

WEB WARRIORS

PHASE - ☺

Date |

Page |

Video

HELLO WORLD — 25/06/2025

How computer get know about website that searched on a search engine?

Ans → HTTP — Hyper text transfer protocol

A ~~link~~ link that ~~we~~ have another link that open another web page

→ Stateless protocol

Every time you ^{visit} visit a page, you are a new user.
No memory of prev. Req.

There is no protocol or rule in ~~state~~ HTTP where state can be maintained.

[THERE are a plethora of protocols]

→ **Sessions**

A session like a class with a teacher, so Teacher only teach to certain Student for that session.

↳ It can store a state of connection of Browser and Server.

→ **Cookies**

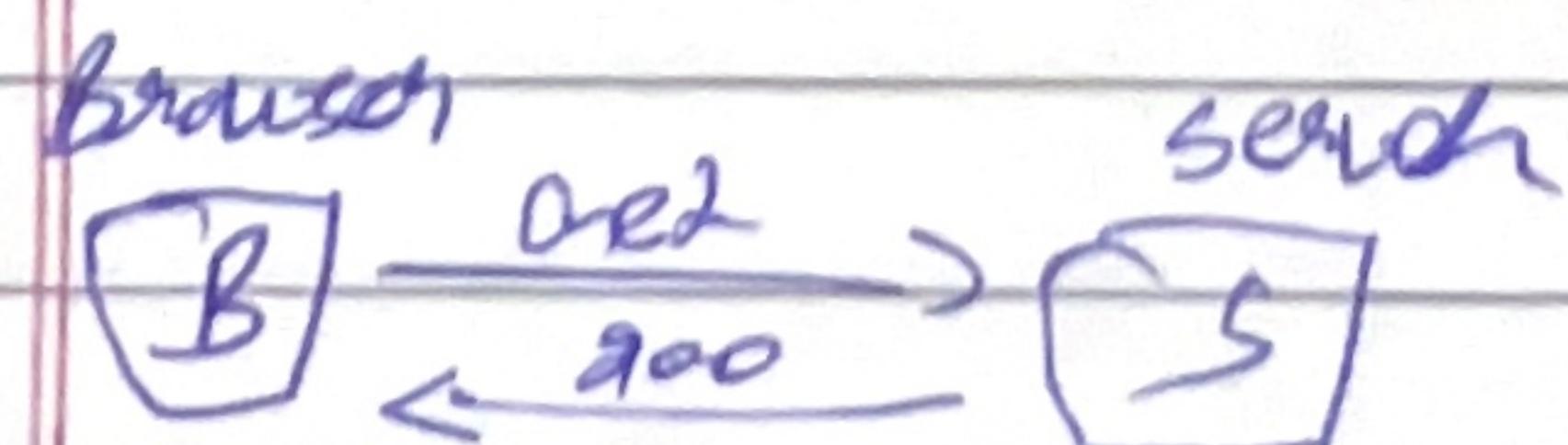
Cookies is a key: value pair. It is just information.

→ **HTTP Header**

Headers also an information. Such as

- ↳ Client
- ↳ Browser info
- ↳ Date time
- ↳ Cookies to store.

→ **Request Response Model**



Request → Response

Types of Request

GET — to get data from server

POST —

DELETE —

when sending a request from browser REMEMBER

- what action to perform (Get, Post) → this is called Verb.
- where to perform (which website to go)
- was it done? — send a code whether response is OK or not OK.

HTTP/2

Replaced by HTTP-2 used more than 80% today.

1) HTTP-means HTTP/1

↳ Fallback

HTTP-1.1 is ~~Fallback~~ & still used.

2) It uses compression. (zip - decrease size)

(there is not released because majority of HTTP 1.0)

3) Encryption [S in HTTPS]

EVERY-thing in computer has Protocols and can be broken.

Date: _____

Page: _____

Kevin

SSL/TLS (TLS)

Every little computation has cost so we don't use encryption in AWS. [in internal servers]

→ what is TLS and TLS certificate?

TLS - Transport Layer Security

Cert. certificate.

