GIT

- It is a Global Information Tracker
- It is a Distributed version control system (or) Source code management
- We are controlling the versions of the applications through GIT
- Every application contains code. That code was maintained by SCM

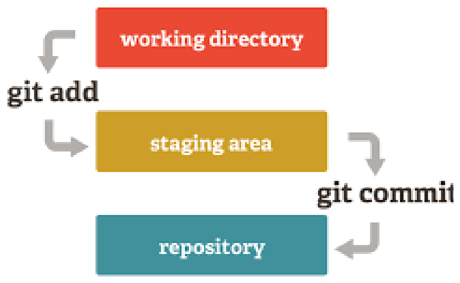
Def

- Git is used to track the files
- It will maintain the multiple versions of the same file
- Git is a platform independent i.e. it will open in all systems
- It is a free and open Source
- Git save time and developers can fetch and create pull requests without switching
- It is 3rd generation of VCS

VCS History

1. SCM(Source code control management)	- To track only one file
2. RCS(Revision control system)	- Track multiple files but not directories/folders
3. CVS(concurrent version system)	- Track multiple files and directories but single user
4. SVN(sub version)	- Track multiple files & directories and multiple users
5. GIT(Updated)	- Distributed version control system

Stages in GIT



Working Directory :

- Untracked files are present here.
- When a new file is created, updated or an existing file deleted, those changes automatically go into the working area.

Staging Area:

Here, changed files are present here. So, we can commit/save

Repository:

- A Repository has all the project - related data
- Repository in git is considered as your project folder
- It contains the collection of the files and also history of changes made to those files

Types Of Repo:

Local Repo:

- The local repository is everything in your git directory.
- local repository contains all checkpoints (or) commits.
- It is the area that saves everything (so don't delete it)

Remote Repo:

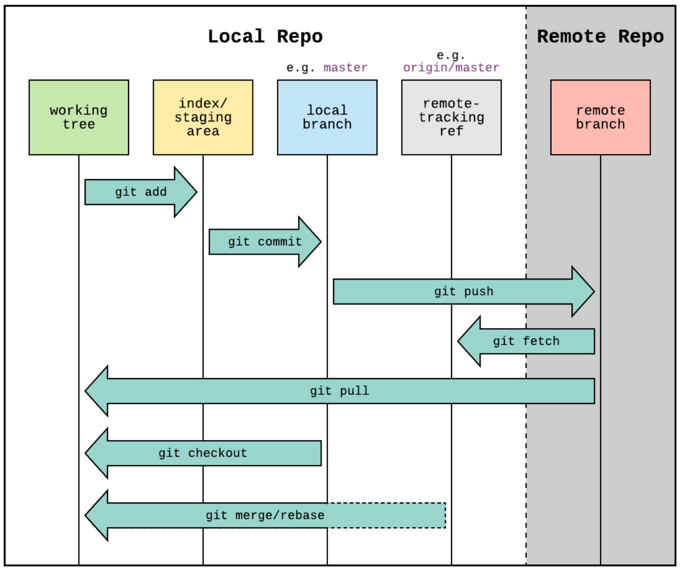
- The remote repository is a git repository that is stored on some remote computer.

- It allows you to centralize the work done by each developer

Central Repo:

- This will be present in our GitHub

GIT WORKFLOW



GIT Installation in Linux

sudo yum install git -y → from ec2-user

yum install git -y → from root user

git --version (or) git -v → check the git installed or not in your local

We have to initialize the empty repository. Otherwise, file will not track/commit

git init . (. represents current directory)

GIT ADD (Track the files)

- [ ] Git add command is straight forward command. It add files to the staging area
- [ ] We can add single or multiple files at once in the staging area
- [ ] Every time we add or update any file in our project, it is required to forward updates to the staging area
- [ ] The staging and committing are co-related to each other

git add filename

git add * (all files)

git add -f * (Forcefully)

git add . (hidden & normal files)

