1/03/2024

chapter 2.

Intro. to Git

Q: What is Git?

→ Git = Global Information Tracker
= Distributed VCS.

→ VCS = Version Control System
||
SCM = Source code mgmt
~~SCM = Software Configura~~

Q. What is the need of VCS?

⇒ VCS stands for Version Control System

(i) is an essential software tool that manages changes to the project codebase over time.

(ii) maintaing multiple versions manually is very complex activity.

(iii) Every changes should be tracked like
who did the change
when he did the change
which changes he did, etc
and all the changed should be maintained.

(iv) overwriting of the code should not ~~be~~ happen.

v) multiple developers work in a collaborative way & share their code.

vi) parallel development must be required.

vii) Eg - Dev A, Dev B - asked to develop a project & client asked to re-code structure the project ~~but~~ several times but atlast asked for 1st version. Expand it.

Q. Difference b/w CVCS and DVCS.
or Types of VCS

Q. Basic terminology of VCS :-

→ working Directory
The folder where we write, create and modify changes in files.
Here version control is not applicable.

→ Repository
here we store files & metadata (vvi)
here version control is applicable

→ Commit
The process of sending files from working directory to the repository.

→ Checkout
Vice versa of commit

Q. What are the benefits of VCS?

Ans i) we can maintain different versions and can choose any version based on client requirement.

ii) With every version, we can maintain metadata like ~~commit~~
commit message
who did changes
when he did the change
what changes he did

(iii) Developers can share the code among themselves.

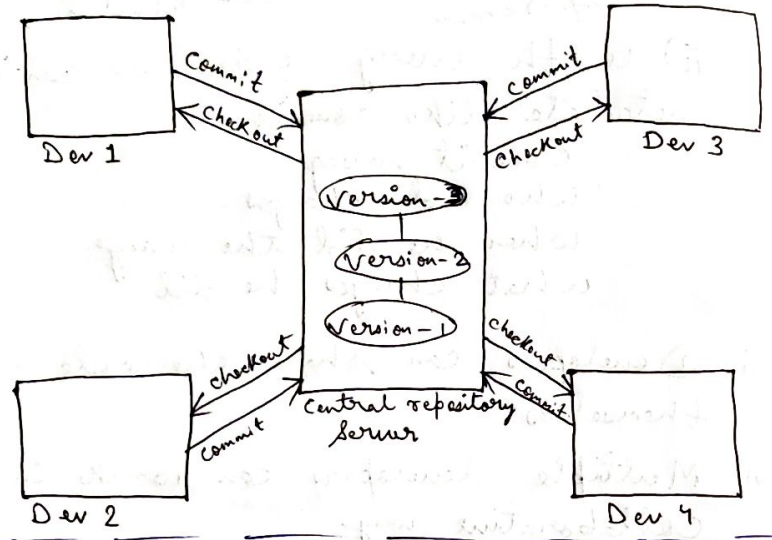
(iv) Multiple developers can work in a collaborative way

(v) Parallel development is possible.

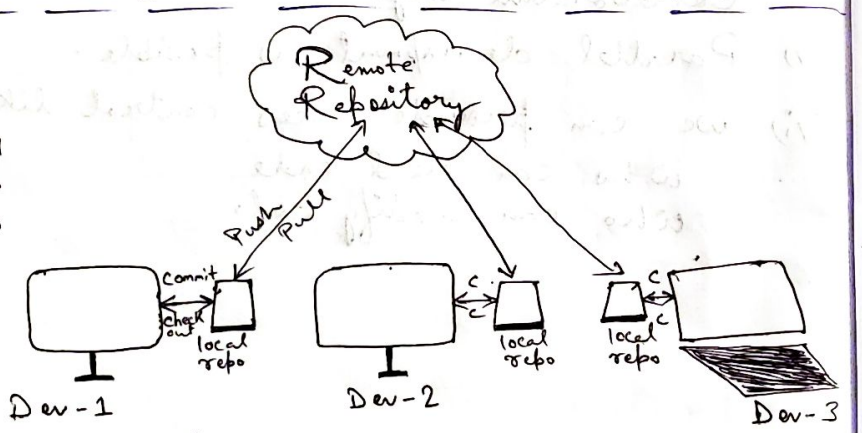
(vi) we can provide access control like
who can read code
who can modify code

Q1

Centralized VCS



Eg:- CVS, SVN, Perforce



Eg:- GIT, Fossil, Mercurial

Distributed VCS

⇒ Explain the above diagrams.

