Git & Github

PART-1

- VCS
- command - line - toal
- Git introduction
- Tracking a Project
- Git additional commands
- Branching

PART-2

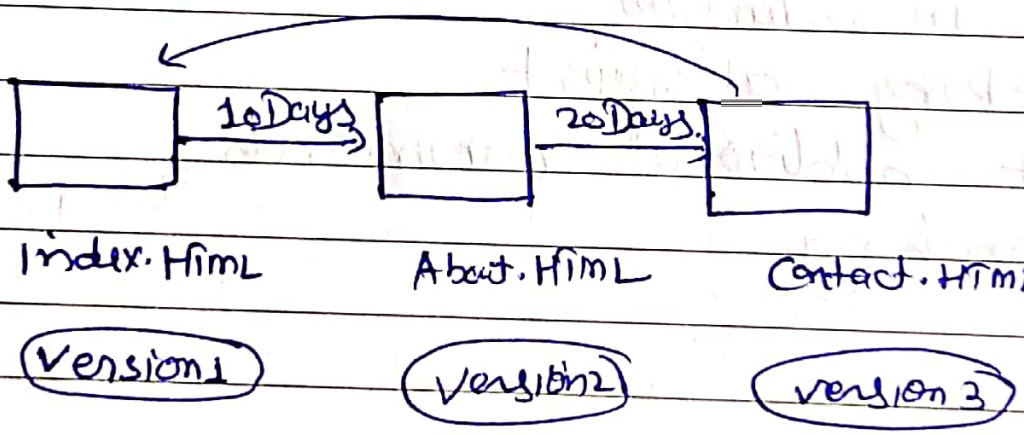
- Merging
- Rebesing
- introduction to Github
- Social coding with Github
- Rewriting History

1

Git

④ VCS (version control system) (git)

- How to you create a Project and how would you manage changes?



(Track the files)

⑤ VCS:-

Version control is a management software Responsible for managing changes to computer programs, documents, large websites or other collections of information.

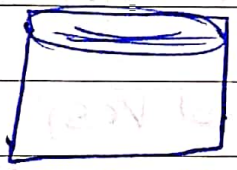
- Easy recovery of file / folders.
- Rollback to Pervious version.
- Inform about us who, what, when, Why changes have been made.

⑥ Local VCS :-

History of VCS
① Local VCS
② centeralized VCS
③ Distributed VCS

① Local VCS :- Changes are Stored in a database, along with time stamp.
→ code is in Local system.

Version1 10 Jan 21
Version2 25 Jan 21



DataBase
ex:- MySQL

Cons:- Project can be lost, if your hard-disk is computed.

② Centralized VCS :-

Repository (folder) :- A directory (folder) where your project resides. it can be a local folder in your computer or a

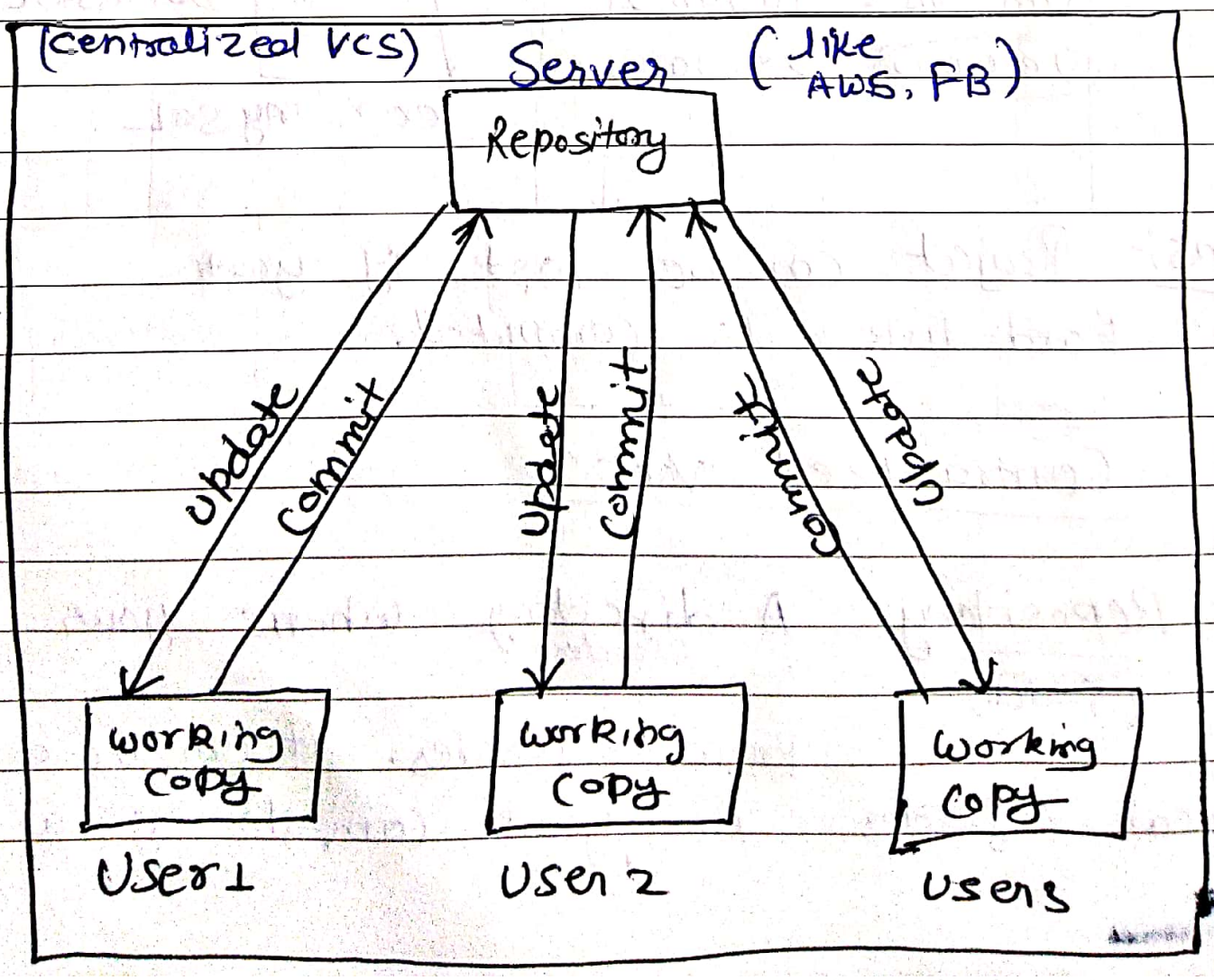
Folder:- In general terms of VCS we say Repository or directory.

remote directory on a Servers.

There are 2 types of repository

1. Local Repository (local computer)
2. central Repository (all folder on Server like Github, AWS, fb server)

⇒ centralized VCS contains just one repository i.e central repository and each user gets to work on the same.



Commit:- Putting the Update over there or making some changes on the server. which control repository.

Cons:- 1. if the central repo. goes down even for an hour.
2. you will lose the project if the hard disk of the central server goes down.

(b) Distributed VCS

Distributed version control system contains multiple repository. each user has their own local repository & there is a ~~to~~ central repository where the final code reside.

→ Provides full back-up of the Project.
→ Git is an example of DVCS.

(9) How ~~to~~ Git Created

Linus Torvalds (creator of Linux Kernel)

(10) Git :-

→ Git is a free and open source Distributed version control system designed to handle everything from small to every large projects with speed and efficiency.

→ Git stores snapshot of your project (not differences).

(11) Git Features :-

→ Collaboration, Storing versions, Analyse the code changes.
→ Distributed, Almost everything is locally
→ Non-linear / Branching, Security / integrity
→ Speed (very fast, bcz C langage is used).

(12) Installing Git

→ Git
→ Download
→ windows
→ (Next) → Next → Next → Next → Next.

(Search) → Git Bash

Git Bash → Running Linux Command
Bash → Terminal in Linux.

Pwd → (Print working directory).

→ cd Desktop (comes to desktop)
→ ls (All items on Desktop)
→ Clear

(14) Configure Git

Whenever you save some version of your project, so git should understand, who is actually save this changes.

1) # to check the git version

```
git --version
```

2) # configuration of global variables

```
git config --global user.name "Monu"
git config --global user.email "manuchanda123"
```

3) # check the configuration

```
git config --list
```

⇒ Because we are making some changes, on different version. Git should understand that done by you and not some else, if you not config. It gives anonymous names. Git does not know who accurately do this commit or changes in this version.

Command Line tool

(15) Introduction & PWD

GUI → Graphical user interface.
✗ ex → Using mouse double click on folder, create file, folder etc.

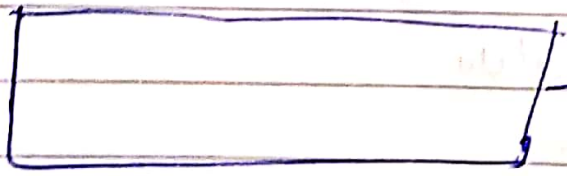
CLI (Command Line tools) / Interface
✓ (use)

(1) Pwd (Print working directory)

/ → root directory

(16) List items :-

(2) ls (list items)



→ Display All files & folders

(3) ls -l (Some flags)

(-l → gives all files and folders list format) on line

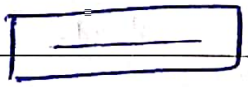
④ ls -a (show all hidden files)

⑤ ls -al

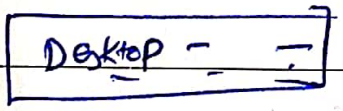
.	→ current directory
..	→ parent directory

⑰ Change directory:-
(cd)

① → pwd



② → ls



③ # wants check what inside desktop.
cd ./Desktop

④ pwd

⑤ ls

Similarly go back

⑥ clear

⑦ cd ..

