
git

SCM (Source Code Management)

- Storage (Any code)
  - Developers (Code)
  - Testing team (Test)
  - DevOps (Scripts)
- Different people from different teams can store simultaneously (Save all changes separately)
- Pipeline b/w off shore & on shore teams
- Helps in achieving team work.
- Track changes (Minute/Minuet(small) level)

SCM Tools

- Git (Most advanced tool)
- SVN
- Perforce
- Clearcase

Important Terminology

- Repository/Depot
- Server
- Work space/Work dir/Work tree
- Branch/Trunk/Code line
- Commit/Check-in
- Version/Version-ID/Commit-ID
- Tag

Important Terminology

- Repository
  - Storage (Folder)
- Server (EC2 Instance)
  - Stores all repos

Important Terminology

- Branch
  - Product is same, so one repo. But different tasks/ideas.
  - Each task/idea has one separate branch
  - Finally merge(code) all branches
  - For Parallel development/implement new ideas.
  - Can create any no of branches
  - Changes are personal to that particular branch
  - Can put files only in branches (not in repo directly)
  - Default branch is “Master”
  - Files created in workspace will be visible in any of the branch workspace until you commit. Once you commit, then that file belongs to that particular branch.

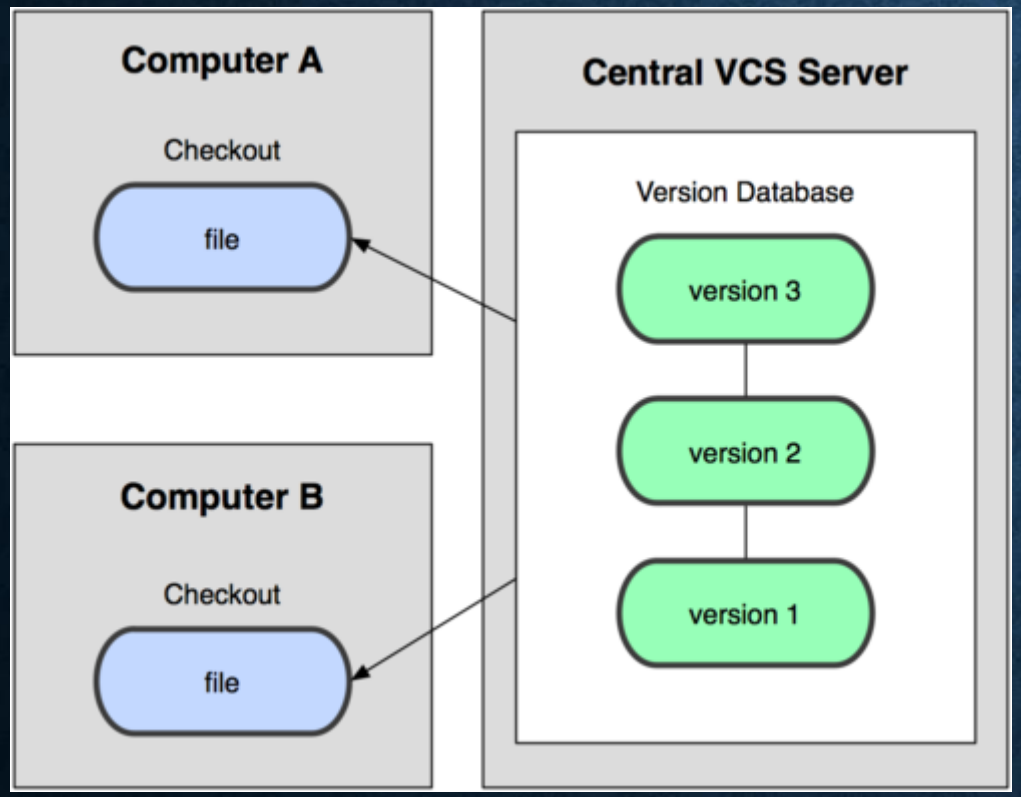
Important Terminology

- Work space/Working Directory
  - Where you work
  - Where you see files physically & do modifications
- Version/Version-ID/Commit-ID
  - Reference to identify each change and who did that change
- Tag
  - Meaningful name (we can not remember Commit-ID)

