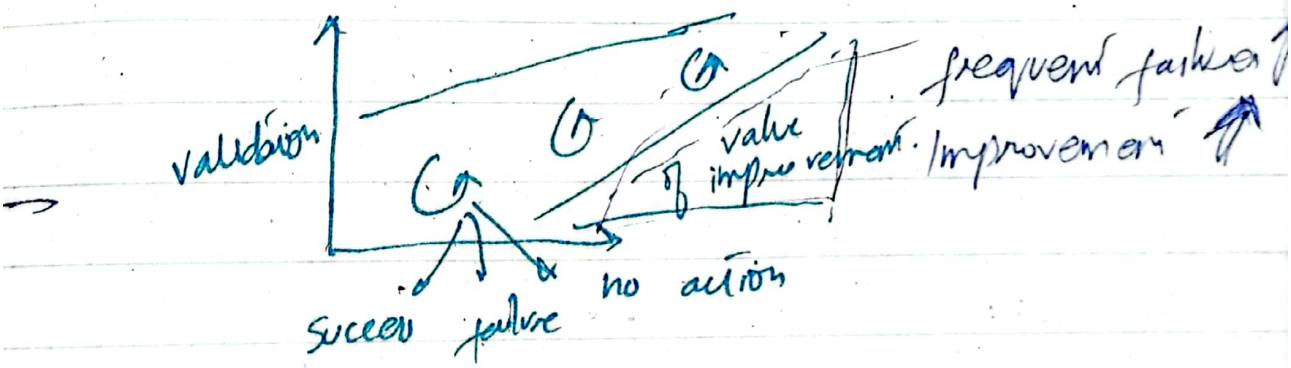
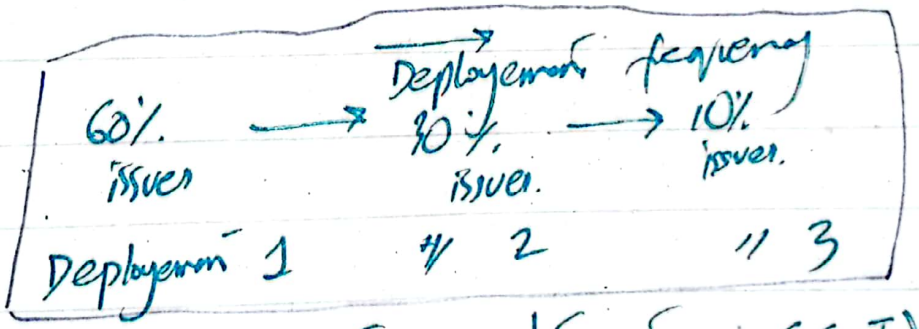
23rd Aug '23 DevOps

- Problem with SDLC : can get or stuck at some stage .. (Disjoint)

- DevOps has cross functional Teams (people from different departments)

- In DevOps, we let output fail. (Aim is frequent feedback). Quick Action can be Taken!




## Continuous Integration (CI)

- Time to Take to push.

Code Taken some Time to push. (100 LOC or 10000

Check code by building on the server no

personal computer.

Continous Delivery (CD)

keep deploying to QA / customer (server)

Version Control

- Normally putting versions on FTP is time taking. Can't rollback efficiently. (on one file or specific place)

- We use branches.

main —o—o [latest versions]

Feature 1 —o local version 1.0
Feature 2 " —o—o

} They are later merged to main

- Manage using repository.

Agile Planning Azure

- Two tools for Project Management

Monitoring & logging

- If a server is too busy / large amount of queues. Traffic Manager / Monitor.
- Check in log (the query causing slowness)

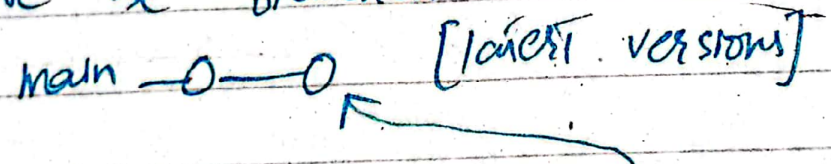
Continous Delivery (CD)

Keep deploying to QA / customer (server)

Version Control

- Normally putting versions on FTP is time Taking. Can rollback efficiently. (on one file or specific place)

- We use branches.




