**Git**

Git is a distributed version control system.

GitHub is centralized repository or storage

**Git workflow**

First create is Linux EC2 instance with SSH and HTTP port.

Update EC2 Instance: yum update -y.

Install git package: yum install git -y.

Create a director with anyone name

Run the following command inside a director #git init (git initialize).

As soon as you run #git init command directory will be converted into git repository and divided into 3 different logical areas or stages which you cannot see.

1) Working directory or work space.

2) Staging/index area

3) Local repository (In backend one hidden directory will created with name .git (which is nothing but a local repository)).

You can write the code or edit the code in working director.

Now you have to commit your code, but before sending your code to local repository send this code in Staging area. Process of sending code to staging area is call add.

You can add multiple files or single file in Staging area and from Staging area you can commit all files of staging area.

Now send code from Staging area to local repository process of sending code to local repository is called as commit/Save (snap shot). Whenever you commit your code one unique commit ID will be generated. You can put tag to commit Id for understanding purpose.

When you commit your code from Staging area to local repository your code first come in (master, main) branch and from (master, main) branch you can push your code to GitHub (inside GitHub branch).

Now you can send your code to GitHub sending code to GitHub is called push.

Now another person is working in Singapore region he wants to edit the same code first he will get the code from the central repository. Getting/accessing code from Central repository is called as pull. As soon as he pulled the code from the central repository code automatically comes in all three stages work space, Staging area and local repository.

By default we get master branch in local repository you can also create your own sub branches and merge those branches with the master beach.

**Advantages of git**

1) Free and open source

2) Fast and small in size as you’re working with your local repository

3) Secure

4) Parallel branching

5) Data is available in multi-location.

**Note**: you can also add commit the directory

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Lab**

Rahul is working in Mumbai and Sai working in London.

Rahul (Mumbai)

1) First create is Linux EC2 instance with SSH and HTTP port.

2) Update EC2 Instance **yum update -y**.

3) Install git package: **yum install git -y**.

4) Create a director with anyone name: **#mkdir mumbaigit**

5) Run the following command inside a director **#git inti**

"When you run **#git init** .git directory will be created it is nothing but a local repository"

To see git package install or not

**#which git**

To see git version

**#git –version**

(git version 2.32.0)

Provide any username and mail ID using following command

**#git config --global** [**use.name**](http://use.name) **"rahul"**

**#git config --global user.email "**[**rahul@gmail.com**](mailto:rahul@gmail.com)**"**

Or

**#git config --global user.mail "**[**rahul@gmail.com**](mailto:rahul@gmail.com)**"**

To see username and mail ID use following command

**#git config --list**

OR **#git config --global --list**

Now create one centralized repository in GitHub

Click on create repository.

Provide name and description.

Click on create.

Create same setup in London as we have done in Mumbai.

1) First create is Linux EC2 instance with SSH and HTTP port.

2) Update EC2 Instance **yum update -y**.

3) Install git package: **yum install git -y**.

4) Create a director with anyone name: **#mkdir londonigit**

5) Run the following command inside a director **#git inti**

Provide any username and mail ID using following command

**#git config --global** [**use.name**](http://use.name) **"sai"**

**#git config --global user.email "**[**sai@gmail.com**](mailto:sai@gmail.com)**"**

Now create file of your code in mumbaigit directory

Check the status of git using following command

**#git status**

**Three type of status are there**

1) Nothing to commit (file is not created in working area) (before creating the file)

2) Untracked file (file is created but not added in Staging or index area) (after creating file)

3) Changes to be committed (now you can commit your file in local repository) (file is in staging area)

Add file to Staging or index area using following command

**#git add <filename>**

OR

**#git add .**

(Dot means all file from current directly)

Now commit file from Staging or index area to local repository using following command

**#git commit -m "message"**

Eg **#git commit -m "my first commit from Mumbai"**

Now if you check status it will show nothing to commit

To see committed username, mail ID date, time and commit msg use following command

**#git log**

To see data inside that file use following command

**#git show <more than 7 char of commit id>**

Now connect or add local repository with centralized repository with the help of following commands or copy commands on Central repository

**#git remote add origin <GitHub http Link>**

To see branch name use following command

**#git branch**

To change the branch name use following command

**#git branch -M <new branch name>**

Eg. **#git branch -M main**

To push data from local to Central repository

**#git push -u origin main**

"In Central repository 2 logical parts are there equivalent to workspace and local repository."

**Sai (London)**

Before pulling data from Central repository first connect or add Central repository with local repository of Sai

Now connect or add local repository with centralized repository with the help of following commands or copy commands on Central repository

**#git remote add origin <github http Link>**

To pull data from central repository to local repository

**#git pull origin main**

To change the branch name use following command

**#git branch -M <new branch name>**

Eg. **#git branch -M main**

Now Sai can edit the file and perform add, commit and push just like Rahul did in Mumbai.

**BEFORE PUSH CODE TO CENTRAL BOTH TEAMS HAVING ONE SYNCUP CALL TO DISCUSS WORK.**

If you have to ignore some file with the specific file format.

Create on hidden file with the name **.gitignore** and mention those extensions in that file

Eg. \*.csv, \*.java

Now add and commit **.gitignore** file

To see recent commit

**#git log -1**

To see 2 recent commit

**#git log -2**

To see all commit in one line

**#git log --oneline**

To search particular commit contain below word in a comment then use following command

**#git log --grep "London"**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Git branch**

\* represent current branch or workspace is pointing to current branch.

Whenever you create any new branch, automatically all data (commits and file from work space) of the existing branch (master) copy to new branch.