HTTP

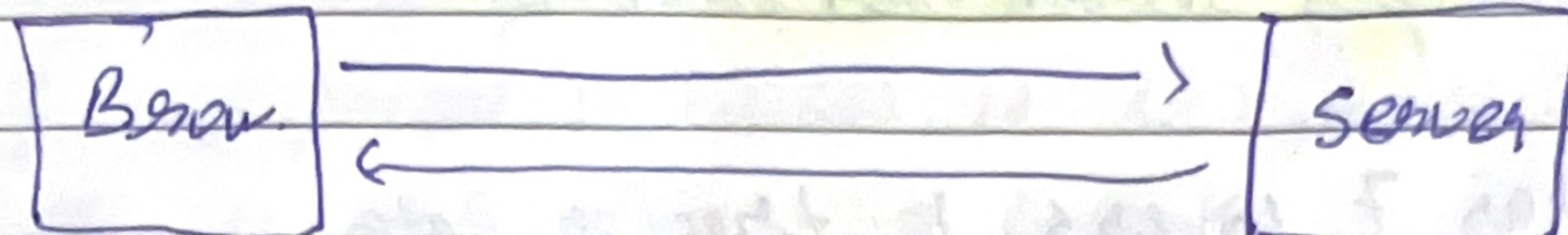
HTTP + TLS → HTTPS

also called TLS certificate and HTTP certificate.

Jargons

- 1) User Agent - Browser or any code that send info to server
- 2) TCP - Transmission Control Protocol
- 3) FTP - File Transfer Protocol
- 4) IP - Internet protocol (name of pc on internet)
- 5) URL - Uniform Resource Locator
- 6) DNS - Domain Name System (server)
- 7) It points URL to IP
- 8) Header - (Pass additional information) - meta data
- 9) Payload - (Actual data - email, password)
- 10) Cache - (Store the data) - temp storage

Enhance about more jargons with us.

Recap

- 1) Set up TCP connection
- 2) Exchange TLS certificate
↳ (HTTPs)
- 3) Send ^{Request} Verb + URL + Data + (more)
- 4) Gets the response back with status code \Rightarrow and Data
(img, CSV, text)
- 5) TCP connection is closed. (stateless)

THAT'S the web

- 1) Cookie vs Cache $\xrightarrow{\text{cache data temporary}}$
- 2) How many bytes in computer? — 65,536 bytes.

Don't share cookie with anyone

Video - 9) Introduction to Web — 27/06/2025

OSI — Open System Interconnection

↳ It has 7 layers, to transfer data.

- 1) Physical Layer (works on) — 0 - 1
- 2) Data Link Layer (works on the Frame)
- 3) Network Layer — (Forwarding, Routing) — IP Addressing
- 4) Transport Layer — (TCP, UDP)
- 5) Session Layer
- 6) Presentation Layer] - Application Layers
→ Servers.
- 7) Application Layer

⇒ What is TCP and UDP

(What is Trade off)

