Will start at 9:10PM

Agenda

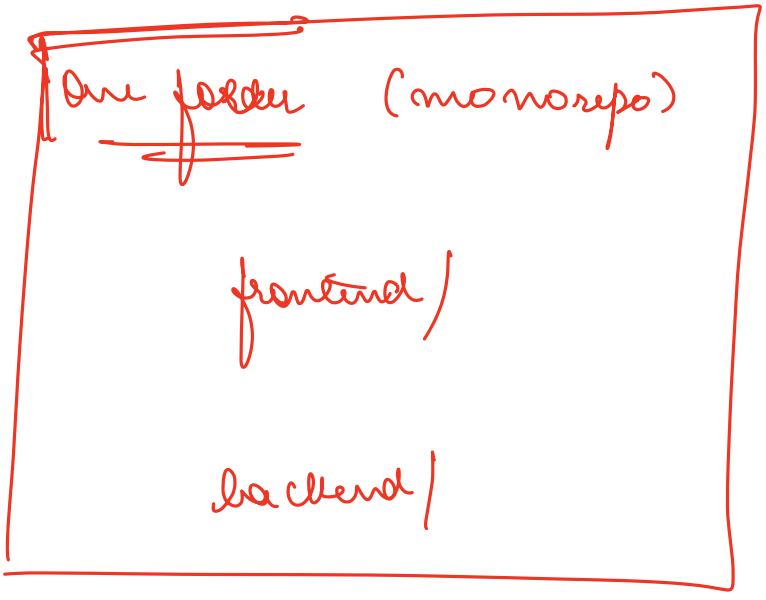
→ Git

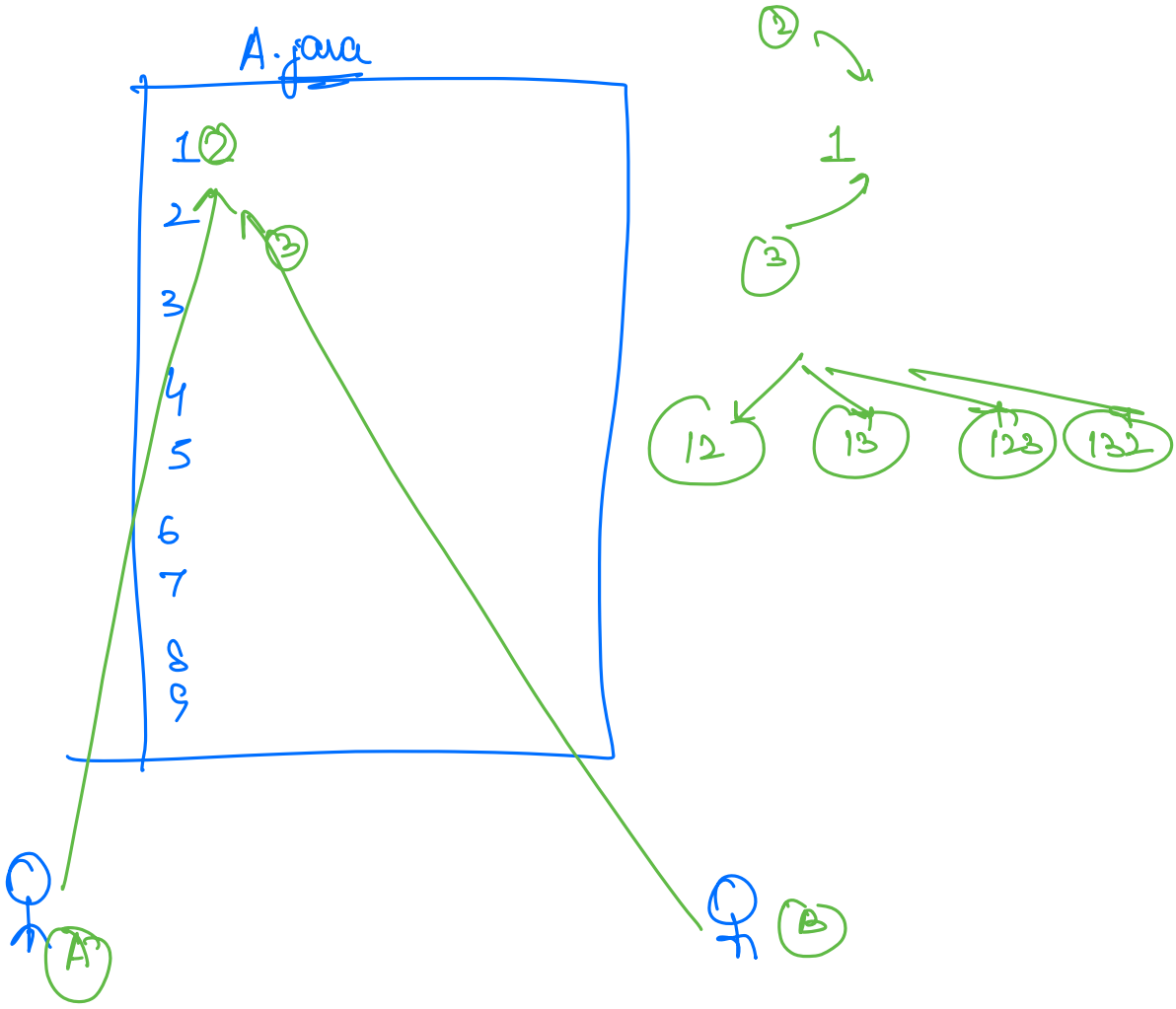
1. VCS and why?
2. Types of VCS
3. Intro to Git
4. Git Commit

BREAK

What are VCS and Why

→ we work as a part of team.

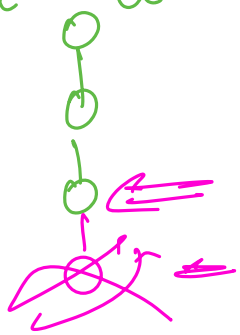




① ~~Conflicts~~ when multiple people working on same file together.

② I might want to know who wrote a particular piece of code and why.

