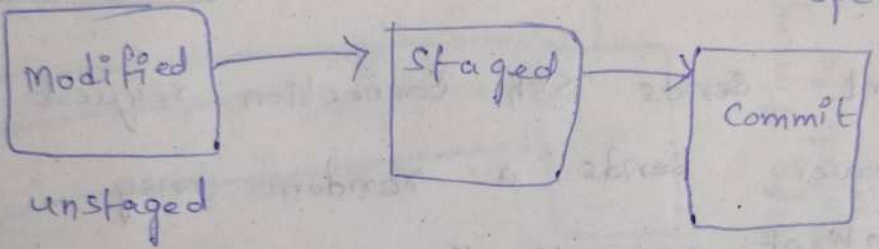
Change the data in the Sai.txt [manually by entering file]

- "welcome to speridian!"

- Run git status [shows modified Sai.txt]
- git add Sai.txt
- git status [Green : modified Sai.txt]
- git Commit -m " Added to speridian'



- Modify the file [Manually by entering file]

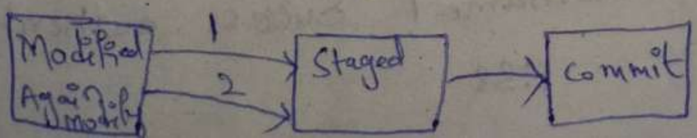
" welcome to speridian"

" Hello"

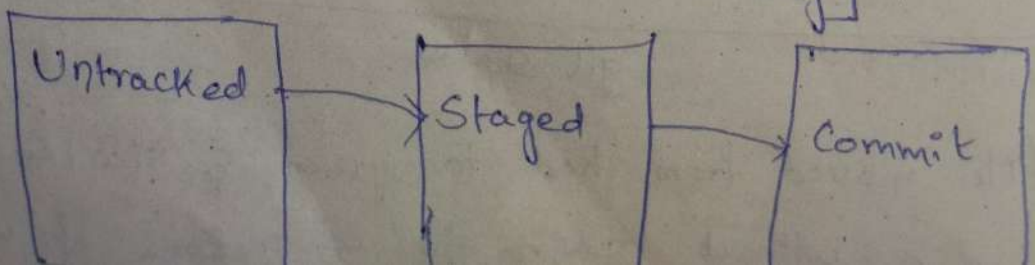
- git status [Shows Red modified : Sai.txt]
- git add Sai.txt [~~Shows Green " : "~~]
- git status [shows Green " : "]
  (Staging)
- Modify the file again [Manually by entering file]

  " Hello Sai "

- git Status [Shows one in staging
  Shows one in modified]



- git config --global --list [Shows uses name, emai
- git config --global user.name " "
- git config --global user.email " "
- git config --global --list [shows the names the we changed]
- [User.name, User.email Should be Same as we given in github]
- git Config --global init.defaultBranch " "
- git config --global --list [Shows the branch we created]
- Create a folder test1
- open git bash inside
- Run git init [Enter inside .git it shows branches, hooks, info, objects, refs, config, description, head]
- echo welcome> sai.txt
- git Status [shows Red(untracked)]
- git add sai.txt
- git status [Shows Green (new file : sai.txt)]
- git Commit -m "Created welcome inside sai.txt"
- git status [shows nothing]

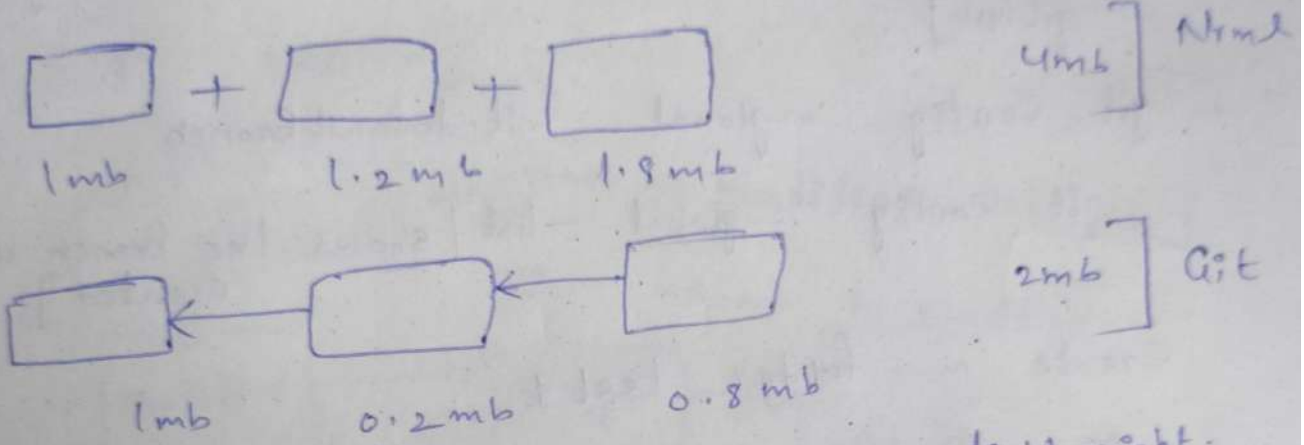


Tracking:-

what did we do [change]