- Manage using repository.

Agile Planning Azure

- Two tools for Project Management

Monitoring & logging

- If a server is too busy / large amount of Queues. Traffic Manager / Monitor.
- Check in log (The query causing slowness)

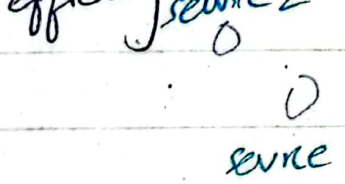
- VSphere (Application insights) for Monitoring.

Infrastructure As Code (IaC)
↳ file with info of Architecture.

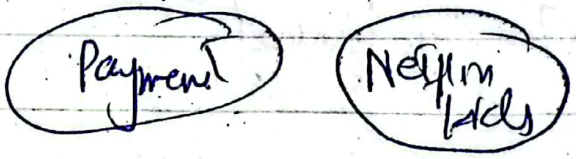
- Microservices architecture / containers.

. Use containers as different micro services & deploy services alag se

scalability ↑
efficiency ↑




Like Netflix




---

Types of Projects where Devops applied

- On New Projects : ① Waste of cost on customer left. ② can apply.

- Existing (Brownfield) Projects (10/15 y/o): People don't want to change. → customization/enhancement → Already process exist.

Greenfield : Newborn Projects. (start up) → new methodology
↳ No old constraints.
↳ can make new rule
↳ on premises / cloud marji
(local)

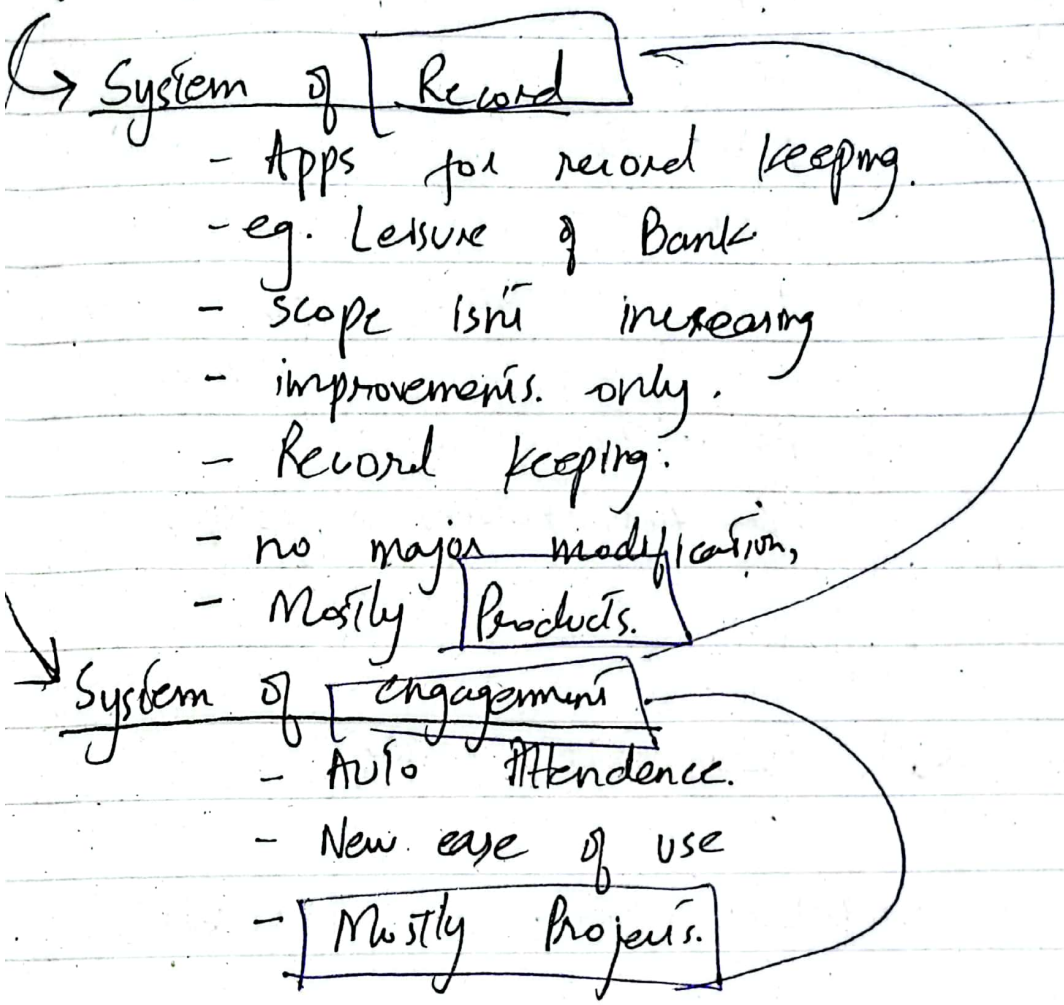
- It can be applied on every project
- DevOps makes the process better. Day 1 you consider deployement.
- Depends on cost

