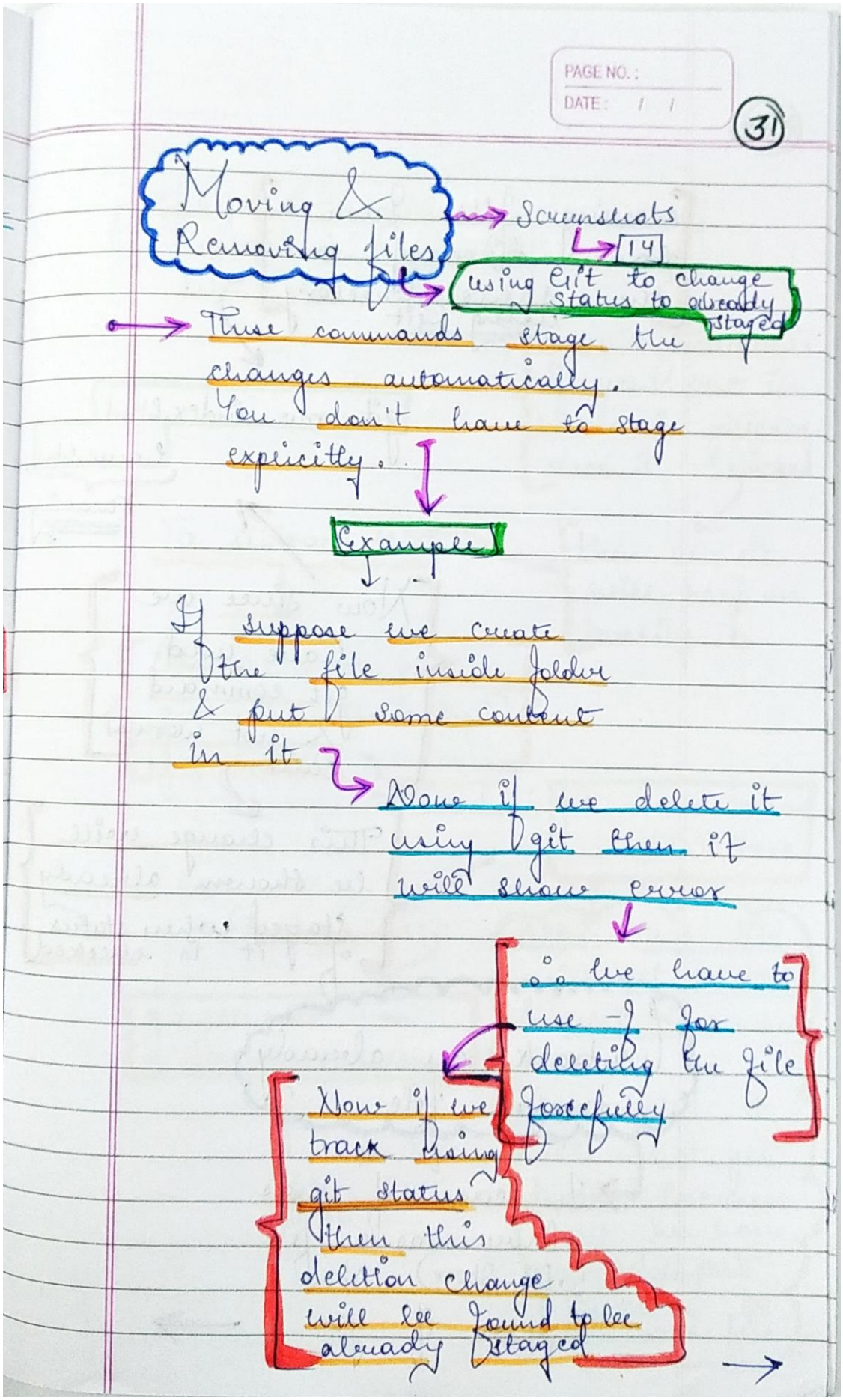

PAGE NO.:
DATE: / /
31
Moving & Removing files
Screenshots
14
using Git to change status to already staged
These commands stage the changes automatically. You don't have to stage explicitly.
Example
Suppose we create the file inside folder & put some content in it
Now if we delete it using git then it will show error
∴ we have to use -f for deleting the file forcefully
Now if we track using git status then this deletion change will be found to be already staged

PAGE NO.:
DATE: / /

Moving file from a ~~same~~ directory to the same directory using Git

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

Renaming

Now since we have used git command & not normal cmd

This change will be shown already staged when status of it is checked

Untrack an already tracked file

In case of mac there was a file (.DS_Store). Now suppose if you →

forgot to create .gitignore and add this to it then

even afterwards if you put its name in .gitignore once it is tracked then also git will keep on tracking it

To untrack this

We have to explicitly give the cmd to ignore this

Cmd : git rm --cached .DS_Store

Finally, do:

git commit -m "Delete .DS_Store"

Now this file will get deleted & next time when it will be regenerated, it will not get tracked because now we have already put its name to .gitignore file.

