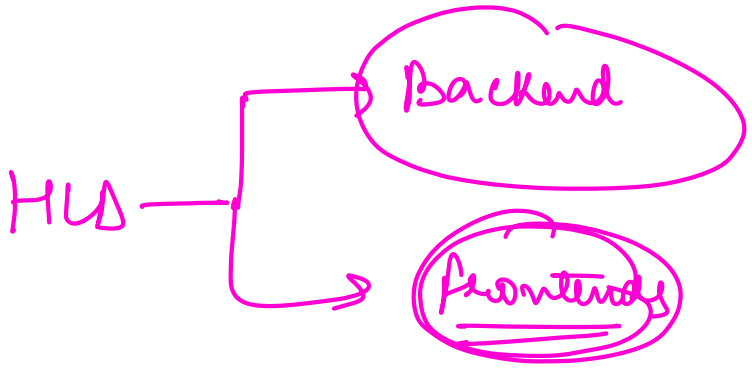
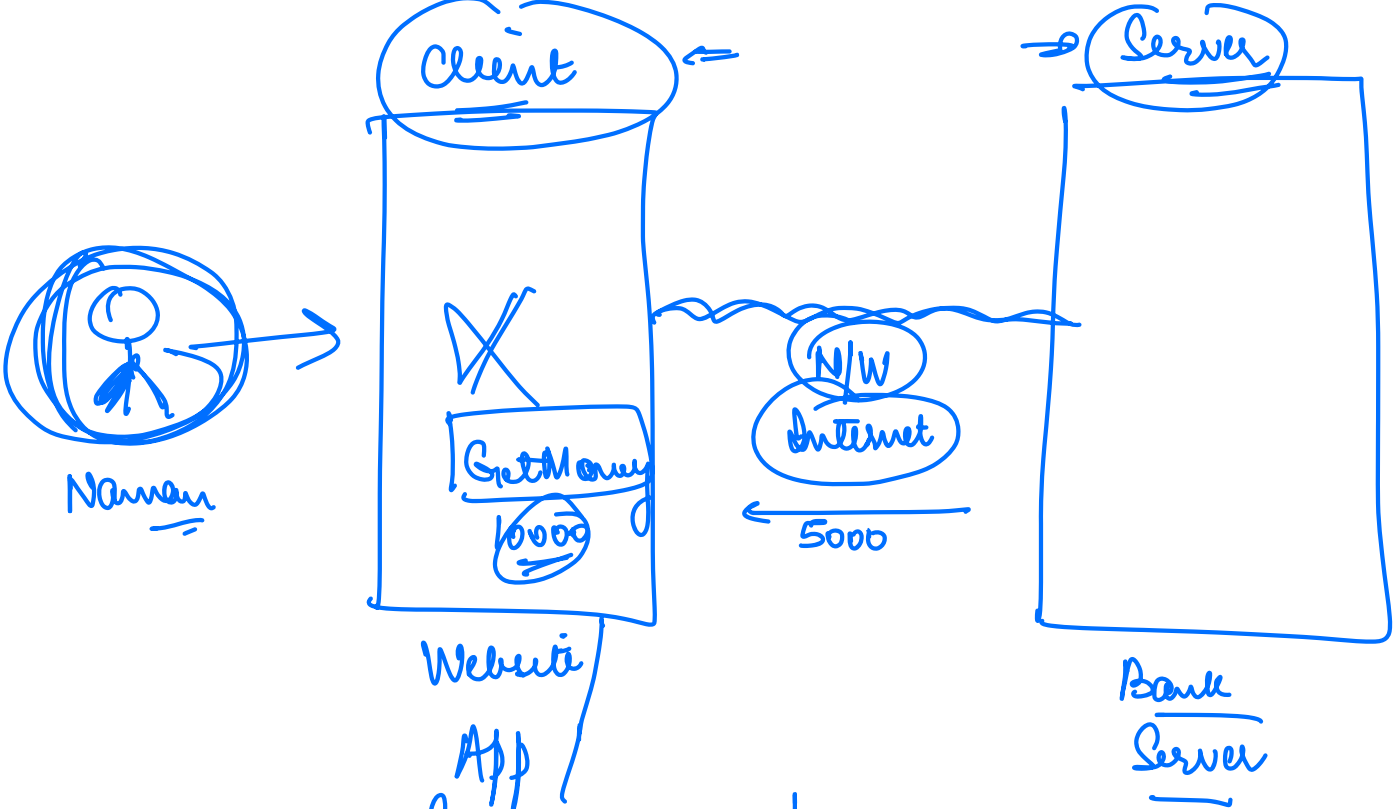
Agenda

