GIT Mithun technologies:

git --version = to check the git version.

download github or any server.

→ cd c:/Desktop

→ mkdir name

→ cd name

1) git init = it is going to create git local repository, which without executing this command, we cannot access any other git commands [it is initializing and .git file created]

Note: After initializing git, three logically areas created, we are not able to see physically.

1) working area (untracked files (red color)) 2) staging area (green color tracked files) 3) Local REPO

→ new files, created files, modified or existing files should be available in working area.

2) → git status = To view your file in which stage, i

Note: git add command is using for working area to staging area [moving files to staging area]

3) → git add . = all file moved to staging area.

→ git add *.filename = moving working area staging file particular file to staging area

→ git add file1 file2 = Multiple files moving to staging area

Note: After completing the developmend file then we go to moving staging area.

* Note:- staging area to local repository (moving copying files using git commit.

4) → git commit -m "first msg" = all staging area files are moving to local repository

add name repository
with name, mapping local repo w
So then, we copy or move to
=> git remote -v => to check wh
correctly or not

7) git Push alias name master or any
or pushing local repo files to remo

Note: after git push entered, ask
password or personal access token
setting → Personal access token, we
Microsoft credentials manager → win

Git :- Git is a DVCS, used to
and maintain the versioning.
it is going to
commit maintains the version

Github :- Github is a hosting
used to manage the source
using Github we can share

5) ⇒ git commit -m "first commit" filename =

moving particular file staging area to local repository

Note
First time you committing in github or linux server, you configure email & username first [one time activity]

⇒ git config --global user.name "any"
git config --global user.email "github mail id"

⇒ git config --global --list = to check configure user details,

6) ⇒ git remote add name (alias) repository url (URL) = replacing url with name, mapping local repo with remote repo, so then, we copy or move to remote repo.

⇒ git remote -v = to check which url mapped, mapped correctly or not

7) git Push alias name master or any branch = copying or moving or pushing local repo files to remote repo

Note: after git push entered, asking github username, password or personal access token [In github setting → developer setting → personal access token. we can generate a token].

Microsoft credentials manager → windows (we can delete password of github)

Git :- Git is a DVCS, used to develop the codes and maintain the versioning. Versioning means each commit it assigns to maintains the version numbers

Github :- Github is a hosting service, which can be used to manage the source code, it will provides the GUI. using Github we can share's the source code & the other's also

Note: Git - github, gitlab, bitbucket, these same works only

SVN client = SVN

Note: Difference between Distributed version control system and version control systems.

→ In DVCS, each developer they have own local repo, first they develop the code, and commit to local repo and Push to remote repo.

→ In SVCS, there is no local repo concept, directly developer work with centralized server or remot server repo directly Pushing.

→ DVCS = Git, VCS = SVN, CVS, TFS

Git flow 1) git init 2) git status 3) git add 4) git commit 5) git remote add 6) git Push

Note: git status, nothing to commit, working tree clean means no files is there to commit, etc.

8)

→ git commit -a -m "Updated any file", (or)
git commit -a -m "any" filename } = previous commited files if we updated again, then we directly use this command [no need to git add]

Note: what are the files in local repo, all files should be in remote repo using git Push, no option is there for particular files. Pushing

9) git log = To display all commit id's in local repo showing last commited, & current branch., press enter to show all commit ids, press Q for quit

=> git log -2 = To showing last 2 commit ids.

=> git log --oneline = showing commit id, commit msg only.

In git log showing detailed information

10) git show commitidxxx = To showing that commit what files are commited. detailed information.

11) git show --pretty="" --name-only commitid = showing only of that particular commitid, display only filenames

12) git clean -n = showing what files deleting (dry run)

13) git clean -f = deleting the files all. [after you creating the files, then when you don't need this files, use with this comm.]

14) git clean -f filename = deleting the particular file only.

15) git reset = all staging area files to working area

16) git reset filename = only particular files staging area to working area.

17) git revert commitidxxx = If already file existed and if any updated content so, that only updated codes deleted if added new files, new files only deleted. in this particular commitid, what are the new files are added or commited those files are deleted from your local repo. only

Note: If changes in github or clone directly and changed files, so, we use git pull origin master.