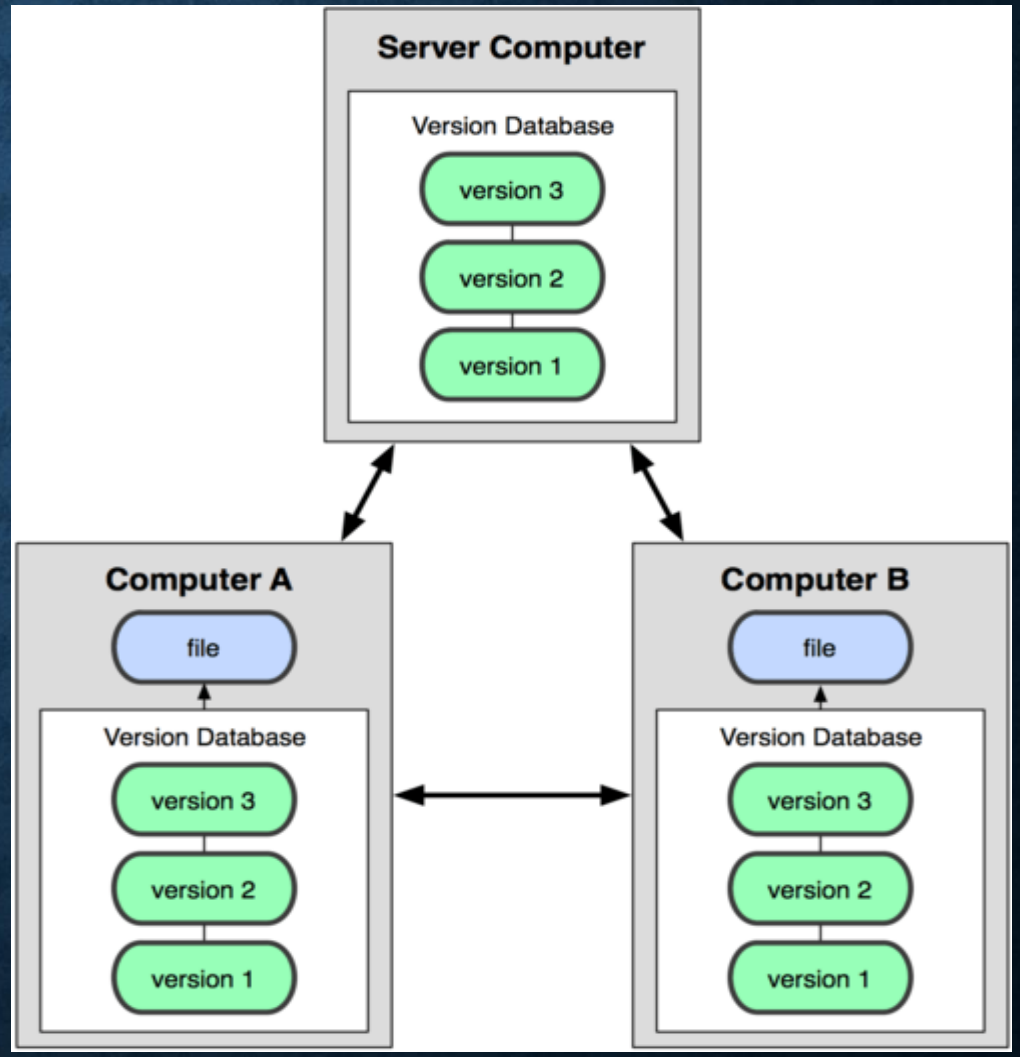
Why only Git?