⑧ pwd

~ → Tilda (Home directory)

cd ~ → go back home

(8) → cd Documents/DSA (Directly go DSA folder)

(9) cd / → Root folder.

(18) Create files / Folders

(1) mkdir name // Folders

(2) * Create 3 folder one time

```
mkdir f1 f2 f3
```

(3) * folder inside another folder

```
mkdir -p f4/f5
```

f5 → f4

f5 inside f4

// Files

(1) * create file `touch file.txt`

(2) * open the file `open file.txt`

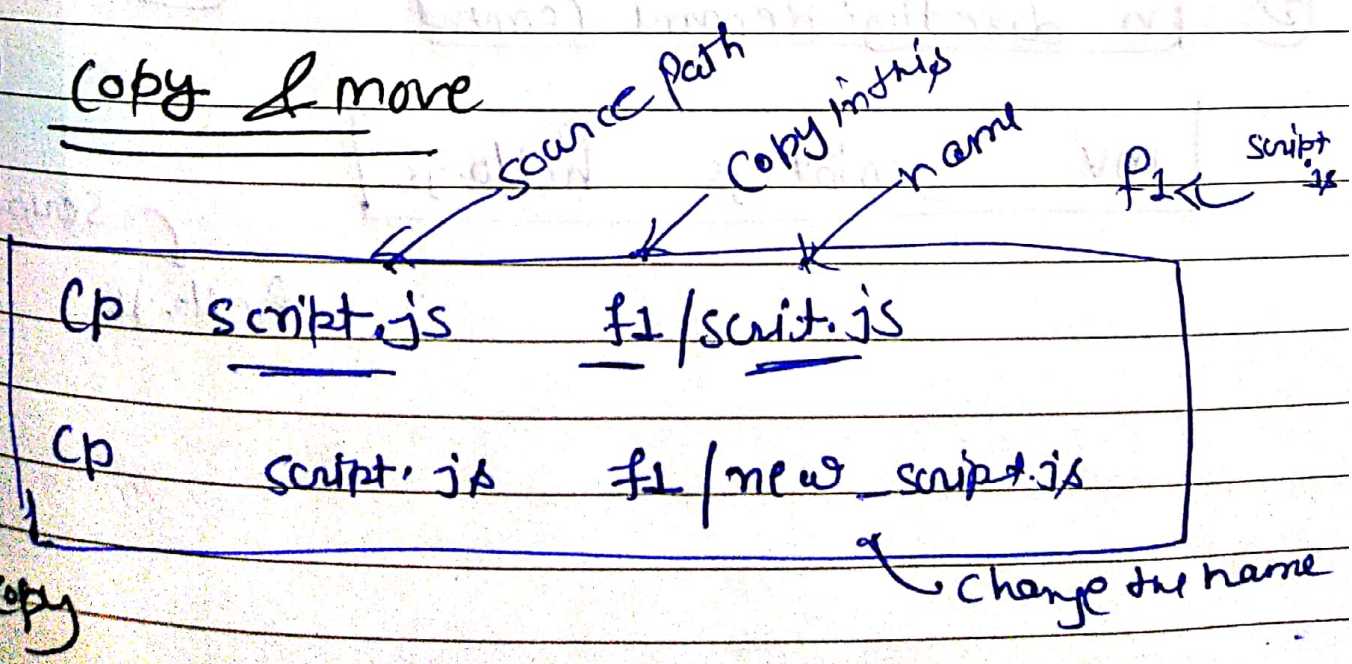
(19) Remove files & folders (rm)

- (1) Remove folder — `rmdir f1`
- (2) Remove file — `rm file.txt`
- (3) file inside folder — `rm f2/file.txt`
- (4) If any content inside folder (like one folder, many files etc). you can't delete directly the folder. so use this command.

`rm -R f2` (-R → Recursively)

(20) Copy & move

(1)



② When you copy the folder in which some file.

```
cp -R f2 new
```

f2 → copy folder
new → name

Move

// move (cut paste)

① `mv index.html f1/index.html`

index.html → (Source)
f1 → folder
index.html → name

f1 ← index.htm

② No directly Rename Comand

```
mv script.js Hello.js
```

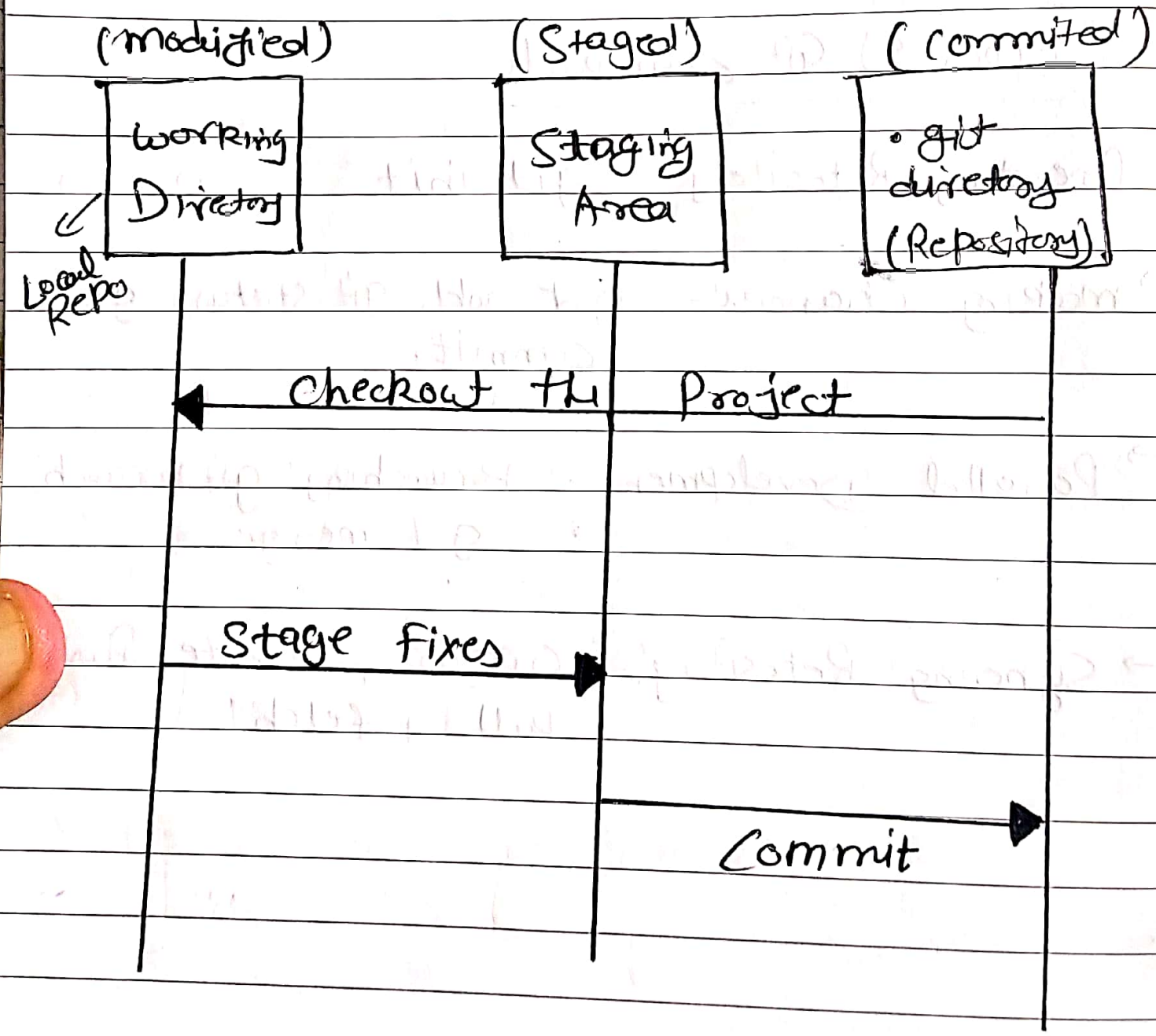
script.js → Hello.js

(21) Git Introduction

Types of Git commands

- Creating Repository:- git init
- making changes:- git add, git status, git commit.
- Parallel Development:- Branching:- git branch, git merge.
- Syncing Repository:- origin, remote, Push, Pull, fetch.