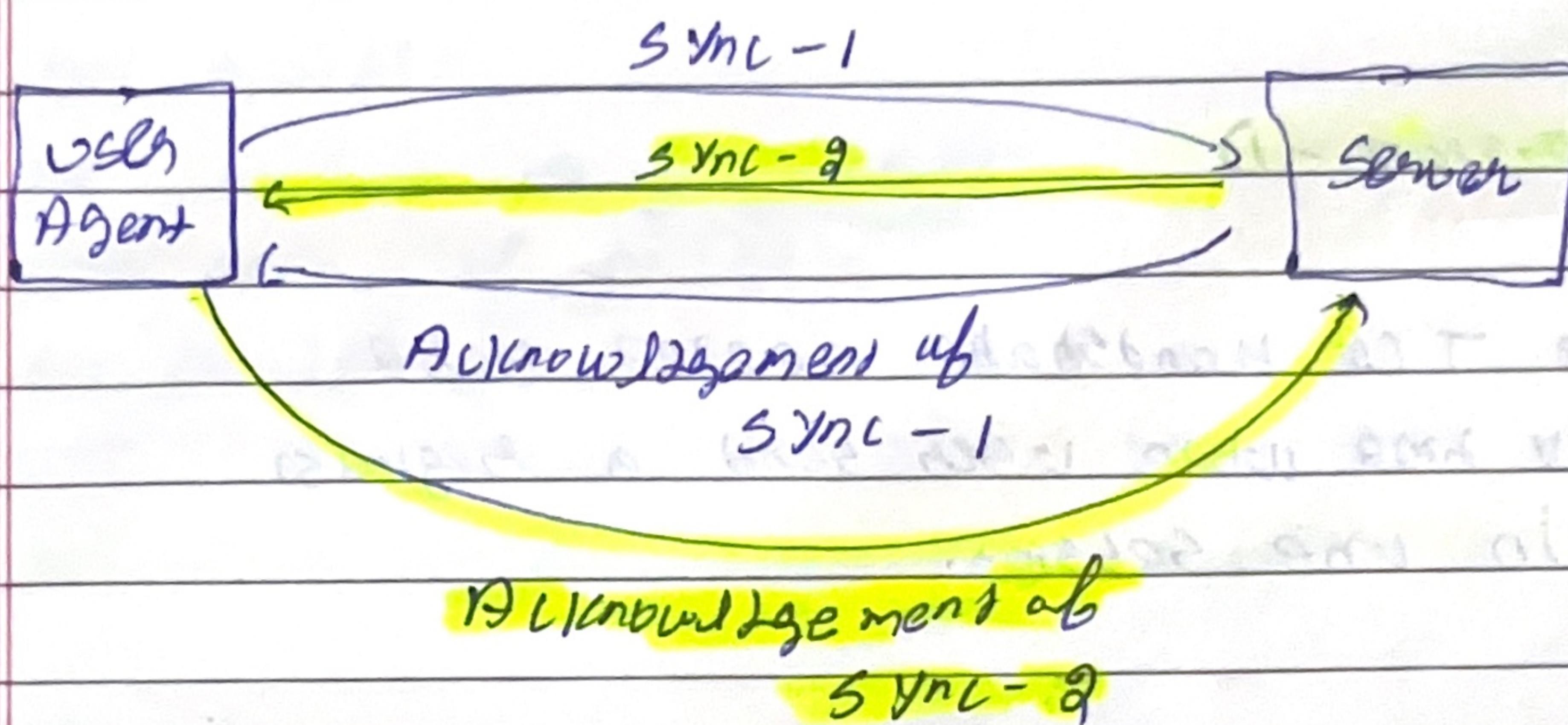
TCP — Transmission Control Protocol

- 1) Reliable
- 2) works in ordered way
- 3) 3-way handshake

Client always have data in chunks also called packets that user to transfer to server.

TCP

- 1) ensures every packet is sent.
- 2) if any packet is missed, it sends retransmission.
- 3) packet will also order the packets.



=> Server send SYN - 2 and ACK of SYN one in a response. (It merges both so send once in spite of two requests)

= SYN - Client



SYN + ACK - Server



ACK - Client

After this - a (GBT, POST begins.

(meaning of offload)

HTTP

versions ↗

V1 — V1.1 — V2 — V3

V.1 (version - 1)

- ↳ Here TCP Handshake needed to be done every time when user send a request even in one session.

V.1.1 (version - 1.1)

- ↳ Here, TCP Handshake required only once in a session.

V.2 (version - 2)

- ↳ It has a feature of Multiplexing which means user can send multiple types of data.

Overhead — Above your head, fixed costs in a company.

Latency — how much time it takes to send requests.

- ~~in TCP~~ it takes more time but reliable.

UDP

- 1) Faster in time
- 2) not reliable
- 3) no checkup ~~all~~ of packets
- 4) no order of packets
- 5) no handshake

→ What happen when type google.com? Client

→ trying to make GET request to use google.

