Introduction to GIT

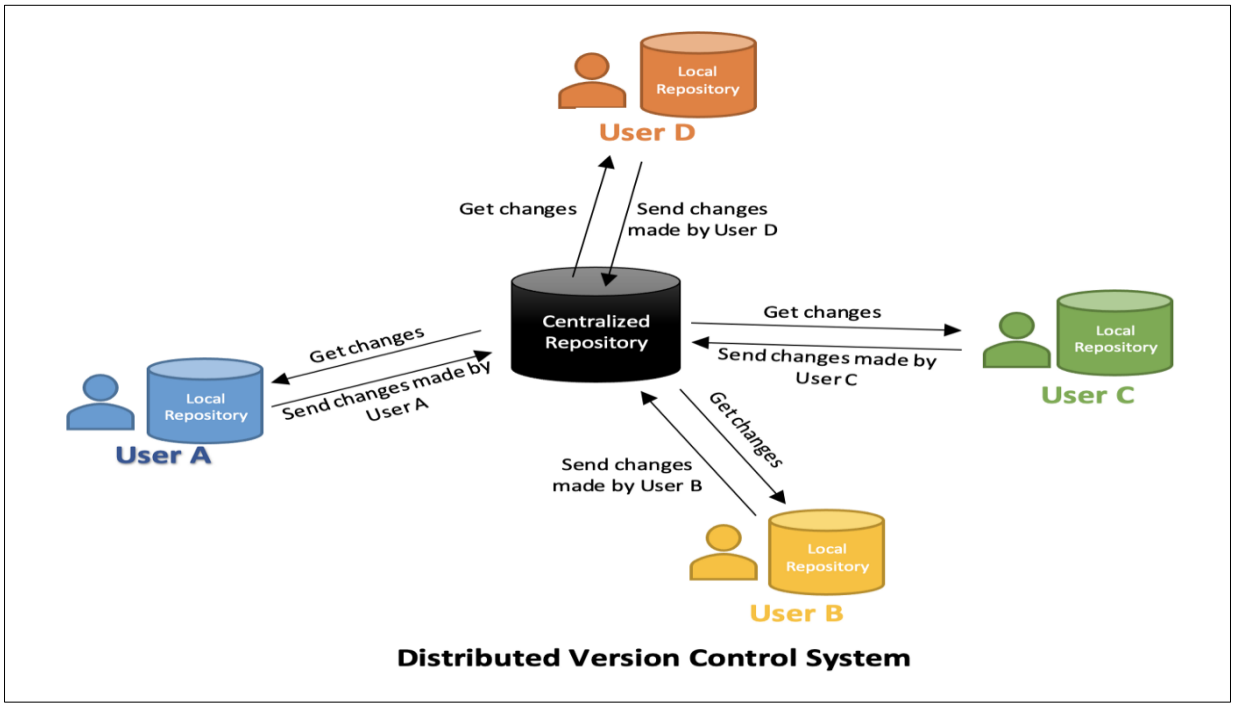
- **Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.**
- **Git was initially designed and developed by Linus Torvalds for Linux kernel development.**
- **Git is a free software distributed under the terms of the GNU General Public License version 2.**

What is repository

- **A Git repository is the folder inside a project. (cloudethix/.git)**
- **This repository tracks all changes made to files in your project, building a history over time.**
- **Meaning, if you delete the folder, then you delete your project's history.**

Git Local & Remote Repositories

- **Git has 2 types of repositories Local & Remote.**
- **Local repository is on your own machine.**
- **Remote repository is on centralized servers.**
- **In working env we use cloud based remote repositories instead of managing it by own-self.**
- **Top SCM remote repos are**
    - **GitHub**
    - **BitBucket**
    - **GitLab**
    - **Code-Commit (AWS)**
    - **Cloud Source Repositories (GCP)**
    - **DevOps repository (Azure)**


Local Repository
User D
Get changes
Send changes made by User D
Centralized Repository
Get changes
Send changes made by User C
Local Repository
User C
Get changes
Send changes made by User A
Local Repository
User A
Send changes made by User B
Get changes
Local Repository
User B
Distributed Version Control System

Install GIT

Understand the setup -

I am using MacBook Pro.