## Brownfield Projects

- Bounded by constraints.
  - bugs in codes get inherited
  - old logics exist in code
  - rectify other's code
  - de "technical Debt"

- Should we apply? Probably Not. Bugs will increase. Technical Debt ↑
- People don't like change, so people won't let DevOps
- time ↑ bugs ↑, apply it early.

## Types of System




## Types of Users

- Canary: Excited users who Test ASAP. Give them the build of earliest possible code. To get feedback. (May crash Hard Disk)
- Early Adopters: A little stable build
- Users: General users.

Deploy → QA → Canary → EA → Users.

## Azure Devops

- Cloud based solution
- Multiple tools: (1) Azure Repos
  (2) Azure Boards.
  (3) Azure Pipelines. (CI, CD)
  (4) Azure Artifacts. (Npm, external code)
  (5) Azure Test Plans.
  (Can Run Automated scripts)

## Github

Why? We had Azure Repos (TFS).

Github was competitor. Both are different completely.

Codespaces: Common environment to write code. (simultaneously)

Repos: " "

Actions: " "

Packages: " "

Later: called TFS
Now: TFVC (Microsoft acquired & improved)

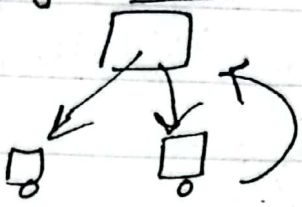
Shelving: store your local code somewhere to avoid HDD crash

## Source Control

Delta → Change set (files with changes)

(SVN)

(1) Centralized Source Control (TFVC) (CVS)




Download ↓ ↑ commit working copy

- No need of Backups. (Just server & copy)
- No individual repository.
- Internet (To upload) (1 server) Download.
  - i) Getting latest code from ☆ central
  - ii) Make changes
  - iii) Commiting

(2) Distributed Source Control (Git)

→ everyone has repository; make changes commit to personal repo only.

→ Problem: Sync problem. (because of personal repository.) local changes. (2 step)

→ No need of internet.

Team Foundation Version Control (TFVC)
- Branching mechanism — now TFS...

* Explore Git in Azure Devops.

[feature, PBI, tasks]

30th Aug 2023

More on Distributed Source Control

→ Clone: repository copied.
→ work on it. Change to original only when merged.
→ Not Automatically synced.
Cons: so many copies so hardware storage large. (but cost of hardware ↓)
→ objects are stored as deltas. (compressed)
Clone: Repo from Server + history (not in Central)
Pulling: getting new changes from repo.
Pushing: Moving your changes to repo

