GIT & GITHUB

SCM/VCS

DVCS

CVCS

CD

App → Code

feature

Deploy

Workstation

IN

Web Server

Production

Life-Cycle

Source Code Management → SCM

Timeline

1 month

1 week

print.....

print.....

print.....

file/code

WorkSpace

WIP

Work In Progress

Working Area

Lines

30th Jan → 10

31st Jan → 20

1st Feb → 30

2nd Feb → 40

marker

* Reputation/Loss

Result of Crash

Backup

Storage Area

Commit Area

Restore

* Plan the backup

Backup Timeline

Rollout

31st Jan

1st Feb

6th Feb

7th Feb

Version 1

Version 2

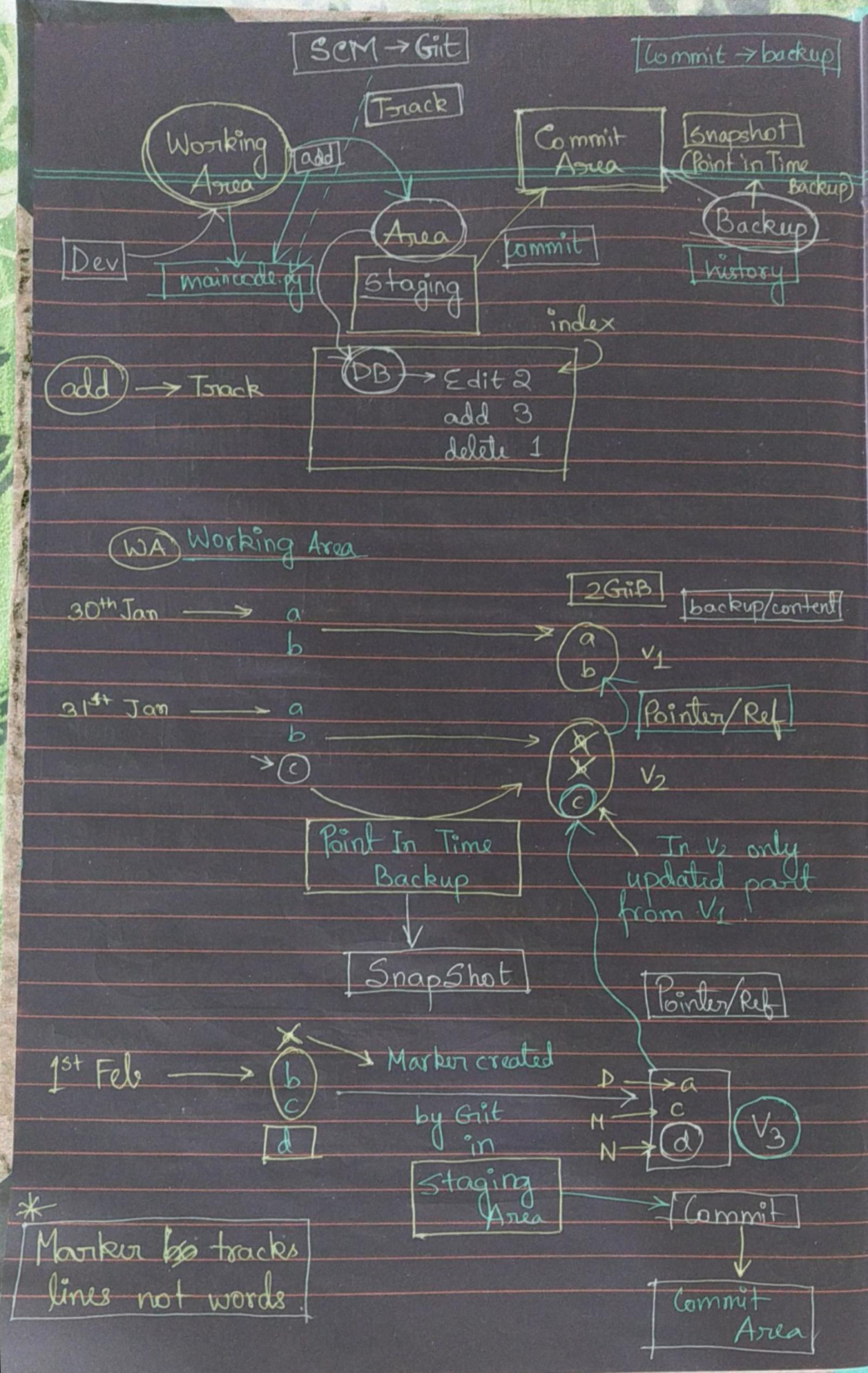
Version 3

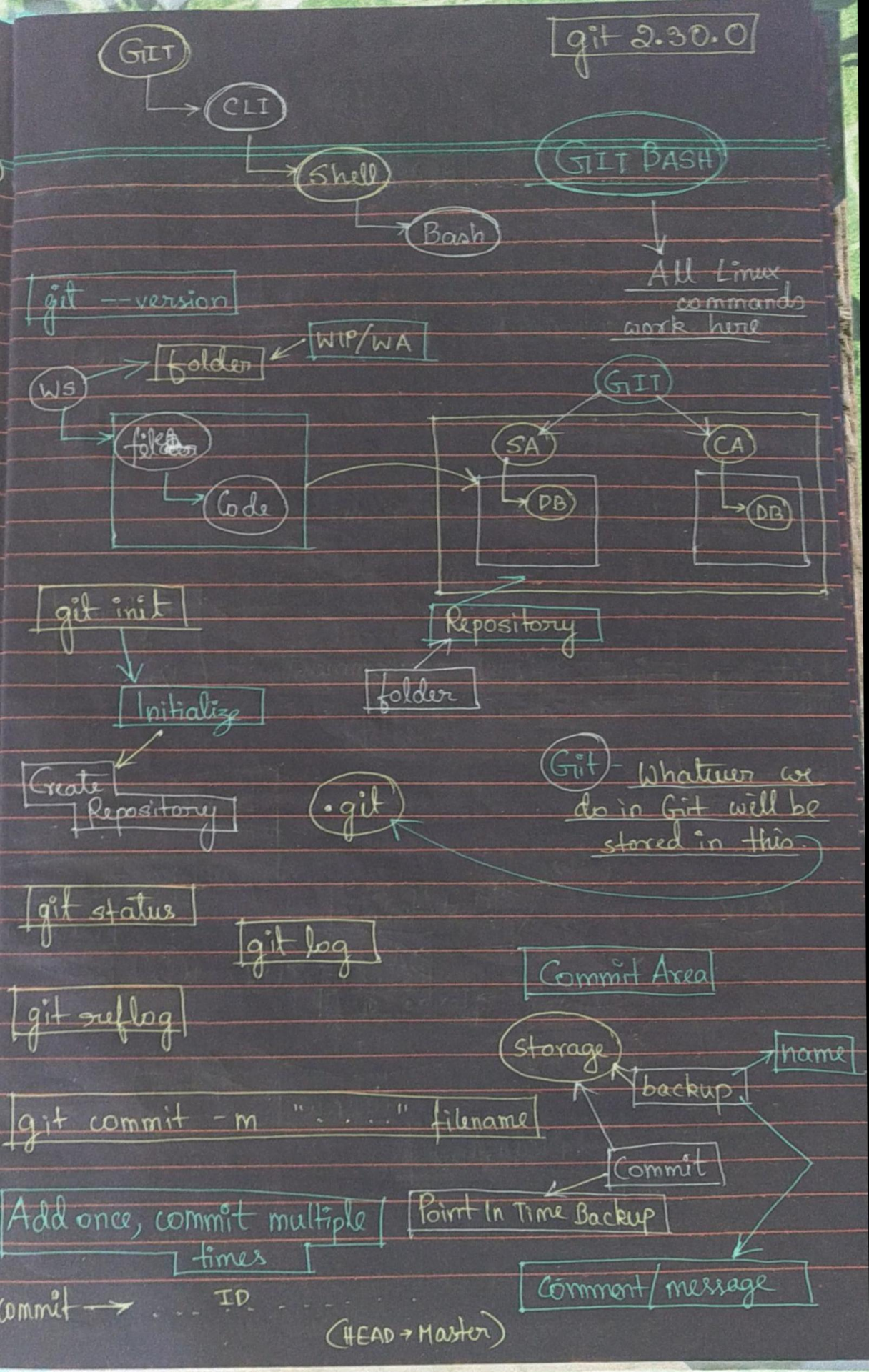
Version 4

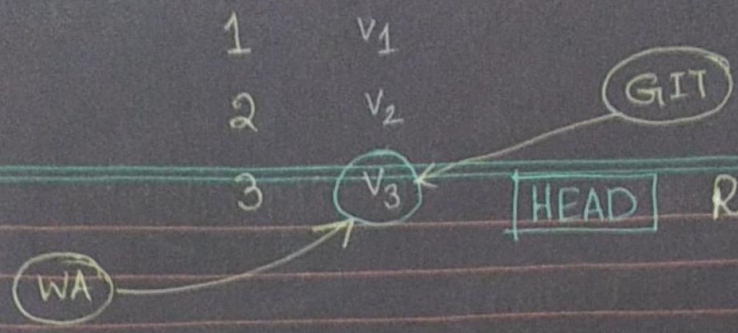
RollBack

VCS → Version Control System

SCM


SCM → Git
Commit → backup
Track
Working Area
add
Commit Area
Snapshot (Point in Time backup)
Backup
history
Dev
maincode.py
Staging Area
commit
index
add → Track
DB → Edit 2
add 3
delete 1
WA Working Area
2GiB
backup/content
30th Jan → a b
a b
V1
31st Jan → a b c
Pointer/Ref
V2
Point In Time Backup
In V2 only updated part from V1
SnapShot
Pointer/Ref
1st Feb → b c d
Marker created by Git in Staging Area
D → a
M → c
N → d
V3
Commit
Commit Area
* Marker tracks lines not words.


GIT
CLI
Shell
Bash
git 2.30.0
GIT BASH
All Linux commands work here
git --version
WIP/WA
folder
WS
folder
Code
GII
SA
CA
DB
DB
git init
Initialize
Create Repository
Repository
folder
Git - Whatever we do in Git will be stored in this
.git
git status
git log
git reflog
Commit Area
Storage
backup
name
git commit -m "....." filename
Commit
Point In Time Backup
Add once, commit multiple times
Comment/message
commit → ... ID ....
(HEAD → Master)