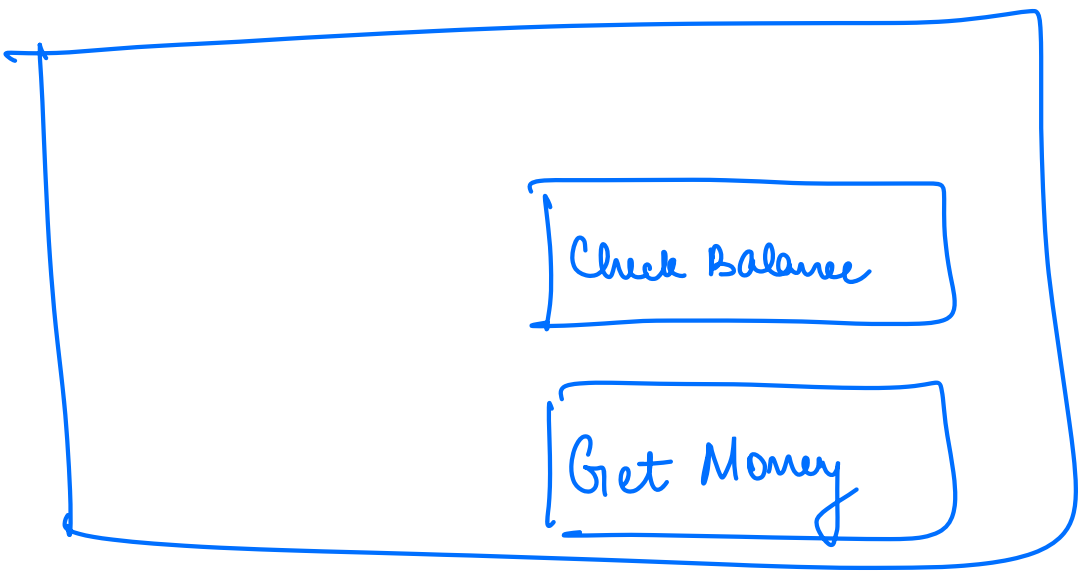
1. What is Backend
2. Expectations wrt this module.
3. Curriculum of the Module.
4. Git
   - → Why VCS
   - → Distributed VCS
   - → Git ~~Commands~~
   - → Git Best Practices

Outcome: Create your first Pull Request

What is Backend







Client

→ user facing part

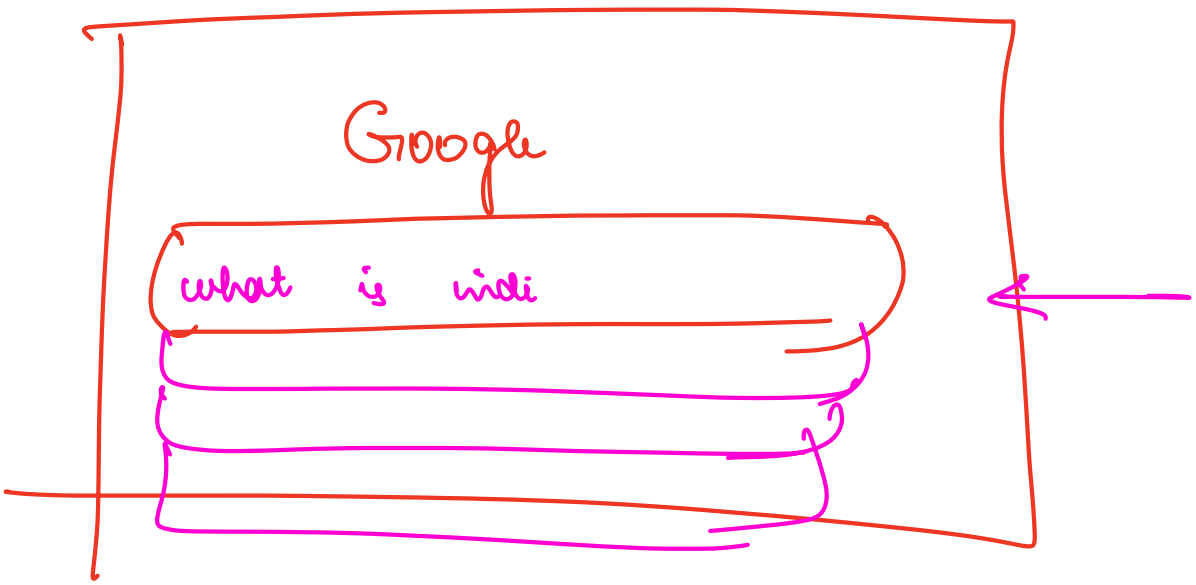
→ website / mobile app^{n} / Smart Speaker

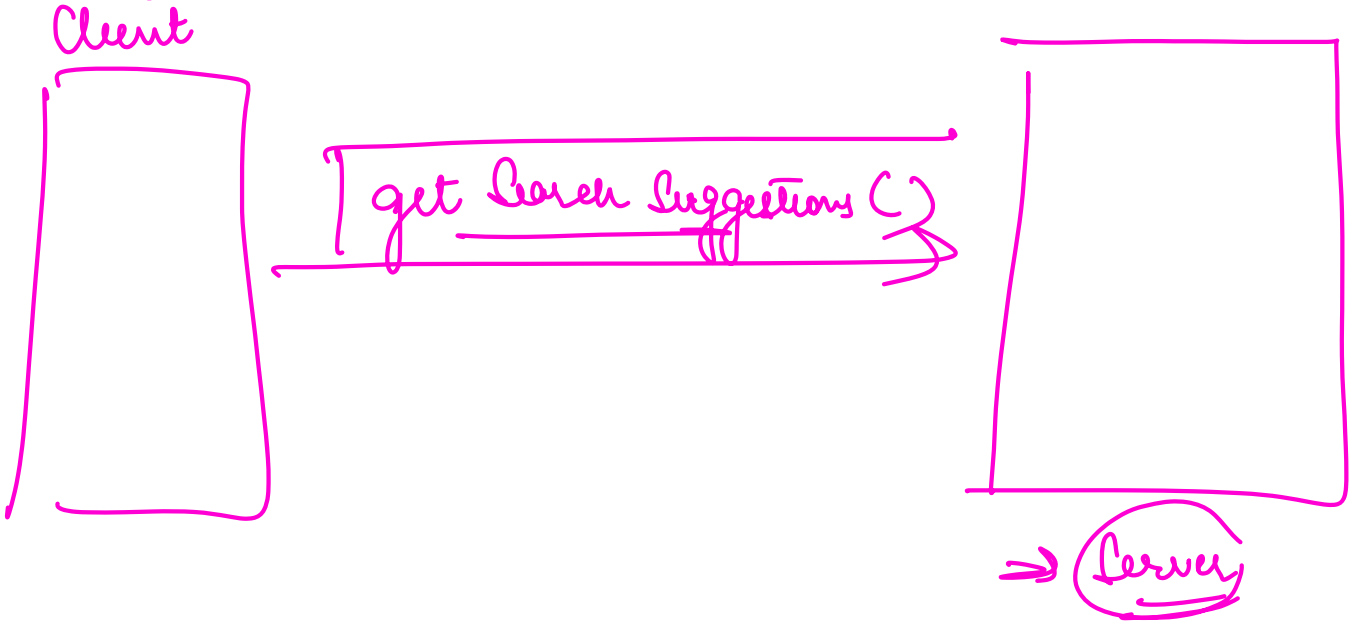
→ requests / Create data

Server

→ Owns data (data is stored)