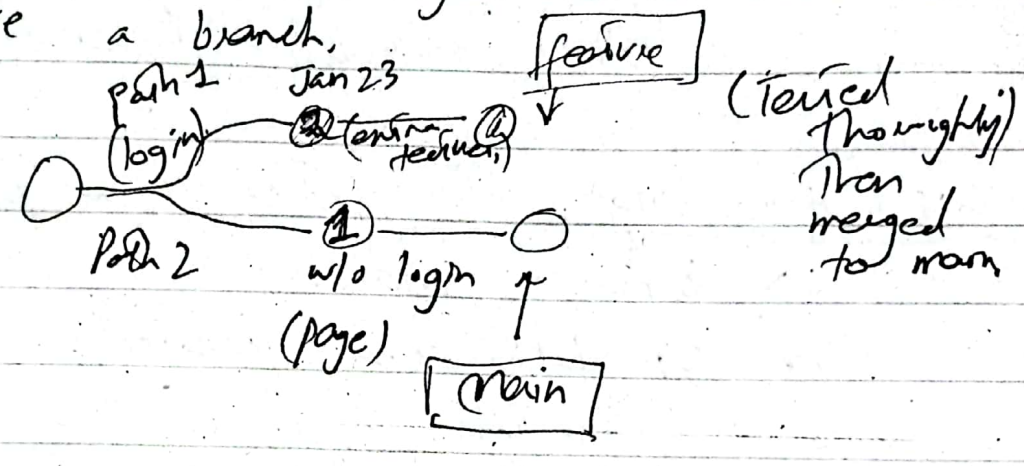
→ History: we can limit it. By default: life long.

Distribute ↑ errors ↑ (bcz of merging)
(To some extent)

→ When Preferred? Independent teams.

## Trunk Based Development

→ Don't push new changes to main. Instead make a branch.




→ Whenever make new changes, create branch then merge later.

Label: We tag those codes. (To check)

Branch: pathway different, with different codes. one stable code, one ~~test~~ on working

- Delete after some time
- History also gets merged.

Pull request: To merge 1 repo to other. or to review codes. / pair programming

: After review, we merge both branches.

- What about Database? (1) Operate scripts, Deploy at customer. Auto created

(2) Create Backup.
other methods?! (Research).

Git for large Files (other than code)
→ consumes space.
→ Git LFS was created. (for company)

Cmd
git checkout → copy code to feature branch from main branch.

saturday. Continuous Integration

Pipeline (One step solution).
like "Fetch - Decode" in parallel.

In Devops, pipeline has an input & output, each main configuration.
- Building code & publishing is time taking.
- so we are coming up with a repetitive technique.

→ When we push/commit it automates "build"
→ integrates all code at a centralized place. & Tells about failure if any
→ Test Automation: Automation Tests run.
→ Deployement Automation: Human factor ↓
Windows/Linux can have different builds.
so generalising is tough. Carefully build code.

Cons
* Dependencies: "can be handled." or can't be deployed if not tested etc.

All of this is automated { Unit test/code based/UI test/API Testing/ Selenium/Performance and stress test

UI Test → only on gfx. Deployment. (CD)
Other class Tests / bugs → (CI)

## Build Automation & CI

- Pipeline builds binaries (Assemblies / DLL files) That run on every machine. Generic executable files.

## Deployment Automation

- Not just on cloud, can be deployed on premise (local)
- low risk step (already tested & deployed locally)
- but risk bcz not deployed on large no of users)

## Azure Pipelines.

→ CI only. checks & ensures integration & successful code build.

→

If loopholes in frontend so check in CD, Otherwise check in CI.

SQL Injection → user write query on front end. (that manipulated all data)

→ Azure Pipelines compatible with most languages. (Marketing technique).

→ Version Control Systems.

. AzureDevops can be integrated with different products (GitHub, Gitlab)

→ Pipelines
Deployement Targets

- Azure
- Cloud
- AWS
- VMachines

## CD

C. Delivery vs C. deployment

↙ customer deployment

↓ kahin bhi. (QA server / developer server)

- CI gives you a published Build.
- CD uses CI's output & deploy

We want to integrate everytime someone pushes code. but deployment we try to do once / least no of times

→ We automate deployment.

Some key terms.....

1. Agent → software running on a system. that does things such as build etc.
   → It is a background service like windows service that automatically gets time / get new updates. It's like a mazdoor (Thekedaar)

2. Artifact → output of CI pipeline.

3. Build → when someone pushes, we get Build 1, then Build 2 (Build version)
   Artifact ki versioning

4. CI/CD → " "

5. Deployment Targets. (Where to deploy)

Agent does build etc. This is called "job"

→ Creating CI Pipeline (Classic Editor) (video)

Asp.Net Project (IT automatically gives template & Tasks)

- NuGet
- Build Solution
- :
- Publish symbol paths (Artifacts created)

Pipeline Variables → for configuration

Triggers → Enable CI. We can customize time of build.