③ I might want to go back earlier how code looked like

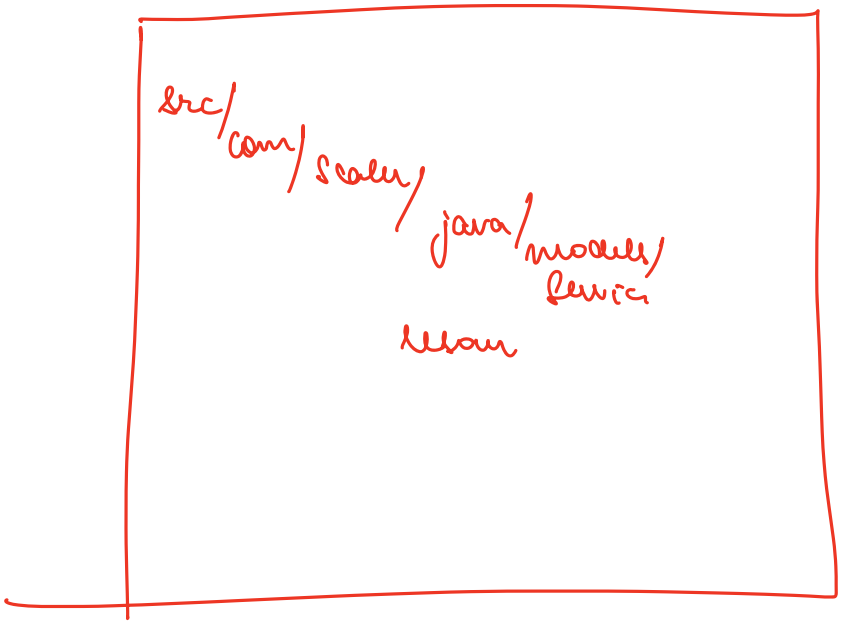


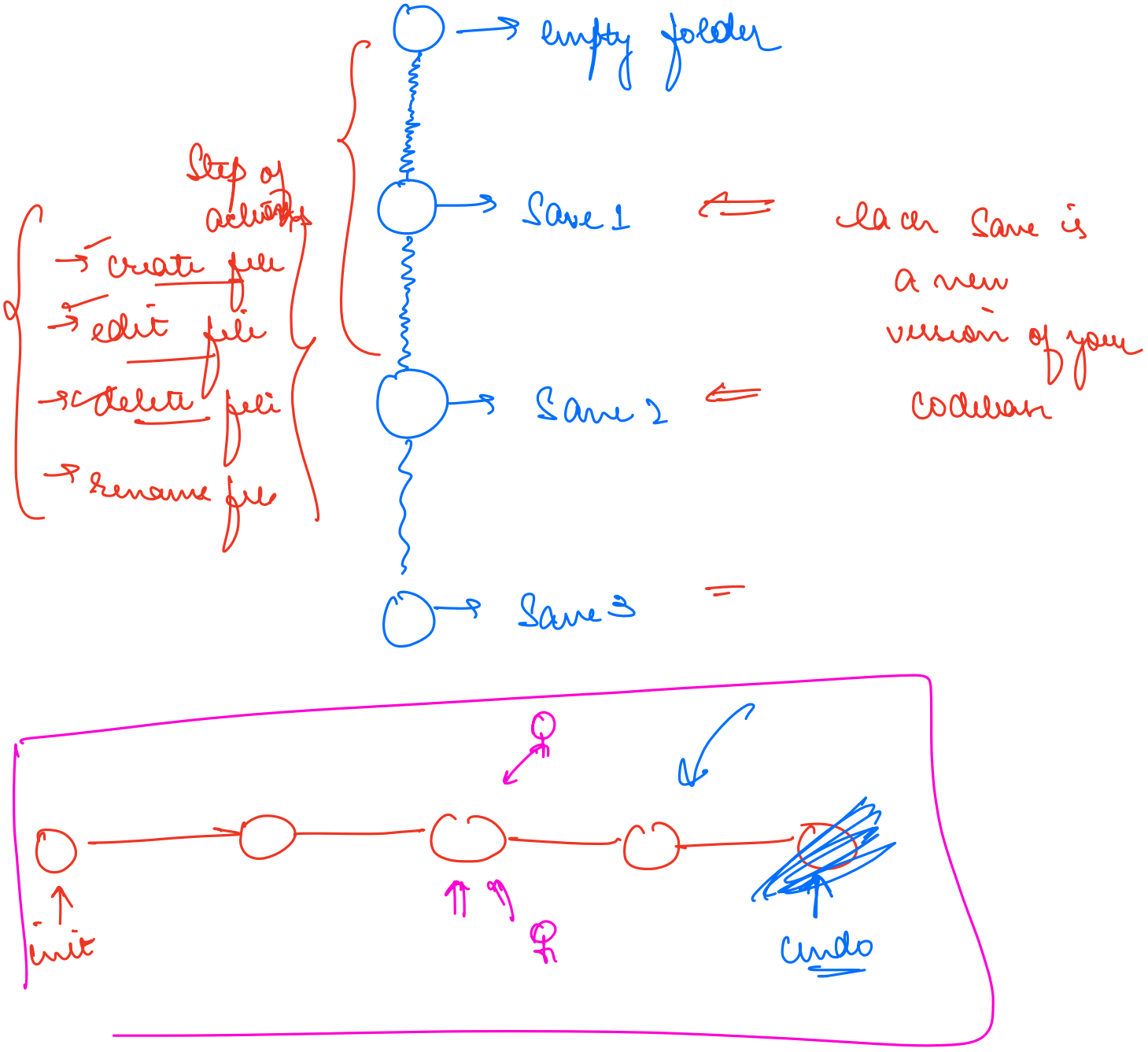
Version Control System

⇒ allows to make versioning of your codebase and thus allowing to store code history.

⇒ Software: allows to maintain versions of your codebase

⇒ nothing but a folder containing a lot of files.