Untrack the files

git rm --cached filename

git rm --cached file1 file2 file3 file4 ...... (for particular files)

git rm --cached *

git rm -r --cached . (for all files)

GIT RESTORE

If you want to discard/undo the changes in working directory use git restore

- git restore --staged filename

GIT STATUS

- ▢ The git status command is used to display the state of the repository and staging area
- ▢ It allows us to see the tracked, untracked files and changes
- ▢ This command will not show any commit records or information
- git status

GIT COMMIT

- ▢ It is used to record the changes in the repository
- ▢ It is the next command after the git add
- ▢ Every commit contains the index data and the commit message
    - git commit -m "enter the commit message" filename
    - git commit -m "enter the commit message"   . (. represents every changed/ tracking files)
    - git commit -m "message name" (to commit all the tracked files)
- At a time track the file and save the file in repository
    - git commit -a -m "enter the commit message" filename
- See the list of commits history in git
    - git log
    - git log --oneline
- If you want see the commit file name
    - git show commitID --name-only

Note: If you enter starting 7 numbers means, it will take it as a commitID

- If I want to see how many times i'm committed the file ?
    - git log --follow --all fileName

Note: We can't commit the committed file. We can commit if that file has any changes

GIT Configure

If you want to give your username and E-mail id to those commits then

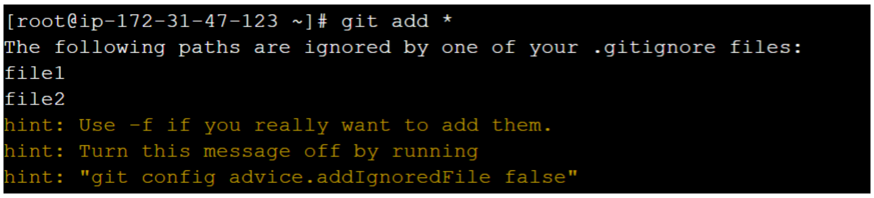
- git config --global user.name "EnterYourName"
- git config --global user.email "EnterYourEmailId"

--global :- If we give global means, that particular commit name or email id applies to all the git repository. If we don't give global also it works but it will work on that particular/current repository

Note: Now, giving the git log command to see changes, it won't work because after configuring we haven't done anything. Now create a file and commit that file and give git log you will see changes as you configure.

GIT Ignore

- ▢ It will be useful when you don't want to track some specific files then we use a file called .gitignore
- ▢ create some text files and creates a directories with "jpg" files
    - vi .gitignore
    - *.txt
    - Now all the text files will be ignore



```
[root@ip-172-31-47-123 ~]# git add *
The following paths are ignored by one of your .gitignore files:
file1
file2
hint: Use -f if you really want to add them.
hint: Turn this message off by running
hint: "git config advice.addIgnoredFile false"
```

- If you really again want that file use → git add -f *

Note: If you want to ignore a file, before tracking only you have to be put. Otherwise, it will be no use

Changing the details for Latest commit

- git commit --amend --author "sandy<sandy@gmail.com>"
- After performing this command, one file opened, don't do anything. perform 'wq' .
  - --amend : It's a flag, if you want to edit the commit details use this command
  - --author : Author details edited
- Roll back to original author details
  - git commit --amend --reset-author
- If you want to edit the message details for commit
  - git commit --amend -m "EnterTheEditingMessage" filename
    - Here filename is not mandatory

How to add a file to previous commit ?

