working directory)

To track the files
→ git add filename

To know whether it is tracking (or) Not
→ git status

Now file is in staging area, if you want you can do some changes to the file. after changes the file will be in modified state. the have to commit

→ git commit -m "Description" filename

here the file will repository stage and you cannot see the states in repository stage. check by git status.

states will be visible when you change the file. when the changes are made the file will be in staging area and have to commit again.

To get the history (recent commits are at top)
→ git log

we can see the commit Id, Author, Date and description.

we can change the time ~~in ssh~~, In .git, config file is there, here name & email are there

To change author
→ git config user.name "Name"

To change e-mail
→ git config user.email "mail Id"

To know the what was the change is done copy the commit Id and
→ git show <commit Id>

To track multiple files at a time
→ git add *
To commit multiple files at a time (dot dot and means all files in staging area)
→ git commit -m " "

To know which files are effecting
→ git show <commit Id> --name-only

To track all files including hidden files
→ git add .

25|03|25

Session - 11

To get only commit Id and message
→ git log --oneline (you will get half commit id)
To get full commit Id to above command
→ git log --pretty=oneline
To get the latest 2 commit from number of commits
→ git log -n 2 (or) git log -2
To get the latest commit
→ git log -1
To get the commits performed to a particular file
→ git log --follow --all filename

if you are having 100 files and you want to track 98 of them. Remaining 2 files should ignored.
→ for this, create a file called .gitignore
→ vim .gitignore
and place the 2 files which you want to ignore & save
after giving git status you cannot find those 2 files in untracked files.
then
→ git add *
again if you want to track those 2 files, then remove the filename in .gitignore folder &
git add filename.

To untrack a file
→ git rm --cached filename
if you want to change the commit message (for latest commit)
→ git commit --amend -m "message"
Likewise to change author
→ git commit --amend --author "name" <mail id>"
it will opens a file. save & exit from it.
The commit Id will also changes by default.

To attach a file which is in staging area to latest commit. then all the files will have single commit Id. (Interview question)
→ git commit --amend --no-edit

To reset the latest commit and the action performed (kind of getting back to previous version)
→ git reset --hard HEAD~1 (commit & changes, file will be deleted)
(head~1 means latest, head~2 latest 2 commits)

To bring back the commit that we deleted you have to remember the commit id (or) message

→ git reflog

copy the commit from here and

then

→ git cherry-pick <commit id you deleted>

it will bring back the file, commit & file

To delete the commit, not the changes (or) action

This prevents confusion of number of commits

→ git reset --soft HEAD~1

we cannot delete the first commit by reset

26/03/25

Session - 12

If you are having 3 commits of 3 different files then you want to delete second (middle commit) & action

To ~~delete~~ a ~~specific~~ ~~commit~~ & ~~action~~

→ git revert <commit Id> (opens a file save & exit)

This revert will not delete the commit, it will deletes the file and it will adds an extra commit in git log saying this reverts commits

To bring that reverted action we perform, again

→ git reset --hard HEAD~1

To delete only the commits, not the action (or) files

→ git update-ref -d ~~Head~~ HEAD

then those files will go to the staging area

To know the branches list
initially when an commit is performed, then master is the default branch
→ git branch

To create a new branch from master branch
→ git branch <branch name>

To get into that new branch
→ git checkout <branch name>
Here will get the all commits and actions from master here we can change the files, actions the commit. This commit will belongs to this branch only.
Inorder to get the actions and commits to master branch from new branch we have to merge
Then you should be in master branch and
→ git merge <new branch name>
you can see the actions and commits in master branch from new branch

To create and checkout at a time
→ git checkout -b <branch name>

To rename a branch
→ git branch -m <old branch name> <new branch name>

To delete a branch
→ git branch -d <branch name>
we cannot delete the branch that we are in, we have to switch to other branch.
for example, you have created a branch and in that you have performed some commits.
Now you cannot ~~this~~ delete this branch cause you didn't merged to any other branch.

even though you want to delete then

→ git branch -D <branch name>

To get deleted branch Id.

→ git branch -d <brand name>

To get a specif commit from the new branch to master branch

copy the specific commit id, go to master

→ git cherry-pick <commit id from new branch>
applies for different files

Merge conflicts (Interview question)

Master

* had a file and having data "welcome" & commit.

This file is copied to new branch by creating it.

New branch

In copied file, we are changing the data like "1234" in the file and commit it

* again you tried to add data in the file & and performed commit "welcome guys"

* when you tried to merg the file from New branch to master then conflict will be raised.

This conflict will be resolved manually by the developers by editing the data in the file.

Practically,

→ git branch
  * master
→ vim aws
  data → welcome

→ git add aws
→ git commit -m "data added" aws

create a brand & checkout

→ git checkout -b New branch
→ cat aws (you will see the same data)
→ vim aws. (replace the welcome data by numbers)
→ git commit -m "data overwrite" aws

Now for these 2 branches having same file with different data in it

Now go to master branch add some data & commit

→ git checkout master
→ vim aws (welcome guys)
→ git commit -m "extra commit" aws

Now merge to new branch

→ git merge newbranch

conflict will raised

→ cat aws (it will shows you conflicted data)

manually remove the unnecessary data by vim aws

again track the file

→ git add aws (cause file will be untracked cause data modified)

→ git commit -m "aws files conflict resolved" (no need to mention file name)
→ git log

Now the comit is done that you wanted from new branch

Session - 13

27/03/25

Git stash :- applies those are in staging area

Theory, If you are having 2 branches which are Branch 1 & 2. Branch 1 has app.java file and code inside it. Branch 2 has app.py along with app.java. Now you are creating a new branch from branch 1 as branch 3. And you have edited some/added data to app.java. Now, if you are not confident about your changes whether it is right (or) wrong. then you want to check the code with branch 1 (or) 2 : here you cannot go to those branches cause you haven't commited. to go to branch 1(or) 2 either you have to Commit (or) use this stash.

This stash will deletes the changes temporarly then you can go the reference branches and come back to branch and undo the delete.

create a branch from master by having another file in it and create branch 3 and perform some initial commit. after add some data and checkout to anyother branch. git will not allows you to do that. then

→ git stash

see the new data added which is not present. Now you can checkout to any branch and come back to this brand and undo the delete action by

→ git stash apply

To know how many stash's you have done
→ git stash list

To delete the latest stash log
→ git stash pop

To delete all the stash list
→ git stash clear

git stash drop →

Stash only applies to data not for files