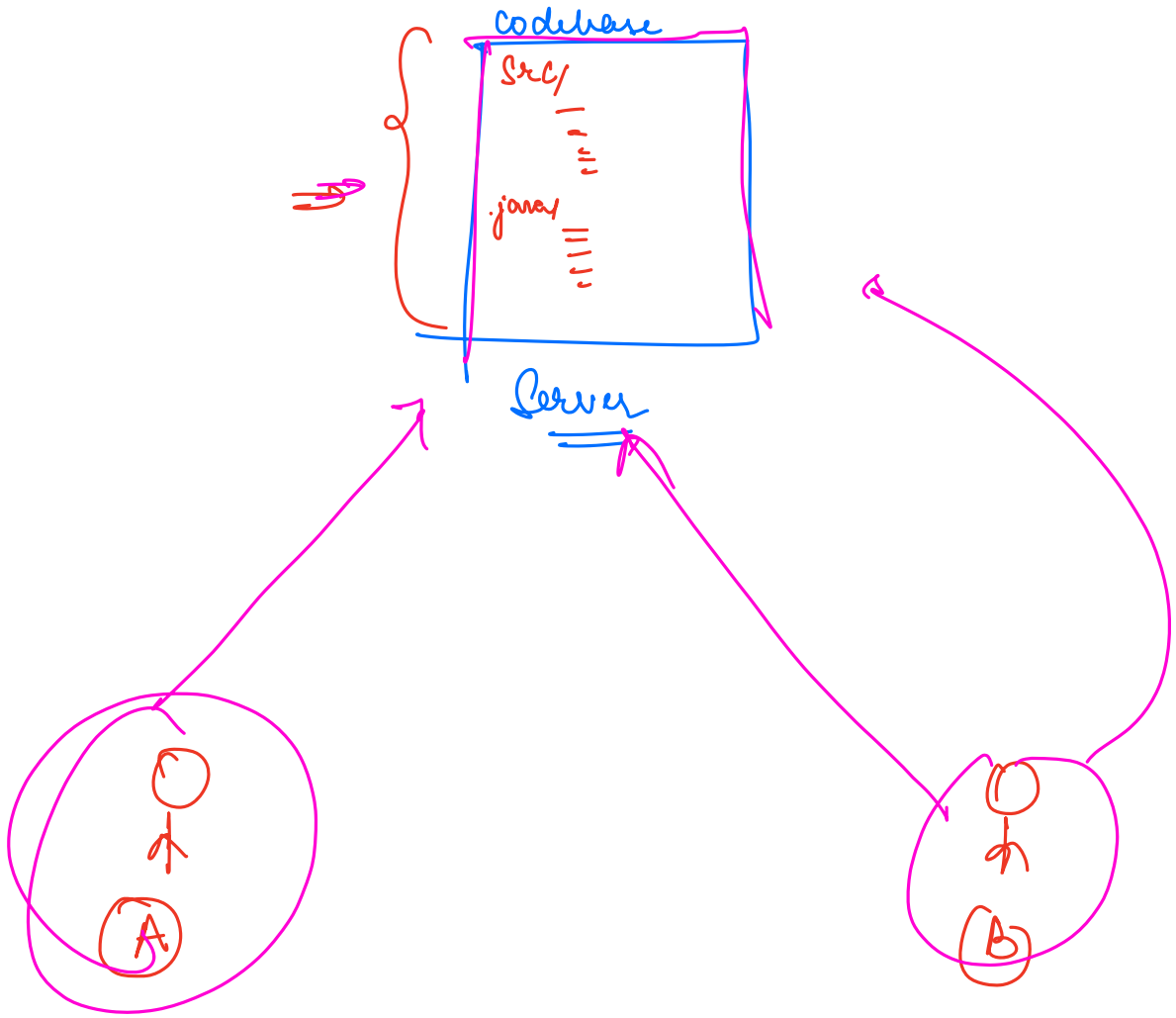

empty folder
Save 1
Save 2
Save 3
Step of activities
create file
edit file
delete file
rename file
each Save is a new version of your codebase
init
undo

Types of VCS

1.) Centralized
2.) Decentralized.

Centralized Version Control Systems

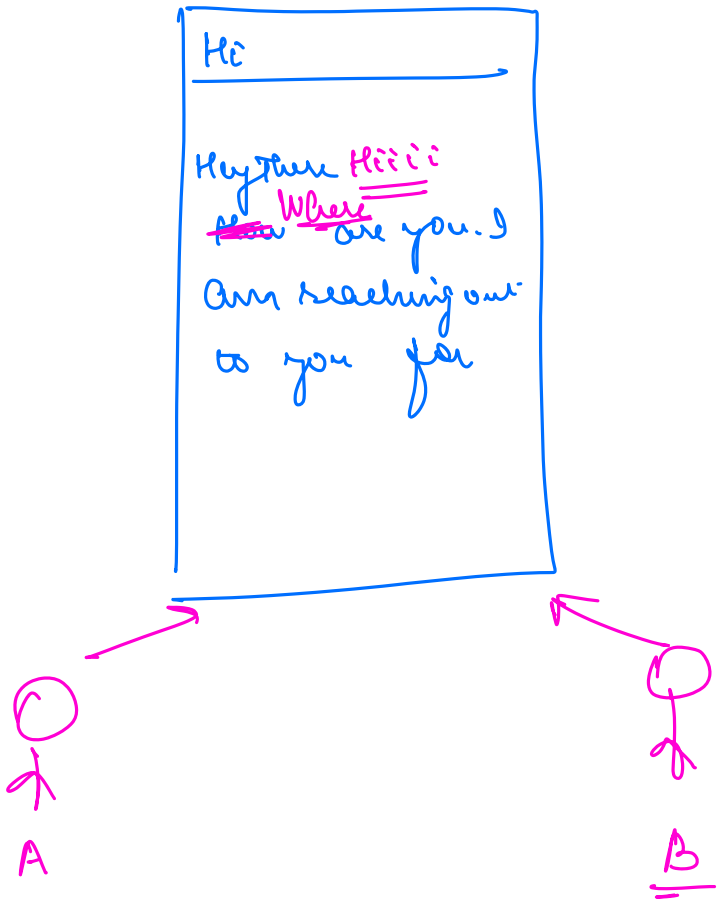
→ Complete codebase is present in a Server.



① Connect to Server

② Make changes to files on the server itself.