The above process only works on only Recent commitId only

branches:

18) git branch = showing all available branches in local repo

→ git branch -r = showing remote repo branches

19) git branch name = creating a branch

20) git branch -d branchname = deleting a branch

21) git branch -m anyname = renaming the current branch

(or)

git branch -m oldbranchname renamebranch.

22) git checkout branchname = switching to particular branch.

Note: first commit your files in progress branch. then switch to any branch

[Basic use case

VS Certified Cloud Practitioner
lidation Number: 9P2JPEK1WNB11TGD

les and Responsibilities:

- **Implemented Ansible for the maintenance of over 3000+ IoT devices, streamlining configuration management and ensuring device integrity.**
- Developed and deployed Ansible playbooks for automating the setup of Docker Swarm clusters, Kubernetes clusters, Redis Clusters, Jenkins, NGINX , PROMETHEUS, GRAFANA, PROMTAIL, LOKI.
- Enhanced security by utilizing Ansible Vault to encrypt sensitive data and manage passwords securely.
- Developed and deployed Ansible playbooks for modifying configuration files and setting up scheduled tasks using crontab, and managing server password changes.
- Implemented comprehensive system monitoring solutions using Ansible, capturing critical hardware metrics such as memory usage, disk space, processor information, OS version, and system bit architecture.
- installing Linux os, antivirus, packages. Vulnerability scan and patching in Linux servers.

MANAGE AND MAINTAIN BACKUP BY USING TAR, ZIP, GZIP. Job automation with cronjobs

a)

23) git diff developmentbranch = showing difference files b/w master and development branch.

24) git diff productionbranch devbranch = showing difference files b/w production branch & dev branch.

25) git merge branchname = pressed branch to particular branch name

git branch -merged = show the merged branches

merge = copying files to pressed branch from particular branch

=> git checkout -b name = creating a branch switch to the branch.

=> git push origin -d branchname = deleting a particular branch in local repo.

=> git push origin:branchname -d branchname = deleting remote branch

=> git branch --merged = show the merged branches.

Pushing local branches to remote

→ git push aliasname --all = all branches pushing to remote

(or)

→ git push alias developement production = particular branches pushing to remote

=> directly you can create branches in remote manually in GUI but not reflected directly to local repo, so use commands

=> git branch -r = showing remote repo branches

=> git branch -a = showing all branches both local & remote

Note:- currently what you pointing or inside the branch, you cannot delete the branch

→ git push aliasname :branchname = remote repo particular branch will be deleted.

Note: defaultly it's Master branch, so we change the default branch in remote repo, then only we delete the master branch if we want

- .gitignore :- what are the files we don't want to push then that files first placed in .gitignore, then that files not pushed to github.

→ create process → vi .gitignore → write or paste the files names in → git status → you can see only .gitignore (not files shown) → git add . → git commit -m "any" → git push alias branchname = only .gitignore file to be pushed and

→ If we want to delete .gitignore or any file in local repo, → rm -f : → git commit -a -m "any" → git push alias branch.

Difference b/w Branch and tags :-

Branch	Tags
1) Branch is mutable	tag is immutable
2) Developme purpose code using branch	after production deployment we add tags like version v1, v2 etc
3) git branch	git tag
4) git branch branchname	git tag tagname
5) git push alias bname	git push alias tag tagname
6) git push alias --all	git push alias --tags

Note: tags can be created only master branch. not default.

best usecase [If we want create another branches

→ there is no option for tags deleting remotely GUI.

7) git branch -d branchname | git tag -d tagname

Note: each release code is updated so creating tags we can create tag's in remote, clicks on tags → release desc → file data etc

git stash :- working area files moved to temporary file we can switch to branch, without commit

→ git stash save "dev branch login" = using save we can see the msg when enter git stash list

→ git stash list = we can see the msg & stash number

→ git stash apply = the files back in working area what you stashed before

Note: After commit ds stash not deleted so, use git stash drop

⇒ git stash drop - above use node.

⇒ git stash pop - instead of using git stash apply & drop

27. git cherry-pick :-

Note: git merge current branch all commit's merged to another branch. we want only one commit to another branch use git cherry-pick

→ git cherry-pick commitid = Particular commitid files saved or stored in present branch.