- Speed
  - Snapshots concept
- Parallel branching
  - Multiple branches at a time unlike other SCM tools
- Fully Distributed
  - Backup copy is available in multiple locations.
  - No need Internet connection. So no network latency.
  - No need central server separately
  - Each work space will have its own repo internally
  - Can create any no of branches
  - Can share code without using central repo
  - That's why we call GIT as DVCS (Distributed Version Control System)

CVCS vs DVCS



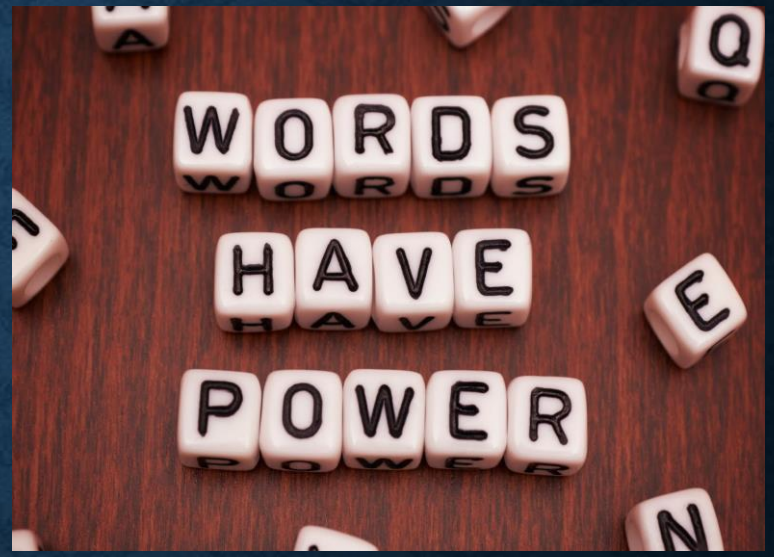
SVN



Git

Important Terminology

- Snapshots
  - Get any previous version (Backup)
  - Represents some data of particular time
  - Stores the changes(appended data) only. Not whole copy.



- Commit
  - Store changes in repo (Will get commit ID)
  - 40 Alpha-numeric characters
  - Concept - Checksum (It's a tool in Linux generates binary value equal to data present in file)
  - Even if you change one dot, Commit-ID will be changed
  - Track the changes

Git stages

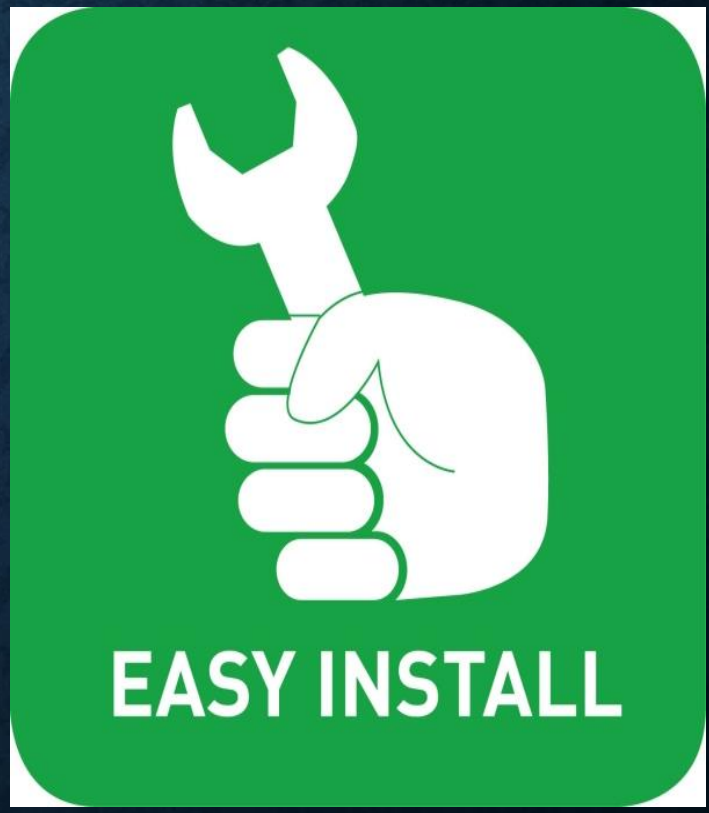
- Work space
  - Physically see file & Modify
- Staging/Indexing area
  - Buffer area
  - Takes snapshot
- Repository (Local)
  - Store changes locally
- Repository (Central)
  - Store changes Centrally

Types of Repositories

- Bare Repositories (Central)
  - Store & share only
  - All central repositories are bare repositories
- Non – Bare Repositories (Local)
  - Where you can modify the files
  - All local repositories are non-bare repositories