Google Doc



Centralized VCS

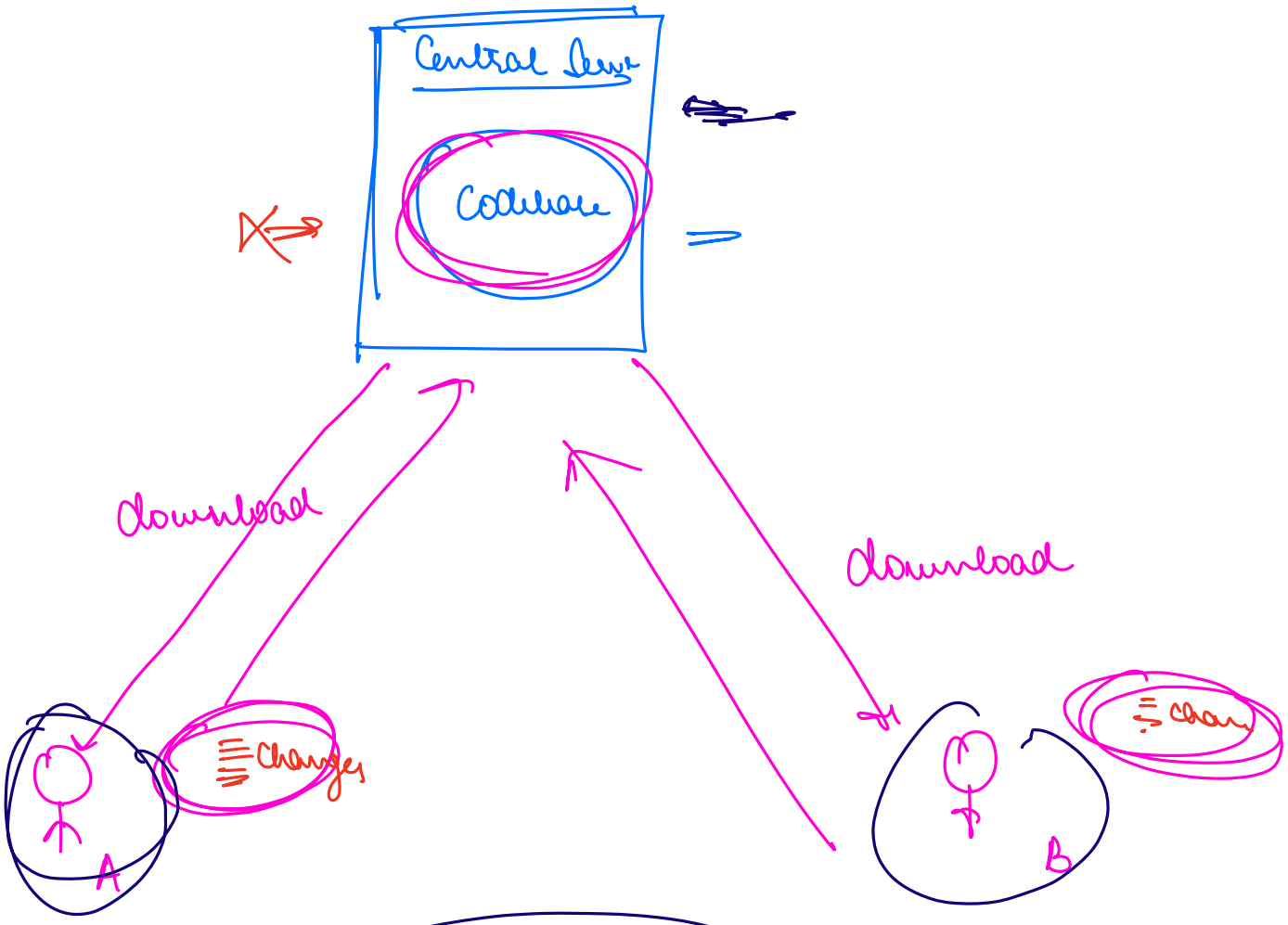
Cons

1. SPOF (Single Point of Failure)
2. Need to be online when making changes

Pros

1. Security
2. Simplicity

Distributed Version Control System



(1) Users download codebase from central server to their device.

(CLONING)

Complete codebase alongwith history

(2) Users make changes to their downloaded codebase.

> » No need to be online while making changes
>
> (Committing)

(3) After doing changes, submit changes on central server

PUSHING CHANGES

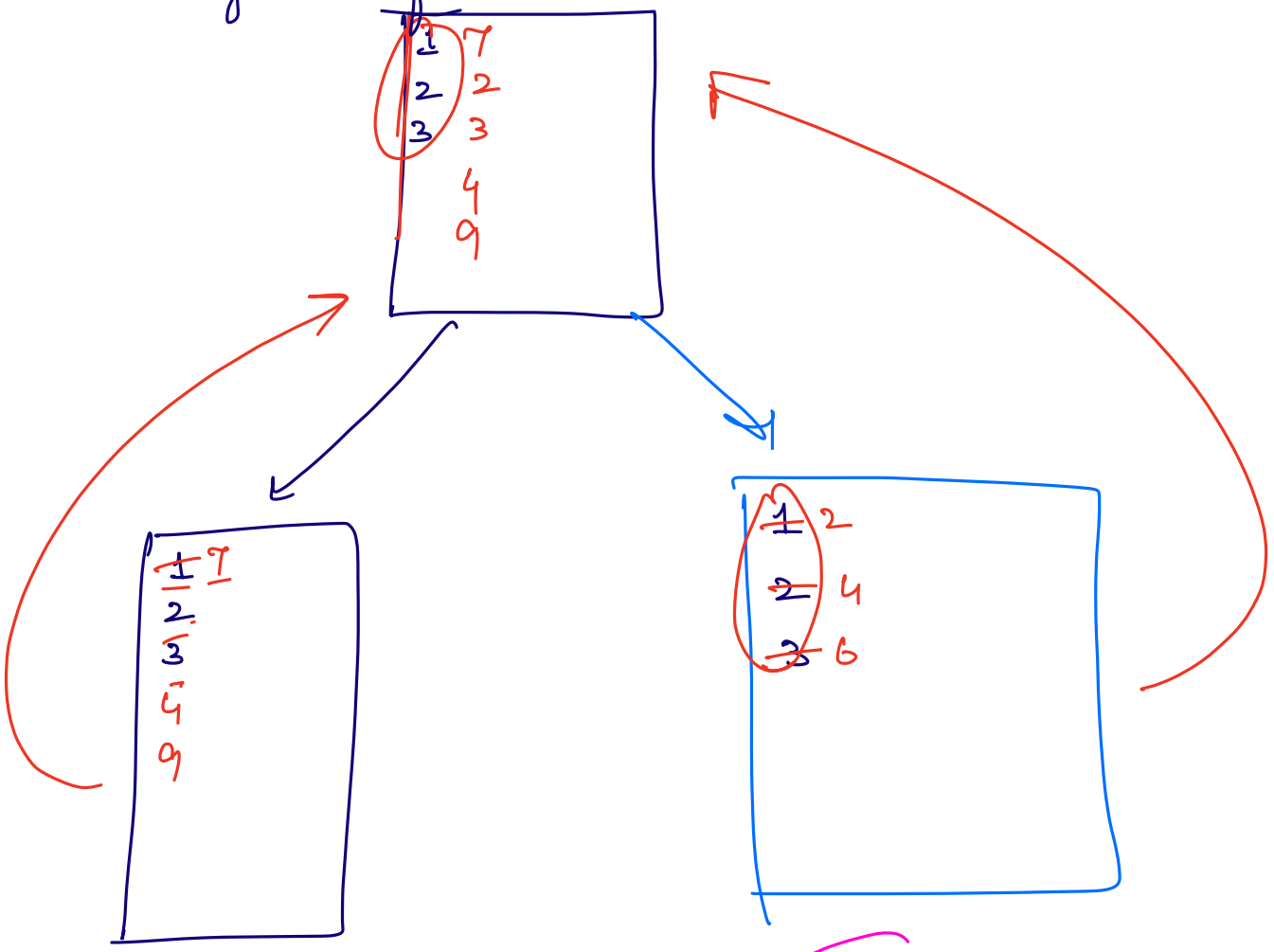
Pro :

1.) No SPOF

2.) No need to be online while writing code

Cons:

1.) Merge conflicts



Simple

highly scalable

Git is the most popular VCS in industries

→ git is a distributed version control servers

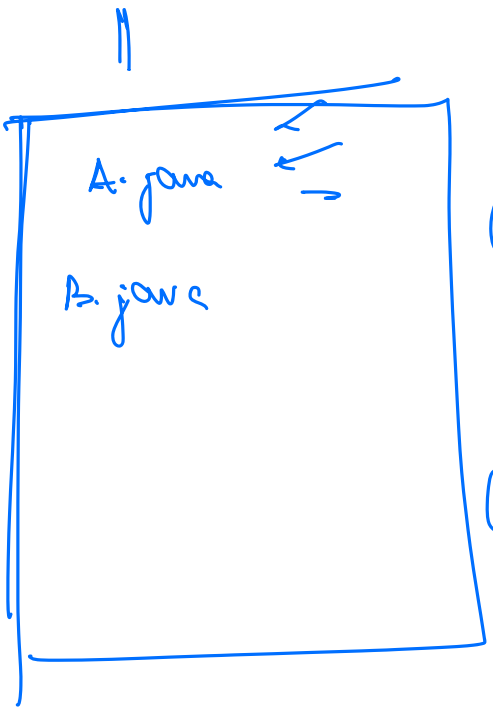
→ Git was created by the same person who created Linux

Linus Torvalds

⇒ Github, Bitbucket, Gitlab

→ Not version control system

→ Central servers that allow to store codebase managed via git.