28. git clone: downloading remote repo files to local repo in one directory, what you store in directory and also automatically mapped to local repo,

→ git clone repourl

Difference b/w git init & git clone :-

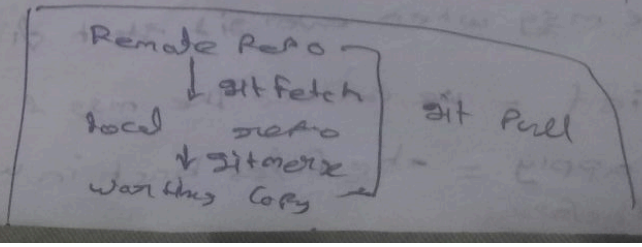
→ when starting new project or we don't have remote repo using git init.

→ already code is there in remote repo, we want to start working on that project, using git clone & downloaded the code. & also remote repourl mapped to local repo at a time

⇒ git fetch alias name branchname = It is going remote repo to local repo only. and it is not going to working area/dir then we use git merge branchname extra commit

→ git Pull alias branchname = remote to local & local to working area also [git fetch + git merge]

eg:



When we use git Pull & fetch :-

⇒ before developing any peice of code first we take updated code that time we use the git Pull command.

⇒ git merge, = if already working in some code, in working area, and ~~developer~~ team lead said some updated code is there in github, so we git fetch first, because we don't want merge's,

Note:- curl and wget commands using downloading softwares in linux. using curl we can call the api

HTTP methods.

1) GET = we can use to get the information from the server

2) Delete = Delete the information or resource from the server.

3) PUT = to update information in the server.

4) POST = to create some information in the server

⇒ Pull requests (Github level) = using Pull request feature, integrate code or merge, one branch to another. ex der to master.

⇒ git merge using in local repo.

Note using fork, we copy files or repository to organization also.

⇒ Rebase, merge, fork, add mefile

⇒ git commit -amend -m "any msg" = change most recent git commit message.

Readme.add:-

⇒ You can add a READ mefile to your Repository to tell other's people why your Project is useful, what they can do with your Project, and how they can get it

✓ [we writing a Msg in Readme file, to user's can understand]

1) What is a version control system?

A) Version control system (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

→ Allow developers to work simultaneously

→ Does not allow overwriting on each other changes

→ maintain the histry of every version.

2 types of VCS

1) Central version control system (SVN)

2) Distributed / Decentralized version control system. (Git, bitbucket)

1) Git: Git is a (DVCS) tool, which handles small as well as large projects with efficiency.

→ It is basically used to store our repositories in remote server such as Github.

GIT	SVN
1) Git is a Decentralized version control tool	1) SVN is a centralized version control tool.
2) Git contains the local repo as well as the full history of the whole Project on all the developers hard drive, so if there is a server outage, you can easily do recover from your team mates local git repo	2) SVN relies only on the central server to store all the versions of the Project file

3) Push

operatio

4) It

3rd

versi

5) clie

share

reposi

local

6) co

done

7) wor

auto

com

2) Wh

A) Su

cre

Subv

3) h

A) Fi

nam

int

3) Push and Pull operations are fast	3) Push and Pull operations are slower compared to Git.
4) It belongs to 3rd generation version control tool system	4) It belongs to 2nd generation version control tools
5) Client nodes can share the entire repositories on their local system	5) Version history is stored on server-side only.
6) Commits can be done offline too	6) Commits can be only online
7) Work are shared automatically by commit	7) nothing is shared automatically

2) What is sub-git ?

A) SubGit is a tool for migrating svn to Git. it creates a writtable Git mirror of a local or remote Subversion and uses both subversion and git if you like

3) how can you clone a Git repository via jenkins?

A) First we must enter the e-mail and user name for your jenkins system, then switch into your job directory and execute the "git config" command.

4) What are advantages of using Git?

A) 1) data redundancy and replication
2) High availability
3) Only one .git directory per repository
4) Superior disk utilization and network Performance.
5) Collaboration friendly
6) Git can to use any sort of Projects

5) How can you create a repository in Git?

A) To create a repository, you must create a directory for the Project if does not exist, then run command ("git init"), By running this command .git directory will be created inside the Proied directory.

6) How you handle the merge conflicts in git?

A) 1) create Pull request
2) modify according to the requirement by sitting with developers.
3) Commit the correct file to the branch
4) merge the current branch with master branch

7) BRANCHing strategies:-

1) master branch ⟶ default branch
2) development branch (dev) → every

team has their own branch => which is created from dev branch.