~~when did we~~

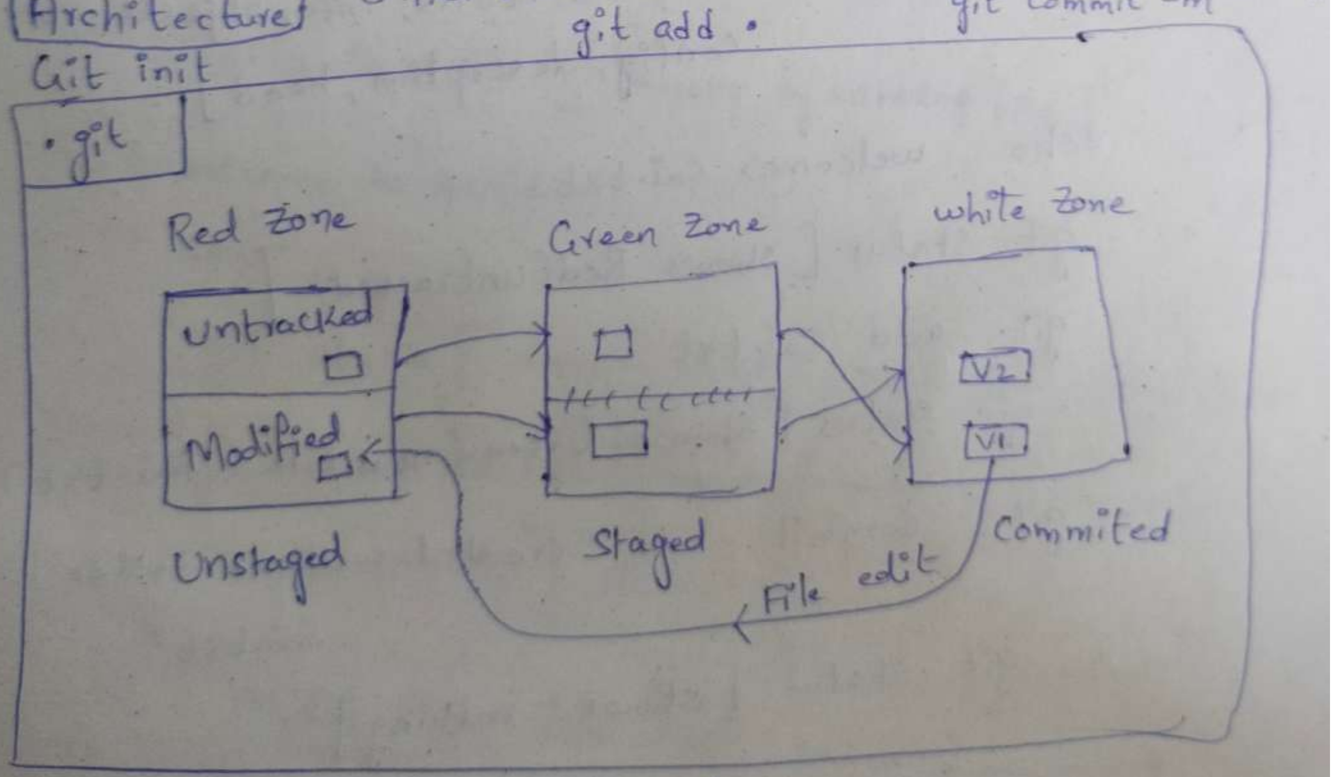
who is the person did [Name, email]



Git maintains the versions in a lightweight.