- git commit --amend --no-edit fileName

DELETING COMMITS

Reset

- git reset undoes changes and moves the branch pointer, discarding subsequent commits
- To remove local commits let's use "reset" command
- Reset commands uses three different options
  a. --mixed : This is the default option(remove commit and move changes to working area)
  b. --soft   : If we give soft means, only commits are deleted not the files
  c. --hard   : It removes commits and permanently discarded changes without our permission

git reset --hard HEAD~1 → This is deleting the latest commit

git log → Head will moved to latest commit

git reset --hard HEAD~3 → It will delete latest i.e., top 3 commits

Delete all commits → git update-ref -d HEAD

Note:- If we perform above command we don't have any commits but inside files data is deleted

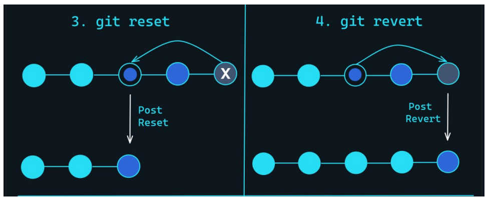
Revert

- git revert creates new undo commits, preserving history
- If you want to undo local/remote commit use this
- It will not remove the commit but it removes changes in the commit and makes a new commit
- Revert used to delete specific/particular commit

git revert commitID

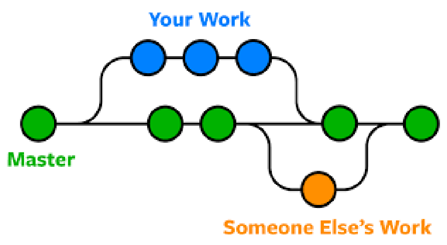
reset vs revert

- Reset removes commit from the history and revert will not
- Reset works for local commits and revert works for local and revert commits



BRANCHES

- A branch represents an independent line of development
- The git branch command lets you create, list, rename, and delete branches
- The default branch name in Git is master



Note:- If you perform git branch first time, you can't see any branch including master also. You can see when you commit something first time to repo. Otherwise, master is not visible

To Creating a new branch	→ git branch branch_name
To see current branch	→ git branch
To switch branches	→ git checkout branch
To create and switch branch at a time	→ git checkout -b branch_name
To rename a branch	→ git branch -m old_name new_name
To delete a branch	→ git branch -d branch_name
To Rename the current branch	→ git branch -M newBranchName

The -d option will delete the branch only if it has already been pushed and merged with the remote branch.

To delete a branch forcefully	→ git branch -D branch_name

Use -D instead if you want to force the branch to be deleted, even if it hasn't been pushed or merged yet. The branch is now deleted locally

Note:- If you create one file inside a branch from master. It will be present in all branches & master. But, whenever you are committing the file into a particular branch. It will not present

Feature Branch

- They are short lived branches.
- We can simply called ephemeral i.e. short lived objects
- we delete it after it is integrated to develop

MERGING

Getting the all commits from one branch to another branch. i.e. getting files and commits from one branch to other

Note:- First, which branch you need to merge, go inside that branch and perform that merging

Eg: so, I'm going into master i.e. git checkout master then you can do merge

  - git merge branch_name
- Cancelling the merge → git merge --abort

GIT REBASE

If you have 5 commits in master branch and only 1 commit in devops branch, to get all the commits from master branch to devops branch we can use rebase in git.

  - git rebase branch_name

CHERRY-PICK

- Cherry pick picks specific/particular commits from another branch and merges with the current branch
- We can pick through commitID

Eg:- If you have 5 commits in master branch and only 1 commit in devops branch, to get specific commit from master branch to devops branch. we can use cherry pick

- git cherry-pick commitID

MERGE CONFLICTS

- GIT makes merging super easy
- CONFLICTS generally arise when two people have changed the same lines in a file (or) if one developer deleted a file while another developer is working on the same file
- In this situation git cannot determine what is correct!

Let's understand in a simple way!

- cat > file : hi all → add & commit → git checkout -b branch1
- cat > file : 0987654 → add & commit → git checkout master
- cat >> file : sandeep Chikkala → add & commit (or) git commit -a

Now perform , git merge branch1

- We have to do manually to resolve our conflicts

How to Resolve merge conflicts ?

open file in VIM Editor and delete all the conflict dividers and save it!