(22) Three Stages Architecture

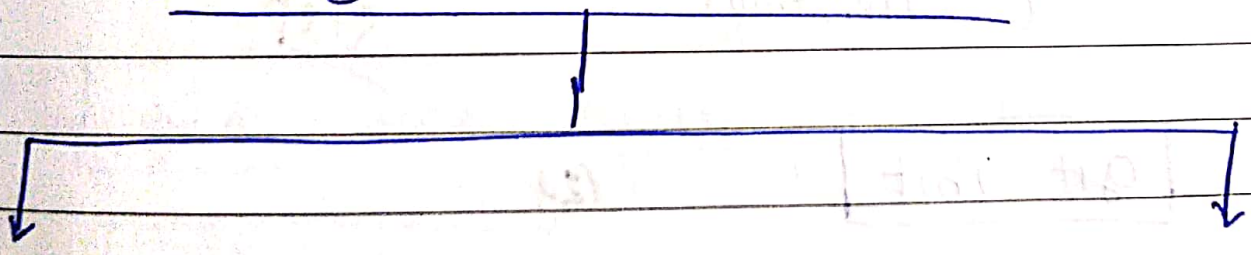


(23) Initialising Git Repository

1. `git init` → This will create a hidden .git folder. That's where git stores everything.

2. `git status` → To track files & folders

Creating a Git Repository



→ Initialize your own local git Repo

→ Clone a remote Repository

→ Create Folder myPortfolio on Desktop and open in vs code write html code, save

(Rightclick

open Git

→ `Pwd`

→ `cd Desktop / myPortfolio` → Folder

→ `ls` (index.html)

Step

→ `git init` (1)

masters

→ `git status`

{ on branch masters
No commit yet
Untracked files:
index.html ← Red

(Modified index.html)
↳ Git does not Know about this file. After → Add to Staging Area and Commit it.

(24) Tracking file

for one specific file

`git add <file name>`

for all files & folders

`git add .` → current directory

Step(2)

Add to Staging Area

→ `git add index.html`

→ `git status`

{ Change to be committed:
new file: index.html ← Green
∴ This file has to be be added to stage Area.

(25) Git Commit (step-3) (last stage)

git commit → git folder

Committing changes

→ moving files from staged area to the git folder is called a commit.

→ making commits, creates version / snapshots of your project.

```
git commit -m "your commit message"
```

flag message

→ `git commit -m "add index.html"`

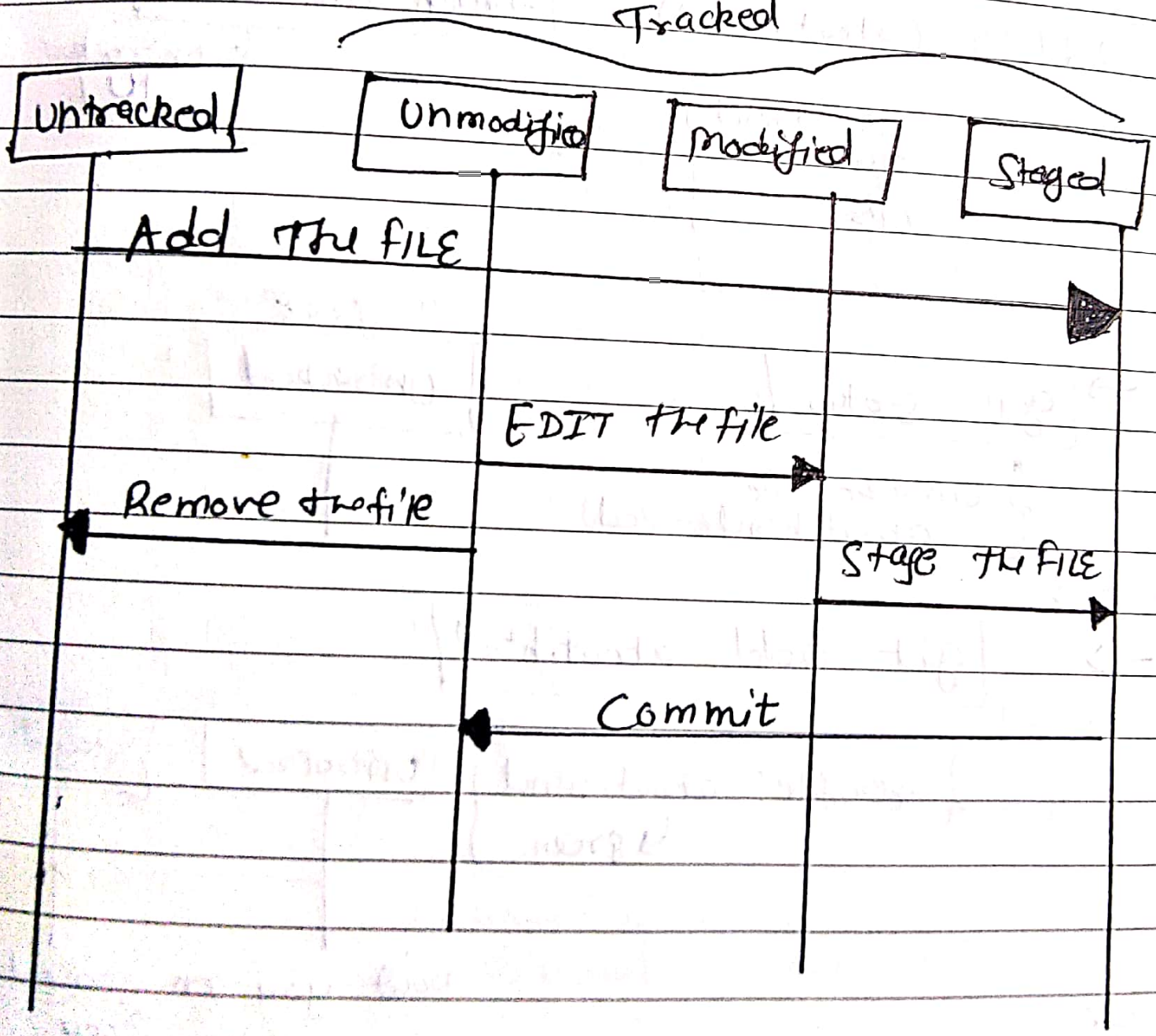
```
master (root-commit) 3963b0] add index.html
 1 file changed, 12 insertions(+)
```

1 file changed, 12 insertions(+)

12 lines of code

∴ Now git understand, git will know that their is a file index.html. i will keep tracking this file. even you will make any changes i will know automatically. Git will know track file. Life long.

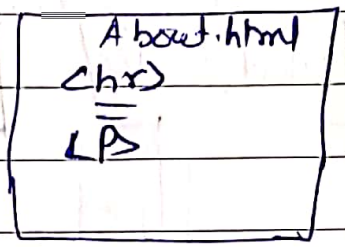
(26) Life cycle of the States of a file



(27) Example of life cycle of file

→ Add file (about.html) `touch about.html`

↳ untracked



→ `git status`

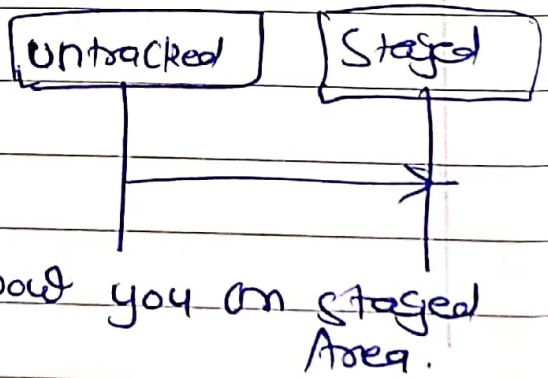
stage

Untracked

{ untracked file
about.html ← red)

→ `git add about.html`

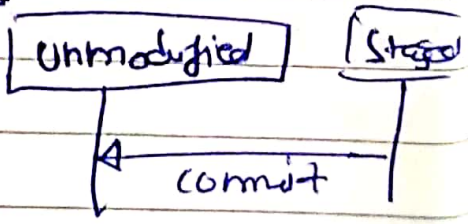
{ newfile: about.html
↳ green }



→ Now you do commit

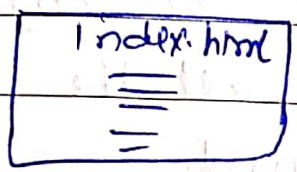
`git commit -m "add about.html"`

{ 1 file Changed, 15 insertion
Create mode 10644
about.html }

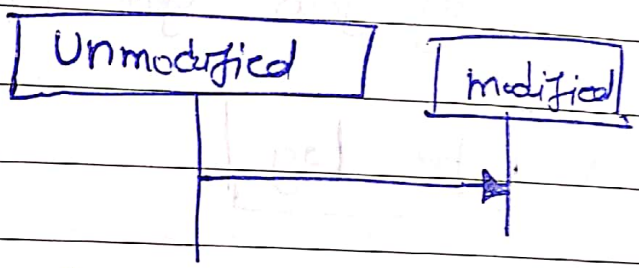


index.html About.html

→ Lets changed the file index.html



(modified
index.html M

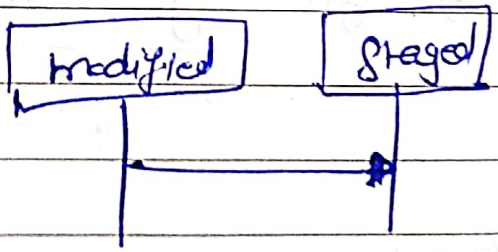


→ git status

{ changed not staged
modified : index.html (Red

→ Add to staging Area

git add index.html

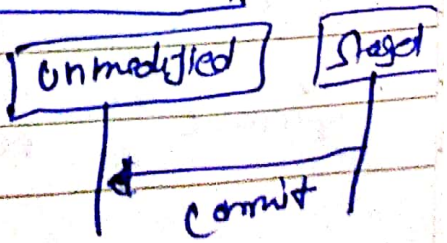


→ git status

{ index.html
↳ green }

It is staged now it is ready to commit

→ git commit -m "update index.html"



(28) Logging the Previous Commit

viewing the commit History
↳ you can see all the Commit / version of your git Project

→ `git log`

← Hash id / Commit id

Commit `f0g7835t9` (every one has unique hash no. for security reason)

Algorith → MD5
→ cryptography

→ `git log --oneline`
↳ (it will give one line commit)

(29) Deleting a Git Repo

Deleting a Git Project

→ `rm -rf .git`

rm → remove
-r → recursively
-f → force fully

(30) Skipping the Staging Area

Additional Git commands

`git commit -a -m "commit message"`

- `-a` → add
- `-m` → commit

(31) What's the differences

`Git diff`

If you want to know exactly what you changed, not just which files were changed - you can use the `git diff`

→ This is used in modified level not use in the staged level or commit level

→ `git diff --staged` → check file Previous commit

∴ to see what you've staged that will go into your next commit.