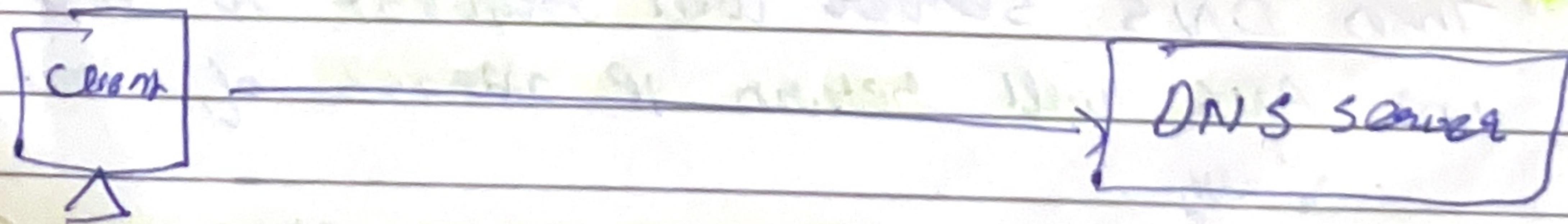
1) To convert Domain name into an IP address or to get IP address ~~so~~ Browser sends request to DNS servers.

2) DNS
It goes to its database and return an ~~not~~ IP address.

IT IS CALLED DNS RESOLUTION

How DNS works

internally?



- 1) DNS send request to **ROOT SERVER**
 - there are only **13** Root servers.

- 2) **Root Name Server** (Root servers)

→ It ~~will~~ return IP address of **Top Level Domain Servers**.

→ **TLDs** (TOP LEVEL DOMAIN SERVER)

These are written below!-

.COM, .UK, .ai, .in

L () L ()
 ↓ ↓
 j j

These are **TLD servers**.

- 3) ~~The~~ DNS server will send request to TLD's particular server and then it will return IP address of that specific domain's (charcode) **ANS** (Authoritative- Authoritative name server)

ANS example

↳ GoDaddy

- 4) Then DNS server will request so that ANS and ANS will return IP address of that godaddy.

certain website. (In our case charcode.com)

DNS ^{server} works on UDP on Port No. #53

USER DATAGRAM PROTOCOL

Browser ~~will~~ store DNS (IP address) in Cache.

(Q) -> what was pipelining?

Video-3) Git and GitHub Master Class - 23/06/2025

→ VCS → Version Control System

↳ It can maintain history of local code in local machine.

→ Name of some VCS

1) Git

2) Subversion

3) Mercurial

4) Perforce

5)

Git

After initiating git creates an .git folder which contains .git that stores every little changes and history.

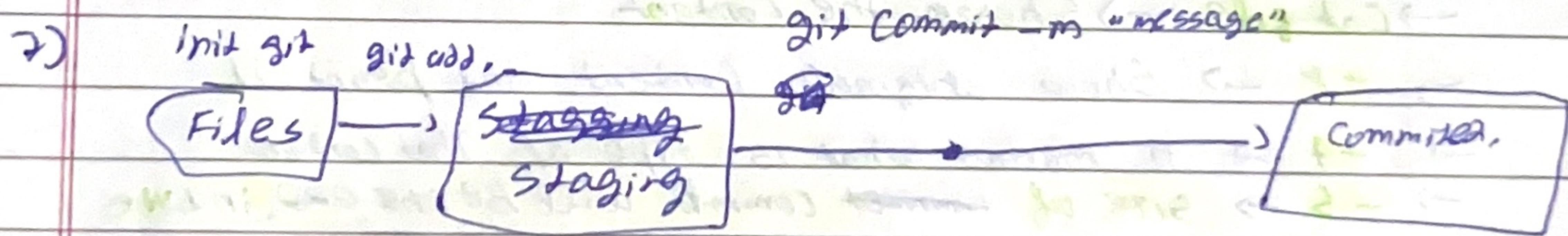
1) U - Untracked - by default git does not track any file so to track file `git add` command can be used.

→ `git add filename`

2) A - ^{Index} Added - That certain file will be added for tracking.