1:30:45

Architecture 3 Tier Architecture

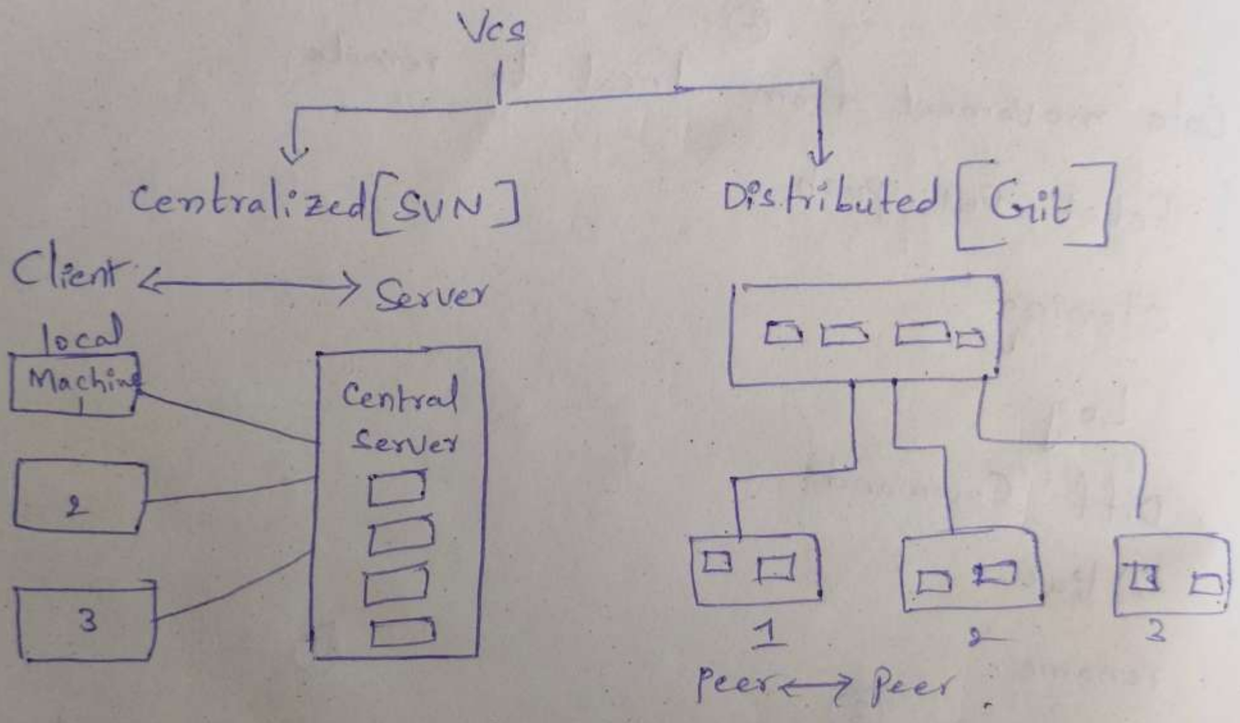


git status

one file moved from Red to green & started editing it again without moving to white Zone if we run git Status what should it will show?

- Fork
- Pull request
- Branching
- Merging
- Conflict

- Rebase
- walkthrough GitLab, Bitbucket, Egit.
- Project.



- Internet lequired
- Single point failure.
- Machines maintaining ~~Versions~~ records only on Central Server.
- Everything is lecorded only on Central Server.

- Versions are maintain both in Central & local Servers.

- If we don't have internet then we cannot do lecording of a Version.

Conclusion: We record only in one place in Centralised Vc

- git add sai.txt
- git status [Green]
- git commit -m " last "

②

✓SSH setup

Git
Client

Github
Server

- Client sends ssh Connection request.
- Server sends a random msg
- client encrypts the msg with pvt key & sends it back.
- Server decrypts the msg using pub key and on authen succeds.

Keys

Pub	→	.ssh/id_rsa.pub
Pvt	→	.ssh/id_rsa

- ssh-keygen -t rsa -C "email"

↳ Should be gith email.

- ssh -T git@github.com

Hi githubusername! Success authenticated.

1:38

⊡ log

```
git log

git log --oneline

git log --author authorname
```

⊡ Alias

```
git config --global alias.lo "log --oneline"

git lo
git log --oneline
```

} → Shows Sameoutput.

⊡ How to delete alias

```
git Config --global --unset alias.lo
```

⊡ Renaming

```
git mv oldname newname
```

⊡ `git remote -v` [Verify any remote Connected to local or not]

⊡ `git remote add origin` [remote url]

↳ To Connect to remote Repo from local.

⊡ `git remote remove <url>`

⊡ `git log reponame/branchname` [Logs from Remot Repo]

⊡ Difference b/w two Commits

```
git diff old commit new Commit --file
```

⊡ Pushing 1st time

```
git push -u origin <remote branch>
```

⊡ next time

```
git push
```

⊡ Cloning a Repo

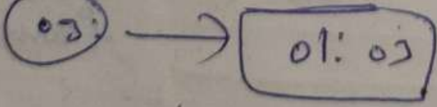
```
git clone < url >
```

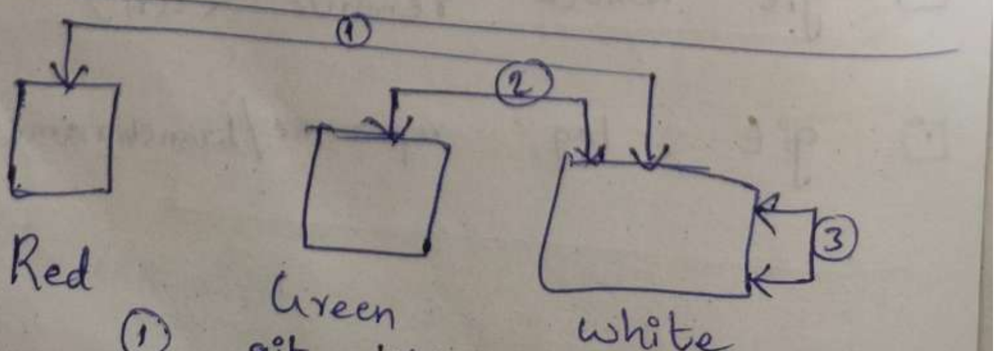
⊡ cloning only a particular branch from remo
Repo.

```
git clone -b <Branchname> < url >
```

⊡ list all files in a directory

```
git ls-files --directory <Directory Name>
```





① git diff filename
② git diff
③ git diff old --staged file

Diff

- git diff — untracked Red — modified Red

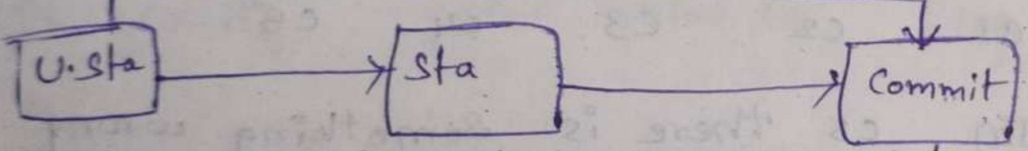
  git add .