(32) Git IGNORE

- Sometimes, we don't want git to keep track of some specific autogenerated files like logs, or build file, .DS_Store etc.
- To avoid them, you can list files or a pattern for files in .gitignore file.

Create log file

→ touch error.log → i don't want this file should be tracked by the git.

→ git status

{ Untracked file:
error.log ← Red

↳ open

server got crashed ✗

** And if you check

→ git status

{ Untracked
error.log

but i don't track it.
but its tracking

* So you need to create Specific file and that file is `.gitignore` *

→ `touch .gitignore`

And in this you write all file

ex. error.log

→ `git Status`

{ Untracked files:
.gitignore [∵ but not show error.log so you can take many files.

`*.log` → for all log file.

→ * ignore folder like

* `mkdir abc`

→ `touch abc/random.txt` (make file in folder)

`.gitignore`
`abc/`

→ If you want add

```
git add.
```

```
.git ignore
.DS - Store
```

→ `git commit -m "add git ignore file"`

→ `git status.`

{ on branch master
nothing to commit, working tree clean.

(33) Moving & Removing files

These commands stages the changes automatically. You don't have to stage explicitly.

Remove

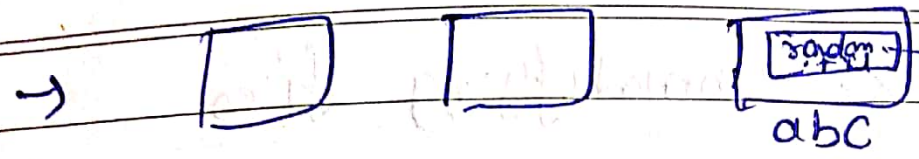
1) `git rm filename.txt`

or

`git rm -f filename`

Move

2) `git mv file name.txt newfile name.txt`



Delete random.txt

```
rm abc/random.txt
```

→ `git status`

{ deleted : abc/rad.txt

∴ But now add to this change into staging area and than commit otherwise this file will not go away.

→ Change file name (index)

```
git mv index.htm home.html
```

(34) Untracked an already tracked file

```
git rm --cached .DS_Store
```
↳ file.

(35) Unstaging & Unmodifying files

→ `git restore -- Staged filename.txt`

(If you are goes unmodified → Staged Area)

→ `git checkout -- filename.txt`

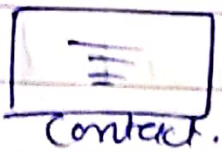
(This command takes your file to the last version / commit.)

→ `git checkout -f`

(Rollback to the Previous Commit, losing all newly modified files)

#

create contact.html



Contact.HTML

→ `git status`

{ Untracked file:
contact.html ← (Red)

→ `git add .`

→ `git status` { new file: contact.html ← green

→ `git restore --staged contact.html`
(for unstaged)

→ `git status` { Untracked
contact.html → (Red)

→ `git add .`

→ `git commit -m "add contact"`

∴ Brother comes and Delete some code (Remove)

→ `git checkout -- contact.html`
↑ space

→ `git status` { clean

(36) Git Alias → Short name

Setting Alias

term → alias.st
Short name → st
Actual Command → Status

```
git config --global alias.st status
```

usage:- git st

ex

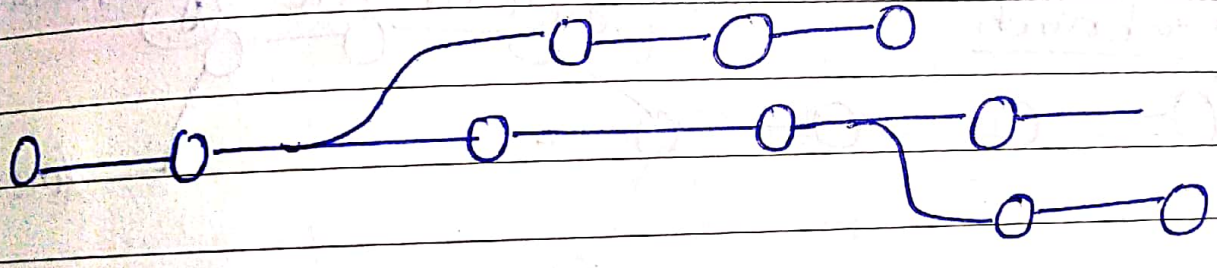
```
git config --global alias.unstage 'restore --staged --'
```

usage:- git unstage file.txt

Non Linear Development: Branching

(37) What is Branching

→ Git follows non linear development.
(Tree like structure its is non linear)



→ A branch in Git is simply a lightweight movable pointer to one of the commits.

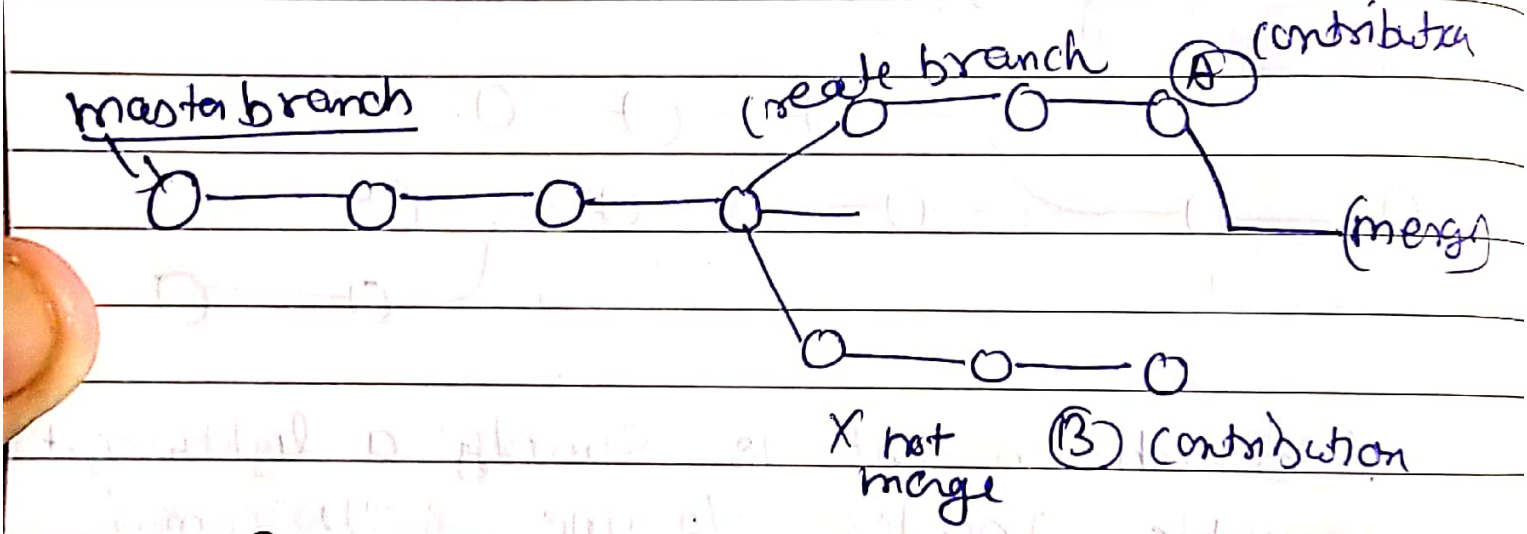
→ The default branch name in Git is master.

→ Branching : Diverging from the main branch.