⇒ Explain the problems with CVCS.

2/03/24

Q. Difference b/w Centralized VCS and Distributed VCS.

CVCS	DVCS
* A VCS where the code repository is stored on a central server. Developer check out code from this central location and commit changes directly to it.	* A VCS where every developer has a copy of the code repository on their local machine. Changes are made locally and then pushed to a central repository.
* Requires a constant connection to the central server for most operation	* ~~Opera~~ Most operations are performed ~~online~~ offline, as developers have a local copy of the repository. Synchronization with the central repository is required only when pushing or pulling changes.
* Due to the dependency on network connection, operations will becomes slow, which causes performance issues.	* Dependence on network is very less, it doesn't affects the performance.
* Organization of central repository is very complex if number of developers and files increases.	* It doesn't create any problem if number of developers increases.
* E.g:- SVN (Subversion)	* GIT
* work Space - commit → Repository	* One repository --- push → one repo ← pull --- other repo

Features and Architecture of Git

History of Git

→ by Linus Torvalds in 2005

→ Early years of Linux Kernel Maintenance (1991 - 2002)

→ Bitkeeper usage (2002):
The Linux Kernel project started using a private DVCS called Bitkeeper.

→ Creation of Git (2005):
After the relationship with Bitkeeper broke down, Linus Torvalds and his team created Git.

→ Goal of Git:
→ speed, simplicity, full distribution, efficiency with large projects like Linux.

What is Git?

→ a distributed version control system tool.

→ most of the people abbreviate GIT as "Global Information Tracker".

→ developed by Linus Torvalds (Finnish), who also developed Linux Kernel.

→ Most companies use Git, these days.

Features of Git

1) Distributed:-
Git is developed based on Distributed VCS architecture, because of which it has several advantages like -
→ Every developer has his own local copy of repo. All operations can be done locally. Hence local and remote repo need not be connected always.
→ most of the operations can be performed locally so performance is high.
→ There is no single point failure as every dev has his own repo.
→ It enables parallel development.

2) Staging Area :-

→ A concept in git where changes are stored before committing (logical / virtual)

→ working directory ⇒ Staging area ⇒ local repo

→ we cannot commit the files of working directory directly. First we have to add to the staging area and then we have to commit.

→ This staging area helps to cross check the changes before commit.

→ This type of layer is not available in other VCS.

→ Git stores files in repository in hash form, which saves space.

→ Internally, git uses snapshots mechanism to store files in repository. All these conversion and taking snapshots happen in staging area before commit.

→ E.g.:- If a sample repository takes around 12GB in SVN, whereas in git it hardly takes 420 MB.
Hence staging area is the most important strength of the git

3) Branching and Merging :-

→ we can create and work on multiple branches simultaneously and all these branches are isolated from each other.

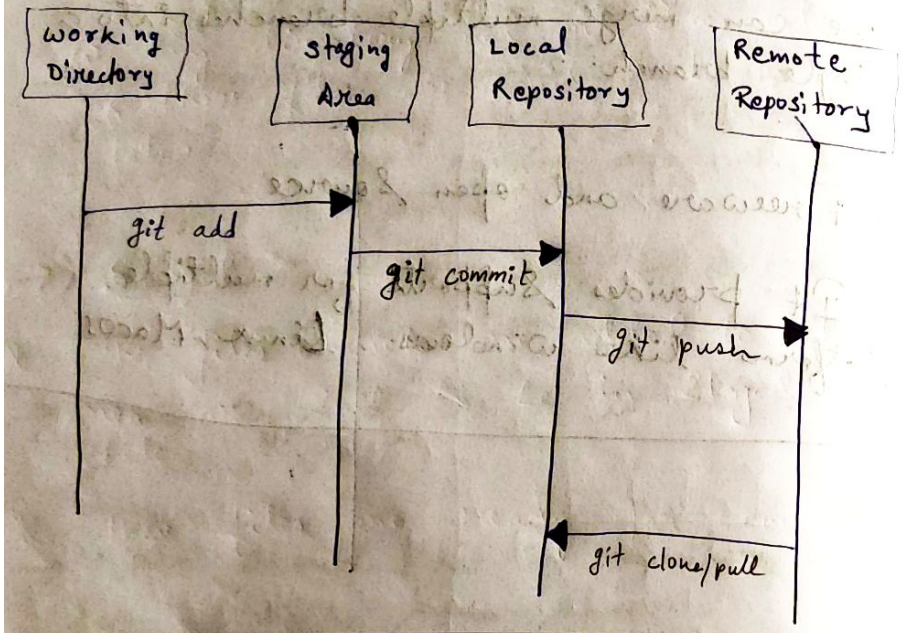
→ we can merge multiple branches into a single branch.