Install & Configure Git

- Launch 2 EC2 machines in 2 regions(Mumbai & London). (Run below commands in both machines)
  - sudo su
  - yum update -y
  - yum install git -y
  - git --version
  - git config --global user.name "Sai/Hari"
  - git config --global user.email "sai/hari6cs@gmail.com"
  - git config --list
  - date
  - date +%T -s “20:58:00”



```
echo "# test" >> README.md
    echo "# test" >> README.md
```

Git Hub (Central repository)

- Go to www.github.com & create account.
- Create a new repository(centralgit) & choose public.

Git commands(in Mumbai EC2 machine)

- Create directory & go inside that
  - mkdir mumbaigit, cd mumbaigit
  - git init (to initialize git)
- Create new file, see status, put in staging area &commit into local repo
  - touch myfile (put some content)
  - git status
  - git add .
  - git commit -m “1st commit from mumbai”
  - git log
  - git show <commit-id>

Git commands(in Mumbai EC2 machine)

- git remote add origin <centralgit repo url>
- git push -u origin master (give github credentials)
  - Verify in github. Can see the file.
- (add some content to the file and repeat the same process. But need not to add again to central repo)
- (observe the differences b/w untracked file & modified file)
- git log
- git show <commit-ID>

Git commands(in London EC2 machine)

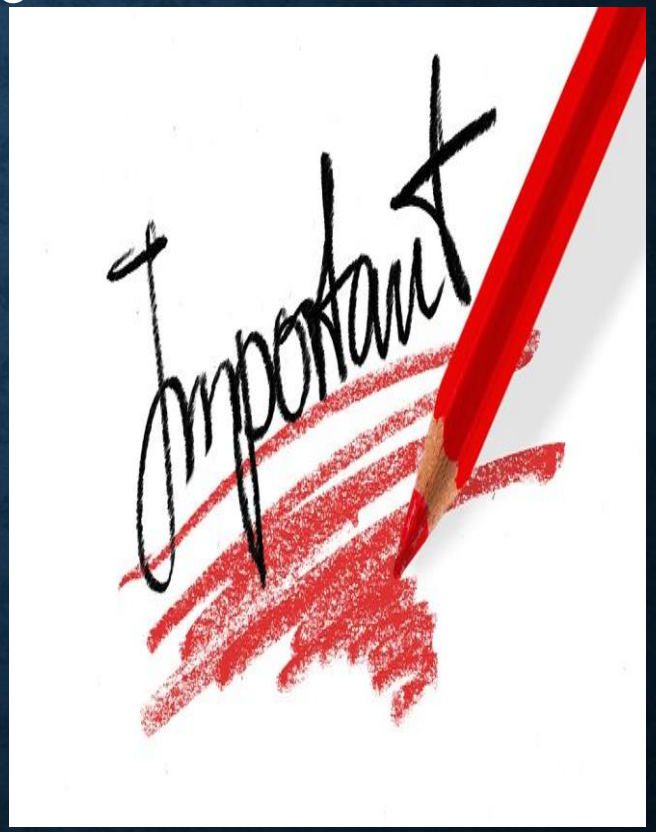
- Create directory & go inside that
  - mkdir londongit
  - git init (to initialize git)
  - git remote add origin <centralgit repo url>
  - git pull origin master
  - git log
  - git show <commit-ID>
  - cat >> myfile (append with some content)
  - git status
  - git add .
  - git commit -m "1st commit from londongit"
  - git push -u origin master

Git commands(in Mumbai EC2 machine)

- To pull latest changes
  - git pull origin master
- Again append with some content, add, commit & push.
  - cat >> myfile (append with some content)
  - git status
  - git add .
  - git commit -m “Any commit msg”
  - git push origin master

Git commands

- Observer the difference b/w modified file & untracked file
- See commit ID's
  - git log
- Local repository will be in hidden mode
- Push changes to central repo
  - git push origin master
- Pull changes from central repo
  - git pull origin master
- Observe the current state
  - git status
- Like this we can push/pull to/from central repo