3) feature branch => every developer has their own branch -> which is created from dev branch

4) hotfix/bugfix branch -> errors/bugs -> which is created from master branch -> errors free -> hotfix/bugfix branch will delete

5) Release branch -> every release has one branch with version -> which is created from master branch. -> once release push to remote -> Release branch will delete.

Note: Branch = To develope the code of our project.

7) merging strategies in locally

A) 1) fast-forward

2) two way or recursive merge or ~~sort~~

3) git rebase (dangerous)

8) How to merge the remote side changes.

A) 1) Pull request -> remote side branches -> github -> in b/w the branches -> "merging is nothing but Pull request"

2) Pull request -> TL/colleques -> approved -> merge Pull request -> conform change.

3) Pull request -> source & destination Source (right side) and destination (left side)

★★★★

1) Centralized version control system (CVCS)

→ It is uses a central server to store all files and enables team collaboration.

→ but the major drawback of CVCS is its single point of failure, i.e failure of the central server.

→ unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all.

→ and even in a worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project. Here DVCS comes into Picture

2) DVCS:

→ DVCS clients not only check out the latest snapshots of the directory but they also fully mirror the repository.

→ If the server goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository

→ Git does not rely on the central server and that is why you can perform many operations when you are offline

→ you can
view logs
you are
→ you
your cha

★★★ Git

client

Repositor

one sing
repositor
→ github

→ You can commit changes, create branches, view logs, and perform other operations when you are offline

→ You require network connection only to publish your changes and take the latest changes

~~AAA~~ Git is a client server architecture.

client → gitbash & server → github

Repository :- Group of Project files to store one single area and each Project has one repository.

→ github has a number of repos.

GIT :=

→ Git is a version control system (VSC) for tracking changes in computer files.

→ Git is an open source Distributed version control system (DVCS) which records changes made to the files laying emphasis on speed, data integrety, distributed, non-linear workflows.

Advantages := Pdf.

version control :-

→ Git is a DVCS, which means that a local clone of the project is a complete version control repository. These fully functionally local repositories make it easy to work offline or remotely

Adding local files to remote repository

1) git remote -v = you can see the data or github links in git.
2) git remote add origin https://github.com/[yours]
3) git push origin master [next sign up with browser, chrome, check the box, ok]. we can see the local files in remote repository

→ git log --online = we can view the local & Remote copies
we can see the commits, data.

⇒ git push superstar master = master data pushing or given to superstar.

**

Pushing Github-code to another server's dc,

→ In Github → setting → click on Add a deploy key → title, copy Paste the key, → Add key.

→ click on <>code → click on code → SSH click on → copy

*

⇒ git clone Paste the id of github. (what you copy in above step)

→ enter: yes

→ ll = you can see the what files in server.

⇒ git log --online | wc -l = we can see the how commit's in number [3 or 5 or etc]

ctrl + o = save

ctrl + x = quit

ctrl + k = delete total.

GIT Fork : sharing or copying one Github to another Github account. using Git Fork.

1) First github login in your ~~to~~browser, create new tab, Paste in search the your wish Github id or required id, fork in the id click on, select repository, create it fork.

2) Refresh your first login, we can see the files or repository

GIT FETch = Coping remote files to local repo. (like Pull)

→ Git Fetch command always followed by git merge command

→ After Git merge = we see the full data [If any changes happend in Remote recently] in already existing file]

→ Git Pull = best use case

Process :-

1) apt-get install git = install git in linux

2) git config --global user.name "RamPrasad"

3) git config --global user.email "leela.challagalli"

* you can track & find who did what change [giving name & email].

4) git init = hidden directory .git will be created. Initializing the git,

→ the git init command creates a new Git repository

→ It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository.

→ Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.

→ git init is only for when you create your own new repository from scratch. It turn's a directory into an empty git repository.

5) GIT ADD :- 5) git add :=

→ Git a. → git add filename

→ git status = showid git file status

git status command tell you where my files residing exactly & what is their status.

→ untracked working changes to

→ Green

* Git add: or multiple git add. going to

→ the git the wor tells Git particular actually

6) git co → git the s project

7) git who c person of the

8) git what → git under it in

→ Untracked files [red colour] = these are in working copy or in workspace then we have changes to be commited

→ Green colour staging area [file name]

* Git add: you create a working, you change and save a file, or multiple files. Then, before you commit, you must git add. this step allows you to choose what you are going to commit.