I have installed Oracle Virtual Box

On the top of OVB, I have installed CentOS 7

SSH from mac terminal trying to SSH to centos Machine

On centos 7

 # yum install git

 # git --version

Initialise GIT repo

 # git init

As soon as you run this command it will create .git directory. This is the directory where GIT will track all the changes.

Setup GIT config (Username & Password)

 # git config user.name cloudethix

 # git config user.email cloudethix_devops01@gmail.com

 # git config --- to see all the options

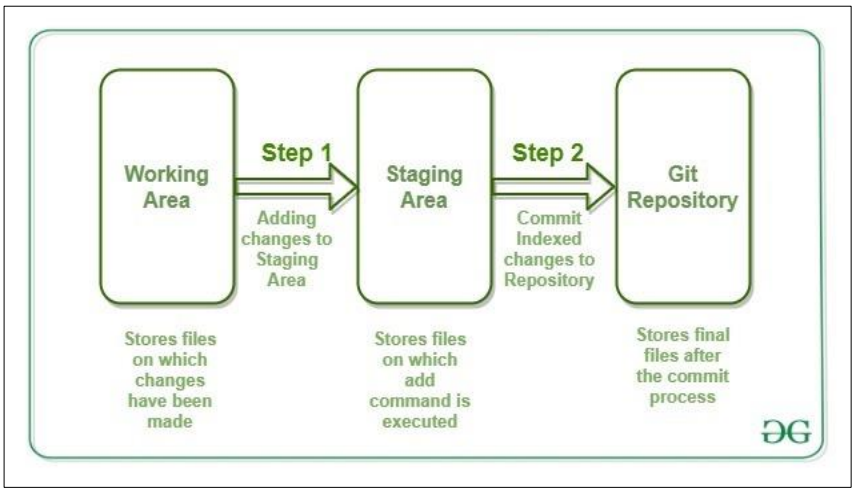
 # git config -l

 # git config -e

GIT life Cycle

- **Below image explain the git lifecycle.**
- **We have created a file in our local repo 01.txt**
- **Our 01.txt file is in working area.**

- **The working area is where files that are not handled by git. These files are also referred to as "untracked files."**
- **When you run the git add command file will go to staging area.**
- **Staging area is files that are going to be a part of the next commit, which lets git know what changes in the file are going to occur for the next commit.**



GIT Operations

Initialize the git repository.
 # git init

This will **add** files to staging area.
 # git add file_name

This will **commit** the file to repository from staging area.
 # git commit -m "MSG"

This will **restore** file from last commit. If u want u can delete the file from local system & check
 # git restore file_name

Suppose 2 file are in staged area & u want to remove one of them
 # git restore --staged

This will **remove** the file from staged area with --cached but will not delete the local file however with -f it will.

 # git rm --cached
 # git rm -f

Add .**gitignore** file to repo & add name of the files that u do not want to commit.

 # .gitingnore

Check the Commit **Logs**

 # git log
 # git log --oneline

It groups each commit by author and displays the first line of each commit message.

 # git shortlog

Filtering the Commit History

 # git log -3
 # git log --after="2014-7-1"
 # git log --after="yesterday"
 # git log --after="2014-7-1" --before="2014-7-4"
 # git log --author="John"
 # git log --author="John\|Mary"

GIT Branching

1. In Git, a branch is a snapshot of the main repository, with separate name.
2. Meaning that, Your main branch is master branch (Master is default branch).
3. You took snapshot of your master branch & promote New branch from that snapshot.
4. Now here, GIT will start tracking the changes separately from both the branches.

List branches

 # git branch --list / -l
 # git branch

Create New branch. (Note Give some meaningful names)

 # git branch branch_name

BugFix
HotFix
Feature
Experimental
Release
Develop

Delete branch.

 # git branch -d

Rename the branch.

 # git branch -m help hotfix

Change the working branch.

 # git checkout hotfix
 # git branch
 # git checkout master

Create new branch and checkout.

 # git checkout -b kdesi
 # git log --oneline --decorate (Here you will notice the HEAD is same for all)

Understand the branchings.