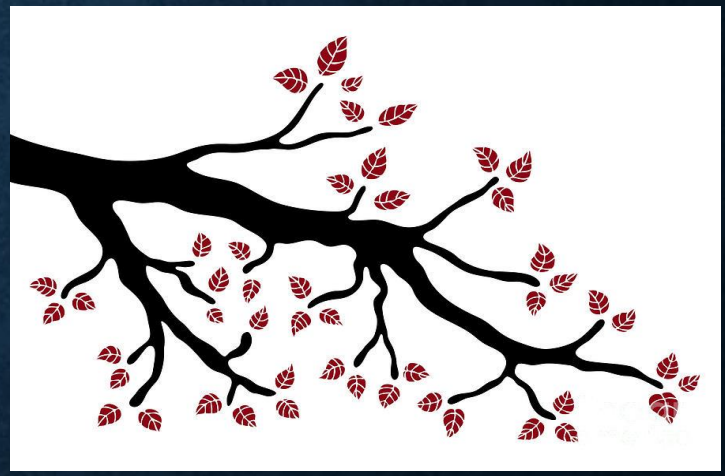


Git commands

- To ignore the file while committing (Retrospective effect is there)
  - .gitignore
  - Steps:
    - Create .gitignore file and give pattern matching (eg: *.class)
    - Add and commit .gitignore file
    - Create some class files and java files for testing and add them by running “git add .”
- Git log options
  - git log
  - git log -1
  - git log --oneline
- To pic commit based on commit msg
  - git log --grep “any word of commit msg”
- To see the content of particular commit
  - git show <commit-ID>

Branching

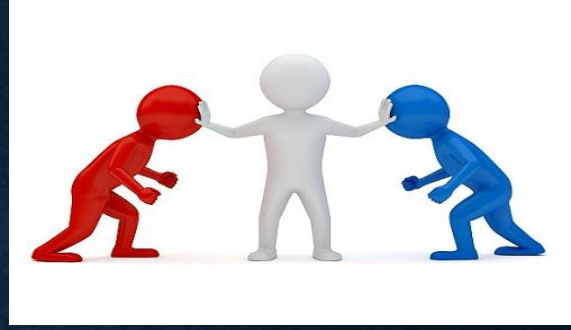
- To see list of available branches (must have some thing)
  - git branch
- Create a new branch
  - git branch <branch name>
- To switch branch
  - git checkout <in which branch you want to go>
- Verify the content in both branches
- Add content in both branches
- Files are not personal to branch until you commit.



Branching (Merge)

- You can not merge branches of different repositories
- To merge branches (Pulling mechanism)
  - git merge <branch name>
- Verify the merge
  - git log
- Push to central repo
  - git push origin master

Git Conflict



- When same file having different content in different branches, if you do merge, conflict occurs.(Resolve conflict, then add and commit)
- Conflict occurs when
  - Merging branches

Git Stashing

- It removes content inside file from working directory and puts in stashing store and gives clean working directory so that we can start new work freshly.
- Later on you can bring back that stashed items to working directory and can resume your work on that file.
- To stash an item (only applies to modified files. Not new files)
  - git stash

Git Stash

- To see stashed items list
  - git stash list
- To apply stashed items
  - git stash apply stash@{number}
  - Then you can add & commit
- To clear the stash items
  - git stash clear

Steps to be followed

- Create any blank file add and commit
- Put some content ,then run “git stash” (repeat twice)
- Go to stash repo for verification.
- Get from stash repo and then add and commit (if conflict occurs, resolve conflict)

Resetting changes (Before commit)

- To reset from staging area
  - git reset <file name>
  - git reset .
- To reset the changes from both staging area and working directory at a time(not individually)
  - git reset --hard

Reverting changes (After commit)

- To revert the changes
  - git revert <commit-id>
- Revert will create a new commit-id (all changes will be tracked)
- You need not to add and commit again. Automatically new commit-ID will be generated.