NEXT PAGE

PAGE NO.:
DATE: / /

Unstaging & Unmodifying files

→ Screenshot → 15

Command: git restore --staged <filename>

If suppose we have staged a file & then we think that it is not fine & want to unstage it then we can use this cmd

- git checkout -- <filename>

This cmd takes our file to the last version/commit on our local machine as well

- git checkout -f

(Rollback to the previous commit. Loosing all newly modified files)

Git Alias → Screenshot → 16

Suppose we don't want to write the complete commands to do our work & want to give short form to them

For setting alias

Eg 01:

Command → git config --global alias.st status

The short form

Command to give short form to

Usage → git st

Now if we will use this instead of git status it will still show the track

Eg 02:

Unstaging code

Alias (unstage)

The short form

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

Command

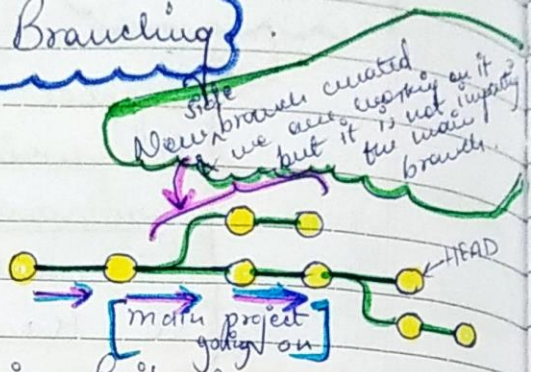
Usage → git unstage <filename>

Command to give short form to

PAGE NO. :
DATE : / /

Non Linear Development : Branching

What is Branching

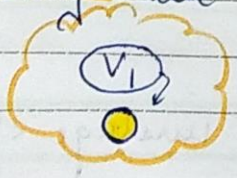


- A branch in Git is simply a lightweight movable pointer to one of the commits.
- Default branch name in Git is **master**
- Branching : Diverging from the main branch.

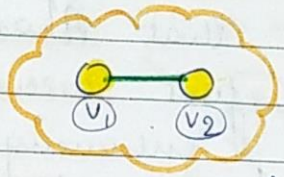
Need of Branching

Example : Suppose we have built a website

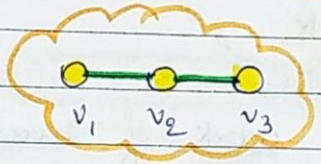
1. Initially we created index.html suppose



(2) Then we added about.html & it is v2



(3) Then we added contact.html



→ And, our code is working perfectly fine. BUT

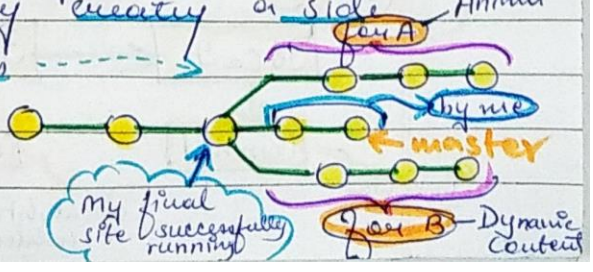
Suppose I have 2 friends A & B

A says that she can contribute animation to my existing website

&

B says to make the content dynamic

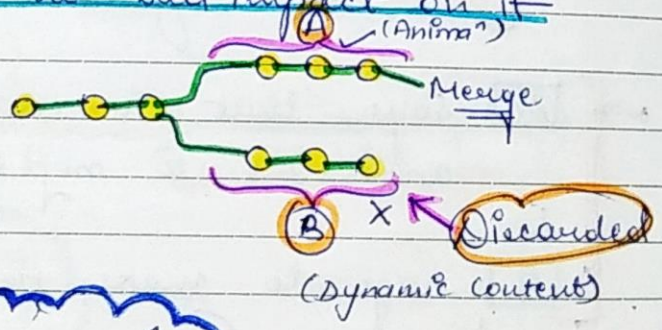
So instead of giving them the main (master branch) to work on, I can give them allowance by creating a side branch like this ------>