4) Freeware and open source

5) It provides supports for multiple platforms like windows, Linux, MacOS

3/04/24

Git Architecture



Types of Repository in Git

① Local Repository
② Remote Repository.

Local repository

→ The repository which resides ~~on the~~ locally in the developer's machine is called local repo.

→ all checkout and commit operations are performed locally by the developer.

→ In Git, Commit is a two-step process:-
first, add files to staging area by using git add command.
Then it needs to be committed to the local repository using git commit command.

→ commit is applicable only for staging area files but not for working directory files.

Remote Repository

⇒ If the developer wants to share his work to the peer developers, then he has to push his local repo to remote repository by using git push command.

⇒ ~~Commit is applicable~~

⇒ Remote repository contains total project code which is accessible by all developers.

⇒ If a developer wants to make a remote repo, a local repo, git clone command can be used.

⇒ A dev can get updates from the remote repo to the local repo by using git pull command.

Life Cycle of file in GIT

Every file in git is in one of the following state.

1) Untracked :-

The files which are newly created in working directory and git does not aware of these files are said to be in untracked state.

2) Staged :-

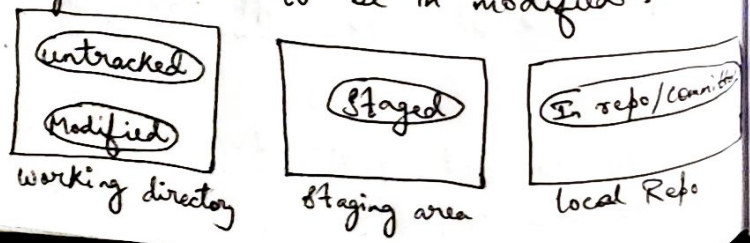
* The files which are added to staging area are said to be in Staged state
* These files are ready for commit.

3) In Repository / committed

Any file which is committed is said to be in Repository / Committed state.

4) Modified :

Any file which is already tracked by git, but it is modified in working directory is said to be in modified state



Git Commands used in Various Situations :

* Start a working area :
  - → clone : clone a repo into new directory
  - → init : Convert working directory into empty git repo.

* work on the current change :
  - → add : Add files to the Staging area.
  - → mv : move or rename files.
  - → rm : remove files or directories.

* History and State :
  - → status : Show the working tree status.
  - → log : Show commit logs.
  - → diff : Show changes between commits.
  - → ~~bisect~~ :
  - → Show : This command gives information about a specific commit, including its metadata and the changes made by that commit.
  - → grep : used to search a string or pattern in the contents of files.
    global regular expression print (grep)
  - → bisect : use binary search to find the commit that introduced a bug.

* grow, mark, and tweak

→ branch : lists, create or delete branches

→ commit : record changes from staging area to local repo.

→ merge : join two or more development histories together with all commit messages of both branches.

```
git checkout ~~branchName~~ main
git merge branchName
```

⇒ goes into main branch

⇒ merge the main branch into a feature branch

```
git checkout feature
git merge main
```

→ rebase : Same as merge but it keeps the commit history clean

```
git checkout feature
git merge main
```

→ reset : reset current HEAD to the specified state.
used to move the repository back to a previous commit.

```
git reset techwelve
```

* Collaborate

→ fetch : used to download changes (commits, branches, etc) from another repositories without automatically applying those changes to current local repository.