Removing files

- To remove files
  - git rm <file name>

    (Then commit (no need to add))

    (Gets deleted from that particular branch)

Delete all untracked files

- git clean -n (dry run)
- git clean -f

Tag

- To apply tag
  - git tag -a <tag name> -m <tag message> <commit-id>
- To see the list of tags
  - git tag
- To see particular commit content by using tag
  - git show <tag name>
- To delete a tag
  - git tag -d <tag name>

echo "# test" >> README.md
echo "# test" >> README.md

Git Hub (Clone)

- Go to www.github.com
- Choose an existing repository.
- Run below command in local machine's current directory
  - git clone <url of github repo>
- Go inside that repo.
- Then onwards, we can push & pull……..

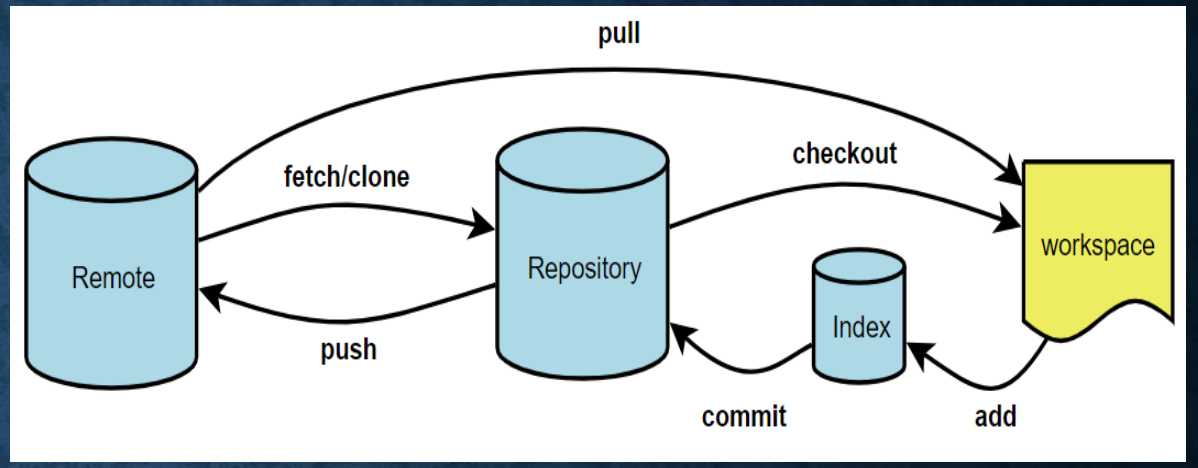
echo "# test" >> README.md
echo "# test" >> README.md

Git Hub (branches & merging)

- Create new branch form master(newbranch)
- Switch to newbranch, create an new file & commit in newbranch.
- Observe the differences b/w master branch and newbranch
- To merge branches in git hub itself, click on
  - New pull request
  - Create pull request
  - Merge pull request
  - Confirm merge
- Go to master branch. You can find new file that u created in newbranch

echo "# test" >> README.md
echo "# test" >> README.md

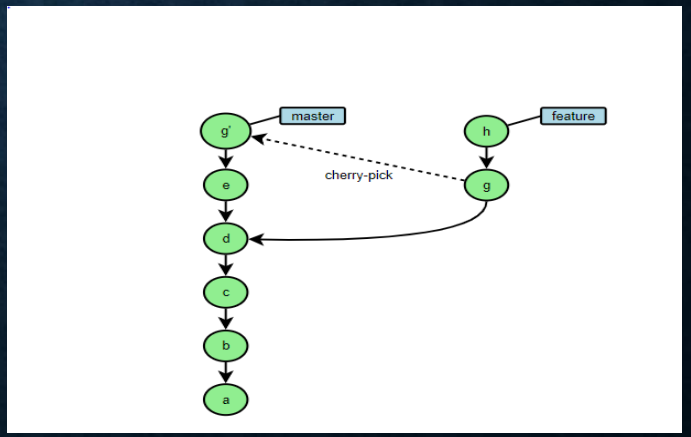
Git Commands



- Pull vs Fetch
  - Pull = Fetch + Merge
  - Fetch = Download from central repo
  - Merge = Installed the downloaded one
  - Pull = Download + Install
- Run git fetch(verify content in file in workspace & commits. There won't be any new change)

  fetch : Download objects and refs from another repository

  pull : Fetch from and integrate with another repository or a local branch
- git diff master origin/master (before pull, if want to verify)
- git push -u --all (to push all branches to central repo)
- git branch -D new (to delete branch(be in other branch))
- git push origin newbranch (to push to newbranch)
- git cherry-pick <commit-id> (to merge particular commit)



