GIT- Global Integration Tool

git --> version control tool used to keep track of versions of files

Types of Version Control System (VCS) :

1. Centralized: SVN

2. Distributed: GIT (SCM - source code management tool)

Commands:

sudo yum install git (For Redhat OS, amazon linux) -- To install git

sudo apt install git(for Ubuntu OS)

git --version → to check the version of git/ to verify whether git is installed or not

mkdir git_practice (Creating a directory for git)

cd git _practice

There are two types of repositories

Central repository - git init --bare

git init --bare → to initialize the central repository.

Local repository - git init

git init ---> create git repo / to initialize a git repository

to set username and email :

git config --global user.name "abc"

git config --global user.email abc@gmail.com → This will set the username and email for all the git repositories in the local VM/ developer's laptop

ls -a --> after git initialization, ls -a is used to make sure that .git is present

vi file1

add few lines and save

git status ---> shows whether files are in local repo or staging area or in git repo

git add filename --> to add given files from local workspace to staging area

ex: git add file1 fil2 -- to add file1 and file2

git add * → to add all the changes from local to staging

git commit -m "added new file" ---> to commit files from staging area to git repo. the versions are tracked

git log ---> shows the history of current branch(shows all the commits)

git log filename -- shows commits of given file(history of file)

git log -2 ---> lastest two versions on repo

git log filename -2 --->to check latest two versions of given file

git log --oneline → shows only commit IDs list.

git log --oneline | wc -l -→ to check number of commits

git log branchname: to list the commits of the given branch without checkout

git log --oneline brachname → to list only commit id of the given branch without checkout

---

ACTIVITY

Modify the same file1 again and add two more lines

vi file1

git add file1

git commit -m "modified file1"

git log

git checkout commit-id ---> used to switch to particluar

check content of file using cat file1

git checkout master ----> switch to latest version

cat file1

add more file file2

repeat above steps

Tag ---> name given set of versions of files. It's easy to remember in future and it indicates milestone of a project or stable commit.

git tag ---> to list tags

git tag tagname ---> used to tag latest commit

git tag tagname commitID -- used to tag a particular commit

git tag -d tagname ---> to delete tag

git checkout tagname ----> switch to tag

git checkout commitID --- to view the version in the given commitID

we can't modify tags once created But we can add multiple tags to a single commit

branch --> is for parallel development. Two people or two teams work on same piece of code for developing different set of features and they can integrate by merging.

For example, we released a product for 6.0 version and we might want to create a branch so that the development of 7.0 features can be kept separate from 6.0 bug fixes.

git branch ---> list branches

git branch branchname ---> create branch from checkedout branch

git branch new-branchname old-branch-name ---> it creates new branch from other branch, we need not checkout to that branch

git checkout -b branch-name existing-branch-name --> create new branch and checkout to new branch

git branch -d branchname ---> delete branch

git branch branch-name tagname ----> create branch from tag

git merge branchname ---> mentioned branch will get merged to checkedout branch

git merge master ----> it will merge latest commit from master to checkout branch

Difference between rebase and merge

Both merge and rebase will combine the changes of two branches but the difference is merge will create another commit called merge commit and the history of the commit ids will not change.

But in rebase, it will combine the changes along with commit history. It will alter the history of destination branch

Commit history of destination branch is preserved in merge but altered in rebase.

git rebase branchname

assignment 1:

1.will rebase creates another new commit ?

2.will rebase takes all the commits and adds it to the tip of the current or it will modify the history and inserts the commitIDS according timelines ?

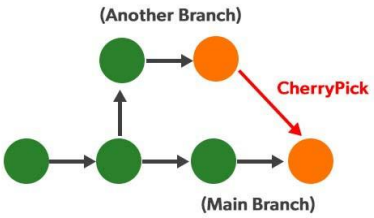
How do you merge particular commit to branch?

Using git cherry-pick

git cherry-pick used to bring particular change from one branch to another

git cherry-pick commitID3 - pick commitID3 and combine the changes to current branch

git cherry-pick commitID3 commitID2 commitID45 - pick multiple commits



What is merge conflict?

If same piece of code on the same file got modified differently on two different branches. When we try to merge these two branches. We get merge conflict.

I don't know whose changes I should consider to merge. So I will contact developers who modified file on branch1 and branch2.

They will discuss and decide which change should go to merge and I will take that change, continue with merge by committing to that file.

git stash ---> avoid commiting files with incomplete changes

When I am working on one branch, in between if i get any critical issue/ bug which needs to be fixed on another branch.

As I have incomplete work which I don't need to commit on current branch

Before I switch to another branch, I need to stash files from current branch, it stores on temporary area.

After come back to current, I can get files back using git stash pop and continue with usual work. so we can avoid committing files on the wrong branches.