- [git diff --staged filename] — Staged Green

  git commit -m " "

- git diff old commit ID new commit ID

- Remote vs local = git diff origin/main



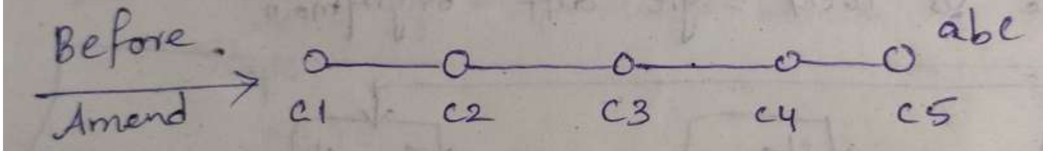
- edi the Same file
- check status [File is in modified U.staged]
- Run git diff.
- Move the file to Staging
- Run git diff
- Move the file to commit
- Run git diff.

- Always rename the file through cli·

- git commit -am " " [only works for modified file not for untracked file]

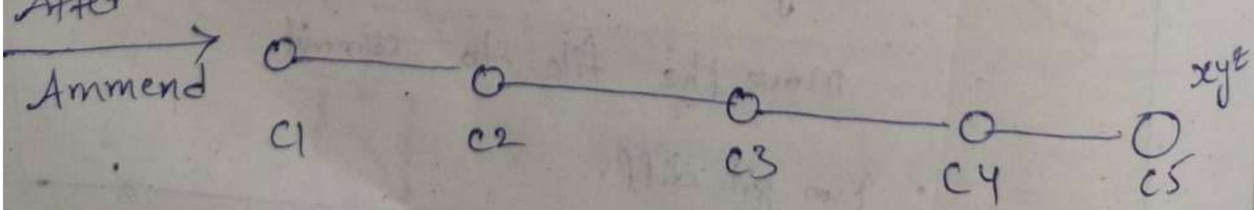
untracked
Modified
unstaged.

02:06:44 Amend



In c5 there is something wrong which is done by me. If i want to hide it from others then we have to use "Amend" Command.

- Amend will leplace the position of c5 and it lemoves the existing ID and leplace it with new ID



Amend will only work if the final commit is not pushed to lemote Repo.

```
· git log -p < filename >
```

```
· echo hi > sai
· git add .
· git commit -m " c1 "

· echo hi am new >> sai
· git commit --amend -am " c2 "
```

Note:- After using amend command c1ID will be replaced by c2ID.

```
□ git config --global alias.lo "log --oneline"
```

```
□ git reflog [It shows all logs including
                    amend also]
□ git remote rename origin github
□ git remote remove origin.
```

□ git ignore

Initially at the time of creating remote repo .gitigno
is also created.

If our developers working on java project
we need to select .gitignore as java.

without # → taken into effect
with # → neglect.

Inside .gitignore for java project

*.class [Compiled class file]
*.log [log files]
*.ctxt [Blues files]
.mtj.tmp/ [Mobile tools for java J2ME]

*.jar
*.war [Package files]
*.nar
*.ear
*.zip
*.tar.gz
*.rar

- ~~echo hi > sai.jar~~
- ~~git status [shows upto date]~~

~~Note:- Instead of showing untracked it will show upto date.~~

~~These type of files are not pushed to remote Repo.~~

- ~~echo hi > sai.jar [where .jar is not in .gitignore]~~
- ~~git add . [Shows sai.jar is in green~~
- ~~[edit the .gitignore and include .jar]~~
- ~~git status [only shows that .gitignore is modified]~~

~~file~~ ~~unstaged~~ ~~staged~~

~~→ If file is in unstaged it will n~~ ~~in .gitignore~~ ~~track~~

~~file~~ ~~unstaged~~ ~~staged~~ ~~Staged~~

~~→ If file is in staged & if added that to .gitignore it will track.~~

□ a.docx → moved to staging area → Then add to .gitignore → git status [shows a.docx]
→ If commit & push then a.docx moved to Remote Repo.