- 3) `git add *` — it will start tracking all files.
- 4) There is a compressed file in .git (hidden folder) that is titled `index`, it ~~will~~ is same files that are added to track.
- 5) to read file command is — `cat <filename>`
- 6) Command — `git status`
 It reads git folder and show in a good LF.
 If you do not want to track it will show a command to ~~track~~ untrack files, ~~it~~ it will be removed from index folder.

ALL THESE COMMANDS CAN READ OR WRITE .git FOLDER.



Commit — it is like a check point or snapshot.
 — Every commit has a ID and hash

8) `git log` — it check history and shows hash of the commit and every related info.

9) M = modified

10) git diff - It shows what change has been made. Shows difference between new and older version of a specific line.

11) objects folder in .git folder

↳ This folder will have a compressed file that is committed in the folder so every commit will have folder named first two characters of its hash code.

12) git cat-file -p <hashcode of specific Commit>

This command will show content of that file as commit.

→ Cat-file → Shows the content

→ -p → Shows original content and print it

→ -t → It means what is type of that content

→ -s → Size of ~~commit~~ commit will be checked in bytes

13) TOP MOST IS ALWAYS HEAD

↳

So everytime a new Commit enters it will become a Head.

Every Commit has reference of it's ~~last~~ past. (last Commit).

14) To see difference b/w two specific commit the command is:-

→ `git diff <hash code> [hash code]`

You don't need to write complete hash code, it will work with ~~• first 5-6 characters of hash code.~~

15) How hash b/w ? Become

It reads code file and hash it (Hash SHA1) and makes it a commit.

16) What is Hash ?

~~Hash~~ Hash is an ~~algorithm~~ ^{function} that can ~~make~~ any text or file ~~into~~ into a string.

git reads every file and make ~~it~~ a ~~hash~~ Hash so it becomes a unique Id. Hence, until -local does not make any changes in their code the hash will not change. Once a change/commit occurs Hash will be changed because it will check all files and a new change will be shown and that would make a new hash code.

17) MAYA - mostly advanced yet acceptable.

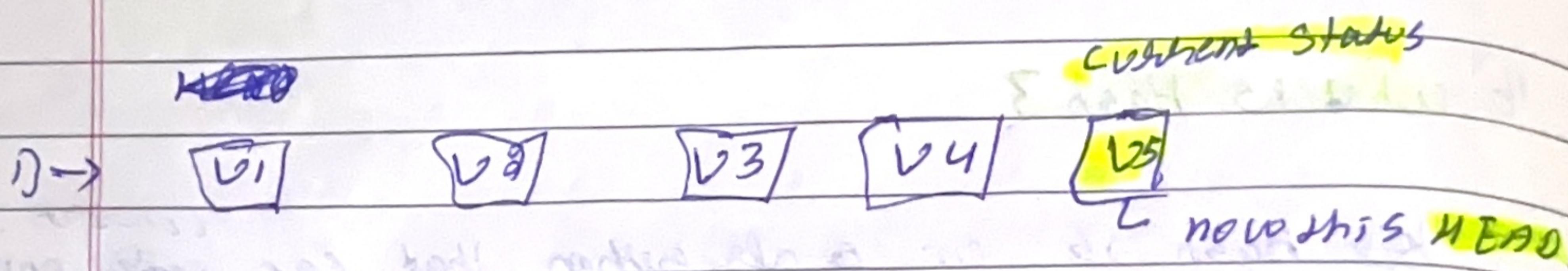
18) How to go back to a past commit (How to gain past)

git reset <hash code of that commit>

above command will change the Head. It will add changes of ~~future~~ (future) then update that were once a present will go into ~~past~~ changes.

in simple - Those changes will be removed from commid.

for example -



2) → Now I want to time travel and go back to [v2] then I will use following command

git reset <Hash Code of v2 commit>



Now Head is version of [v2] and

[v3] [v4] [v5] will be removed and they will go into changes. These 3 files as commits objects will be removed from object files of .git folder.

19) To permanently delete or discard lateral commit
the following command will be used! -

→ git reset --hard <hash code>

Now those changes can not be taken back. --hard
means ~~permanent~~ permanently deleted.

20) git log --~~one~~ oneline

It will show history in one line. All commits and
their msgs in separate lines in a clean way.