 # git branch
 # git checkout kdesi
 # echo "Hello moto" >4.txt
 # git add 4.txt
 # git commit -m "Added 4.txt"
 # git log --oneline --decorate (Here you will notice the HEAD is now changed)
 # git checkout master
 # ll (Here file 4.txt is not Present BCZ it changed in another branch)

Merge branches

- Suppose you have below branches
  master ----> Default branch
  release ----> Main branch
  feature/dashboard ---> Here dashboard team is working

feature/login ---> Here login team is working

- Suppose you are running one application & its code is in release branch.
- Later team came up with idea that there shud be some new changes in Dashboard and new login functionality shud be added.
- You created 2 branches i.e feature/dashboard & feature/login for both the team respectively.
- Here dashboard team has updated the code to the feature/dashboard branch & those changes are tested.
- Now you want to add dashboard functionality to be added to your application. So here u will have to merger the code from feature/dashboard to your main branch release.

```
# git branch
# git checkout release
# git merge feature/dashboard
```

- After some days login team has also completed thier work & its time to add login functionality to your application.

```
# git branch
# git checkout release
# git merger feature/login
```

Remote repository

- Create account to GitHub.
- And create 1 repository to the GitHub. once you create the new repo, u will get below msgs.

create a new repository on the command line

```
echo "# cloudethix_devops" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin
https://github.com/SandeepDawre/cloudethix_devops.git
git push -u origin main
```

push an existing repository from the command line

```
git remote add origin
https://github.com/SandeepDawre/cloudethix_devops.git
```

git branch -M main
git push -u origin main

Initialize remote repository

 # git remote add origin https://github.com/SandeepDawre/cloudethix_devops.git

 # git remote -v
origin https://github.com/SandeepDawre/cloudethix_devops.git (fetch)
origin https://github.com/SandeepDawre/cloudethix_devops.git (push)

- Here push will not work with the HTTPS origin due to sec issue. We will to do it with SSH keys.

- Generate SSH keys to your linux machine with below command
  # ssh-keygen

  - *Copy data of id_rsa.pub file & paste it to GITHUB -- Profile --- - settings --- SSH key, & Add public key there*

Remove the HTTPS Origins & SSH Based origins.

 # git remote remove origin

 # git remote add origin git@github.com:SandeepDawre/cloudethix_devops.git
 # git remote -v
origin git@github.com:SandeepDawre/cloudethix_devops.git (fetch)
origin git@github.com:SandeepDawre/cloudethix_devops.git (push)

Push Data Remote

 # git push origin master (This shud work now)

Clone Remote Repository

 # git clone https://github.com/SandeepDawre/cloudethix_devops.git

Pull Request (PR)

- Suppose you are working in one branch locally say test branch.
- You made changes locally & you push the changes to remote repo under test branch.
- Now you want to merger the Test branch to the master branch.
- So while working in organization, we just can not merge the code to the master branch as it is main branch of your code base.
- Here, instead of merging Test branch to master branch directly, we create the Pull Request.
- Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub.
- Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the master branch.
- Meaning that we can add reviewers to review your code and once approved by reviewers they can merge it to master branch.

Create one branch on local repo.

```
# git branch -l
# git branch test_pr
# git checkout test_pr

OR

# git branch
# git checkout -b test_pr
  (This will create new branch and enters to it)
```

Add some file and commit to the local repository

```
# echo "Hello This is test pr test" > test_pr.txt
```

```
# git status
# git add test_pr.txt
# git commit -m "Test PR file is added"
```

Now Push This new Branch to Remote repository.

```
# git remote -v
# git push origin test_pr
```

- Now go the Remote repo & check the test_pr branch.
- Here when you witch the master branch, You will some messages like "This branch is 4 commits behind release."
- Here, in order merge the test_pr branch to master, we will have to create the PR & we need to add reviewers to PR.
- Reviewers can review the changes & approve the same, reviewer can merge the test_pr branch to master branch.
- In organization there are some rules set, we can only merge the test_pr branch to master only when it is approved by at least 3 team members.