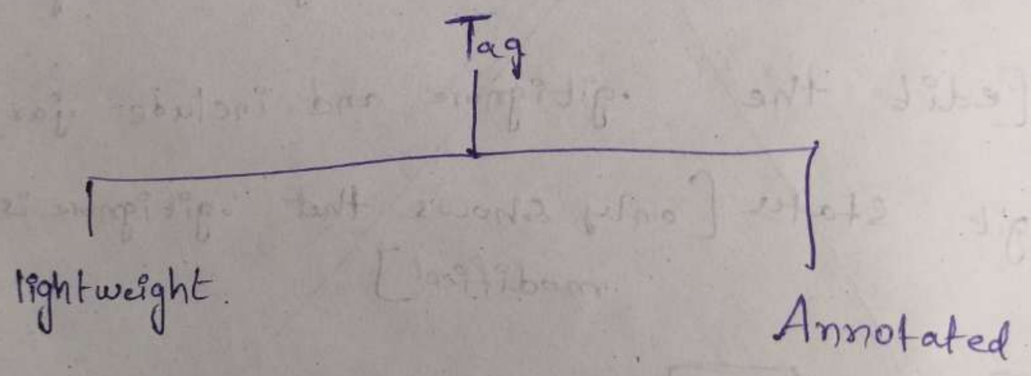
□ a.docx is added in .gitignore → edit a.docx still status shows upto date.

If we create one file and if it shows untracked, modified it is clearly a not a ~~untracked file~~. gitignore file.

- git config --global --list

Imp ⊡ git config --global Core.excludesFile [.gitign

⊡ Tagging 03:50:09



- git tag --list

- ~~git tag bID817 < any~~

Lightweight tag

⊡ git tag < Tagname> < commit ID>

⊡ Tag that we added is in yellow colour.

Annotated Tag

⊡ git tag -a < Tag name> < Commit-Id > -m "message"

⊡ git tag -n [shows tags with msgs]

- Annotated tags store extra metadata such as Tagger name, email & date. ~~Emp~~ This is imp data for public release.
- Leightweight tags are essentially "bookmarks to Commit". They are just name & pointer to a Commit.

- git ~~de~~ tag -d < tagname > [For deleting tag].
- git show < tagname > [Shows author, Date]
- git push --tags [Push tags to remote repo]
- git push --delete origin < Tagname >
  ↳ Delete tag from remote.

6

⊡ Stashing 04:13:04

⊡ echo hi > test.txt

⊡ git Status

⊡ git add .

⊡ git commit -m "c1 "

⊡ git log --oneline.

⊡ ~~ecbo~~ Add Some matter to test.txt.

⊡ git status [modified test.txt]

⊡ The modified file I want to move it to Stashing area

⊡ git ~~status~~ stash filename

whatever the Content that is inside the file will be removed and moved to Stashing area

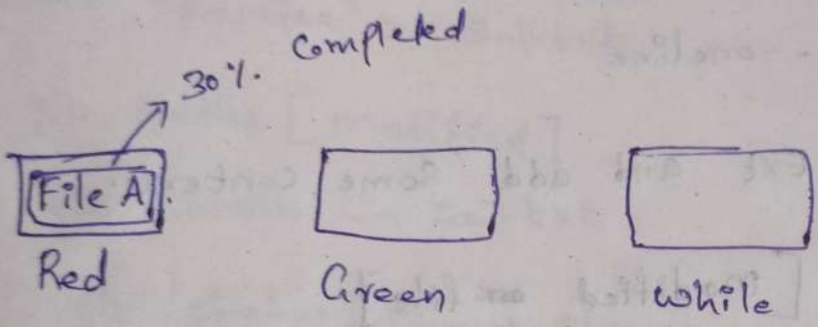
⊡ git Status [Shows our branch is ahead of 1 Commit instead of Showing modified test.txt]

⊡ git stash list

Stashing:-

A Separate space given by git if we dont want to mark it as Commit and we dont want to move data from local to remote then we will go for Stashing.

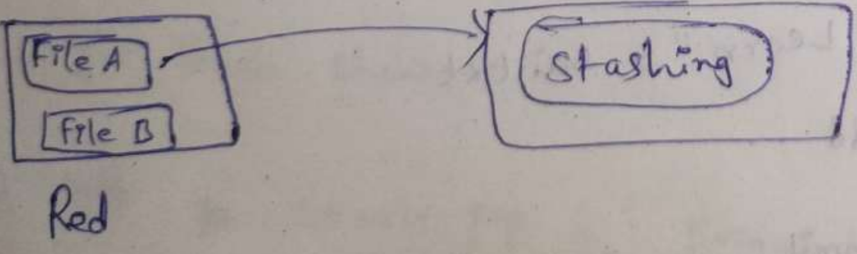
Stashing Area



If a file is not completed and we want to keep in separate area.

File A → Half done.

After few days there is a situation arise that you need to work with another file. only after finishing File B then only come to FileA.



In such cases git provides Separate space & it removes file from Red A and move it to Stashing area.

□ Create test.txt

□ git Status

□ git add .

□ git Commit -m "c1"

I. □ git status

□ git log --oneline

□ open test.txt and add Some content.

□ git status [Modified ~~as~~ file]

□ Now I want to move the file to Stashing are

□ git Stash --filename [After running this Command data is demoved from the file]

□ git status [It doesn't tell that test.txt i exist here]

□ echo "Learn" > sai.txt

□ git add .

□ git commit -m "c2"

□ echo "Lets enjoy" >> Sai.txt.