Ex: I am working on branch5 and I have finished only half of the work. If there is a bug in master , emergency issue which needs to be resolved ASAP. In this case we cannot commit or we cannot delete the changes. We have to use git stash.

touch t1 t2 t3

git add *

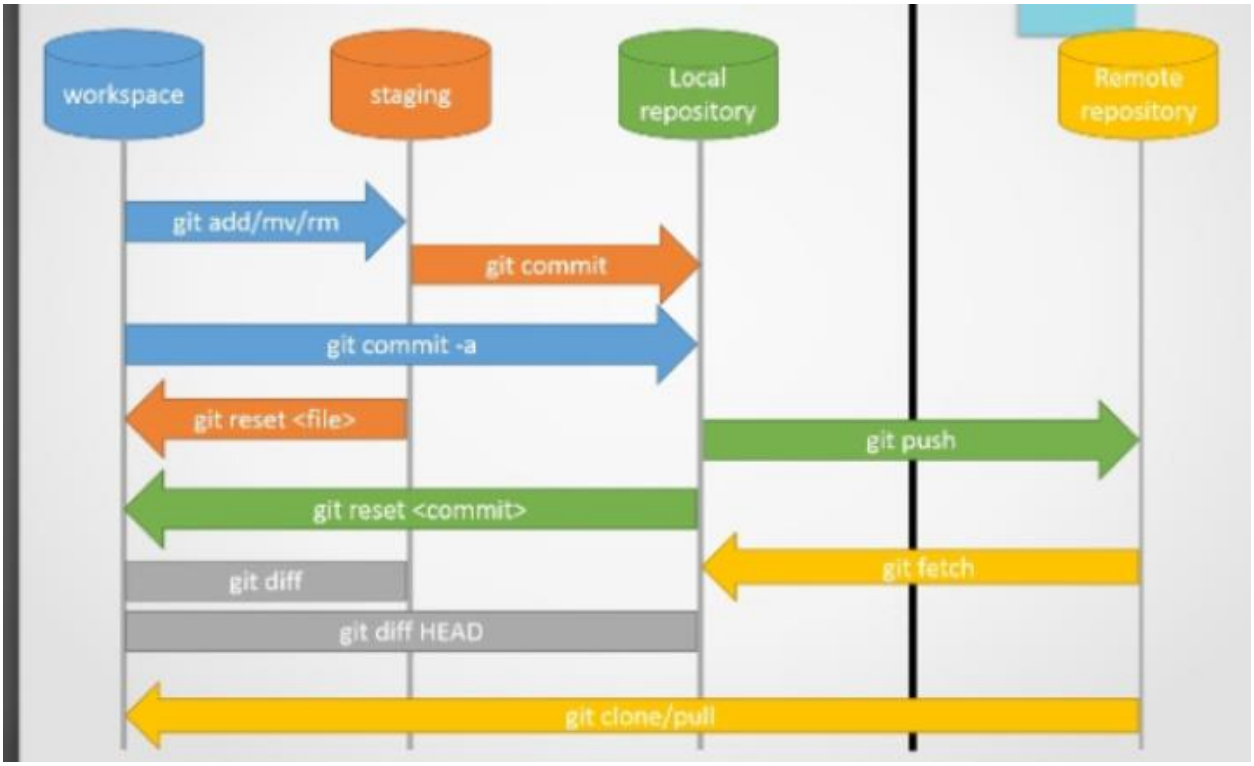
git stash ---> moving files to temporary area

git checkout another-branch

git checkout current-branch

git stash pop ---> bring files back

GIT ARCHITECTURE/ GIT FLOW



workspace ---> local area where we modify project related files

staging area or index area ---> it is intermediate area where we save changes

git repo ---> commit changes and commit ID will get created

---

github is a tool to host our central repository.

We have 3 types of settings in repo

public - anybody can copy

ex: https://github.com/DeekshithSN/CICD_Java_gradle_application

private - Only the owner can see the code

Internal - Company employees can access. other cannot access

git clone --> used to bring whole repo to local workspace for the first time

git clone /home/ec2-user/central_repo or git clone <url>

If It is a private account. it will ask for username and password

Earlier(5 years ago): the password for github account used to work

Now we access using secret token called **PAT**

To generate secret token:

top right corner --> click on settings --> left side , go to developer settings , PAT (personal access token), create new token , give the permissions and generate --> copy paste the token

git push ---> used to push changes from local workspace to central repo

git push /home/ec2-user/central_repo

git pull --> bring changes from central repo to local workspace and merges automatically

git pull /home/ec2-user/central_repo

git fetch --> will bring changes from central repo to local workspace and stores it in separate branch. You can review it and merge it to local workspace if required