→ logics on servers can be very complex.





Backend : Everything wrt the server

⇒ Good in working with data systems

→ Kafka, DB, Cache, Elastic Search.

Expectations wrt Backend Module

Why this Module (technically complex and comprehensive)

1. Getting projects in resume that demonstrate ability to get started from day 0.
2. Teaching best practices around common pitfalls that software engineers do.

Do

1. Understand how real world dev looks like
2. Explain common tools, nuances, concepts that will help you become a more proj developer.
3. Get a project worthy to show in your resume that gets your resume shortlisted.
4. Get you to work with adv infra layers (Kafka / ES / Redis / SQL / MongoDB)
5. Teach Best Practices.

Don'ts

1. Spoonfed You : we will ensure you are never lost but will only show you dir^. You will have to read / implement / try
2. No memorization concepts :

Curriculum

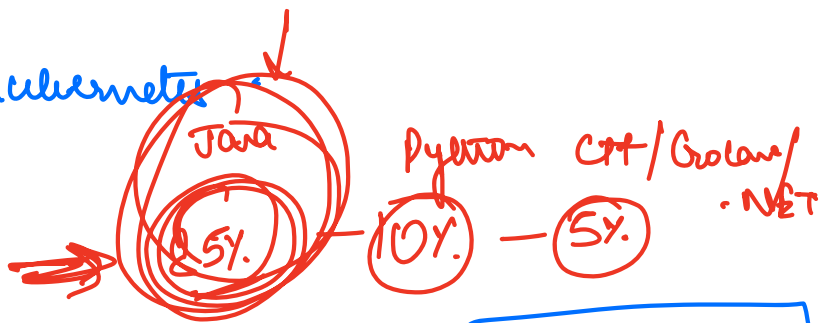
1. Foundational Adjacent Concepts
   - → Git
   - → Command line
   - → Dev Setup.
2. Project
   - → Setting Codebase.
   - → Imp APIs
   - → Connecting to DB
   - → Common Features : Auth → OAuth, 2FA, Logging, Paging, Sorting, Filtering.
   - → Unit Testing.

③ Adv Concepts

- → Kafka
- → Redis
- → ES
- → MongoDB
- → Dist Tracing / Logging
- → Implementing Payments
- → Microservice.