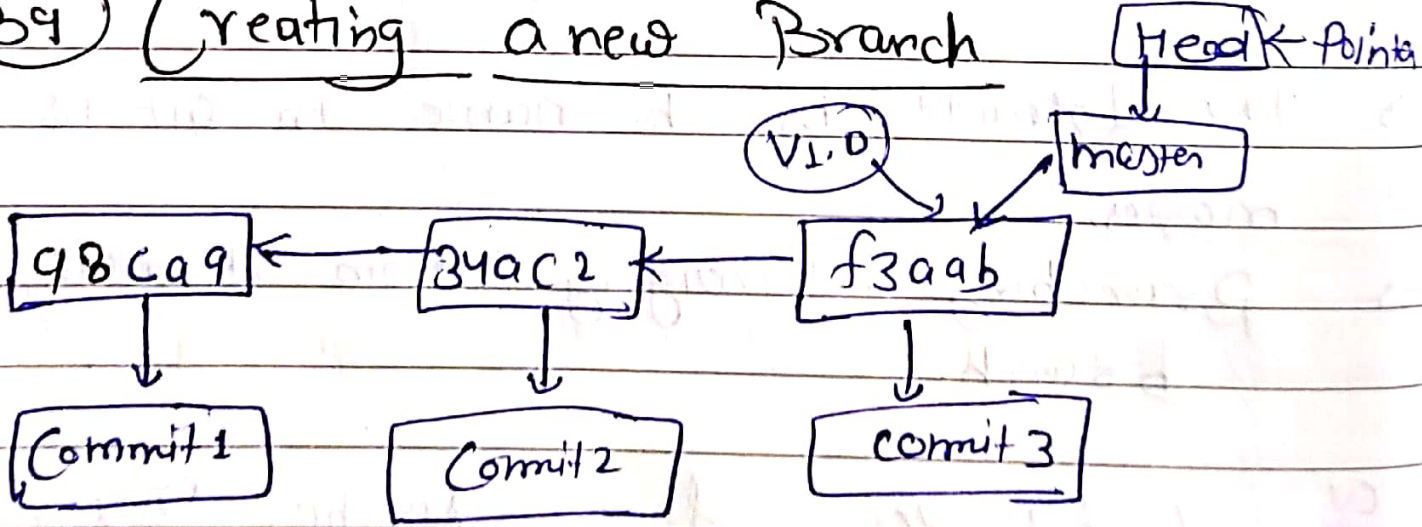
→ (All VCS like Apache, Mercual etc).

(38) Why do we need branching

→ We can contribute
→ We can changing on file to another file.
→ use for opensource. etc.



(39) Creating a new Branch



Get all the branches in your Project

`git branch` → Total branch

`git branch -v` → last and recent Commit in that branch

* → current branch (master)

creating a new branch

`git branch branchname`

ex { `git branch`

develop
* master

40 Switching to Branches

moving b/w branches

Checkout an existing branch

`git Checkout branchname`

ex { `git branch -v`

develop
* master] → 2 branch (Now you are in master)

If you want to move develop branch and you want to work with that

→ `git checkout develop`

{ Switching to branch 'develop'

→ `git branch -v`

{ * develop (Now you are in develop)
master

→ `git log`

{ HEAD → develop, master] (Head is pointing to Develop)

Creating and Checkout to a new branch

→ `git checkout -b branchname`

used for creating new branch and directly go to that branch.

* let if UX branch does not exist, but you want to create the branch and move to branch immediately and checkout to that branch.

ex → `git checkout -b UX`

{ switching to new branch 'UX'

→ `git branch -v`

{ develop
  master
* UX

→ And you want to switch master

`git checkout master`

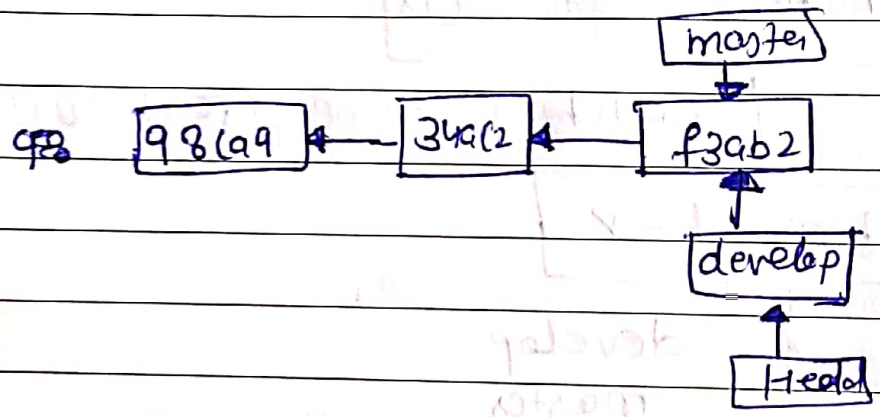
(41) Working with branching

Go to develop branch

contact.html (code change)

→ `git checkout develop`

{ switched to brand 'develop'



→ `git status`

{ modified: contact.html

→ `git diff` → all line of code del, add or sub

OR

Add one more file (feature.html)

→ `git status`

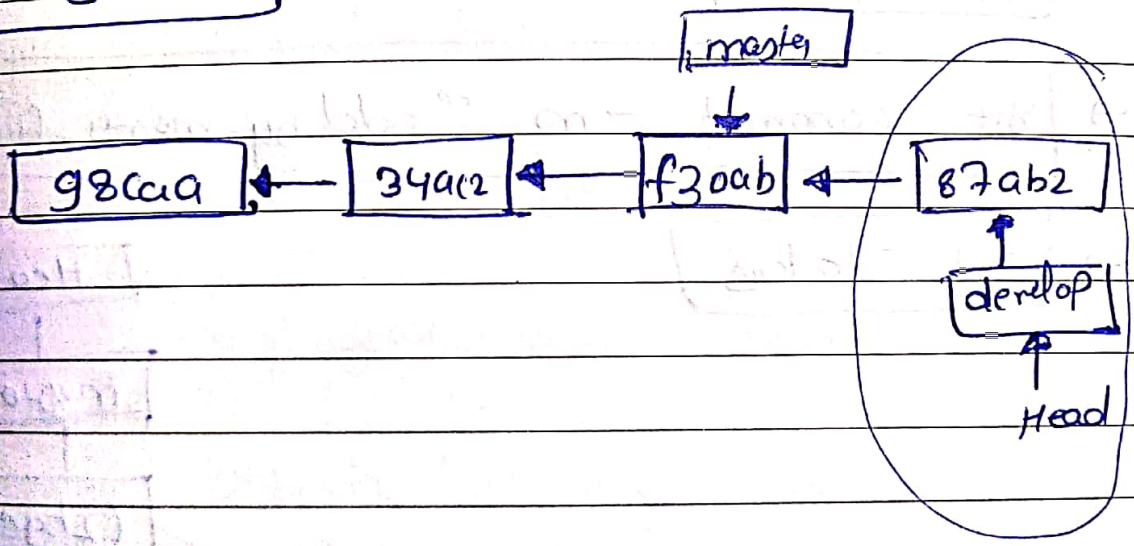
{ modified: contact.html
{ untracked file: feature.html

→ git add .

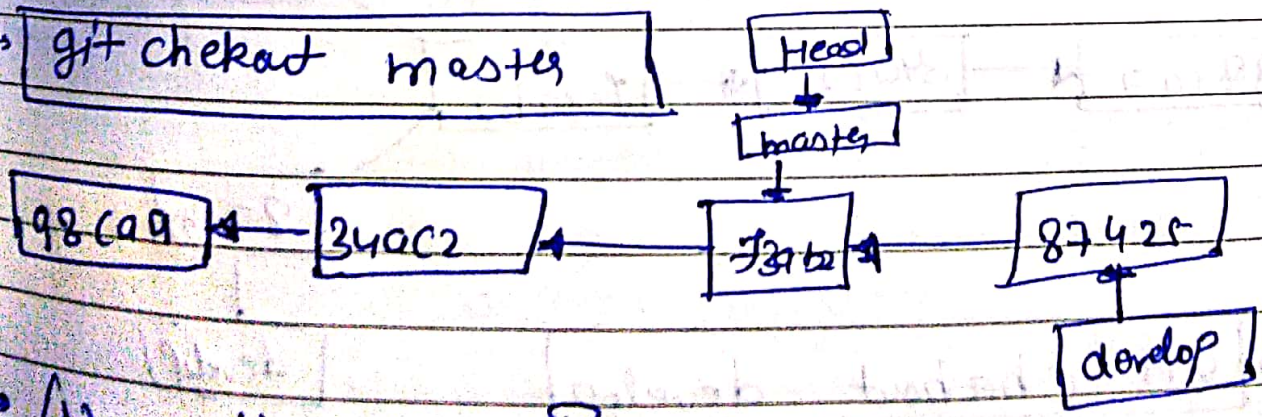
→ git status { modified: contact.html
newfile: featur.html } → Green

→ git commit -m "improved contact.html"

→ git status



→ git chekat master



→ (Now it will show previous, which was master class, Not show the changes)