Run Pipeline: Do Agent settings (Hardware / OS)

Upon saving, code can be built, artifact created

Note: It's upto us whether we build on each push or put in Queue. (for small Teams it can be good)

→ Azure Devops do everything on an Agent.
→ Agent is given Tasks (Jobs). such as build.

## Pillars of CI (Important)

We need 4 things To implement CI.

1. Version Control. (Git etc)
2. Pkg mgmnt (CDN, NPM)
   ↳ can update easily
3. Tool that has CI features.
   ↳ Jinkins
   ↳ Azure Devops
   ↳ TeamCity
4. Automated Build process (Release)
   ↳ Apache Ant
   ↳ NAnt
   ↳ Gradle

For Pipeline creation,

1. Classic Editor (UI)
2. Yaml File (customization ↑, more used ↑ works everywhere)

Other method to create Pipeline

↳ YAML (it has limitations)
(see table in slides)

---

Agents (Two Types)

↳ Microsoft Hosted Agent
- ↳ Microsoft Machine (Build server)
  - It has all things/softwares
  - ↳ On error (4 minute cut off time)
  - ↳ not free.

free version has limits ←

↳ Self Hosted Agent
- ↳ your own laptop/cloud/Hosting
- ↳ we maintain everything
- ↳ no limit of build. (no cut off time)

Which is best?

Microsoft has your code! Security issue. Most companies self host. But self host is hectic. Microsoft destroy VM's automatically. But in self, we do it, we maintain.

## Job Types

- Container Jobs.
- Deploy Jobs.
- Agent Pool Jobs. → list of Agents.
  - ↳ self / ~~MS. Hosted~~. Pool of Agents is assigned tasks. Windows / Linux pools. Put in Queues if busy.
- Agentless jobs
  - ↳ MS. Hosted. Everything Automatic.

## Communicate with Azure Pipelines (Agent)

- Agent tells every after every few seconds. to that it has not Tasks. (Pull Notification)
- Agent that ask first get tasks.
- Communicate with each other
- Works on SSL port
- Pull Notification. (to reduce load on Azure Pipelines)
- Push Notification: Server asks agent: DONT DO! load ↑

→ Token based communication: login with Gmail. token generated. user validated (example)
→ Task finished, token destroyed.
1 token → ping (initially)
2 token → task based.
Tokens get expired too (security).

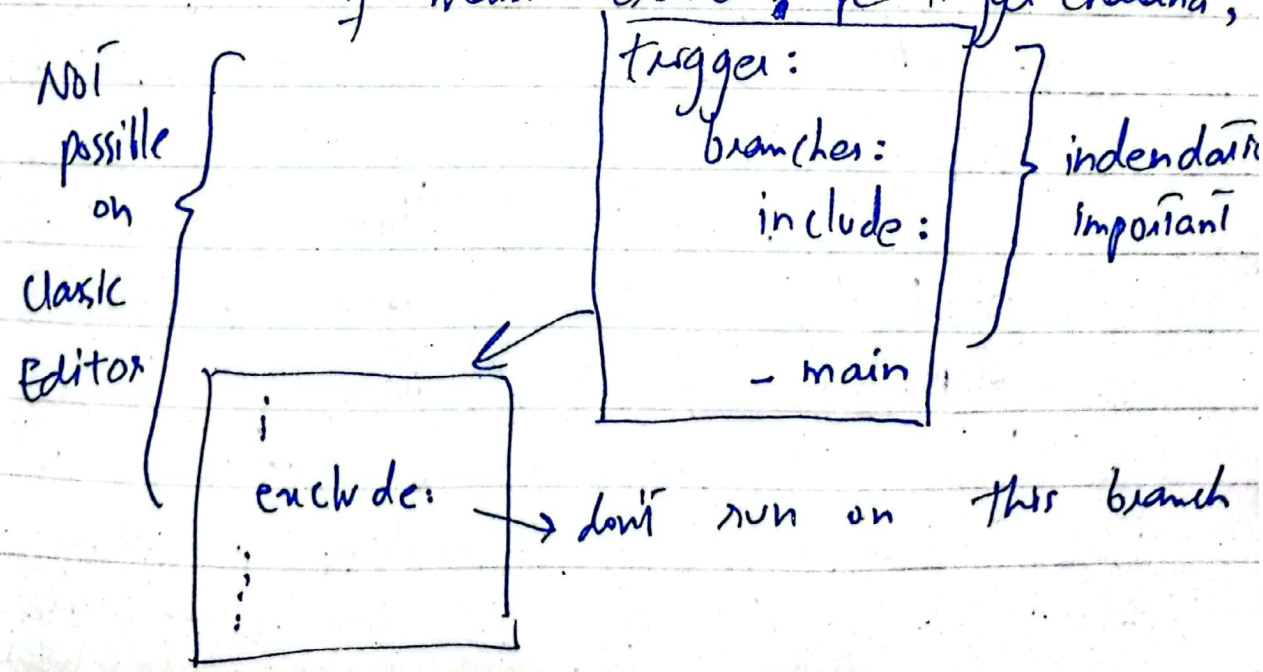
(video) YAML (communicate with Agent)