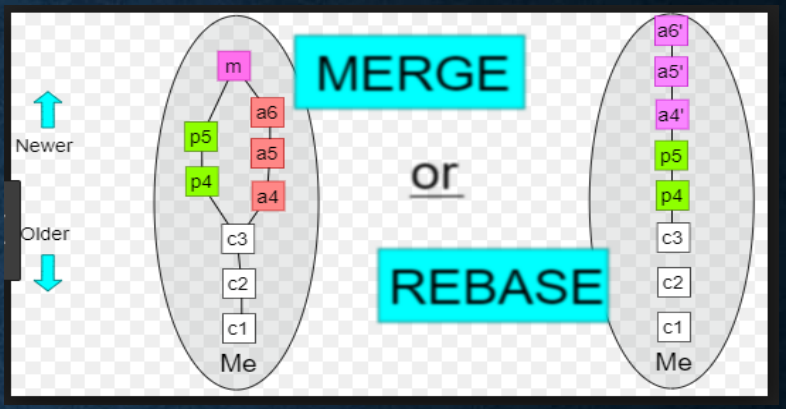
echo "# test" >> README.md
echo "# test" >> README.md

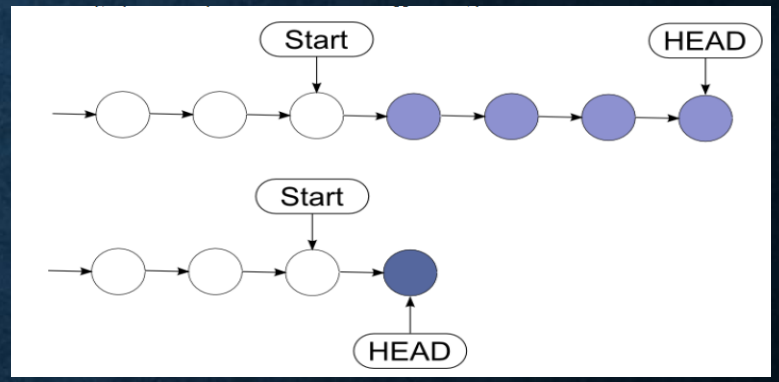
Git Commands

- Git hooks (Web hooks) : To set permissions & configure email notifications
- Git merge vs rebase

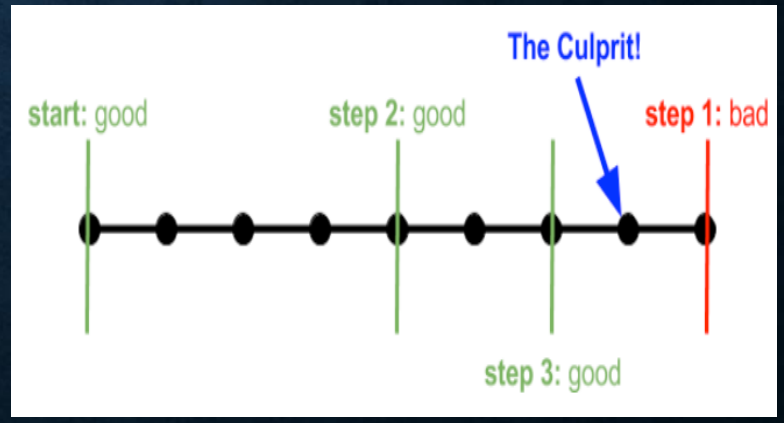
  merge : Join two or more development histories together

  rebase : Reapply commits on top of another base tip





- Git squash : To move the multiple commits into its parent so that you end with one commit. If you repeat this process multiple times, you can reduce *n* commits to a single one.
- Git Bisect : **git bisect** is a tool that allows you to find a bad commit. you don't have to trace down the bad commit by hand; **git-bisect** will do that for you.



git remote remove origin