Now you will add new file to master branch

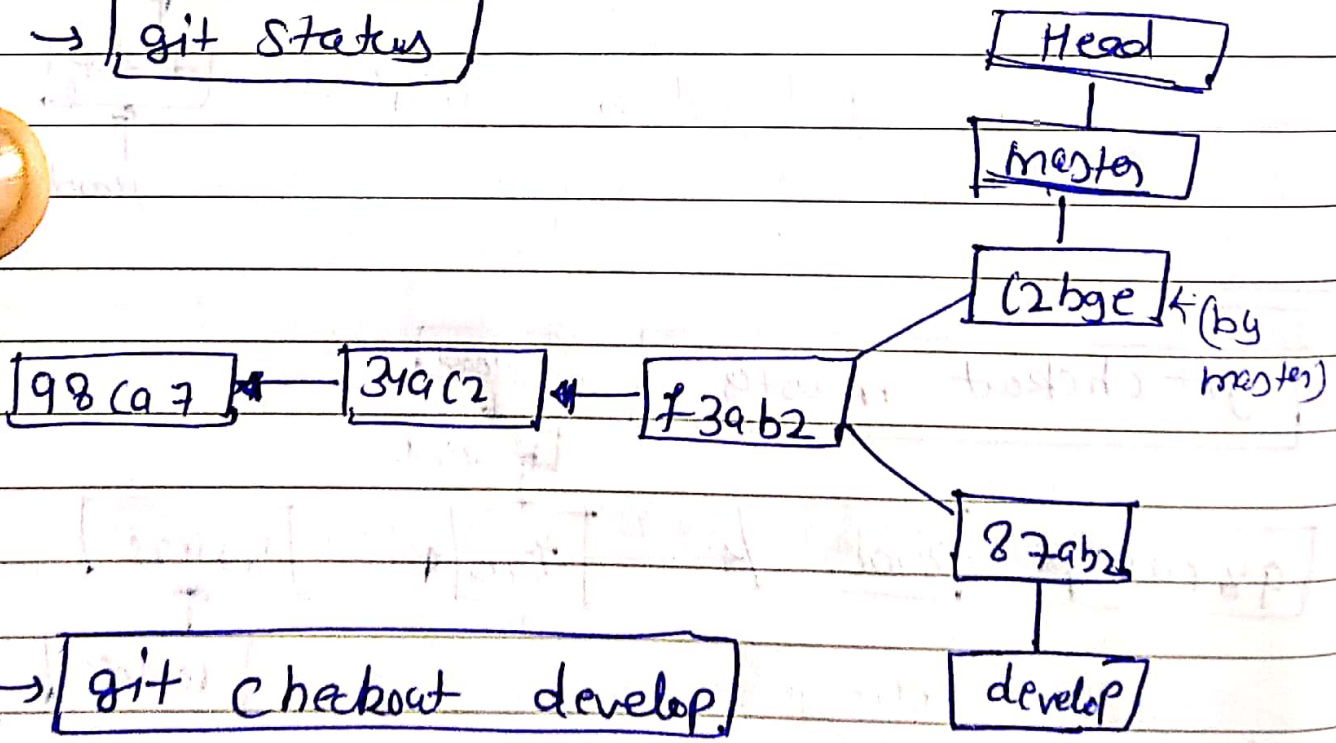
→ `git Status`

⇒ { Untracked file.
by_master.html ← Red

→ `git add .`

→ `git commit -m "add by_master_file"`

→ `git Status`



→ `git checkout develop`

{ `feature.html` `.Power` but `by_master.html` not present

That was in this commit only.

(42) Branch logging

checkall update for all branch

```
git log --oneline --graph --all
```

(43) Deleting a branch

Deleting a merged branch

```
git branch -d branchname
```

(Bcz now you can change file cod, add file).

error:- The branch 'develop' is not fully merged.

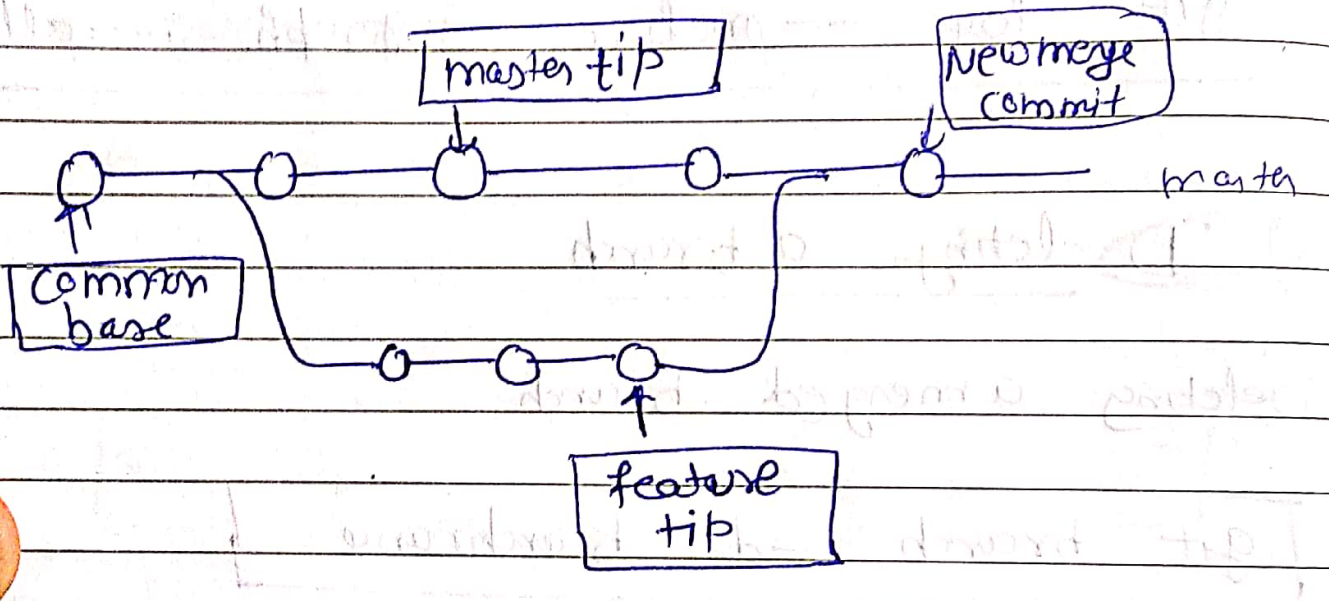
→ But If you want to delete it.

Deleting a non-merged branch

```
git branch -D branchname
```

Non Linear Development : Merging

44 Merging Introduction



45 Adding Functionality

at develop branch

* Change contact page

→ `git commit -a -m "better contact Page"`

-a : Skip staging area

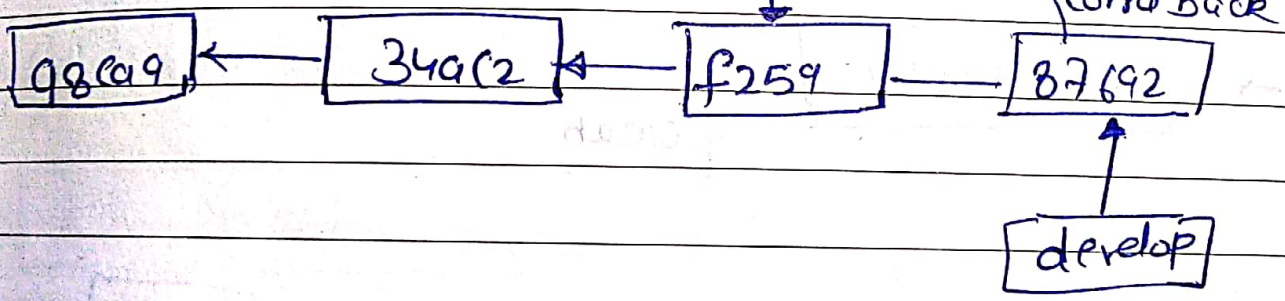
But you to some changes in this, but your boss give task.

go to | index.html changes

→ git status

& clean

→ git checkout master } Go back to master branch



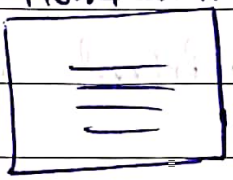
(46)

Basic Merging

→ one way you can change directly the changes in file in master branch but this is not good.

→ So create, a new branch (and go, we do all fixes)

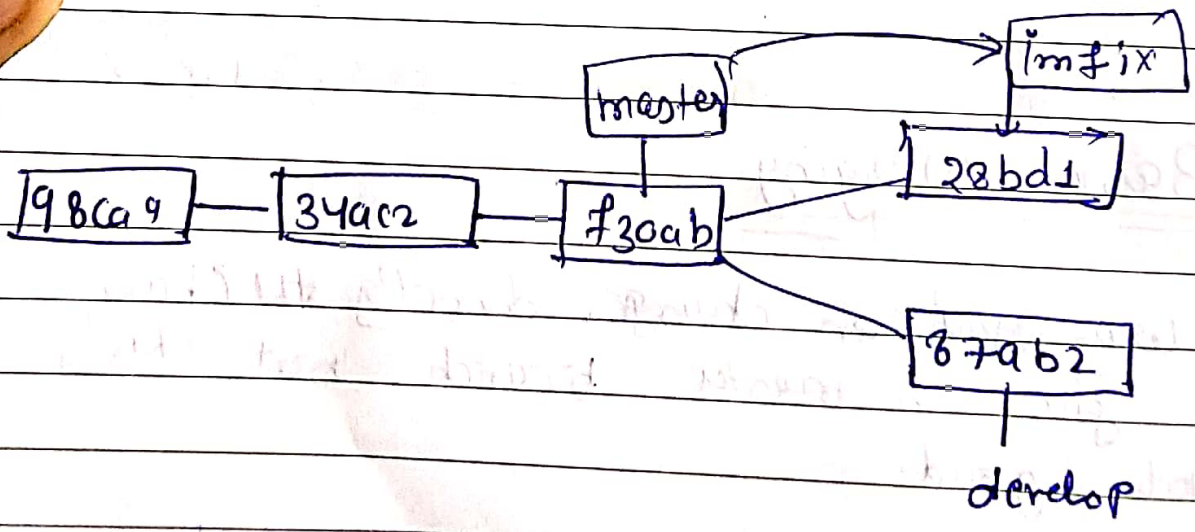
→ git checkout -b imPfix → create new branch