git reset commit id filename → Rollback to Version 1

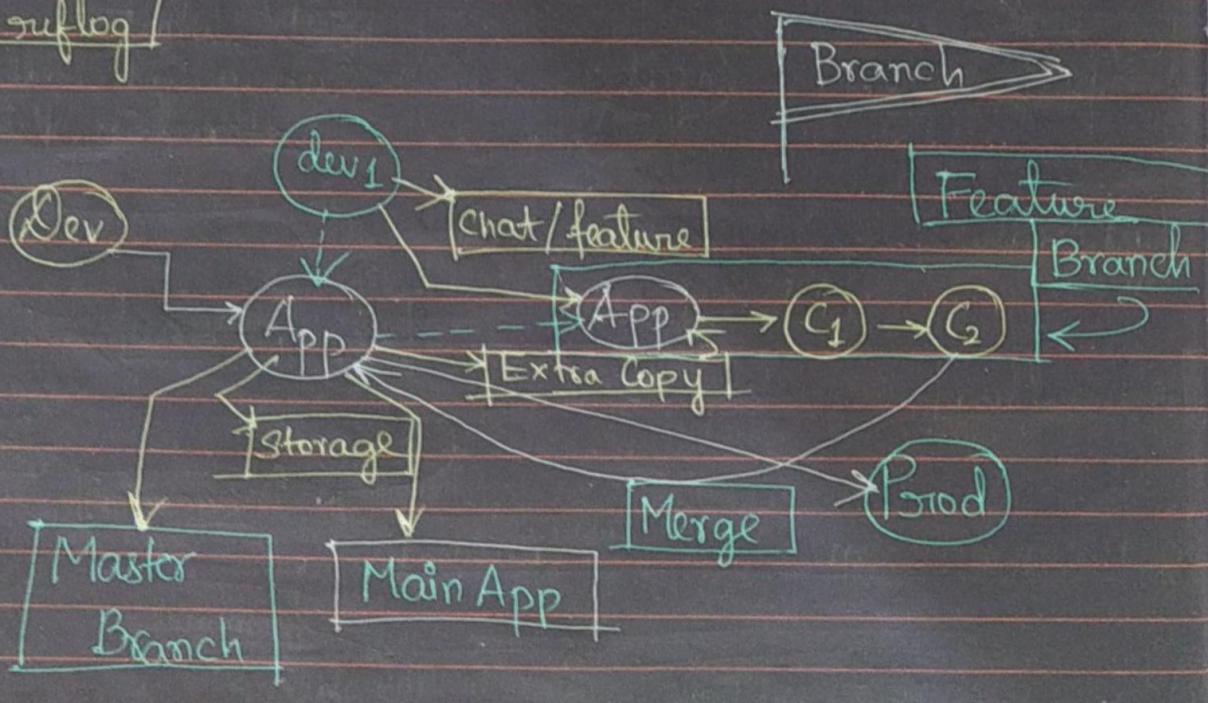
git checkout -- filename

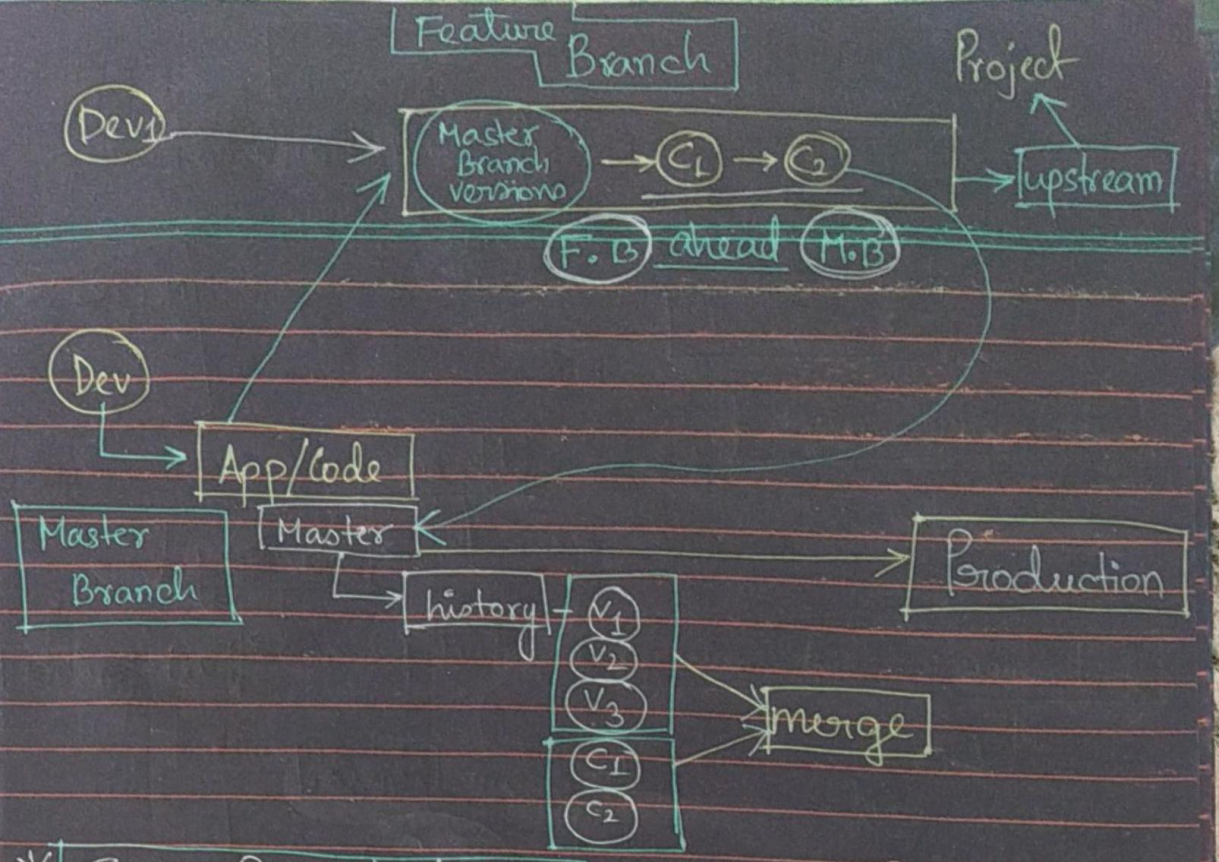
git reset commit id filename

git checkout -- filename → Rollout to V3

git log → --oneline → Comment
→ Entire History

git reflog



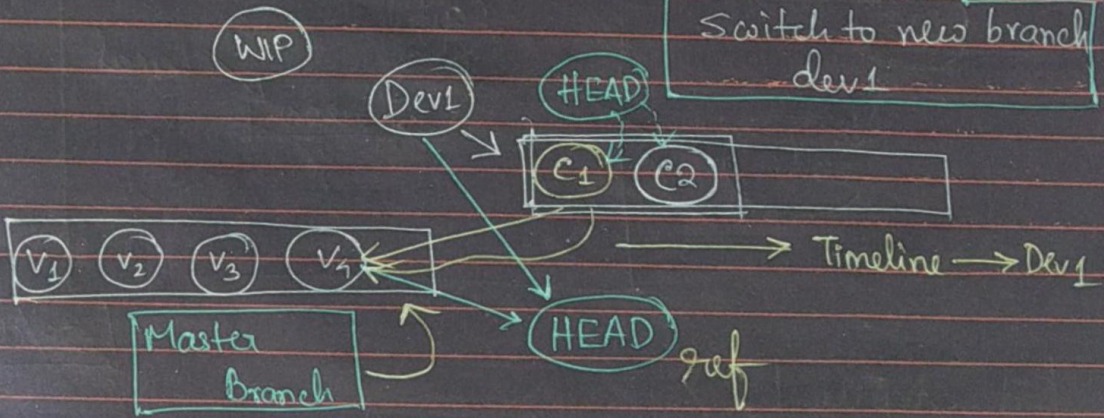


* Every Branch have their own personal timeline

git branch

git checkout -b dev1

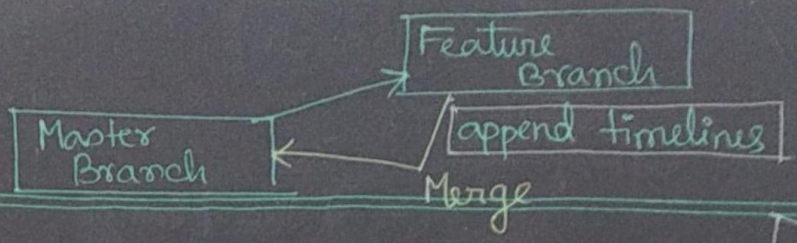
Create and switch to new branch dev1



git branch -h

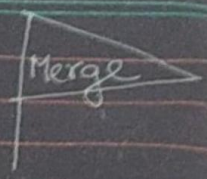
git branch --show-current

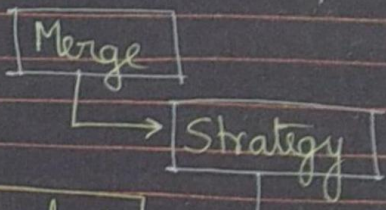
git checkout master



git checkout master

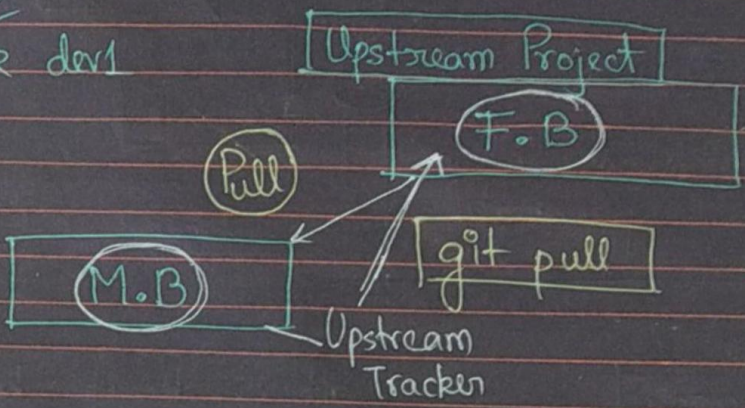
git merge dev1

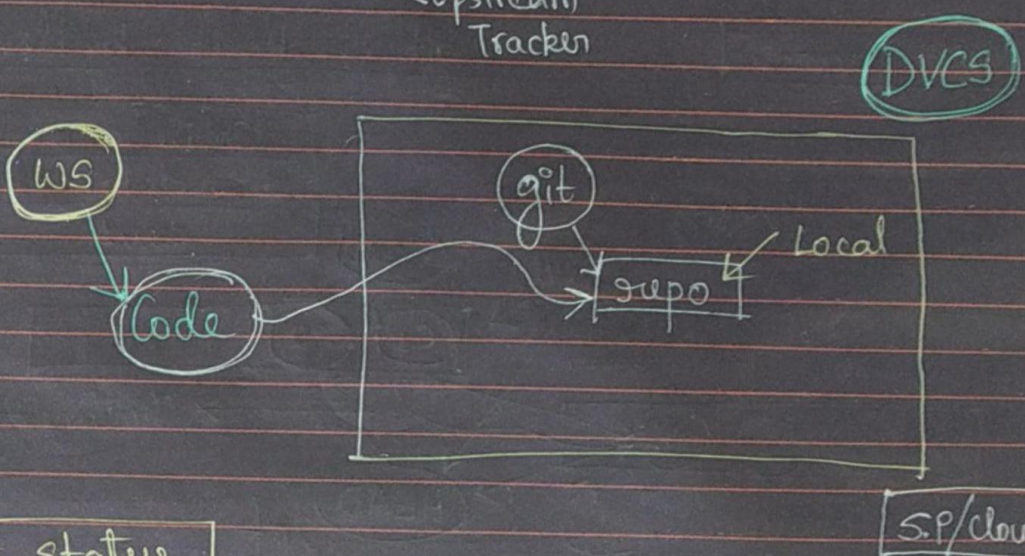




git branch --set-upstream-to=dev1

Track dev1





git status