add git to that file and commit it with the command

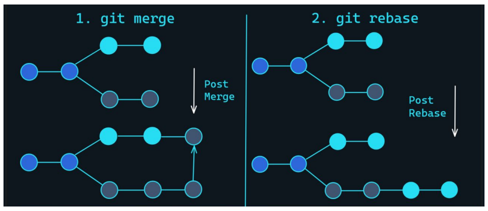
STASH

- Using the git stash command, developers can temporarily save changes made in the working directory.
- It allows them to quickly switch contexts when they are not quite ready to commit changes
- And it allows them to more easily switch between branches
- It saves changes in working and index areas and saves it to a different location and making a way for other important tasks
- It can be access only inside the branch
- Generally, the stash's meaning is "store something safely in a hidden place"

Apply the stash	→ git stash (or) git stash apply
Remove the stash	→ git stash clear
Check the stashes	→ git stash list
Remove the last stash	→ git stash pop
Remove particular stash	→ git stash drop stashID eg:(git stash drop stash@{0})

MERGE VS REBASE

- When there are changes on the main branch that you want to incorporate into your branch, you can either merge the changes in or rebase your branch from a different point
- Merge takes the changes from one branch and merges them into another branch in one merge commit
- Rebase adjusts the point at which a branch actually branched off (i.e. moves the branch to a new starting point from the base branch)
- Generally you'll use rebase when there are changes are made in main/master branch that you want to include in your branch.
- You'll use merge when there are changes in a branch that you want to put back into main
  - to merge : git merge branch_name
  - to rebase: git rebase branch_name
- Overall,
  - git merge combines branch changes with new merge commits
    git rebase moves branch changes on top, creating a linear history



If I delete/modify the branch in local, I want to see the changes in server → git fetch -p

GITHUB

- It is a web-based platform used for version control
- It simplifies the process of working with other people and makes it easy to collaborate on projects
- Team members can work on files and easily merge their changes in with the master branch of the project
- In github, default branch is 'main' branch
- If you want to push code from local to central we use github

Readme file: It is a text file, it tells about the particular repository what is the version and what actually the code does, how to use/see the repo, all information developers are write inside a readme

Blame: This command presents the developer details at each and every li ne

Commands

Linking the repo from local to central → git remote add origin url

push the code from local to central → git push -u origin branch_name

Note: without commit you can't push the file

Remove the repository → git remote rm origin

push multiple branches at a time → git push -u origin --all

Delete github branch from local → git push -u origin --delete github branch_name

See all branches (local & central) → git branch -a

see the repo whether it's linked to local/not → git remote -v

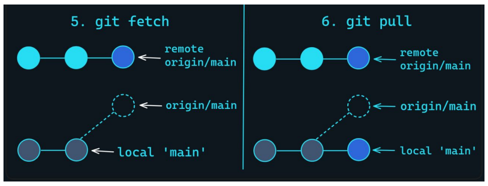
see the commits of remote repo → git log origin/branch_name

Checkout to central branches from local → git checkout origin/branch_name

git pull

git pull fetches and auto-merges remote changes

- From local to central we have to send our code use git pull
  - git pull origin branch_name (For Specific branch)
  - git pull
- git pull = git fetch + git merge
  - git fetch origin branch_name
  - git merge origin/branch_name



- fetch : Inside the central whether we have the changes/not. we will know through by git fetch

(or)

It will download remote commits to local but it will not merge. i.e. git fetch downloads remote changes without auto-merging

(or)

You can review remote changes before you merge

Pull Request:

- It is nothing but a merging in GitHub
- Using this pull request we can also resolve github merge-conflicts
- select 2 branches and compare and do merge
    - base branch - less files will present in that branch
    - compare branch - more files will present in that branch

Git Cloning:

- git clone is used to get the code from the github
- To clone a git repo we need to have a repository and also check our pwd
        - git clone repo_url
- now we just cloned the files in repo-A to repo-B
- But before cloning we need to add and commit our files

Git Fork:

A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project

(or)

Fork is used to get the other repositories to our account

Git Branch Protection rules

Using this feature

1. We can lock a release branch when we stop supporting this release
2. You can create branch protection rule to merge changes to main through PR