YAML File Format (Build#1, abc, dd-mm-yyyy etc)

① name ③ Jobs.
② which branch to be triggered on. (main)

o Triggers. (First write this)

if main branch pe trigger chalana,



~~Triggers~~

## Multiple Branches

```
trigger:
  branches:
    include:
      - feature/*
```

→ feature ki sub-branches

```
paths:
  include:
    - webapp/**
```

→ agar yahan changes honge tab hi execute hogi CI pipeline

Naya kaam ~~pe~~ feature branch pe

build, test (these are jobs).

**Jobs:-** (Sequential) but sometimes parallel.

By default, it is sequential

```
jobs:

- job: A
  steps:
    :

- job: B
  steps:
```

Sometimes same agent do both jobs. or not

→ testing independent can be done by different Agents.

(Important)

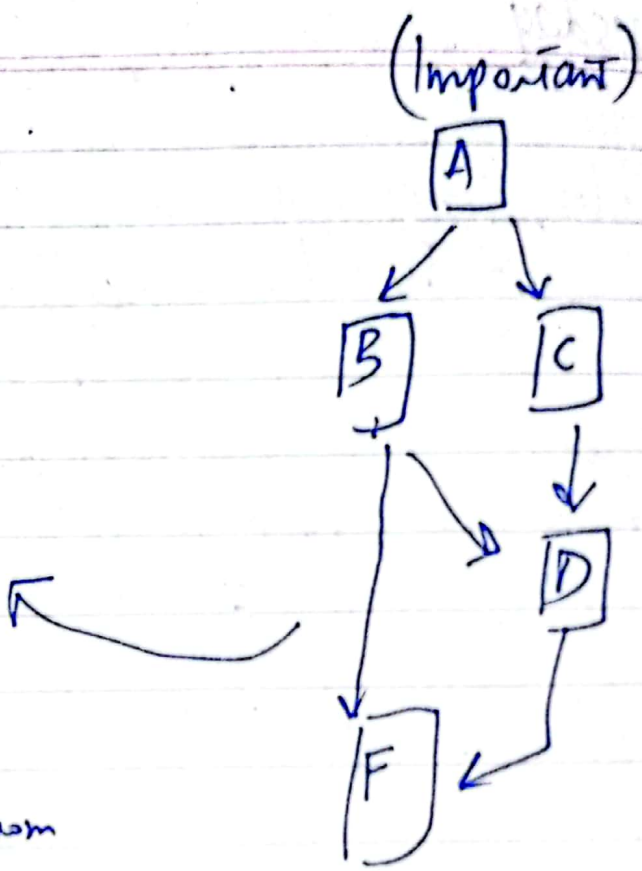
## Dependant Job

... A

... B

depends on : A

. . . . . .




Write code from dependency diagram.

→ Steps. (Tasks) kya chalana hai. Actual thing.
↳ ex. Nuget downloading etc

Stages → Jobs → Steps. → Tasks.

# Tuesday

## Jobs (continued)

↳ Build
↳ Test
} requires tasks to be performed.