If you create a new file or edit any file in your new branch and forget to add and commit then that while is visible in each and every other branch until you commit that file.

Note: workspace common for both main and new branch

(File created in workspace will be visible in any of the branch workspace until you commit that file. Once you commit that file then that file belong to that particular branch)

Default branch master branch.

Changes are personal to the particular branch.

You can create any number of branches.

Branch concept is useful for parallel development.

To see branch name use following command

**#git branch**

To change the branch name use following command

**#git branch -M <new branch name>**

Eg. **#git branch -M main**

Command to create new branch

**#git branch <new branch name>**

Eg.**#git branch new**

Command to switch branch

**#git checkout <new branch name>**

Eg.**#git checkout new**

**Branch merging**

Being present in main branch copy and paste the data of new branch or vice versa.

It is pull mechanism or copy paste mechanism.

You can't merge branches from different repositories.

Command to merge branch

**#git merge new**

When you are having same file name in both branches and different data inside those file and trying to merge new branch with main branch then merge conflict occur.

If data is same no conflict.

You need to fix that particular file in a main branch, like where you want that data from the new branch file in top of the previous code or in bottom of your previous code or you have to overwrite your previous code.

If you want to delete a branch use of following command

**#git branch -d <branch name>**

"Being present in the branch you can't delete same branch"

How to push to other branch?

**#git push origin source:destination**

Eg. **#git push origin main:new**

How to pull from other branch?

**#git pull origin source:destination**

Eg. **#git pull origin new:main.**

Source is local repository

Destination is central repository

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Git Stash**

Git stash is a temporary storage in your repository.

If you have to perform multiple task (log in page, logout page) in single file in one branch then you go for git stash concept.

"First add and commit empty file before using for stash concept."

Command to send data to stash

**#git stash**

Command to check stash store

**#git stash list**

(Stash@{0} recently add new

 Stash@{1} previously add old)

Command to bring back task from stash

**#git stash apply stash@{0}**

(Any number task u can get 0 1 2 etc)

After finishing your first task and add and commit that file, when you try to bring your second task back conflict occurs because of previous task code already present in a file. Better to delete the content of file before bringing second task or clear the conflict.

When you send data from file to stash it is cut and paste and when you bring back your data from stash to file it is copy and paste.

Command to clear stash

**#git stash clear**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Git reset**

After add file to stating/index area if you want to undo/remove file from Staging area then we go for reset concept

Use below Command

**#git reset <file name>**

OR

**#git reset .**

To remove file from both work space or Staging area use below command

**#git reset --hard**

"Reset concept work before commit"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­­­­­­­­­­­­­­­­­­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Git revert**

Revert command help you to create new commit id where you can mention please ignore previous commit it happened by mistake.

Command

**#git revert <previous commit id>**

As soon as you run the above command one file get open in vim editor mention your message in first line save and quit and file from workspace and Staging area will be deleted and new commit id will be generated.

We cannot do anything after commit. We cannot delete commit file.

"Revert concept work after commit"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Git remove**

If you want to remove any file from workspace which is already committed file then we go for git remove concept.

Command

**#rm -f filename**

OR

**#git rm filename**

Once you remove that file from work space automatically snapshot will be deleted from Staging area.

After using remove command please do commit

**#git commit -m "message"**

How to remove untracked file?

Untracked file = file not being added and committed is call untracked file.

Or file is only in workspace.

**#git clean -n**

Above command will show it would remove those many files

**#git clean -f**

Above command will remove all untracked file.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Git tag**

Tag is a meaningful name given to commit id

Command to apply tag

**#git tag -a <tagname> -m "Message" <commit id>**

Command to show list of tag

**#git tag**

Instead of commit id now you can use tag name

Command to delete tag

**#git tag -d <tagname>**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Github clone or clone**

It creates a local repository automatically in Linux machine (instance) with the same name as in GitHub account repository name.

Command

**#git clone <github url>**

"Clone concept is used when your GitHub repository already present"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Whatever things you are doing in your insurance using command you can do the same thing in GitHub GUI. But it's not recommend.

If your Central repository having any extra file which is not present in your local repository and you’re trying to push some files from your local repository to central repository conflict occur, so first you need to pull from central repository and then you have to push from local repository.

"While pushing if you are getting any error then first pull and then push"

How to check your local repository and Central repository having same data?

Push from your Central repository it will show everything up to date.

You can find previous version of file in commit of GitHub.

**Public vs private central repository**

To push we have to give credentials

If central repository is private it will ask credentials at time of pull.

If center repository is public it won't ask credentials at time of pull.

How to push and pull files in central repo without credentials????

Need to generate key in instance using following command

#ssh-keygen

Enter enter

Now copy this key from /root/.ssh/id\_rsa.pub file to central repo

Go to setting

Click on ssh and gpg key

Click on new ssh key

Provide title any name

Copy that key here

Click on add ssh

Now using ssh link add local repo to central repo.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Day to day task**

1) Being a devops guy we are going to write to script all those script we are storing in git and github.

2) We integrate git with jenkings, Maven and many other tools.

3) Need to provide training to others on git how to use

4) Sometimes you need to push code here you can take requirements in which branch and merging and all

5) Need provided credentials to client so they can pull code and do check

6) Sometimes we take remote session install git and setup repo and all we do

7) Forking we do means copy repo of github to another github

8) We can provide access to other developer without sharing credentials with ssh key we can also take back/revoke access by deleting ssh key.

What are the possible errors?

1) If you forget to provide name an email while committing it through error and while pushing it through error.

2) You can't clone Private repository without providing credentials.

3) In ssh key if we miss any char

4) Some we run git command out side git repository.

5) If forgot to open ssh port.