□ git Status [Modified Sai.txt]

□ git commit -am"modified"

□ git status

□ git log --online [It doesn't show anything about test.txt]

□ git stash ~~list~~ list

□ echo "practice" >> sai.txt

□ git status [modified]

□ git stash -- sai.txt

□ git status [shows clean]

□ git stash list

□ git ~~status~~ stash show [last modified stash]

□ git stash ~~-p~~ show -p

□ git stash show 0 [shows which file is on 0 position.

□ git stash show 1

□ git ~~to~~ stash pop 1 [Brings back test.txt ~~to~~ to from stashing area]

□ git stash --test.txt save "test.txt"

□ git stash list [shows 0 in test.txt
" 1 in sai.txt]

File	stash position
A	0

If we add File B

File	Stash position
A	1
B	0

□ git stash show [Shows the last ~~modified~~ Stash]

□ git Stash Clear [Removes all data from stashing area]

↳ Do not follow.

Note:- "Git stash" will only work if the file touches staging area.".

⊡ clean

⊡ git clean -n [will tell which files are going to be ~~[illegible]~~ have an impact]

⊡ git clean -dx -n [will tell which all dir, files will delete]

⊡ git clean -fdx [f - force, d - directory, x - ignore files]

⊡ git clean -f -d [To delete all files & dir]

Note: Clean only for untracked.

⊡ cache

git

⊡ touch abcd.txt

⊡ git add .

⊡ git commit -m " c3 "

⊡ git rm --cached abcd.txt

[Shows 2 copies
1. deleted abcd.txt [in staging area]
2. Untracked abcd.txt]

⊡ git commit -m " c4 "

⊡ git status [Still shows abcd.txt still in untracked]

⊡ Add abcd.txt to .gitignore

⊡ git status [abcd.txt Vanish, modified .gitignor

· Rebase Should not be done in group activity

Main	Dev	Main
m1	~~8~~d1	m3
m2	d2	

- git Checkout dev
- git log [Shows m1 m2 d1 d2]
- git rebase main
- git log --oneline [m1 m2 m3 ~~fa~~ ~~fa~~ d1 d2]
- git checkout main
- git merge dev.

Rebase: Remove the ~~p~~ attachment and reattach ~~wi~~ ~~to the~~ ~~updated~~ at the end ~~ad~~ edge of branch.

GitLab → Used by Devops experts.

Atlessian projects Comes with Bit Bucket, jira, Confluence → US projects.

· Pull Request in GitHub is called "Merge Request"

□ Merge with Conflict

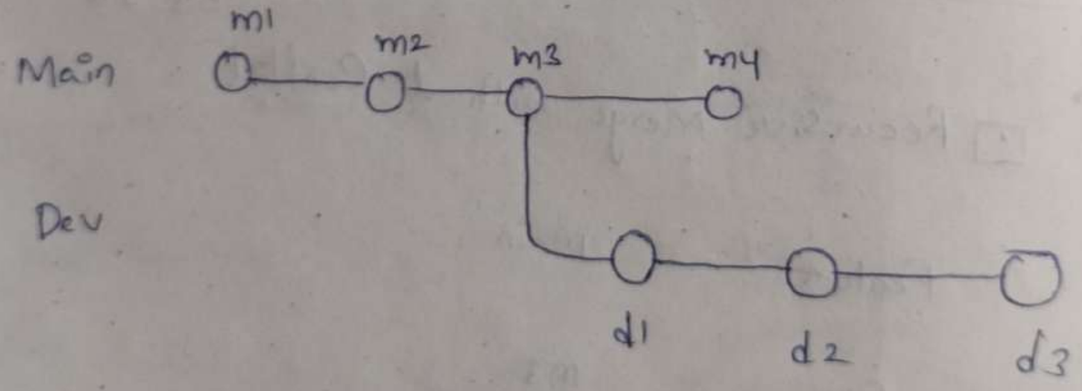
Main	feature	Main	feature
m1 (a.txt)	f1 (file1)	m3 (a.txt)	f3 (a.txt)
m2 (a.txt)	f2 (file2)		

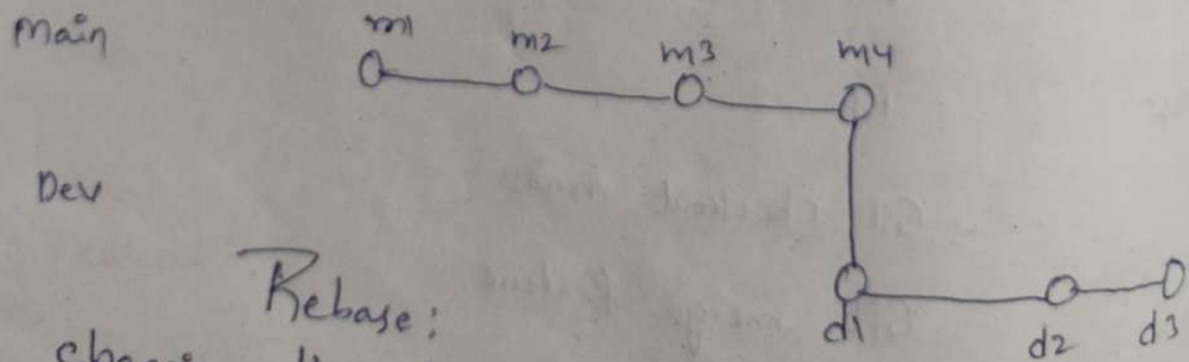
```
Git checkout main

git merge feature
```