Tasks
- Should be Atomic. (sab chalenge)
- 1 job poori 1 machine pe. (mini tasks inside)

- dependsOn (optional)

## Steps are Tasks

- copy x to y
- zip.
} ex's of tasks.
- powershell ki script. (diag & drop)
- some tasks are predefined.

## Stages

↳ Build
↳ Test
↳ Deploy.

Single kam → Task
Multiple tasks → jobs.

Ex We need to Build on 32, 64, Linux jobs

job1 job2

to Build → Nuget, NPM → Tasks.

Different machines.

---

## Multi-Pipelines

→ Let's say web App, Mobile App, API service
↳ 3 pipelines here, we want to build at once
↳ 1 YAML file.

resources:
keyword: pipelines.
repos
containers.

---

→ For reusability in YAML jobs, use template for common Task.
→ similarly in steps, stages ....

Pipeline Stage (Slide)

Stage: Build
"
"
"
"

} CI

Last Stage (CD)

- CI + CD = 1 YAML (in Github)
- 2 different (Azure).

YAML - container → VM ka cut-down version

↳ container: linux
image: Ubuntu 16.04.
ports map hongi

---

resources:
type: Github

Wednesday

# Git

→ **Branching**

When? → Team size ↑, Multiple improvements on a product.

→ V2 has more bugs, so we have V1
→ Main/Master → most stable branch.
→ No fixed rule. Depends on ~~its~~ developer.
→ Feature branch pe kisi specific functionality implement karna.
→ Advantage? ⇒ Use a B. mechanism which doesn't make the process slow.
→ In this course, 2 mechanisms.
  (1) Trunk Based development
  (2) Forking workflow.

(Distributed)

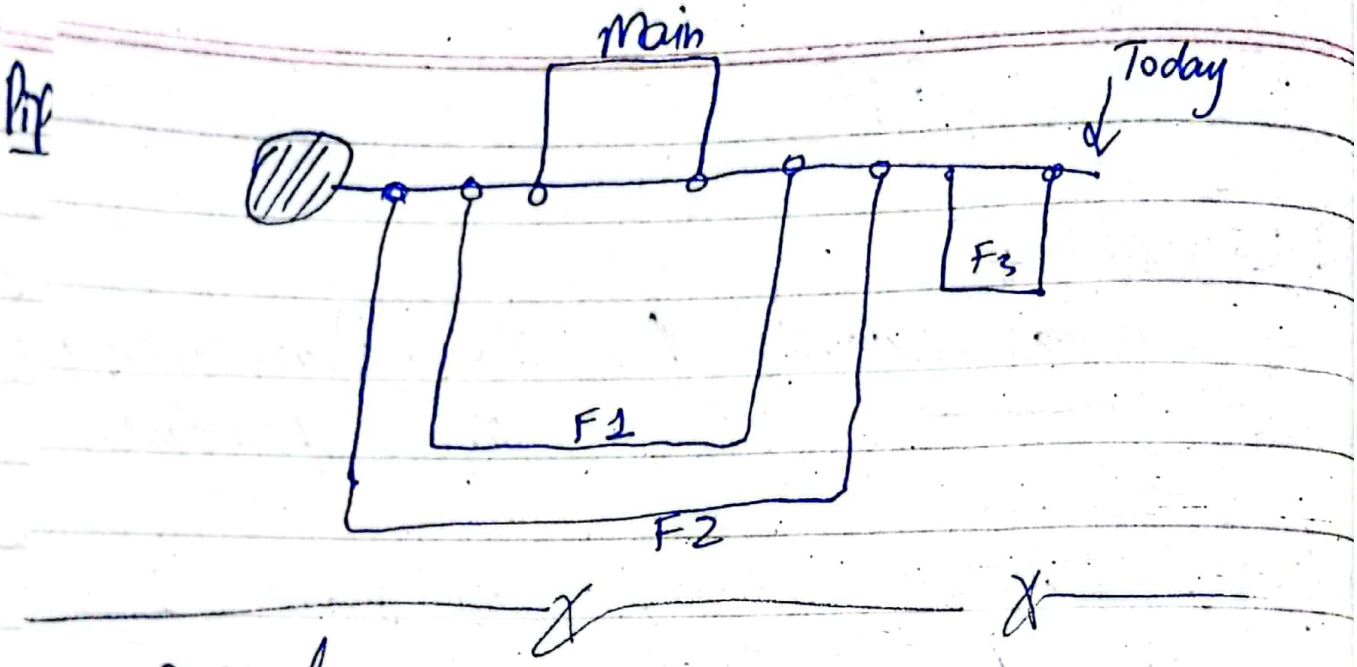
## (1) T. Based Dev.