④ Deployment

- → AWS Basics
- → CI / CD
- → Dockerization. / Kubernetes



→ All impl will happen in Java / Spring Boot

⇒ Ecom Platform

→ No UI

→ Individual Project

→ 18 classes

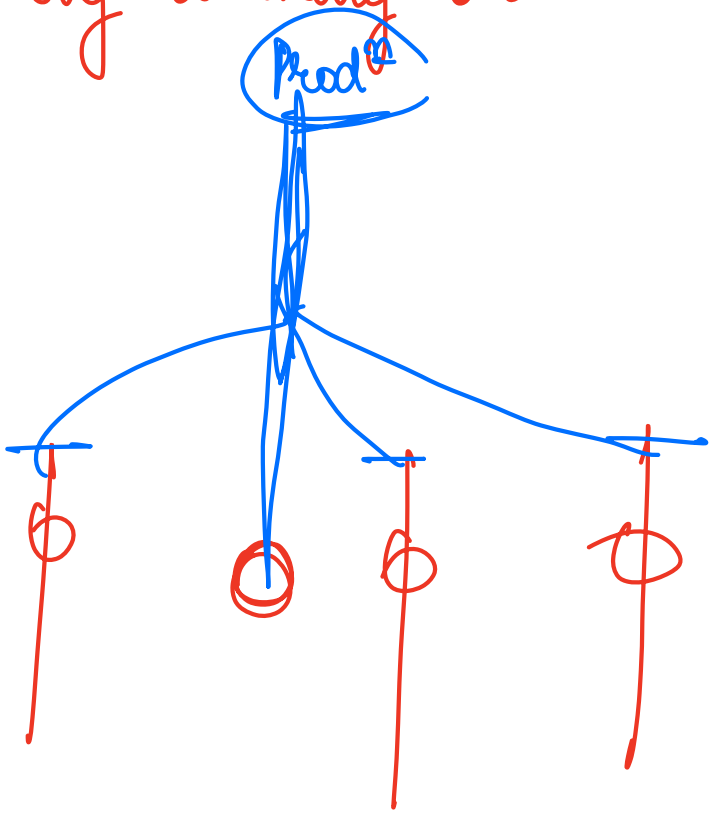
① IDE

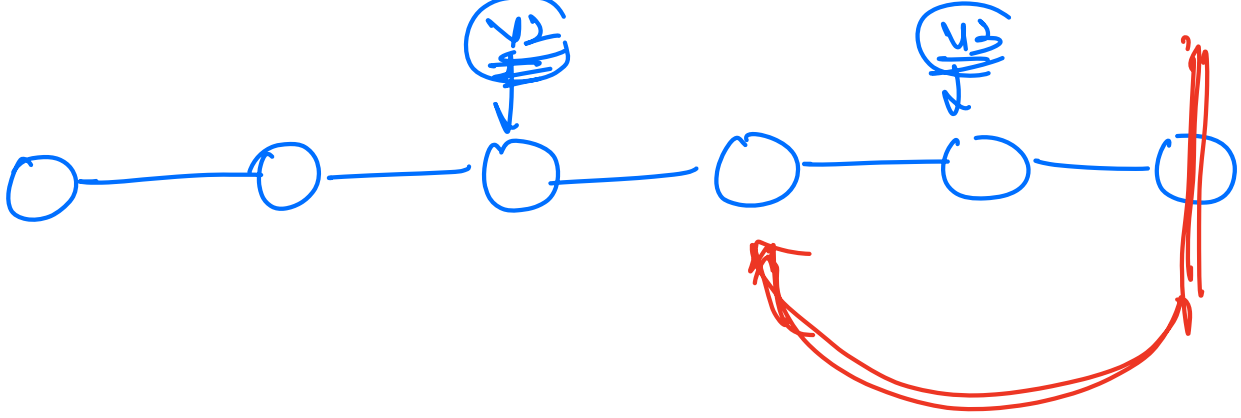
Fill form for Github Dev Pack by 25Aug 6:00PM

Git

Version Control Systems

→ 100s of eng working on same codebase.





(1) Bug

Why might we want to keep history of code base

(1) So, as to go back to how code looked like on a prev date.

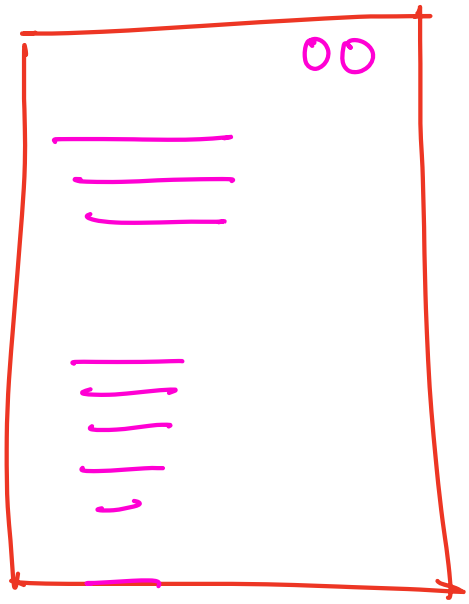
(2) to keep a track of work everyone is doing.

Version Control → Keeping a track of how code looked at a particular time

Types of VCS

(1) Centralized VCS

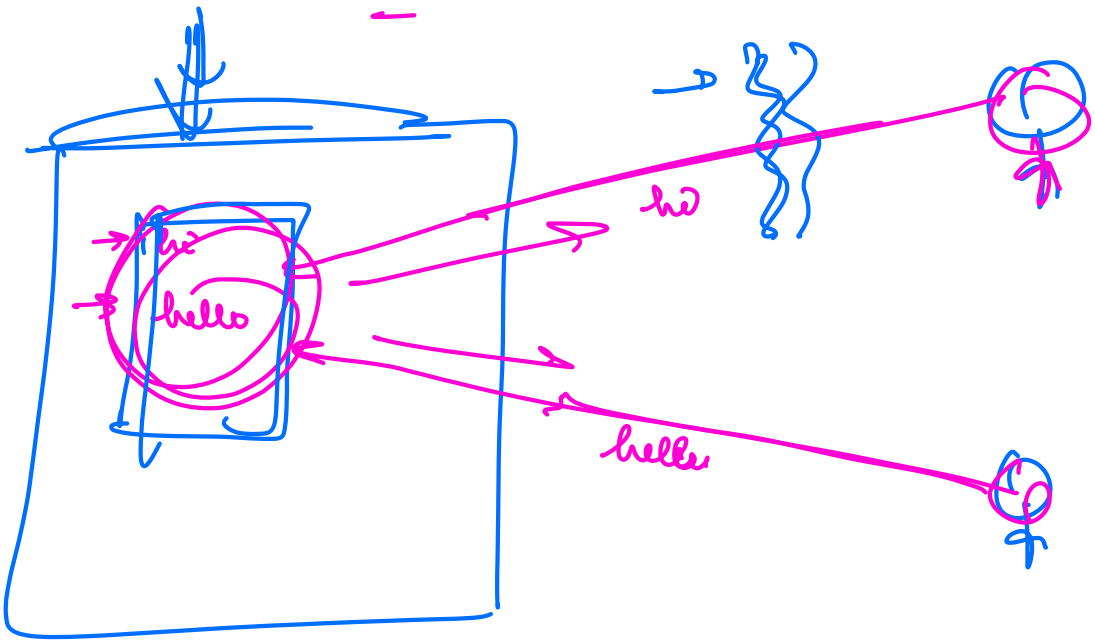
(1) A single server where the exact history of data is being



maintained.