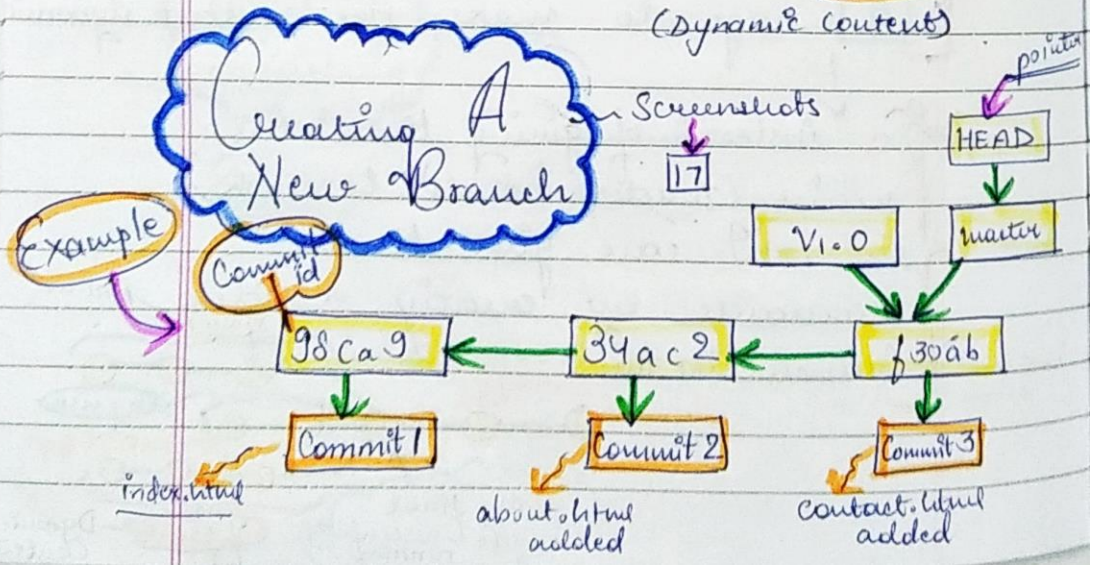


Now suppose both A & B are done with their work & I find that A has done animation part successfully and it is enhancing the quality then I will MERGE her work

While

B's work of making content dynamic isn't proper & so I can discard it & also I have my prev. website successfully running with no bad impact on it





On putting `git log` cmnd, we find

HEAD → master

It means HEAD is pointing to master

`git branch -v`

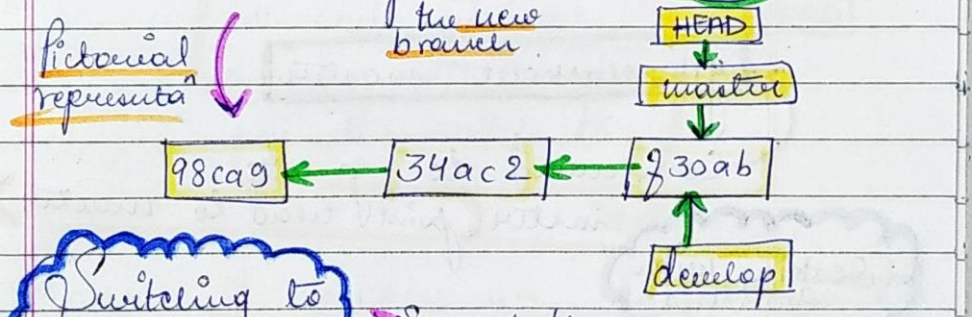
Command to see all the branches

`git branch develop`

develop: desired name given to the new branch

Command to create a new branch

pictorial representation



Switching to Branches → Screenshots 18

Command → `git checkout <branchname>`

PAGE NO.:
DATE: / /

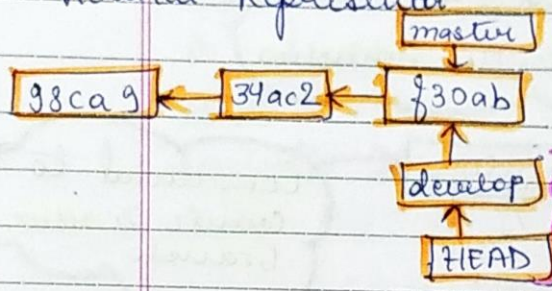
Creating & checkout to a new branch

Command → `git checkout -b <branchname>`

New branch with desired branchname will be created & HEAD will also start pointing to it.