↳ Main Branch
  ↳ login
  ↳ logout
} feature branches or Trunks

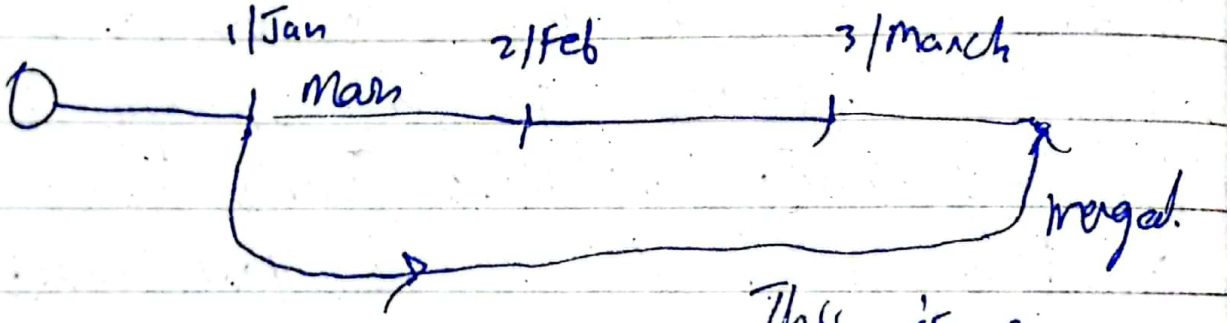
- One developer can have more branches to work on.

Hot fix: ~~for~~ release with fix only. (bugs removed)

retro fix: like Hot fix,




Centralised




This is a problem

So we can branch from each month's release.

## Tags

- Tags are labels. We comment when we push. Becomes easy to search. Go to label if we want to go back in time

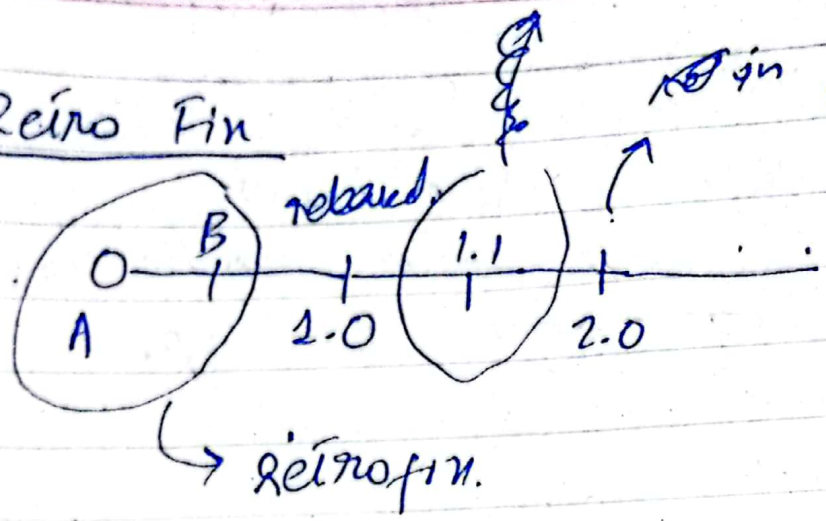
History also gets merged.

→ We merge the feature branch in main.
Should we delete feature branch?
↳ Maybe. Several feature branches complex, on huge project. Disk space:
↳ Maybe not.

## Forking Workflow (better)

Difference with clone: We can't push into main. We have a copy.
- Collaborate. Kaam kay liay use hoti hai.
- Third party / Open source code.
- Request is sent to merge.

## Retro Fix




## Add Commits

## Open a Pull Request (PR)

- when merging we make a PR (both TB & Forking)
  - for code reviews.
  - to take help.
- rather than sending code, send PR.

## Github Actions (video)