--> GitHub
--> Repository
--> Switch to Branch
--> Pull Request
--> Select Base Branch (Where you want to merge)
--> Select compare (From which branch it should merge)
--> Add Title & Write meaningful comments so that reviewer can understand.
--> Create PR.

- Here reviewer will get the notification or you have to reach them to approval.
- They will review all the changes and approve if valid.
- Once it is approved by 3 reviewers, one of the reviewer can merge to the master branch.

GIT Fetch & Pull

- So far, we have merge the test_pr branch to the master branch remotely.
- However, master branch on locally for all other users is not the same, since we have just merge the test_pr branch to master branch remotely.

- So, here we need to Fetch the latest changes from remote master branch to local master branch & then we need to merge the remote master branch to local master branch to see all the latest changes.

```
# git fetch origin master
# git branch -l
# git checkout master
# git merge origin/master
```

OR

- Here instead of running above 2 commands, we can simply run git pull command, which will fetch and merge to the local master branch from remote.

```
# git pull origin master
```

GIT FORK

- Suppose you are a team of 20 people who have deployed one Open Source tool from your code repo. (20 Members/Maintainers)
- Since it is open source project, there are millions/multiple people who are contributing to the project.
- Now suppose You also want to contribute to that Project.
- So here, You will clone the repo locally & created new branch. You made some changes & created pull request to merge your branch.
- But here, maintainer does not want to add PR to main project. So how to create the PR which is not the part of main project.

GIT Fork will help you.

- With Fork you are creating your own copy of the original project. You can experiment with the project without impacting original project.
- At one stage, suppose you have developed something & now want to contribute. Then we need to create the pull request to our local copy & add maintainers as reviewers.
- So that maintainer can review the new feature & provide his comments.
- If they finds all good, & your feature can be a part of that tool, then they can merge the PR which was created on your copy to the main project code.

GIT Merge Conflict.

- Merge conflicts usually happens when you merge the branches.
- Suppose you have one project remotely & it is deployed from master branch.
- As said, Developer will fetch the latest changes locally from remote master branch.
- Suppose dev_01 is working on file called setup.txt in master branch, Dev_01 made some changes to the file & committed it locally. Later it is commited to the master branch.
- At the same time, Dev_02 has created new branch from master & he is also working on same file i.e. setup.txt.
- Lets see below example.

git branch
git checkout **master**
git branch **secondary**
vim setup.txt
Follow the instructions…
git add setup.txt
git commit -m "setup.txt is added to master branch"
git status

git checkout **secondary**
vim setup.txt
Read the instructions…

git add setup.txt
git commit -m "setup.txt is added to the secondary branch."
git status

git checkout master
git merge secondary
Auto-merging setup.txt
CONFLICT (add/add): Merge conflict in setup.txt
Automatic merge failed; fix conflicts and then commit the result.

- Here you may get merge conflict issue

- How to Solve the merge conflict Open file cat setup.txt.

```
<<<<<<< HEAD
Follow the instructions…
=======
Read the instructions…
>>>>>>> secondary
```

- The above output shows that the setup.txt file contains the content added in both branches with some extra symbols.

- The seven less characters (<<<<<<<) with HEAD has added before the committed content of the master branch, and the seven equal sign characters (=======) has added before the committed content of the secondary branch.

- The seven greater than characters (>>>>>>>) has added with the secondary branch name at the end of the file.

- Here, the less than character indicates the current branch's edit. The equal sign indicates the end of the first edit. The greater than character indicates the end of the second edit.

- when you run the GIT status command you will get merge conflict issue.

```
# git status
> # On branch secondary
> # You have un-merged paths.
> #   (fix conflicts and run "git commit")
> #
> # Un-merged paths:
> #   (use "git add ..." to mark resolution)
> #
> # both modified:      setup.txt
> #
> no changes added to commit (use "git add" and/or "git
commit -a")
```

- How to fix, open the setup.txt & keep the line that you wish to keep in code & then add & commit the same.

```
# git add setup.txt
# git status

# git commit -m "Merge conflicts are fixed by keeping
file from secondary branch."
```