If we execute `git branch -v` command then

Pictorial Representation



We will find that at the same commit id the branch is created & also head is pointing to this new branch

`git checkout master`

Command to go to master (point head to master)

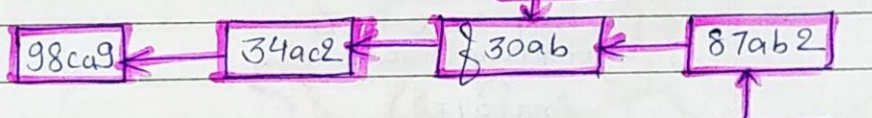
Working with Branches

Now, we will first switch our HEAD to the branch we need to work with using `git checkout` cmd → `git checkout develop`

Now suppose we make changes to contact.html & also add one new file & then do `git add .` & then `git commit`

This whole thing will be done to the develop branch i.e. the side branch

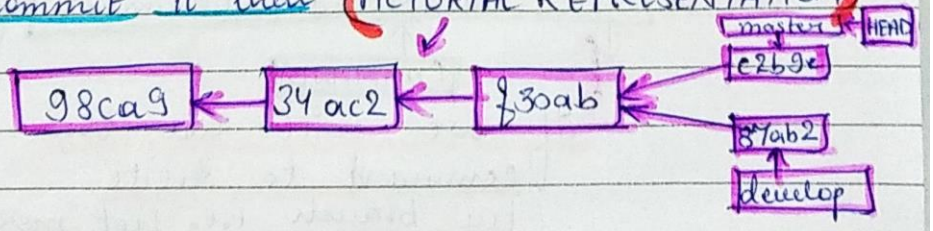
PICTORIAL REPRESENTATION



NOTE: Your working-code i.e. code on local machine will change as per the switch to the branch made

Now if suppose we do `git checkout master`

and start making the change & add & commit it then PICTORIAL REPRESENTATION



PAGE NO.:
DATE: / /

Branch Logging

Screenshots
19

Suppose in the new branch created, we have made some modifications but WITHOUT adding & committing, if we will try switching to other branch like master

NOT POSSIBLE

Confusion
In my case it is working (means POSSIBLE) ??

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

Command to see the pictoral graph (like representation) of the branches.

Deleting Branches

Screenshots
20

Deleting a merged branch

```
git branch -d <branchname>
```

for deleting merged branch

COMMAND

If we will use this command to delete the branch i.e. not merged →

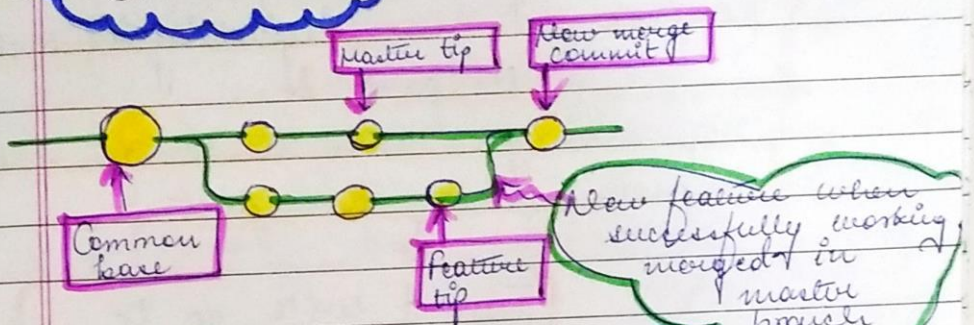
→ It will give an ERROR.

```
git branch -D branchname
```

For deleting non-merged branch

Merging of Branches

Screenshot 21



→ Branching (creation of new branch) is done in order to take a precaution of not messing up with the main branch.

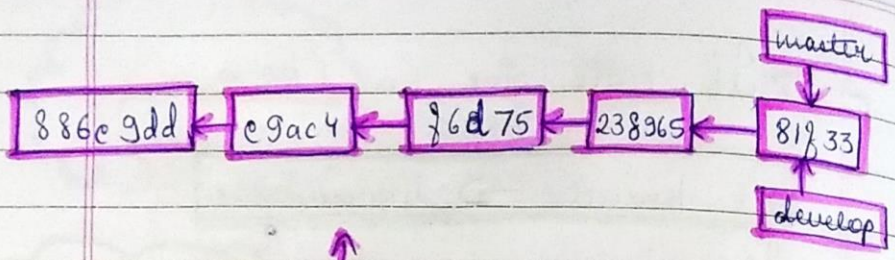
Adding Functionality

(Merge Branches : Example)

→ NEXT PAGE

→ This command can be used also to delete the branch in which no change has been made

(44)