→ Some chayes in Home.html 

→ `git status` { modified : home.html ← Red

→ `git commit -a -m "fix home.html"`

→ `git status` { clean



⇒ If you want the merge `imfix → master` then you need to come and checkout master/fix

→ `git checkout master`

Basic Merge

→ `git merge impfix`

{ master branch is updating }

HEAD → master, impfix

(47) Recursive Merging

After that you can delete imp branch

`git branch -D impfix`

↳ go back to develop branch (bcz we develop contact form)

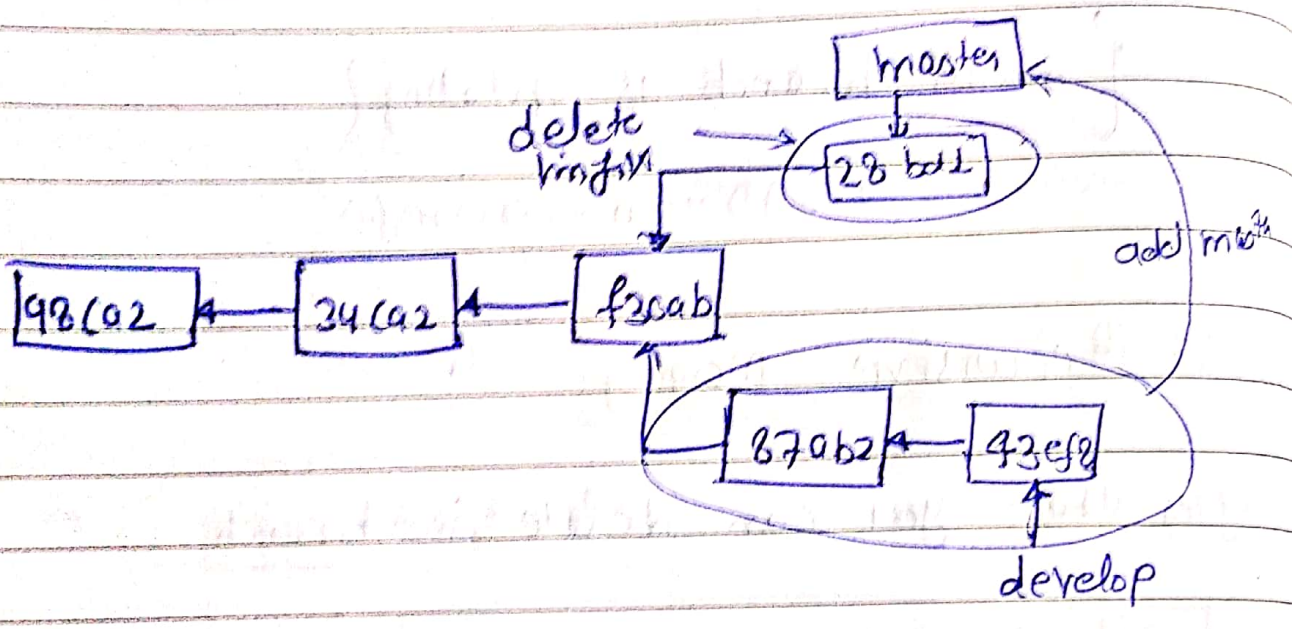
→ `git checkout develop`

→ code change
contact.html

→ `git status`

{ modified: contact.html

→ `git commit -a -m "improved contact form"`



→ Not easy to merge

→ `git log --oneline --graph --all`

→ first come to master branch

`git checkout master`

→ `git merge develop`

[merge branch 'develop'

Esc +

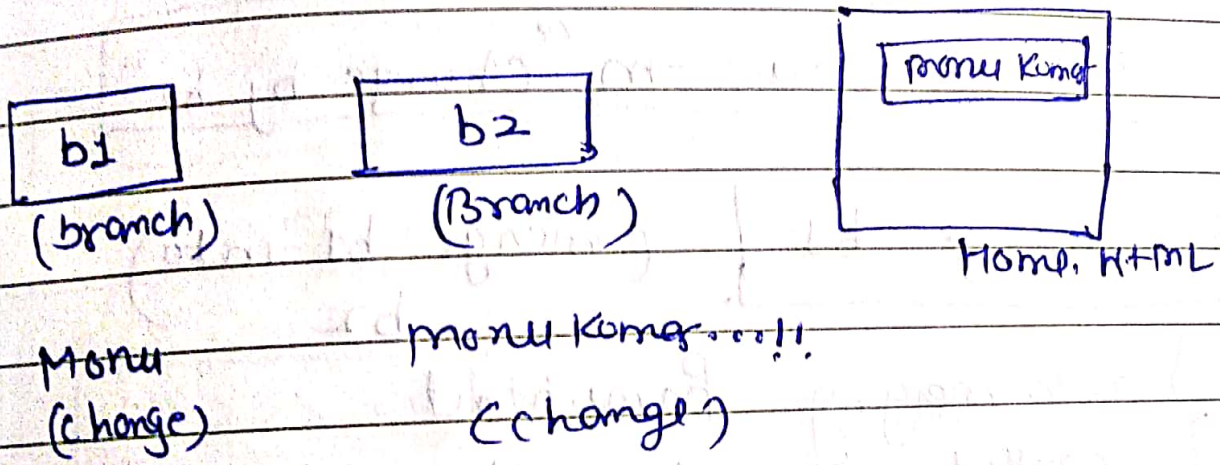
`:wq`

write quit

{ merge by 'recursive strategy'

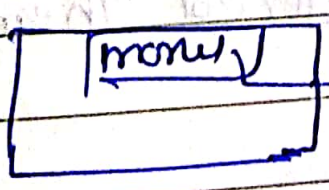
(48) merge conflicts

changes in a file made by 2 different branches, trying to get merge result in a merge conflicts.



→ master (Now Present master branch)

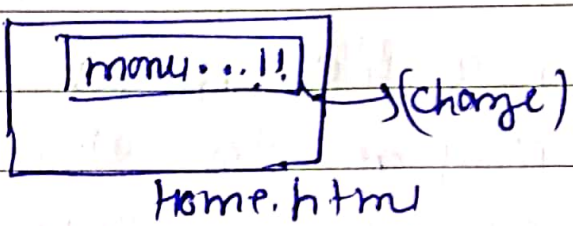
→ git checkout -b b1 (Create new branch b1)



→ git commit -a -m "Reading change by b1"

→ git checkout master

→ `git checkout -b b2` (creat branch b2)



→ `git commit -a -m "change by b2"`

→ `git merge b1` (merge b1 into b2)

auto-merging home.html
conflict : merge conflict in home.html
Automatic merge failed:

→ git can not automatically all the changes we do manually fixed them.

(49) Resolving Conflicts

See (VSC)

<<<<<<HEAD → The conflict start from here is your current branch.

<h1> Rohit Kumar...!! </h1>

(This have to change in your current branch)

=======

<h1> Monu </h1>

>>>>>>>> b1 (Incoming Change) → This is incoming change from another branch b1.

You have Add manually.

→ git status → on branch b2
You have unmerged branch

// If you use
(Accept current change)
→ It will remove incoming change

// :j wq → Accepting incoming change
↳ git will remove current change.

→ And other wise you can write change manually.

→ `git add .`

→ `git commit`

→ `Esp + :wq` { git commit
merge branch 'b1' into b2

(50) Git Branching Workflow in Production

Branching Workflow

Long Running Branches
→ master
→ Development

short

Topic Branches
→ Authentication
→ UI changes

(51) Rebasing Introduction

Integrate changes from one branch into another,

- merging
- Rebasing

(52) Rebase a Branch

Now → master branch

→ `git checkout -b experiment` // create a new branch

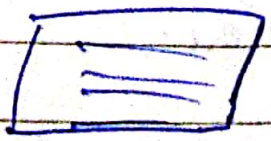
→ `touch experiment.html`

test code

→ `git add .`

→ `git commit -m "experiment"`

(Again update code (one more commit)



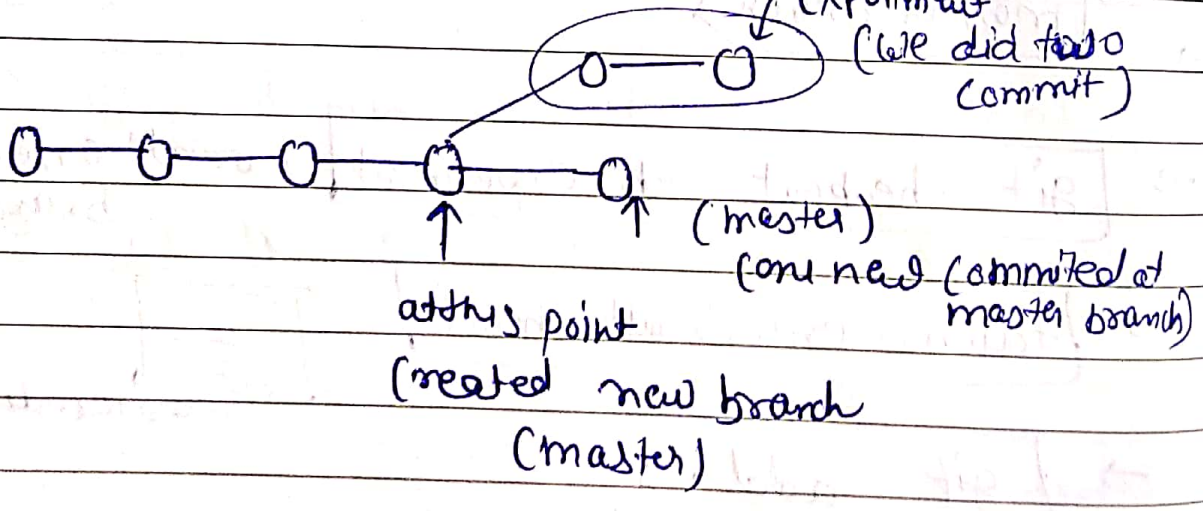
→ With the rebase command, you can take all the changes that were committed on one branch and replay them on a different branch.