Merge:-
Bringing together of two diff info into a Single p
Fast-forward Merge: No changes in parent branc

Recursive: change in both parent, local bran

□ ReBase



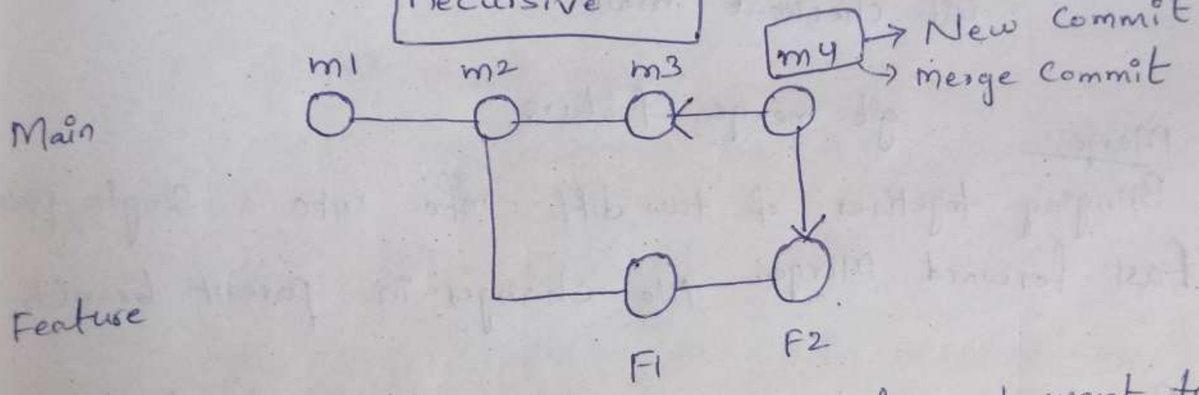


Rebase:
changing the base of the branch from m3 t
m4 Commit

Fast-forward: It's a technique to re-attach the update to the parent branch.

Note:- It is only used when parent branch ~~is~~ has ~~Known to us.~~ no changes. who is asking us to merge.

Recursive



Both branches have different info and want to merge

03:30

□ Recursive Merge without Conflict

Main	Feature	Main
M1	F1	M3
M2	F2	

. Git checkout main

. Git merge feature

· To See all the branches on local & Remote

git branch -a [Green → local
Red → Remote]

· Renaming

☐ git branch -m <oldname> <newname> ✓

☐ git branch -m <Name of we want to assign to our branch>

To View Remote branch list:

· git branch -r

git reflog → It ~~main~~ tracks our work as a process and it will record everything.