21) config file in .git folder

There is some data in this config file.

Q3) added command from Github :-

git remote add origin -----
^{'link'}

That above mentioned command will add a link so
no ~~is~~ it's remote.

Q3) ~~git remote~~ → git remote -v

It shows Github repo and that means accessing
to my branch from this -

PULL

If you want to fetch changes, do from here

If you want to PUSH change, do from here.

PULL → Fetch

PUSH → Send

Q4) To Push

git push -u origin main

-u → upstream

It means push to origin upstream and main
branch.

WED - 03) Git and GitHub Masters class - Part - 2 — 29/06/2025

1) Git Branching

Every branch is a different path of GitHub so Dev can work independently that can be merged later with main branch.

⇒ How to make a branch?

a) git branch

↳ It will show name of branch

b) git log

↳ It shows current history line log.

c) git branch <new branch name>

↳ Creates a new branch.

d) git checkout <new made branch>

↳ User will move to that branch.

⇒ Merge branch to main branch

a) git merge <new branch name>

⇒ one command to make new branch and enter into new branch

a) git checkout -b <branch name>

If branch exist it will switch to that branch
otherwise, first it will create a new branch and then enter or move to latest branch.

g) How to add and commit ~~one~~ once.

command

→ git commit -am "message"

a - added

m - message

git lens and git graph are two extensions of
vs code

3) ~~Squash~~ Squash

Suppose you created your own branch and then added 3-4 commits, if you add it without squashing, your main branch will get polluted with so many commit ~~that~~ which are not even needed if you can resolve the problem with just one commit.

So while ~~adding~~^{merging} custom branch to main branch, if you added multiple commits in custom branch in main branch it will show one commit.

If you are making a feature, instead adding every commit to ~~the~~ main branch, it is considered better to make one commit than show ~~on~~ about that feature.

⇒ How to do it?

a) first go back to main

↳ `git checkout main`

b) `git merge --squash <custom branch name>`

↳ this command takes whole custom branch code into single code (clones file) ~~and~~ but it is pending to commit new changes in main branch using regular commands.

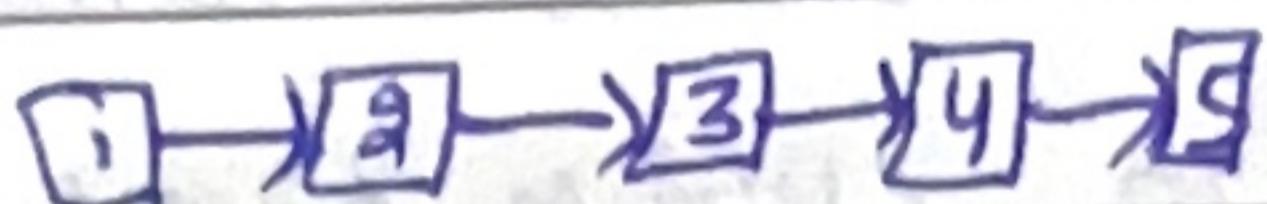
4) Rebase

It re-writes the timeline of ~~new~~ branch.

Command -> "custombranch: git rebase main"