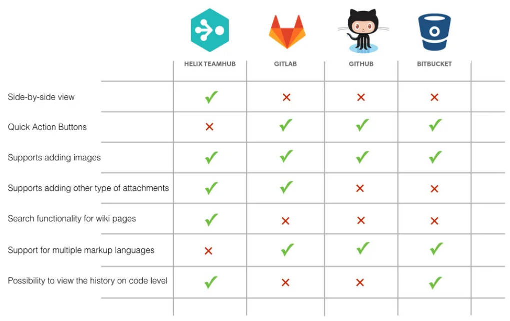
Advantages:

- Speed
- Simplicity
- Fully Distributed
- Excellent support for parallel development, support for hundreds of parallel branches
- Integrity

DisAdvantages:

- windows support issue
- Entire download of the project history may be impractical and consume more disk space if the project has long history

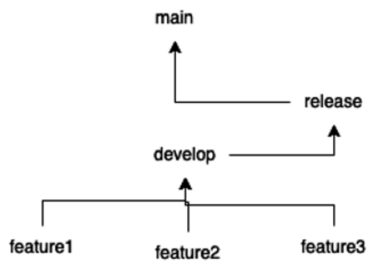
Comparison



	HELIX TEAMHUB	GITLAB	GITHUB	BITBUCKET
Side-by-side view	✓	✗	✗	✗
Quick Action Buttons	✗	✓	✓	✓
Supports adding images	✓	✓	✓	✓
Supports adding other type of attachments	✓	✓	✗	✗
Search functionality for wiki pages	✓	✗	✗	✗
Support for multiple markup languages	✗	✓	✓	✓
Possibility to view the history on code level	✓	✗	✗	✓

BRANCHING STRATEGY

Branching Strategy is differ from team to teams



- ☐ main → This is the final branch, this branch should contain clean and bug free code. that's why do not directly push changes to main
- ☐ feature → This branch should be used for adding new features
- ☐ develop → This branch is used for integrating features and after integrating features we will test it. If there are any bugs occur fix in this branch before merging to main
- So, every sprint we need to do release. For managing releases we're suggested to have release branch. So, release means the changes made by our development to production team
- So, from feature we integrate to develop feature is created from develop, and then after integrate back to develop
- Once you have features reads i.e. develop branch is ready. we create a release branch
- ☐ release → This branch used for managing release. This branch contains commits (or) features specific to that release. And we merge this release back to main
    - So, every time you have release you must create release branch like release 1, 2, 3, ...
- ☐ hotfix → we use this branch for fixing production defects. And this is created from release and make sure this fix is included into the next release. that should be taken by DevOps people

    eg: we are having defects in release 1 and we fix those defects using this hotfix branch

This is one of the common flows used in real-time

Trunk-Based Strategy

- Trunk means main branch
- we will not have so many branches in this approach
- we will just have like features
- we directly merge changed to main to features

It won't help in bigger teams. It is helpful only for less team i.e. 1 or 2 developers

Follow good naming conventions

- Naming conventions will help and it is easily communicate with people without explaining them what exactly you are doing
- Recommended branch name conventions
    - feature/sandy/APPS-4727

- Naming convention for release branch
    1. release-0.0.1 (Major.Minor.Patch) (semantic versioning)
    2. release-0.0.2
    3. release-0.1.0

Patch: There is a bug, if you want to fix it that is called patch release Eg: tier puncture patches etc..,

Minor: Adding 3 (or) 5 features

Major: A framework change means

Git Hooks/Scripts

We can execute custom scripts (or) specific operations

for eg: we wanna have a JIRA id in a every commit message

Advantage:

- When you go with this approach, we know very sure what is the commit, which task the commit is made
- If you go to JIRA It should all commits made particular tasks
- There are server side and client side hooks. Git supports both
- When you perform operations on server side, server side hooks comes into the picture

- [ ] By using the 'whatchanged' sub command, we can see what has changed over time.
    - git whatchanged --since='2 weeks ago'