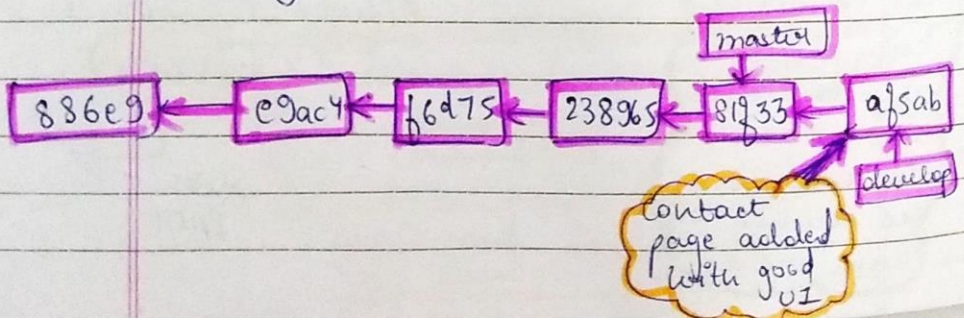


[In screenshots]
(Pictorial representation)

Suppose I am ~~creating~~ improving contact.html
UI part &
my manager asks me to go and improve some other part

Then I will go to the master branch & create a new branch (suppose imp_fix) and then will do the reqd. changes (agreed by manager) in that branch

So we did adding using `git add .`
& then committed using
`git commit -am "better contact page"`



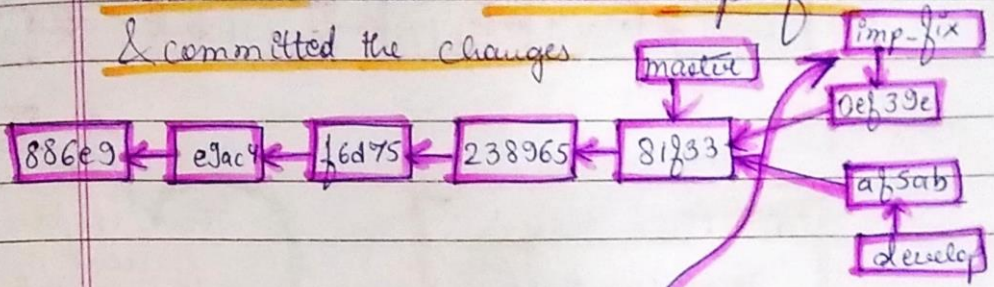
Now we will get back to master & do the changes asked by the manager.

↳ do → git checkout master

Basic Merging

Screenshots [22]

We did changes to index.html i.e. in the branch imp_fix & committed the changes



Now I want to merge this branch (imp-fix) to the master branch

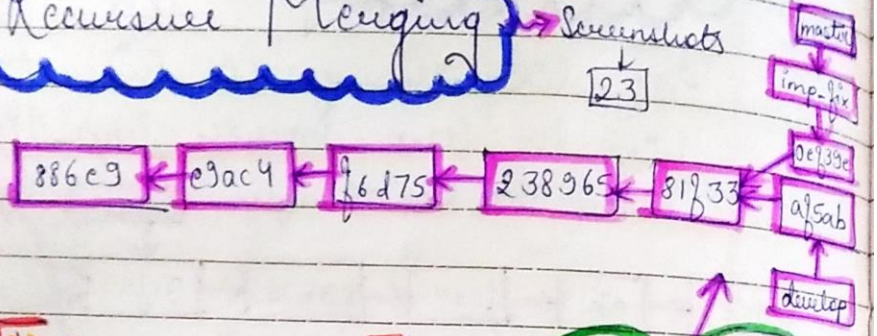
Steps

1. Go to the branch you want to get merged to other branch (means master)
   → git checkout master
2. Now use cmd → git merge <branch_name> imp-fix
   ↑ Branch that we want to get merged to master

PAGE NO.:
DATE: / /

NOTE → `git log` command only shows the commits of the branch in which we are currently present.

Recursive Merging → Snapshots → 23

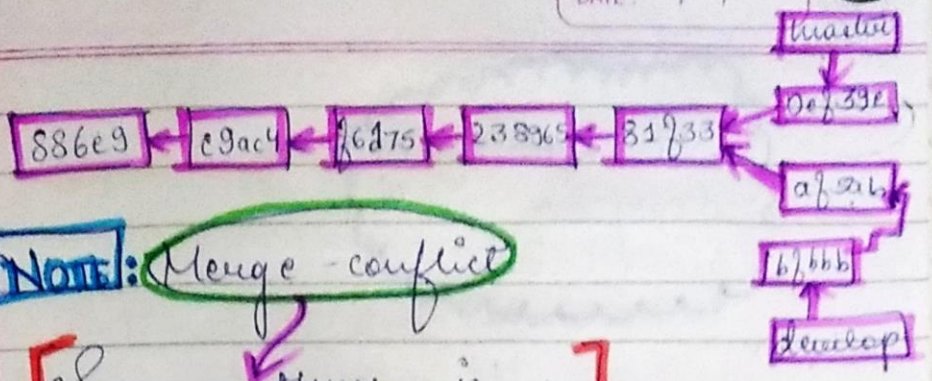


NOTE: We only & generally keep the master branch updated

This is the present scenario & since master is also updated to imp-fix, we will delete the imp-fix branch