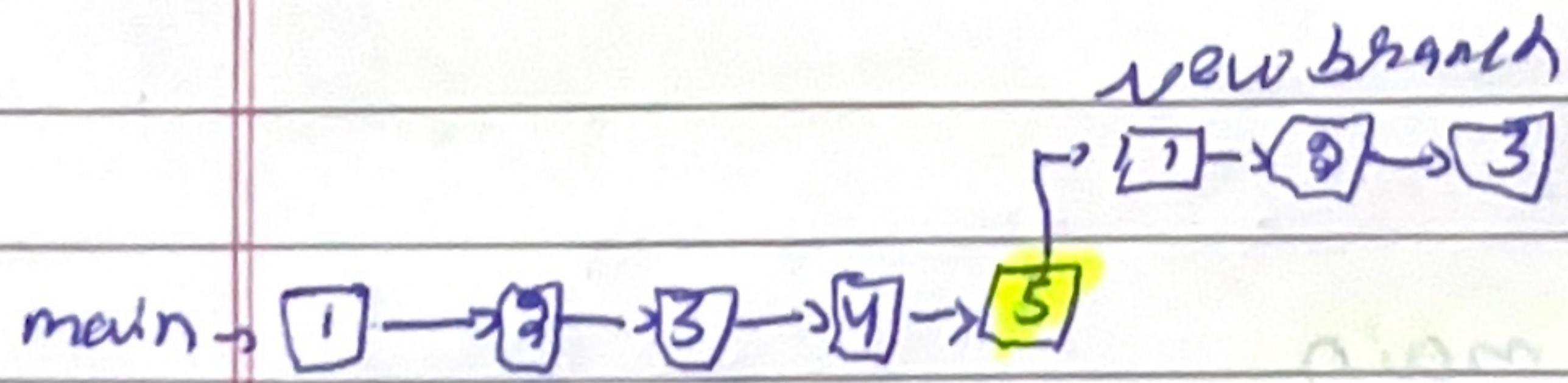
Let's understand with examples

1) main branch



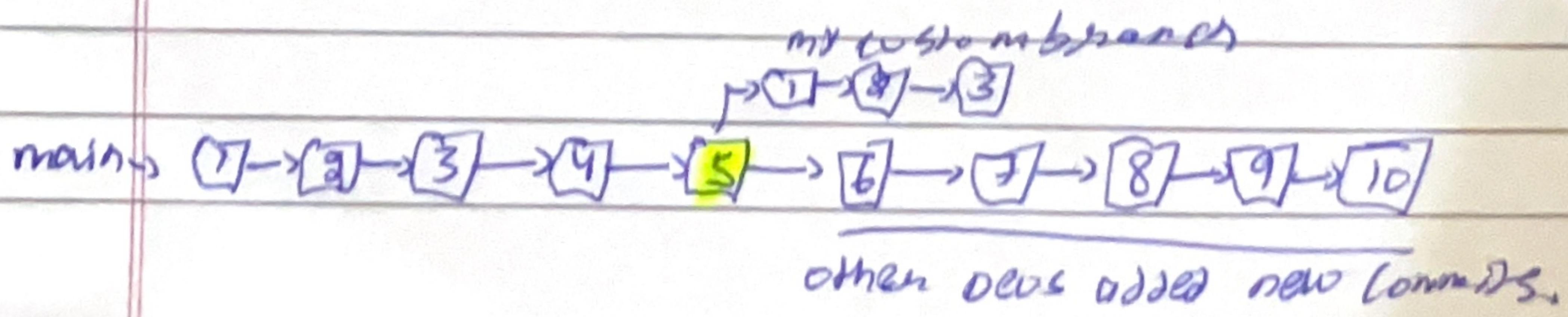
These are Five commits in main branch.

2) Creating a new branch



added 3 commits to new branch.

3) Now, I started a new branch from 5th commit of main branch, however, while I was working on some features, other developer add new features in main branch so I left behind in the main branch and I can not merge feature in the main branch where I left.