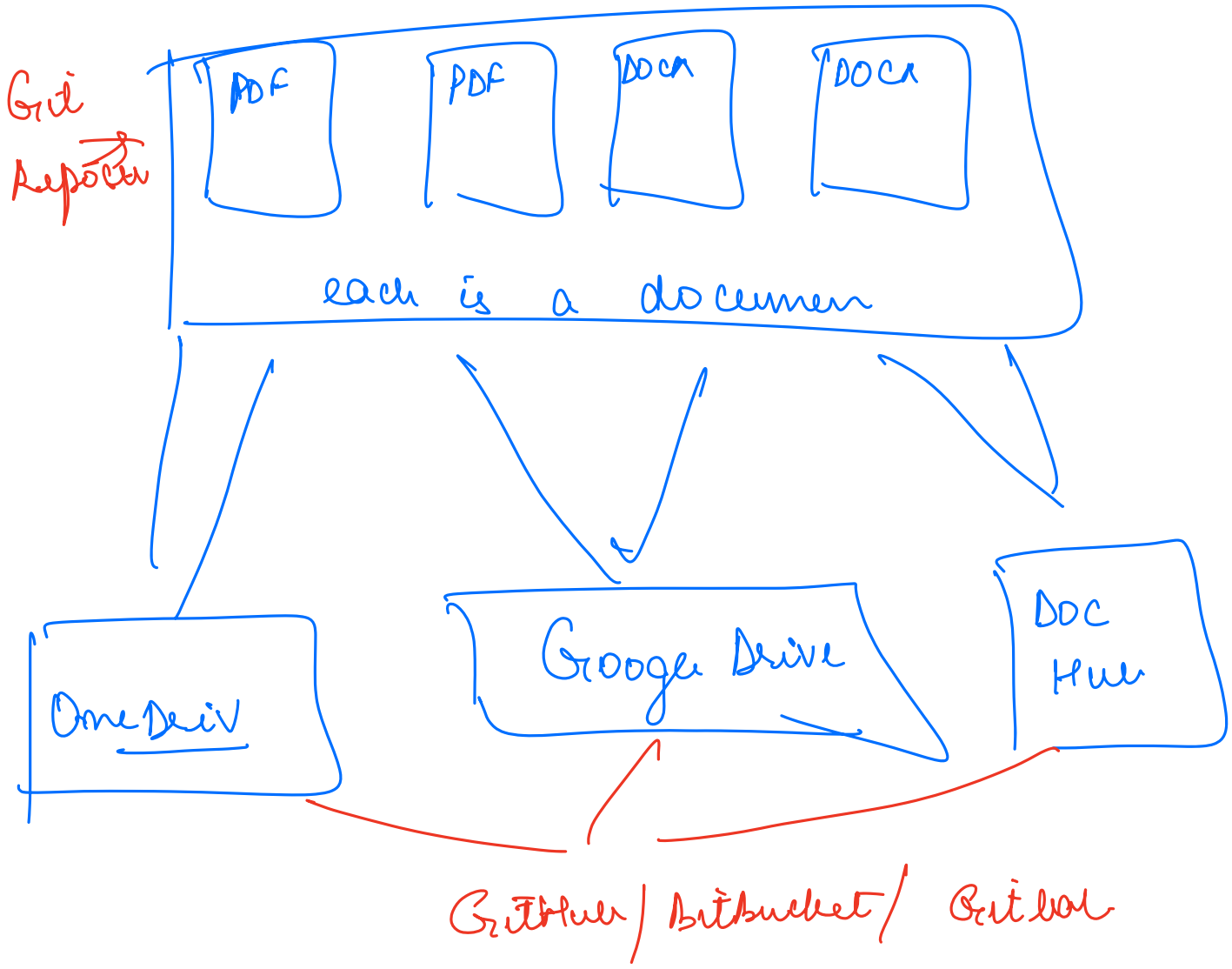
(1) Naman creates a folder

(2) Naman decides to use git to manage versions of the folder

(3) Naman decides to use Bitbucket to store this Git folder (Repository)