→ ~~pull~~

→ push : used to send files from local repo to remote repo.

→ pull : used to bring changes from remote repo and automatically apply them into local repo.

4/03/024

Git Initial Setup.

⇒ Before first commit, we have to configure username and emailid, so that git can use this information in the commit records.
We can perform these configurations with the following commands:-

⇒
```
git config --global user.email "abc@gmail.com"
git config --global user.name "Falna"
```

git config --list ⇒ gives details of configured user.

Remote

⇒ A remote in git is a common repo that all team members use to exchange their changes. It can be hosted on services like Github, GitLab or Private server.
When we done a repository, git automatically creates a remote named "origin" that points to the original repository.

⇒ `git remote` ⇒ gives list of remotes.

⇒ To add a remote, use the below command:

```
git remote add <remoteName> <url>
```

Eg- git remote add upstream https://github.com/upld.git

Branching

⇒ Branching is like creating a copy of the main (original) repository. Each branch is isolated and does not affect original repository. After the purpose of the new branch is fulfilled, it is merged into main/original repo.

git branch	→ lists all branches.
git branch branchName	→ creates new branch
git checkout name	→ switch ~~goto~~ to "name" branch
git branch -d name	→ deletes "name" branch if it is merged & no longer needed.
git branch -D name	→ deletes "name" branch forcefully.

6/03/24 Topic 13: (master and HEAD)

Git References

→ For most of the commands (like git log, git diff etc), we have to provide commit id (or Hash value) as argument.

→ But remembering commit id is very difficult, even 7 characters also. ☹

→ Git provides some sample names for these commit ids. We can use these names directly. These are just pointers to commit ids. These sample names are called references or refs.

→ References are stored in `.git/refs` directory as text files.

→ There are multiple types of references like HEADs, branches, remote ~~etc~~, tags etc.

MASTER

→ master is the name of the branch.

→ It references (pointer) to the last commit id. Hence, whenever we want to use last commit id, simply we can use "master".

→ git show master
git show 49aa8d7 } both will give same result.

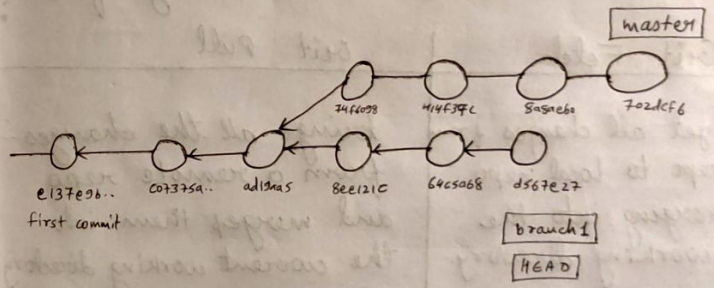
→ All branches ~~or names~~ files are stored inside `.git/refs/heads` folder.

HEAD

Head only points to a branch name that has recent commit (not a commit id)

→ HEAD only points to a branch name where recent commit has taken place.

→ If two branches have recent commits, then it points to both branches.



DETACHED HEAD

When head is directly pointing towards a particular ~~any~~ commit. How to do this?

Ans git checkout commit id

Eg- git checkout c0737sa

Advantage
Time Travel

Todo

① Create one detached head & go to a commitid & do some more commits there, those commits will not be associated with any branch. You can ~~do e~~ delete or prune them by changing branch

If you want to save any commit from detached head timeline:
- go to lastest commit in detach timeline
- make a branch here (git branch keepAll).
- boom, all commits are now associated with KeepAll branch.

Q. Difference b/w git pull and git fetch?

Ans.

Git Fetch	Git Pull
→ used to get all changes from remote repo to local repo without merging into the current working directory.	brings all the changes from a remote repo and merges them into the current working directory.
→ repo data is updated in the .git folder	The working directory is updated directly.
→ No possibility of merge conflict.	merge conflicts are possible if remot and local repo have done changes at same place.
→ command: git fetch branchName	command: git pull branchName
→ Review of commits and changes can be done	updates the changes to the local repo immediately.