Only work if upstream set

git pull

If git tracker set for upstream branch, this command will show me if master branch is behind or at par with upstream branch.

SCM

GIT

VCS

version / ref / commit ID

local ← VCS → repo

T1 T2 T3

clone clone clone

IP

Centralized Server

Private Public

Own Team

Centralized → Anybody can connect

CVCS

OWN → IP

S.P / cloud → GITHUB

repo

git init

local repo

WS

.git

VCS

VCS

VCS

App

github → REMOTE

repo

git add .

If we want to upload our local git repo to CVCS we should not initialize GitHub Repo.

GITHUB

`git remote -v`

`git remote add` (name) `githubrepo.git` → Connect local repo to cvcs repo

`git push mygithub master` [.git folder uploaded]

Myth: `origin` is a predefined keyword

`git push <name> <branchname>`

Multiple name

Multiple Repo

mygithub

my...

Remotews

Remote Repo

Origin

Github

Localws

Local Repo

PUSH

PULL

Git

Different Branch

Then we decide where to push.

Different Branch

`git pull <name> <branchname>` → Download Update

`git fetch` → Download timeline from Remote System

`git fetch` → `git pull`

git bash → Credential Manager

`git branch --set-upstream-to=mygithub/master master`

`git fetch` `git status` `git pull`

git show commit id

git diff fbc3 c6be

git diff c6be fbc3

To check
extra addition, deletion....

Git diff tool → p4merge

Local Master Branch → Remote Branch

(ahead) dev1 Branch

git push --set-upstream mygithub dev1

Merge = Compare & Pull Request

dev1

master

base: master ← compare: dev1

Merge Pull Request ← Create Pull Request

Merge Confirm

git config add user.email

" . . . . . "

git config --global -e

git clone repo-name