(4) Naman's friend can download from that

Documents



After Break : Learn Git

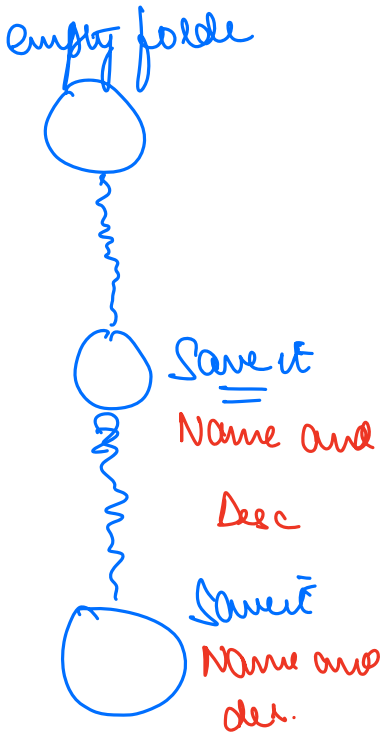
10:15 PM

SVN
Subversion

Book ⇒ Book Shop

Git ⇒ GitHub

(1) Git Commit

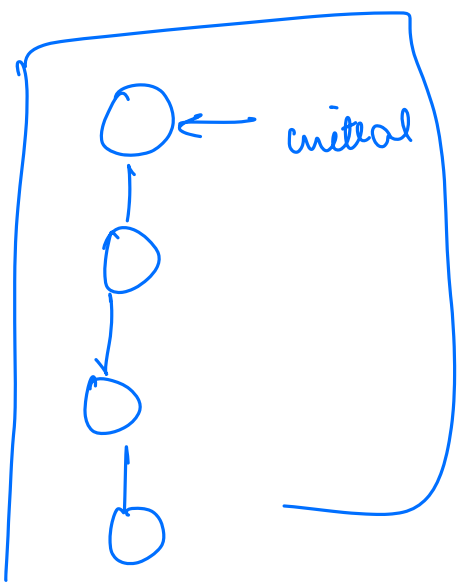


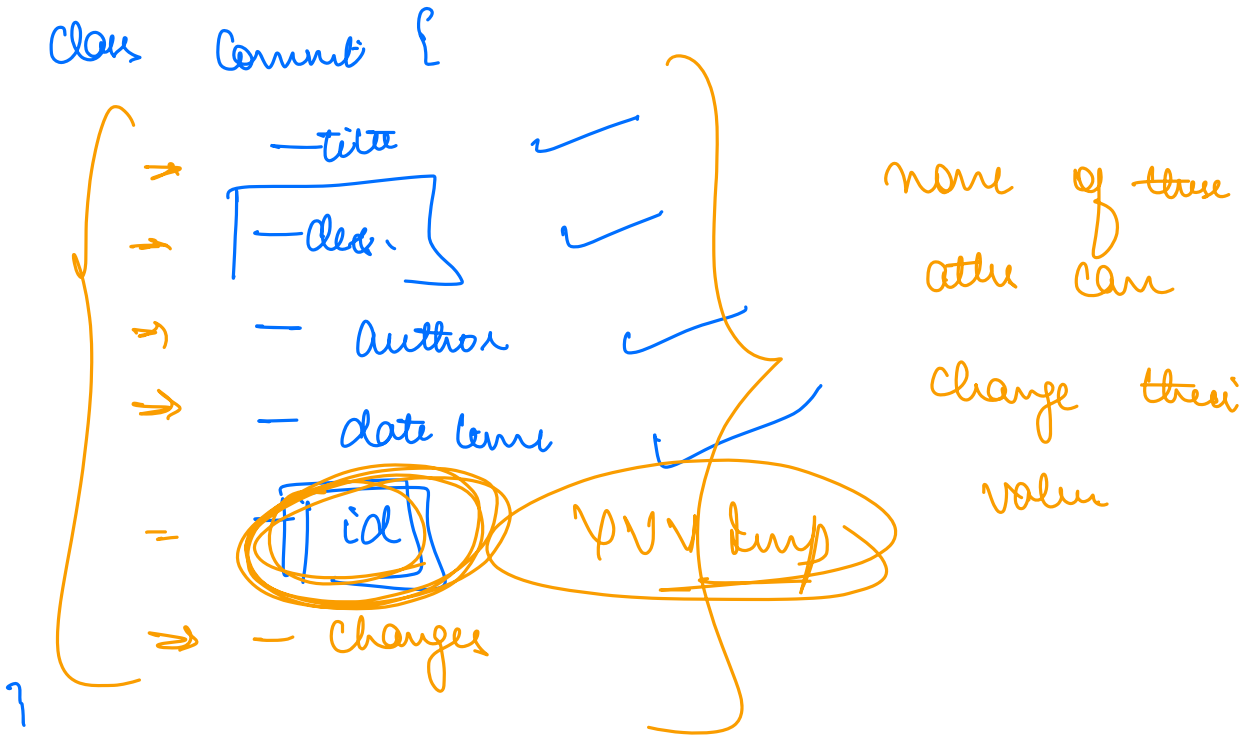
Good practice: commit often

⇒ whenever done any meaningful change

Save <=> Git Commit

⇒ nothing but a series of changes that have been made since the prev commit





⇒ every commit is immutable

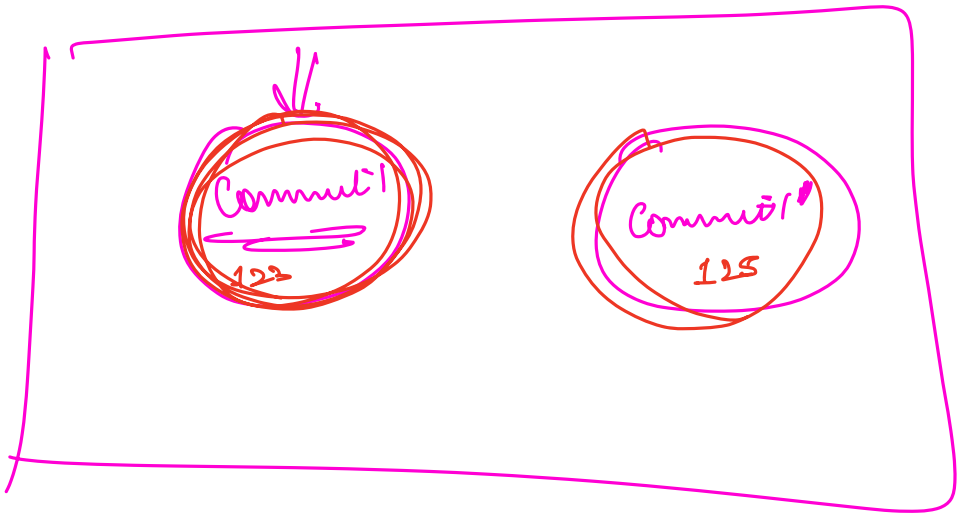
→ when you have made a commit you can't change any attribute of that

⇒ If you try to change anything, it doesn't mutate the commit, instead creates a new one.