GIT Rebase

- Suppose you have one project & all code base we have in master branch.
- Suppose you are working on one feature, say login feature. So you will create your branch ***feature/login*** from master.
- Since feature that you are developing is one the important & big feature, you took time to develop it.
- Meanwhile, other developers in your team has developed profile feature and they committed code to the ***feature/profile*** branch and later merged branch to the master branch.
- Here you are still developing the login feature and it is yet to be committed.
- Here due to latest ***feature/profile*** merge, remote master branch is ahead of you local master branch.
- Here you also want the latest changes from master branch to your working ***feature/login*** branch.
- You can achieve this by merge master branch to your own branch ***feature/login*** & once merged, new commit ID will be created.

```
# git checkout master
# git pull origin master
# git checkout feature/login
# git merge master
```

- But this way a new dummy commit is added. If you want to avoid history you can **rebase**.

```
# git checkout feature/login
# git rebase master
```

- As soon as you rebase the master branch, your working branch will get all recent changes from master branch.
- Now you have the recent changes in your working branch & with that you have completed login feature development.

- Later you can then merge your login branch to the master branch.

```
# git checkout master
# git merge feature/login
```

- So what is rebase, changing the base without commit ID. Meaning that we are putting one branch on the top of another branch. Instead of merging, we just moved feature/login on the top of master branch.
- Now feature/login will have all the latest contents of master branch.

GIT Reset & Revert.

- While working with the Git there might be chances of mistakes.
- Suppose you committed something which you did not want to commit. So in that case you can revert the commit by GIT reset command.

```
# echo "Revert Test" >revert.txt
# git add revert.txt
# git commit -m "Hey Cloudethix, revert file is added"
# git log
# git revert 8fbaf35
# git revert --no-edit 8fbaf35
```

- GIT Revert will preserves history.

- Another way is reset.

```
# echo "RESET Test" >reset.txt
# git add reset.txt
# git commit -m "Hey Cloudethix, RESET.txt file added"
# git log --oneline

# git reset --soft HEAD~2
```

(*Soft will keep the file locally and stage area*)

```
# git reset --hard HEAD~2
```
(*Hard will delete the file from both places*)

- GIT RESET will also delete the history.

GIT Stash

- Suppose you are working in develop branch and you have added couple of files to staging area which you want to commit.
- Suddenly your manager has asked that there is some issue with application, which was deployed from release branch.
- Your manager told that issues is due to ***mprog.py*** file in release branch.
- so in order to fix the issue you have to switch the release branch to modify ***mprog.py*** file, but when you switch the branch from develop to master, all the changes from staging area will be gone.
- If you want to preserve that change, You need to add that in stash.

```
# git checkout master/develop
# echo "Stash Test" >stash_test.txt
# git branch
# git add stash_test.txt    (added to stage area)
```

Here you have been informed to fix the application issue.

```
# git checkout release
```

You fixed the issue & you are switching back to working branch.

```
# git checkout master/develop
# git status
```

Here you will see all the staged changes are gone. So how to save those changes. GIT Stash.

```
# git stash
# git stash list
# git stats pop
# git stash show stash@{0}
```

```
# git stash pop stash@{0}
# git stash drop stash@{1}
# git stash clear
```

GIT REFLOG

- Suppose you committed one file to the repository & later you realised that this was not required.
- And then you did the hard reset which will delete the file from repository and local system.

```
# echo "Hello Cloudethix test RESET 01" >RESET.txt
# git add RESET.txt
# git commit -m "RESET.txt file added"
# git log --oneline
# git revert --no-edit a164bbe
```

- But after sometime, you again realised that this was much needed file. However, you have already deleted file with hard reset. Now how to restore the file which is hard reset.

- Here, git reflog will help you. It will give you all the logs details that you have taken against the repositories including, merges, revert, reset etc.

```
# git reflog
# git reset --hard HEAD@{3}
```

GIT Amend

- commit --amend is used to modify the most recent commit.
- It combines changes in the staging environment with the latest commit, and creates a new commit.
- This new commit replaces the latest commit entirely.

```
# echo "Hello Amend test" >amend.txt
# git add amend.txt
# git commit -m  "added amend.txt filr"
# git log --oneline
# git commit --amend -m "Adding amend.txt file to
enable login settings."
```