→ the git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. changes are not actually recorded until you run git command.

6) git commit -m " msg something "

→ git commit is used to create a snapshot of the staged changes along a timeline of a Git projects history.

7) git log = we can see the details of who changes the file, commit for what purpose, person's email, time. [git log command displays all of the commits in a repository history], [gives the commit details]
showing

8) git diff :- what I modified in my files i.e what are the changes done to the file I edited.

→ git diff command helps you see, compare, and understand changes in your project. You can use it in many different situations, e.g. to look at current changes in your working copy, Past

changes in commits, or even to compare branches

→ git diff is used for displaying the difference between any two revisions in your repository

→ git log is for displaying a set of revisions, potentially including the diff between each revision and it's Parent.

★ Head = It means the latest commit

9) git add . = we can add all files on current folder

10) git Push = the git Push command is used to upload local repository content to a remote repository [github]

11) git remote add name github ID or any online repository

= then next time you will only name sufficient [reducing time for Id]

storing Id in that name [default name is origin]

12) git Push name master

master = master copy of your files or original copy

then asking

1) github username :

2) Password :

then we can see files in github

13) git Pull = the git Pull command is used to fetch and download content from a remote repository

→ It gets copied to a versioning database & file gets merged with your work space

Versioning database = local repository

→ git Pull name master
⇒ git Pull name master
(master → branch name)

14) git fetch = the changes are copied to a local repository [view the changes only]

→ git fetch
→ git fetch name [online repo name]

* To get Pass the errors, first fetch or Pull.
* entering gitPush sometimes we get error's [out of sync]

15) GIT BRANCHING :-

Branching is used in version control and software management to maintain stability while isolated changes are made to code.

The git branch command let's you create, list, rename, and delete branches.

→ git branch name = to creating a branch
→ git branch = to show all Available branches
→ git checkout name = switching to specific branch
(name ← branchname)
→ "*" means showing the specific branch.
→ If we want to add the files or creating then we 'commit' after we save it in branch.
→ git Push origin branchname = Pushing local repo branch to remote repo.
→ git checkout -b name = This command
(name ← branchname)
helps to create a branch & switching to branch.

→ git merge name = we can merge present branch to particular branch [one to another]
(name ← branch)

→ git branch -d name = adding merged branch.
[master branch always in,] master branch is merged

→ git branch -D name = deleting unmerged branch

→ git branch --merged = show the merged branches

16) Merge conflict :- when two branch names same or and same data in merging branches then merge conflict happens.

→ To overcome this problem,

→ git config --global merge.tool kdiff3

kdiff3 → most popular merge tool.

→ git mergetool = merge tool opens. [POP- comeup]
this is specifically for merge tool

17) Git stashing :-

It saves all the changes et to a temporary location, these changes can then be reapplied anywhere.

→ Git stashing is a way of creating a checkpoint for non-committed changes.

18) Branch Rebasing :-

Git rebasing is used for we can see the changed two branches [git log showing 1]

17) git s
whe
stash i
files g
18) git
19) git
20) git
21) git
=
22)
Co
→
23)
→
→
→
→

17) git stash save 'first msg' → single quotes

when you donot do a commit, then you wont stash, stash is told, you changes before you commit, then files go to stash.

18) git stash list = Showing your stash files.

19) git stash clear = to clear stash in branch [entire stash]

20) git stash POP = recent stash deleting

21) git stash drop stash@{0}
→ copying & Paste it.
= This is delete to Particular stash or demand.

22) git rebase master (or) branch (you specifing) : you can see commit messages in your branch,
→ files are also comming in to current branch.

★
23) git revert HEAD = we want to undo the last commit.
↳ Last commit

→ even for Revert there is a commit is recorded
→ can you do a Revert of revert, yes using commit, revert id, [because of revert commit id]
→ if you want to revert a commit as well
→ HEAD = Most Recent commit

24) git reset commitId = If you want to
↑
Pasteid

change to particular comait view head position
(or) last commit this id showing. [It will keep the files as it]

25) git reset --hard commitId = If you want to
delete particular comait

26) Git apply

27) git config --global merge.tool meld
↑
Merge tool widely used one

28) git reset soft comitid = comit's will be deleted
but files in local repo

29) glo = showing local & Remote repositories [Head, Master]