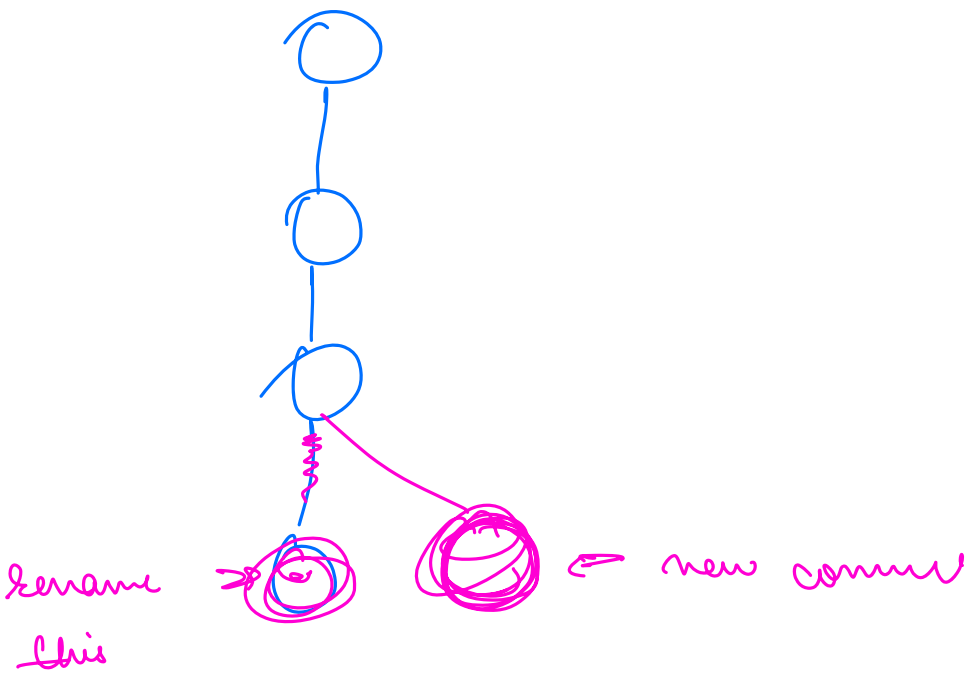
→ id

→ name

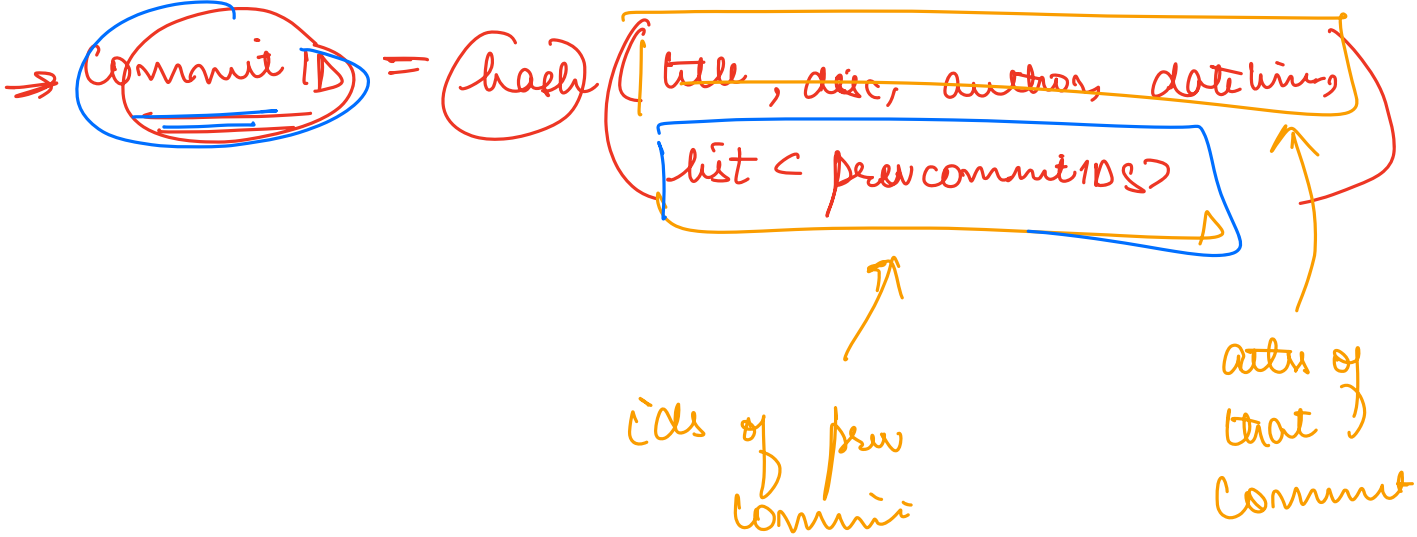
→ desc.

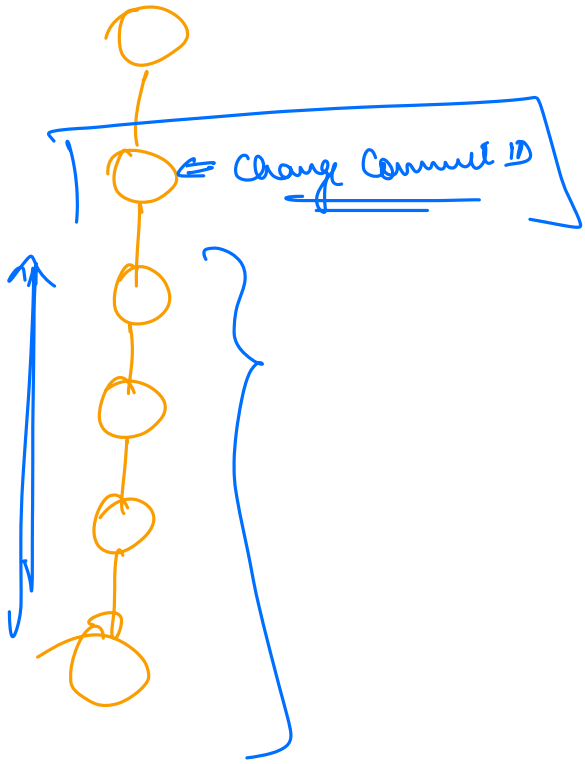


→ Once a commit is made it is never lock

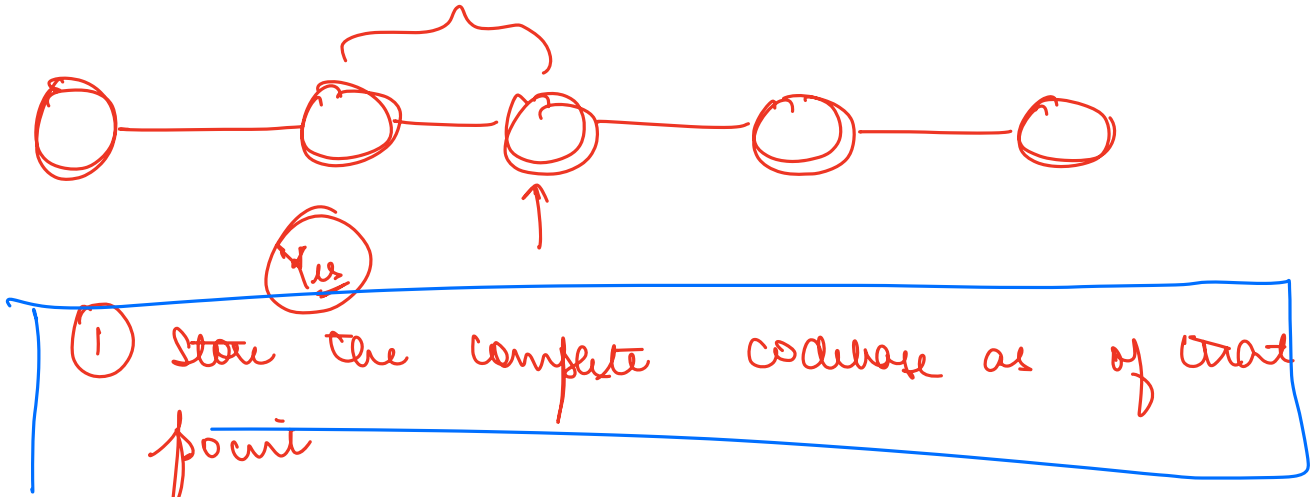


Git ensure no one can play with git history





How Git Works



No

(2) Store only the changes since per commit

Cons

1 ⇒ Space of my git repo will be huge
eg linux codebase → 10 Million of lines

Pro going back to how code looked like at any moment in very easy

Git stores only the diff b/w 2 versions in every commit (Deltas)

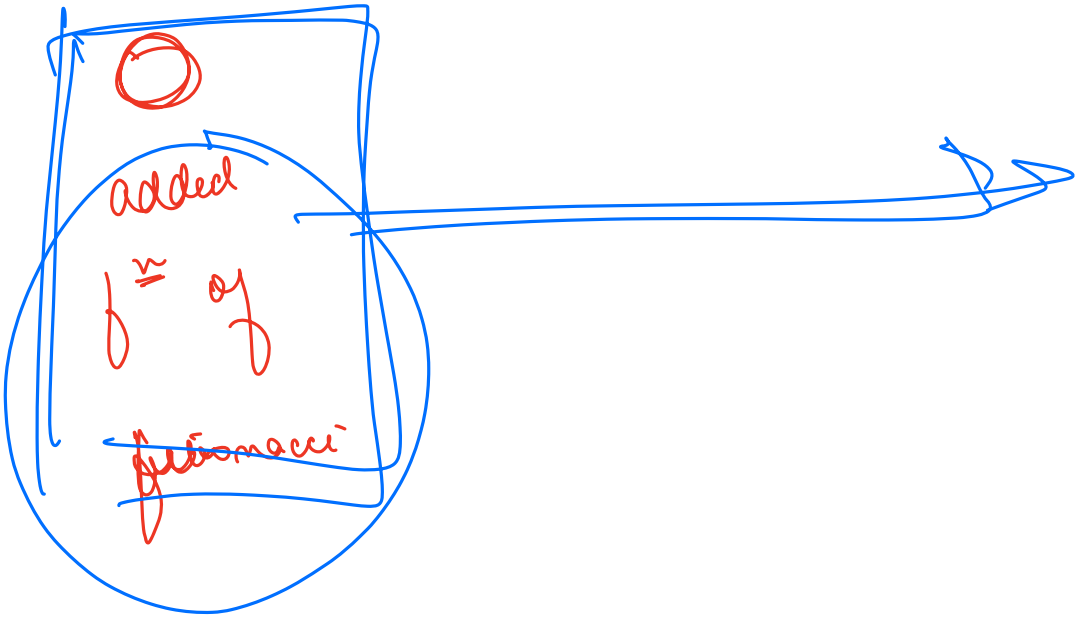
Con: If I have to see how code looked like as of a particular time