Now we are making some updates in our develop branch

```
git checkout develop
Doing some modifications
git commit -a -m
"improved index.html"
```



Note: Merge conflict

Suppose there is a master branch & we are at it, we make some changes to code at a particular line

WHAT WE DID IN MASTER BRANCH

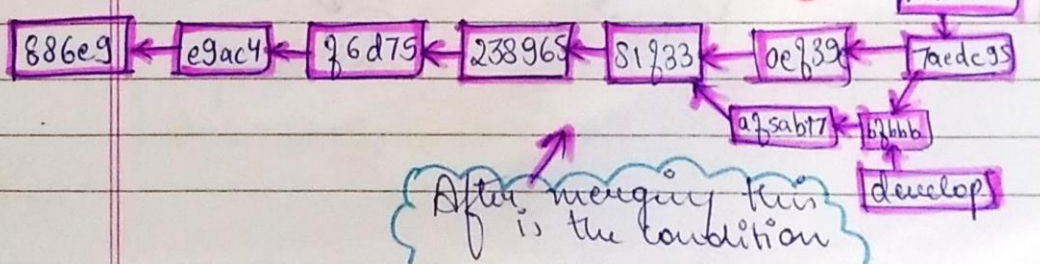
Now suppose in the develop branch when we are, we again make some changes in the same line

WHAT WE DID IN DEVELOP BRANCH

And now when we will try to merge these 2 (master with develop) merge conflict will occur

MERGE CONFLICT OCCURED WHILE MERGING

On VS code it will give us options to select the type of change we want



After merging this is the condition

PAGE NO.:
DATE: / /

Git Branching workflow in production

Branching workflow

Long Running Branches
- Master
- Development

Branch that exists till the project survives

Topic Branches
- Authentication
- UI changes

Created for some particular work

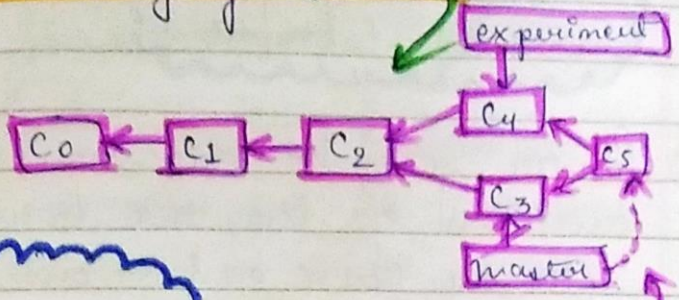
These are short term branches

Rebasing

Integrate changes from one branch into another

- Merging
- Rebasing

So merging is this

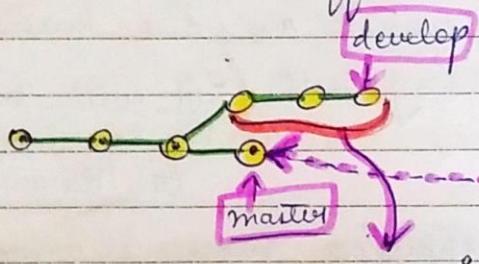


Rebase a Branch → 25 Screenshots

[3 way merging]

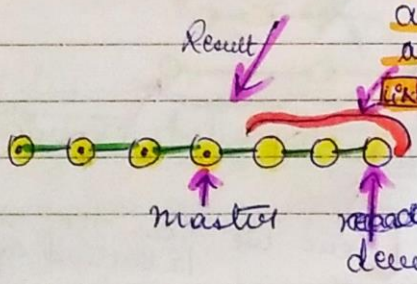
Actual outcome is same as merging

{With rebase command, we take all the changes that were committed on one branch & replay them on a different branch.}



{Go to the branch you want to be appended to master
Do
↓
git rebase master}

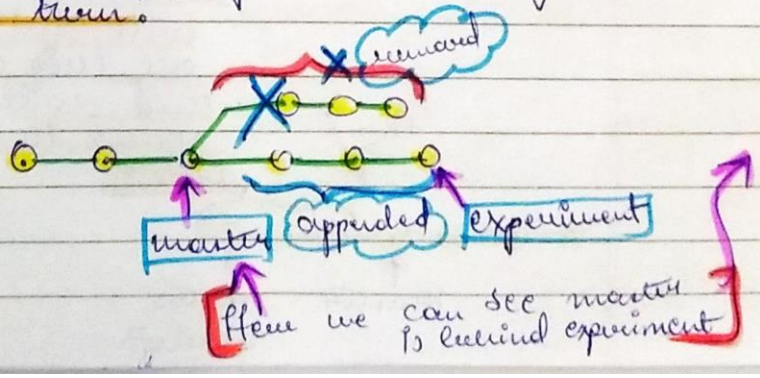
Rebasing will put all the changes and will append them here