② Need to be online always when maintaining changes

③ SPOF

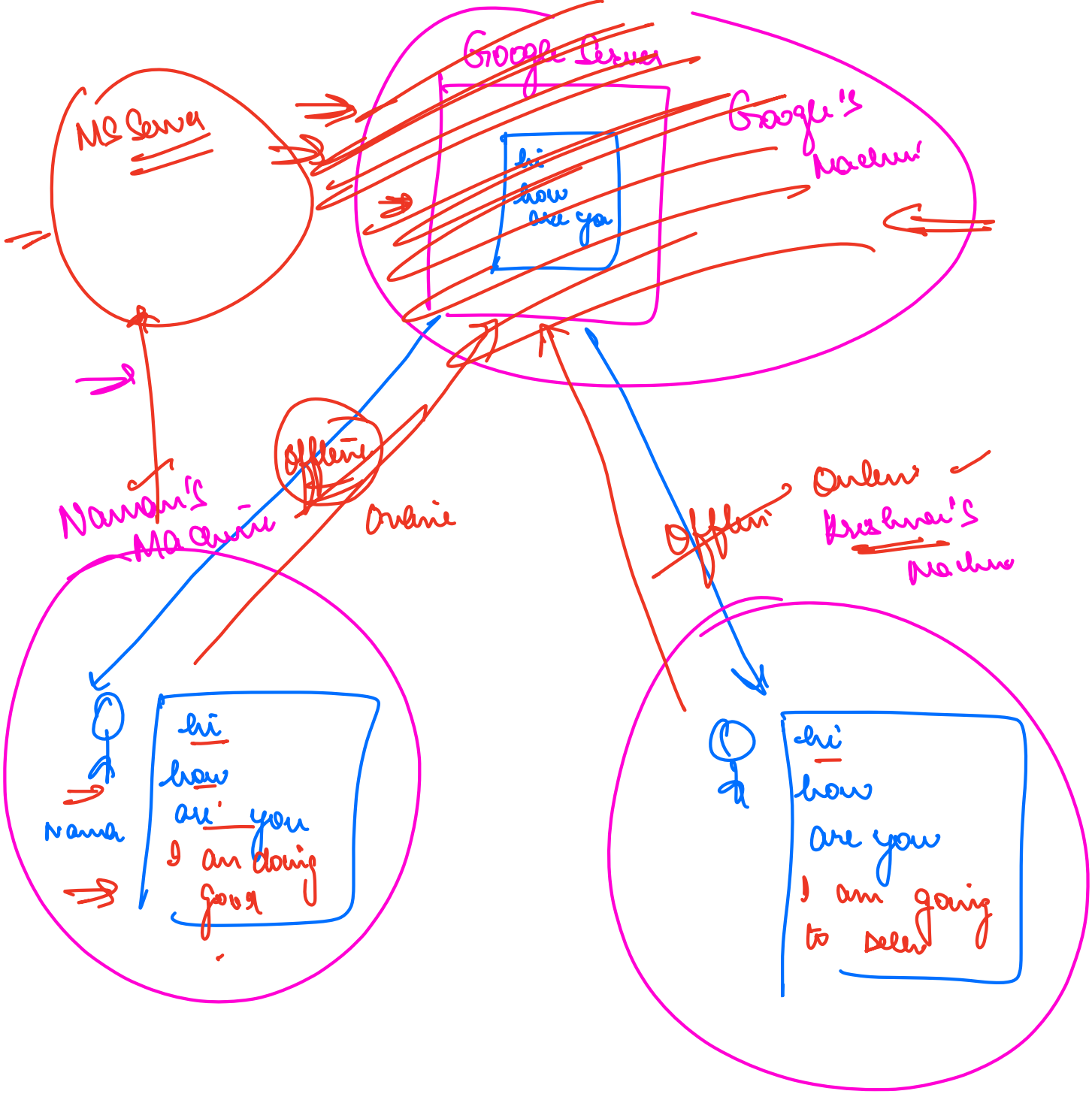


② Dist Version control systems

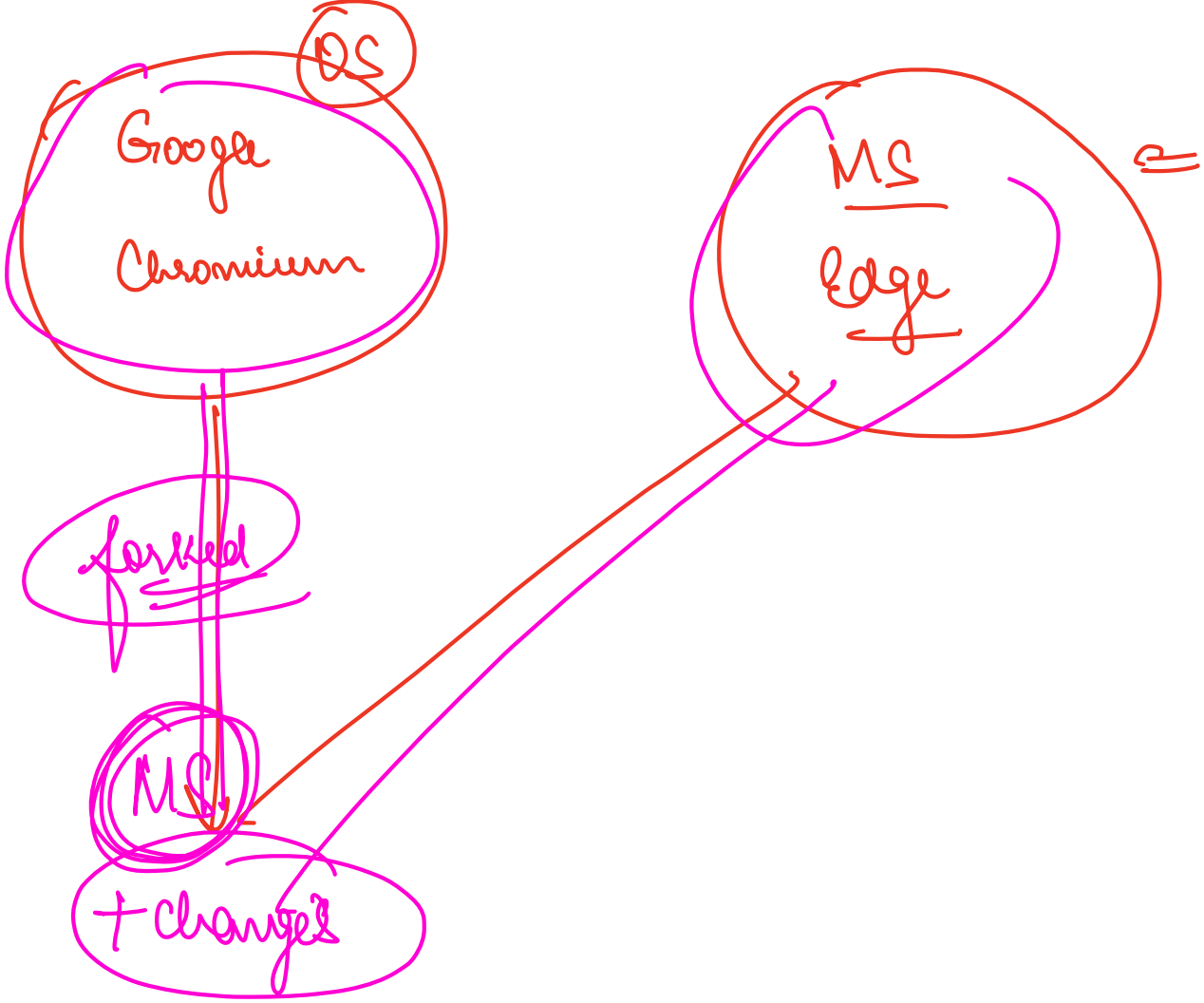
→ allow people to work independently, even offline