→ `git commit -a -m "experiment2"`

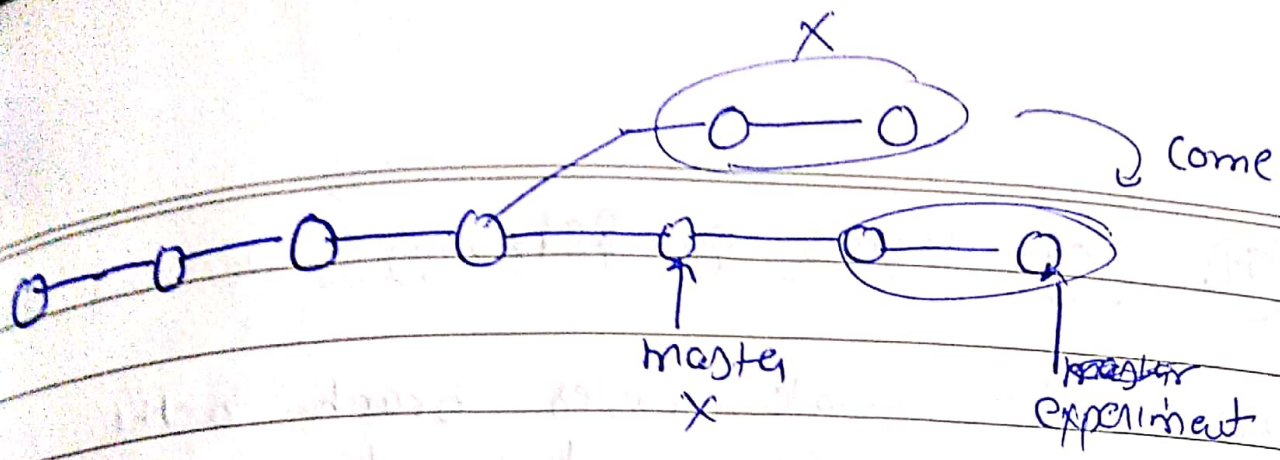
→ `git checkout master`

Add change by_master.html

→ `git commit -a -m "Update by master"`



→ Rebasing will take these two commit (experiment) and will delete two changes and it will add same commit ones here.



→ Here we will do opposite (Checkout experiment)

↳ → `git checkout experiment`

→ `git rebase master`

→ `git log --oneline --graph --all`

This is not format of branch { * * *

Head → Experiment

→ It's like linear Change.

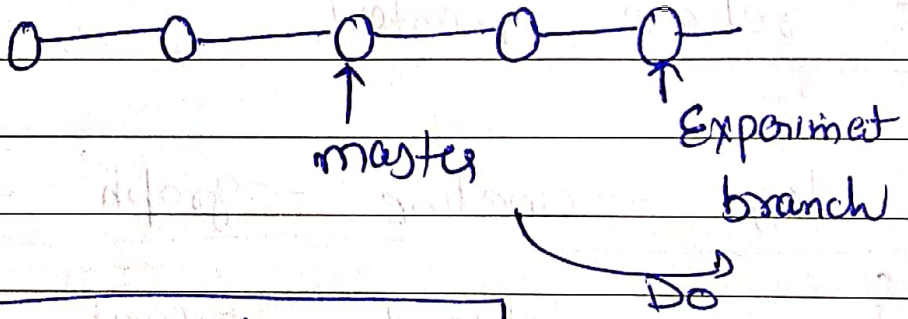
(3) How Git performs rebase internally

(Already done ↑)

(54) Key Points of Rebasing

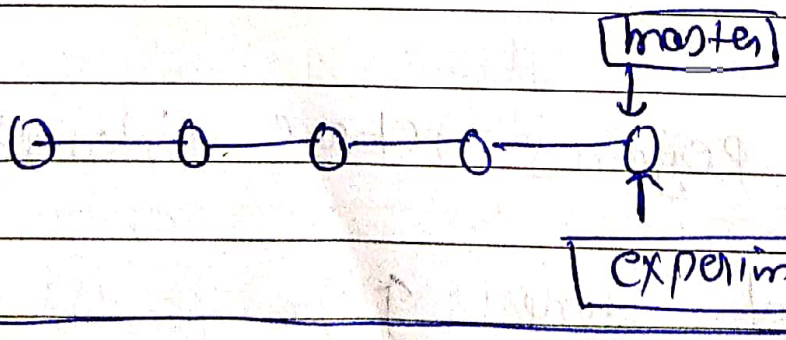
(How to be update master branch to the recent commit (experiment branch).

→ Now (master branch is two commit behind the experiment branch)



→ `git checkout master`

→ `git merge experiment`



→ `git log --online --graph --all`

{HEAD → master, experiment)

→ Now you do not need experimet branch so delete it.

`git branch -d experimet`

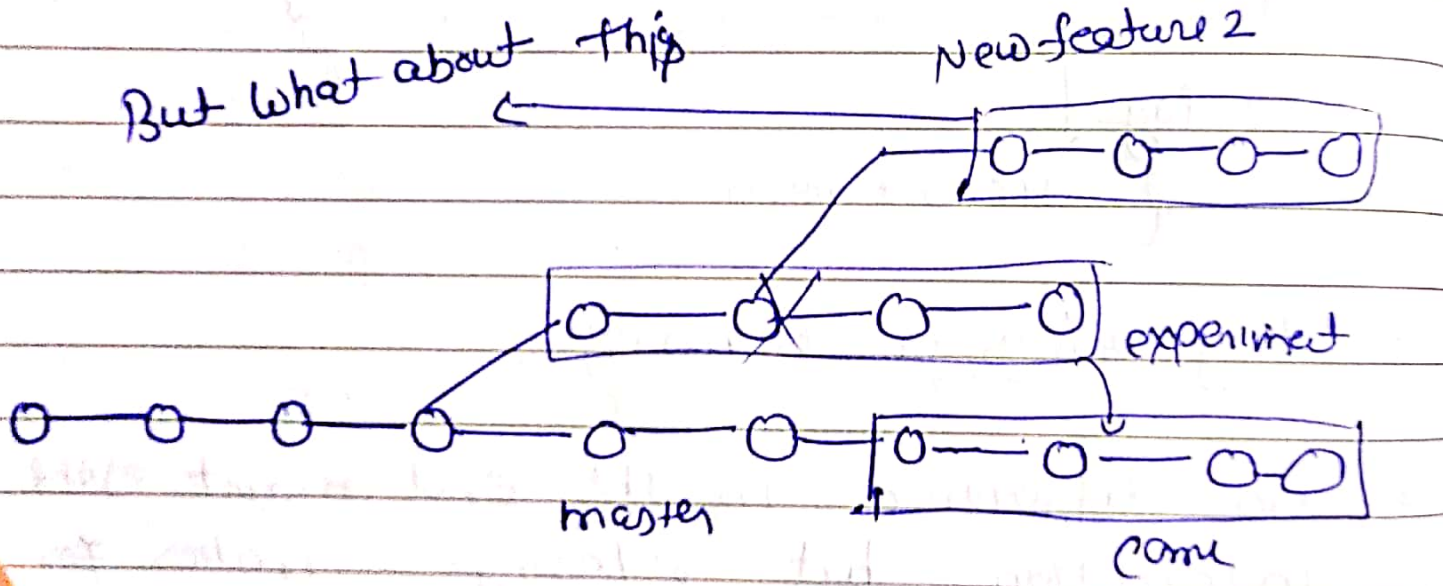
→ `git log`

HEAD → master

Key points on Rebasing

- No difference in the end product of the integration but rebasing makes for a cleaner history.
- The log history looks like linear history: It appears that all the work happened in series, even when it originally happened in parallel.
- Often, you'll do this to make sure your commits apply cleanly on a remote branch.

(55) When you Should not use Rebase



(56) Merge Vs Rebase

one point of view is commit History is a record of what actually happened.

The opposing point is The commit history is the story of how your project was made.