PAGE NO.:
DATE: / /

How Git performs rebase internally

1. Pointer goes to the common ancestor of the two branches (the one you're on & the one you're rebasing onto)
2. Gets the diff introduced by each commit of the branch you're on.
3. Saving those diffs to temporary files.
4. Resetting the current branch to the same commit as the branch you are rebasing onto (Git will reset the experiment branch to the end of master).
5. Finally applying each change in turn.



(Rebasing leads to linear development)

To bring the master to the end

Do this:

→ git checkout master
→ git merge experiment

Key Pts. on (Advantages) Rebasing

- No difference in the end product of integration but rebasing makes for a cleaner history.
- Log history looks like a 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.

PAGE NO. :
DATE : / /

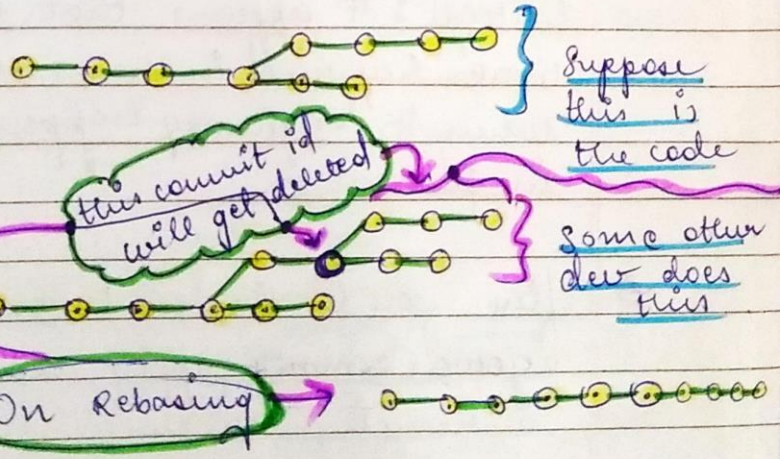
When to NOT use Rebasing

Do not rebase commits that exist outside your repository and that people may have based work on.

(the commit ids of the appended part versions will be different)

[While rebasing, we're abandoning existing commits & creating new ones that are similar but different]

Explained



- If we are working with other collaborators we shouldn't rebase
- It is ok if we are working locally. (to rebase)

Merge v/s Rebase

- One point of view is Commit history is a record of what actually happened
  - Merging shows this
- Opposing point → story of how your project was made
  - Rebasing shows this

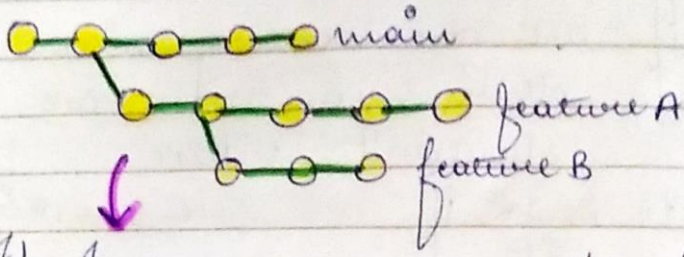
Add this to code part of some other dev was dependent on this commit id which is now deleted

PROBLEM CAUSED

NEXT PAGE →

PAGE NO :
DATE : / /

54



If I want to merge the feature B branch to the main branch.

Command to be used

Git rebase --onto feature A feature B

Introduction to GitHub.

Git v/s GitHub

Git is a version control system that lets you manage & keep track of your source code history.

GitHub is a cloud-based hosting service that lets you manage Git repositories.

If you want to work with open source projects →

That use Git, then GitHub is designed to help you better manage them.

Exploring GitHub

After signing in to GitHub we can see the repositories we have already uploaded & can also upload the repos.