→ no single source of truth

→ the whole history is stored on the machines of everyone involved.



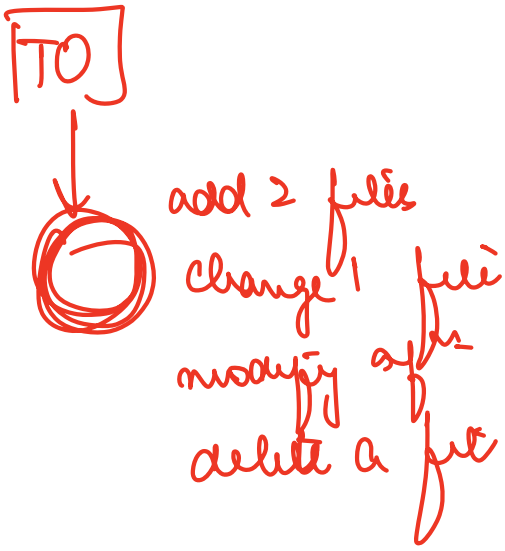
→ DVCS allow users to resolve such conflicts themself.

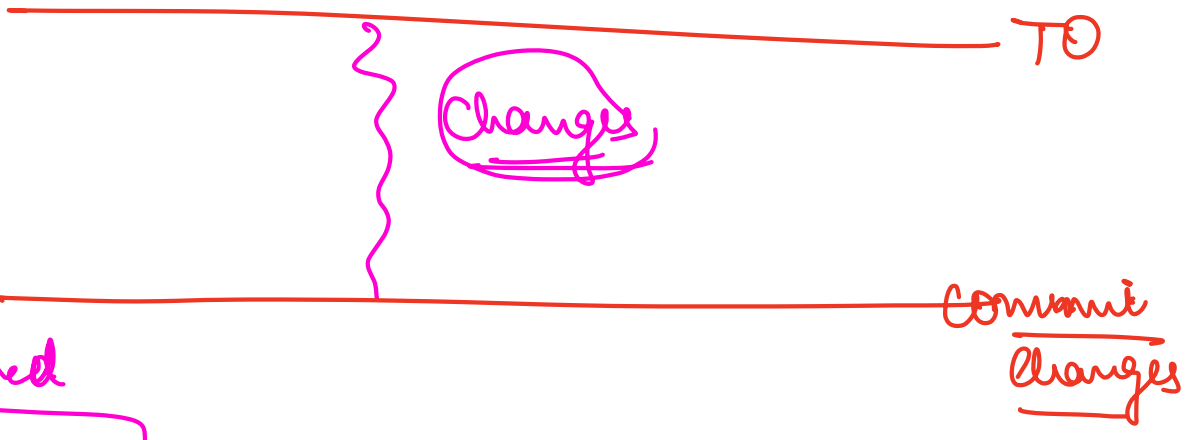