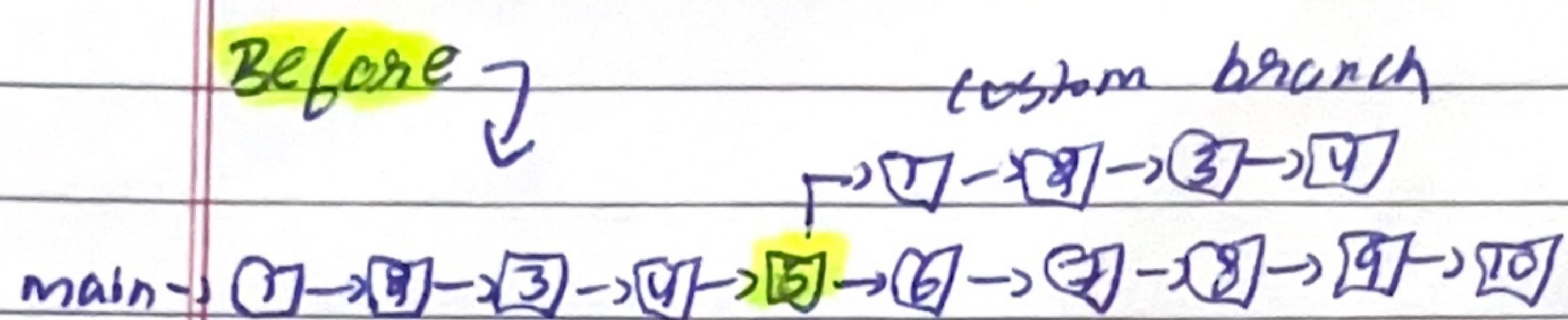


Here, I can not merge my custom branch because it can conflict or I also can not squash main branch into custom branch or custom into main branch.

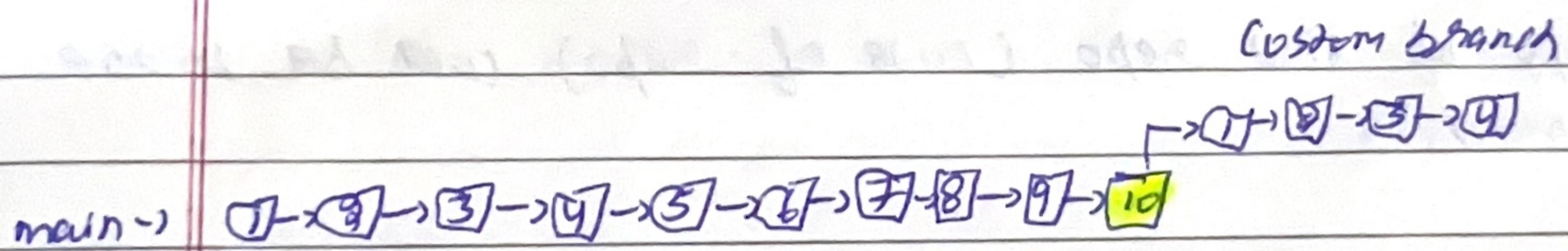
Technically, I can squash branches but if I squash custom branch in main branch, it will raise conflicts.

So, easy and one good way to is **REBASE**.

Q) Rebase will rewrite whole timeline:



AFTER REBASE

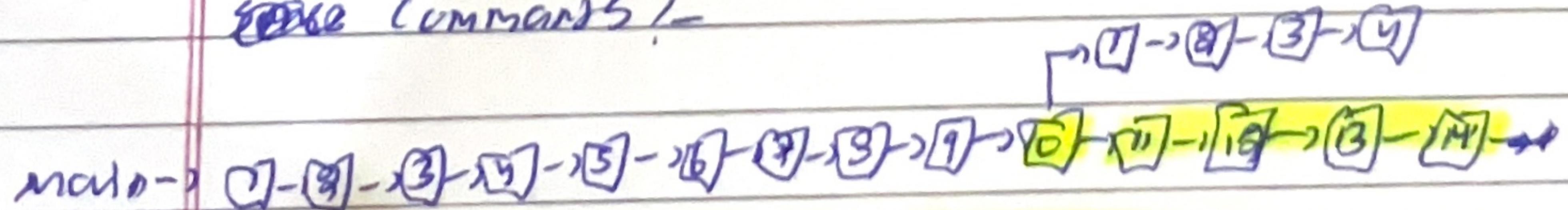


Hence, it will move branch head and its last commit of main branch will become first commit on head of my custom branch.

Final Result

A)

Now, I can merge custom branch into main branch using ~~these~~ commands:-



I added to main branch. I can squash and add only one commit also in main branch. It depends on company rules.

5) Not Much Used Commands

- 1) git revert
- 2) git reset
- 3) git cherry-pick

6) How to make ^{copy} of a Repo?

- 1) Go to Github
- 2) Find a repo
- 3) Click on Fork

7) git clone and use

A copy of that repo (code of repo) will be in the local machine.

~~PHASE - ☺ is COMPLETED~~