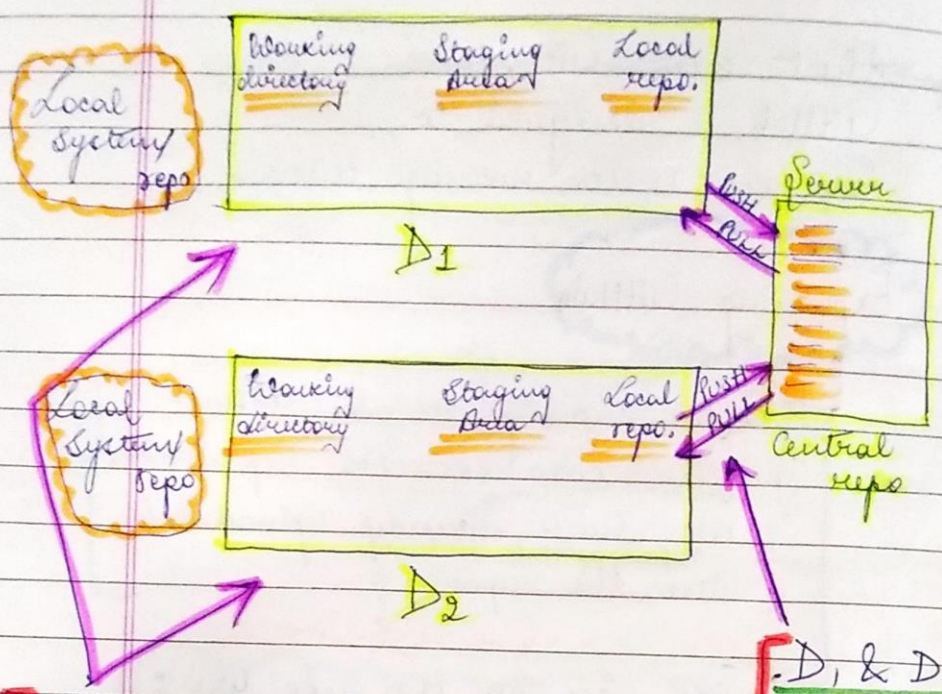
Star sign on the repo use: If someone stars the repo on your profile means in case any update happens to it in future, they will be notified about it.

Need of Central Repository

If 2 or more developers want to work on the same project then there is a need of central repository (server side)

→

PAGE NO.
DATE : / /



Different developers need to work on same project

D1 & D2 pushing & pulling code from central repo

Creating Repo in GitHub

Screenshots 26

Step 1: Right Top side → click [+], Click "New repository"
Step 2: Give Repo name
Step 3: Give Descrip" (optional)
Step 4: Select "Public" or "Private".
Step 5: Select options for "Initialize this repository with".

Step 6: Click "Create repository"

NEXT

See, it will show some commands in order to create a new repo on cmd line

Let's create a folder

Cmds:

1. echo "# udemy-github-repo" >> README.md

Need not have same name as our github repo

Will create a readme file

This file gives the detailed explanation about the github repo

2. git init

Initialize a repo as git repo. so that it starts tracking it

3. git add .

Adding everything in the repo to the staging area

4. git commit -m "first commit"

committing the changes / creating the version

PAGE NO.:
DATE: / /

58

5. `git branch -M main`
   (Not needed generally) → Renaming master branch as main.

6. `git remote add origin https://github.com/pragatii11/udemy-github-repo.git`
   → adding the place where we want to push our code

7. `git push -u origin main`
   - -u = upstream
   - will push the code on GitHub
   - will task afterwards
   - finaley we would be able to see the local repo's files on GitHub

Personal Access Token

NOTE: Before pushing the code to the GitHub, we need to authenticate (that we are the real user)

Authentication ways:
1. id & pwd. (✗ Removed ∵ less secure)
2. PAT (✓) (Personal Access Token)
3. SSH (Most secure)

Setting token authentica"

Step 1: Open GitHub account, move to top right, click profile icon

Step 2: Click settings in drop down. Come to left bottom, there is an option - "Developer Settings"

Step 3: Top left → 3 options
→ Click Personal access token

Step 4: Select "Expiration" after clicking "Generate new token".

Step 5: Click on everything you want the token to have an access to.

Step 6: In the end, click on
Generate Token
→ You will get the token
→ Note: keep it secure don't give it to anyone

USE
→ Suppose you are pushing the code & github asks for our id & password but we don't give our original password, then we can give this instead.

PAGE NO. :
DATE : / /

PROBLEM → Suppose by mistake we lost this or forgot that where we kept this token

SOLUTION → We can use **git cache** to store this.

Different OS have diff. ways
(for MacOS → Keychain access)

Steps → Click

Step 1: Control Panel

Step 2: User Accounts

Step 3: Credential Manager

Step 4: Edit Generic credential

→ github.com

→ Change the password to the token generated

Step 5: Click **Save**

PAT works over HTTPS

So if we have cloned repo on HTTPS use PAT

If repo created on SSH, use SSH