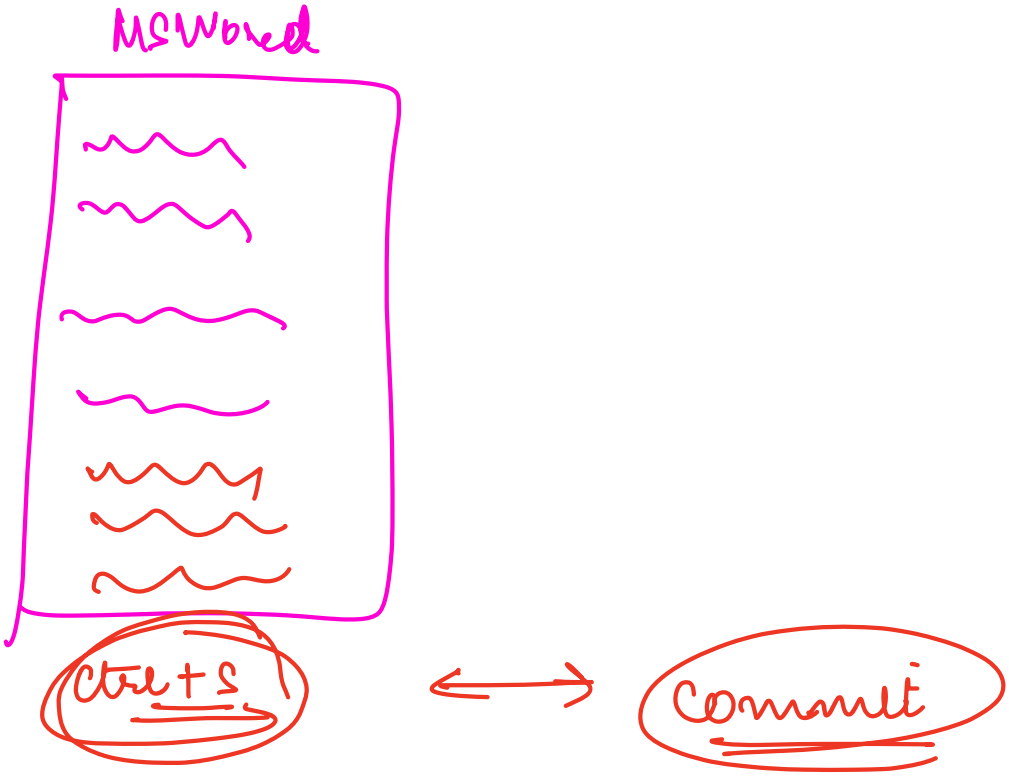
→ Git is a distributed version control system.

→ was created by same person who created linux
Linus Torvalds

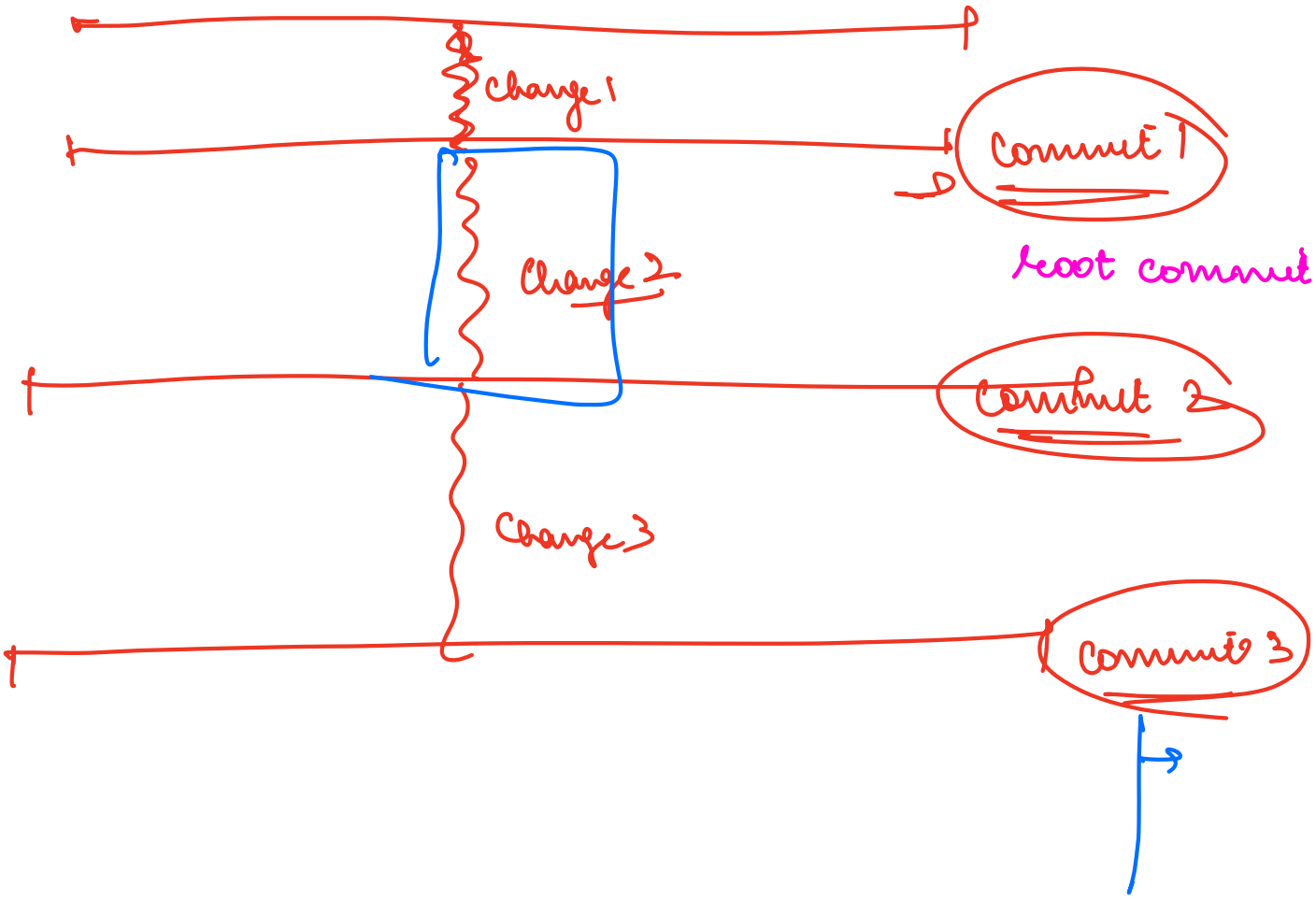
Commit (what is a git commit)