→ require some time to construct

PROS:

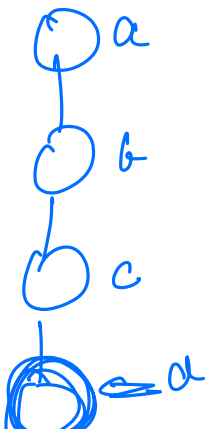
(1) less space

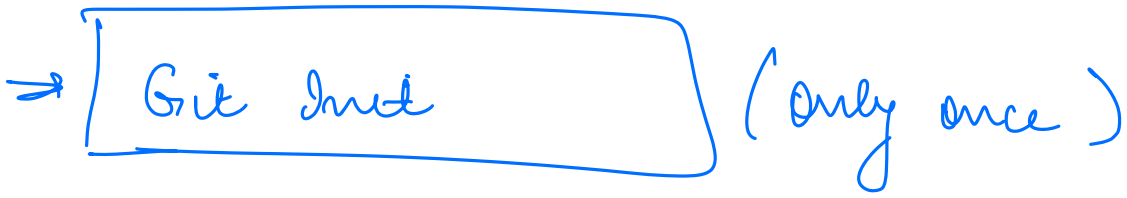
Some other Project

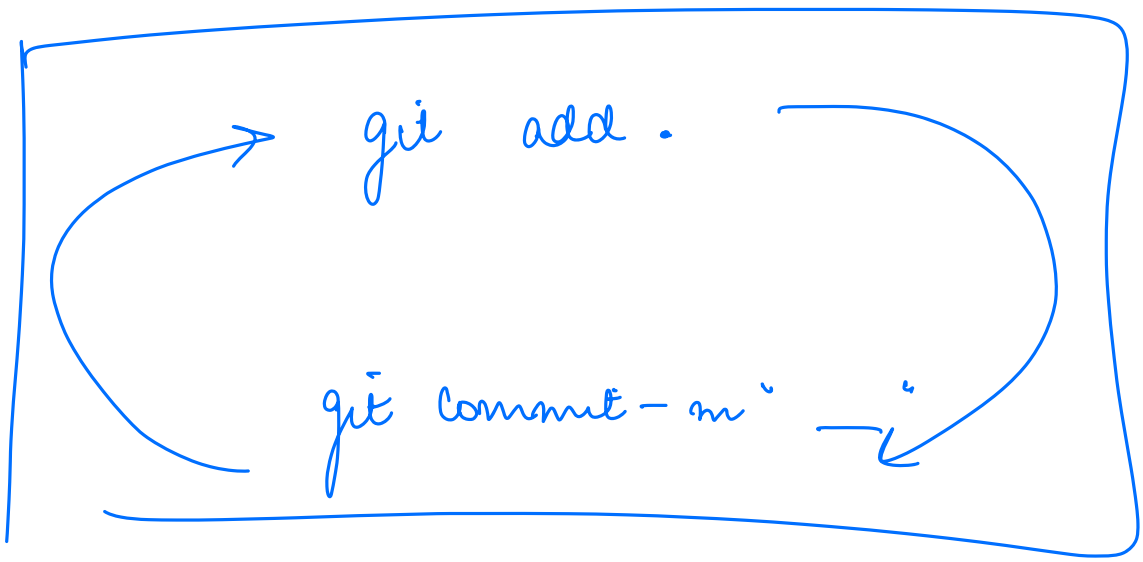


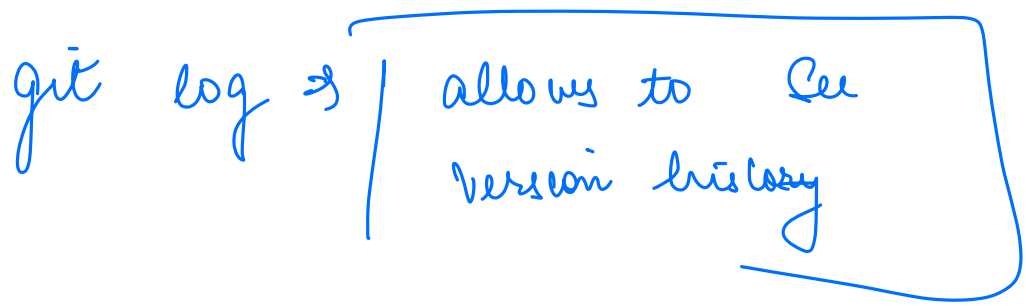
Because a commit is only storing deltas, can use code written at one place at another.

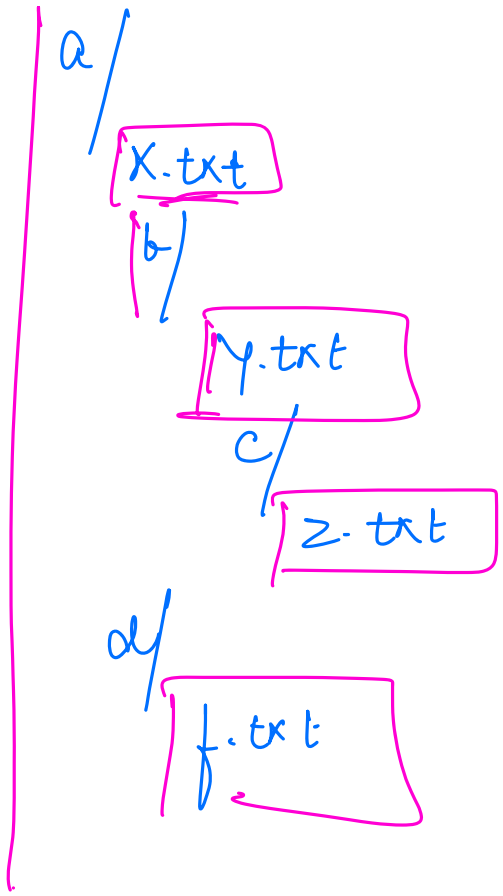
⇒ every git commit has atleast 1 parent (the commit from which changes were made)




Git Init
(only once)


git add .
git commit -m "..."


git log
allows to see version history



a/ git add .

↓

all files in current and child directories

.patch

1. Install Git on your machine
2. Practice first 3 sections of learn git branching (5 tutorials)