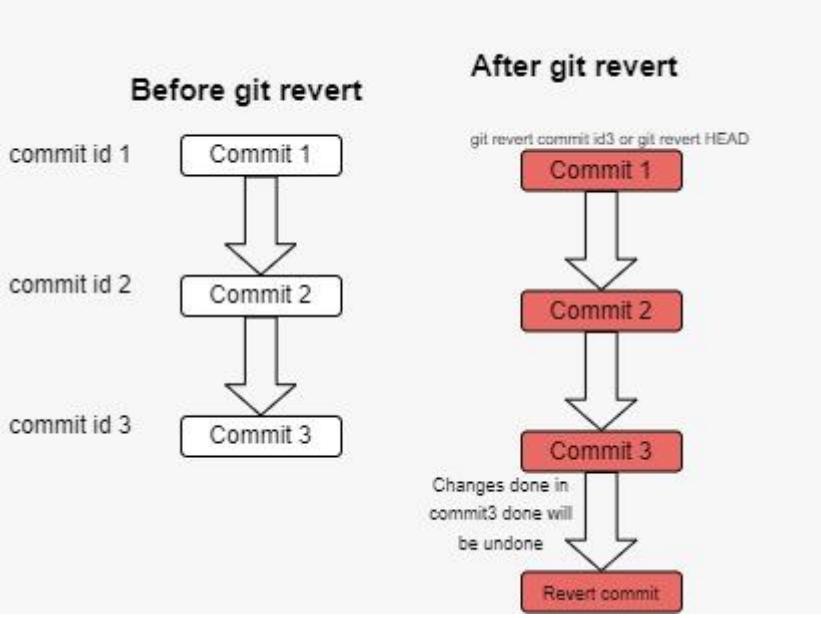
git fetch /home/ec2-user/central_repo

Pull request - its a request raised from a developer to update his changes to a central repo or from dev branch to master.

it will check for merge conflicts. Reviewer can review the changes and can choose to merge or cancel the merge

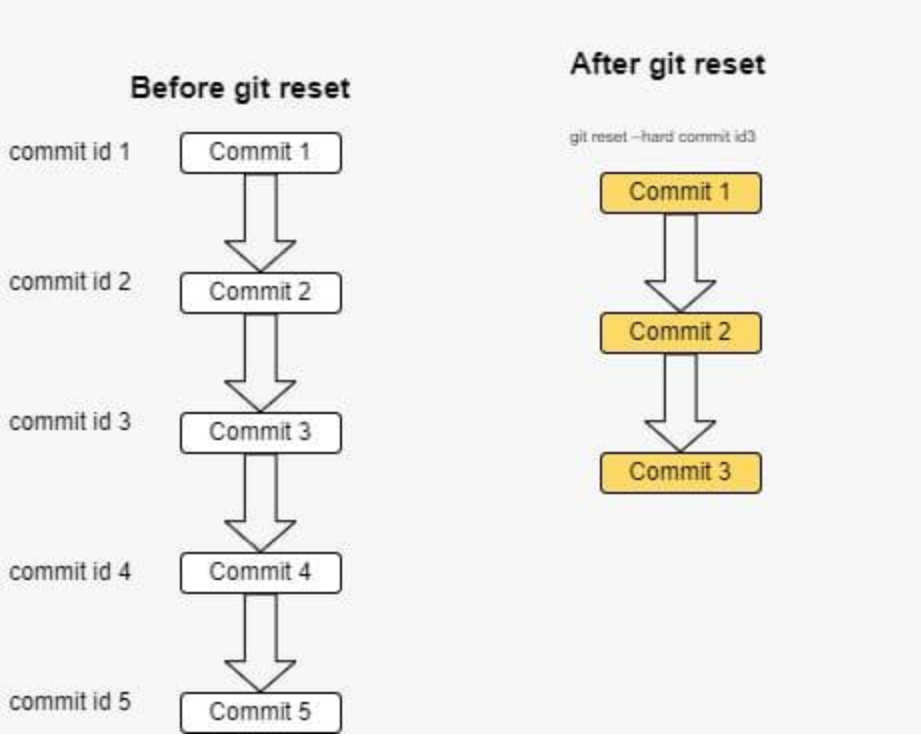
Revert –

git revert commitID --> this comand will undo the changes of the given commitID.
+ It will create a new commitID for reverting



Reset -

git reset --hard commitID --> reset will take me to the given ID and deletes the changes done in the subsequent commitIDs and erases the history



Difference Table

git reset	git revert
Unstages a file and bring our changes back to the working directory.	Removes the commits from the remote repository.
Used in local repository.	Used in the remote repository.
Alters the existing commit history.	Adds a new commit to the existing commit history.
Discards the uncommitted changes.	Rollbacks the changes which we have committed.
Can be used to manipulate commits or files.	Does not manipulate your commits or files.