→ history of project in git is nothing but a series of commit



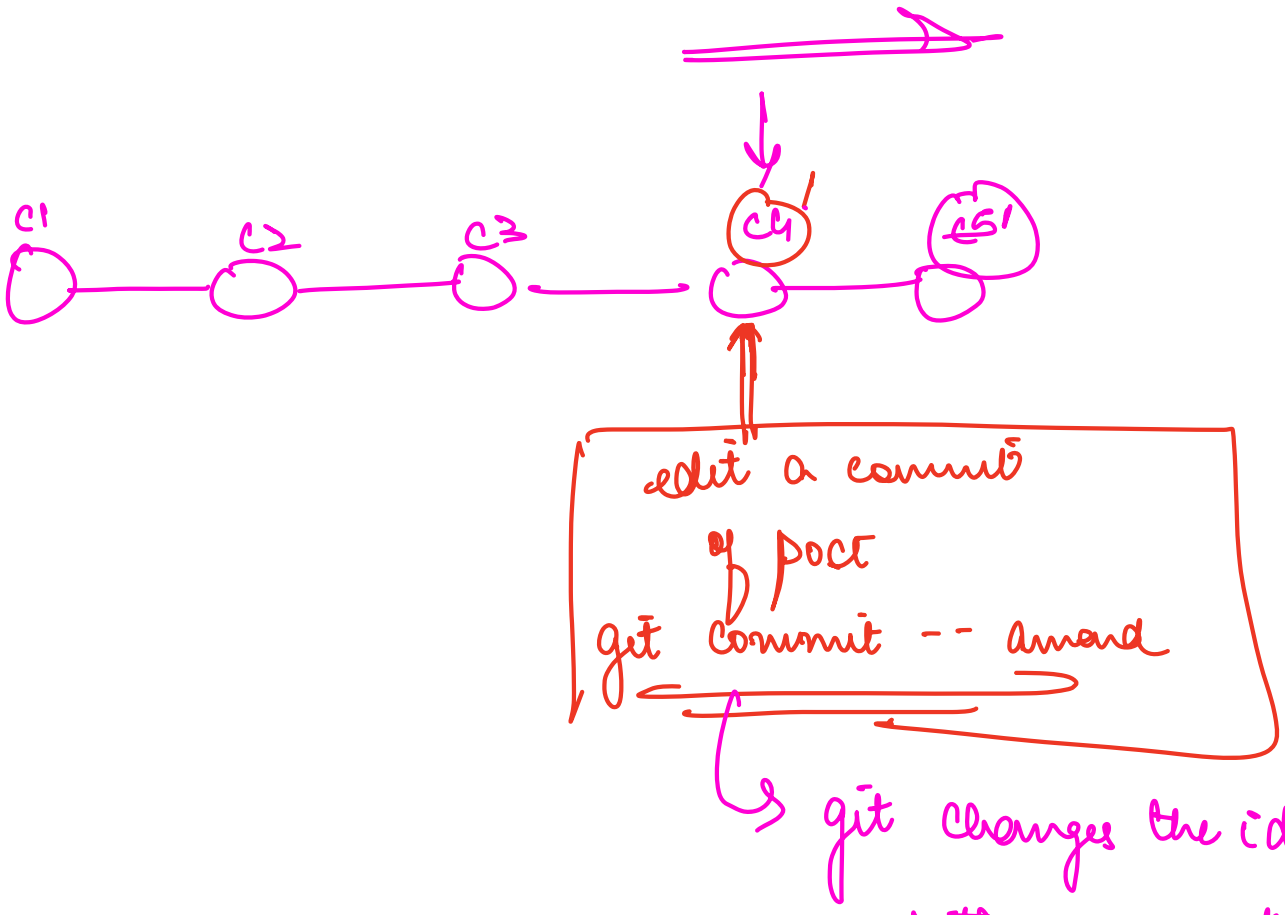
every commit

- → message
- → description
- → author
- → date
- → commit ID

} explain what those changes do

→ every commit is immutable → can't be changed

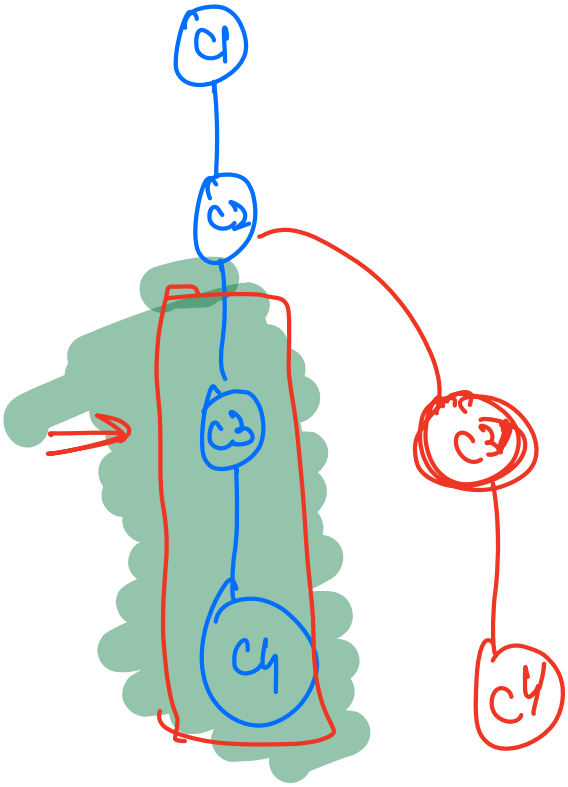
→ Once you make a commit you can never edit it



In git, id of a commit is a function of id of all prev commit as well.

id (New commit) = hash (id-1, id-2, id-3, id-4)

→ The prev commits never get lost



→ all of the changes (commit) that were made in the project will always remain

(1) When a commit is done would you want to create a copy of the complete project as of that commit and store

Or

(2) = you store only the changes since prev.

delta / patch

30-40 GB

→ Assoc with every commit

→ Changes made since the prev commit

→ Keep changes small
→ No long commit
→ make commits as frequently

→