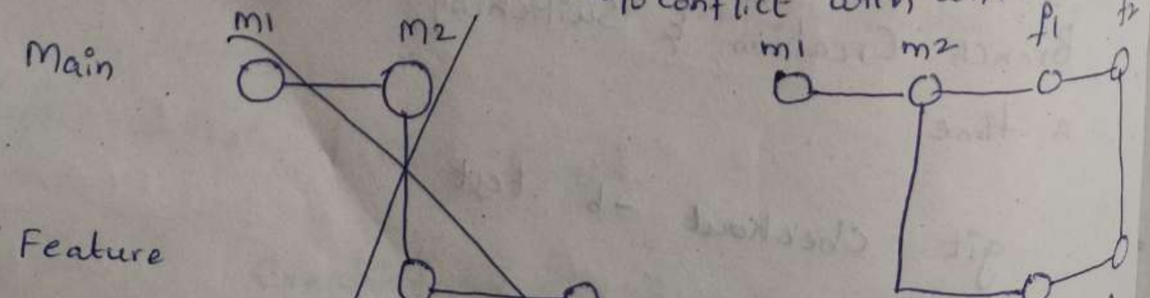
☐ Merge

Fast forward

Recursive

w/o conflict

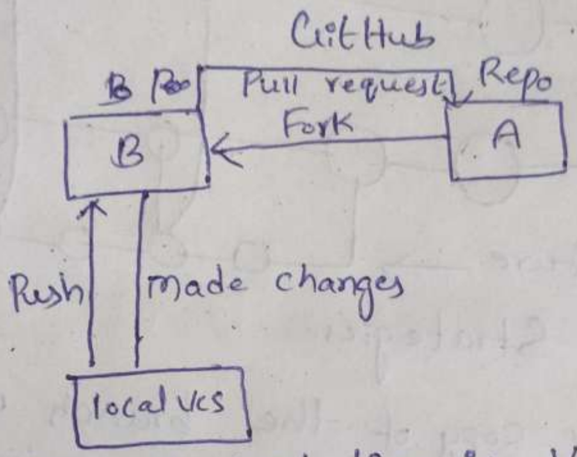
with conflict



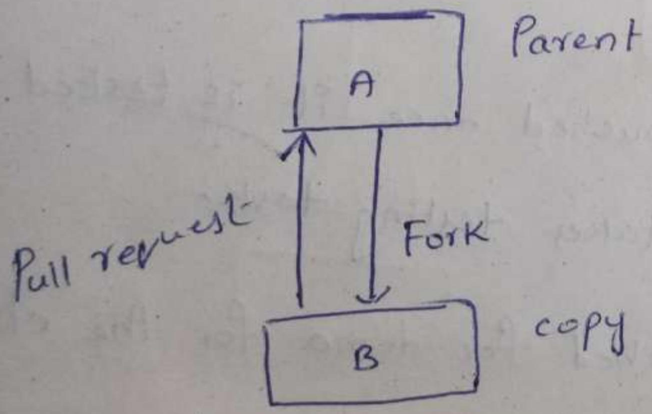
Rather than Creating Commit fast forward will update the chang
the into main branch.

Fork:- whatever the changes that applies to Parent Same will be applied to forked Repos.

Fork & Clone → changes wont Come to cloned ones.

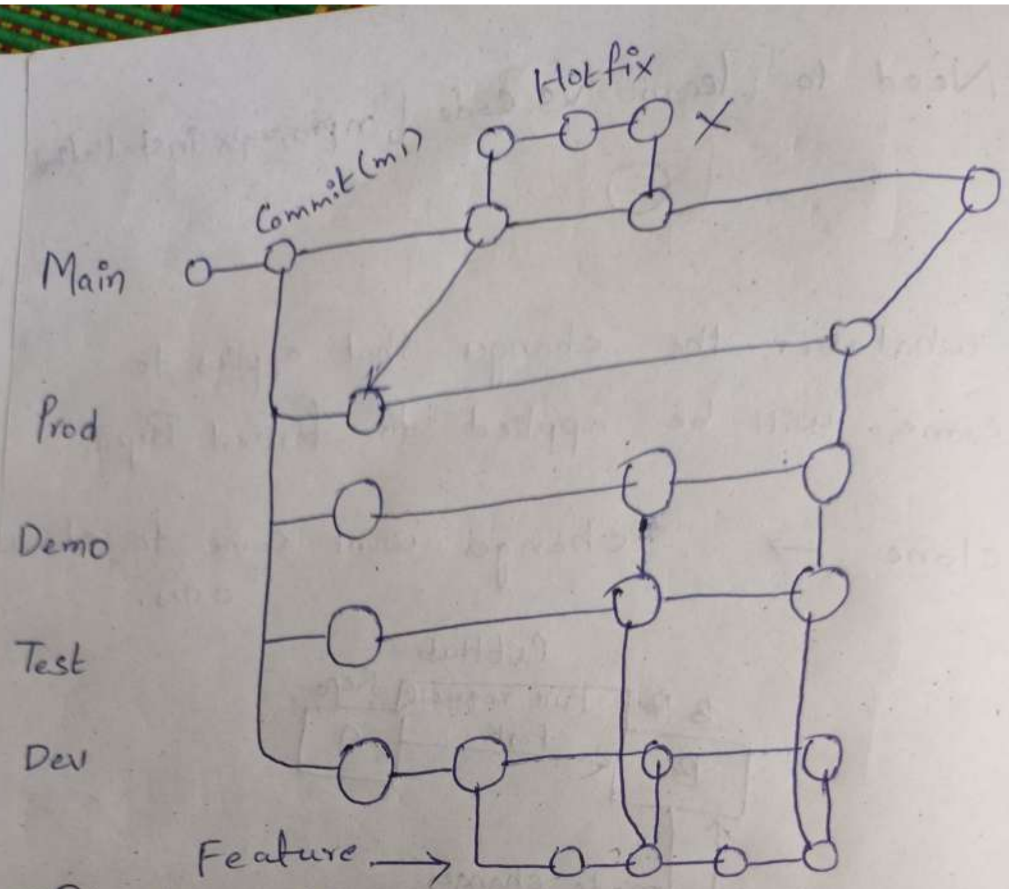


Through Fork our Contribution is visible in another Project.



If B made any changes and want to Send those changes to A then need to request Pull request only if Parent (A) accepts then only changes will be pushed to A

01:54



□ Branching Strategies.

Main: Master copy of the branch where we push only tested code

Dev: Master Branch for developers.

Testing/stage: code is pushed once it is tested in Dev. QA team takes testing task.

Pre-prod: Code is pushed for demo for the client

Bugfix/Hotfix: Fixing Bugs.

Feature: Creating Various features.

□ Branching

- git init
- touch abc.txt
- git add .
- git commit -m " m1 "
- git status [shows ntg to Commit]
- git branch [shows main branch]

Creation of branch:

- git branch < name of branch>
- git branch dev
- git branch [shows 2 branches]
- git branch prod
- git branch [shows 3 branches]

To change to a particular branch:

- git checkout dev

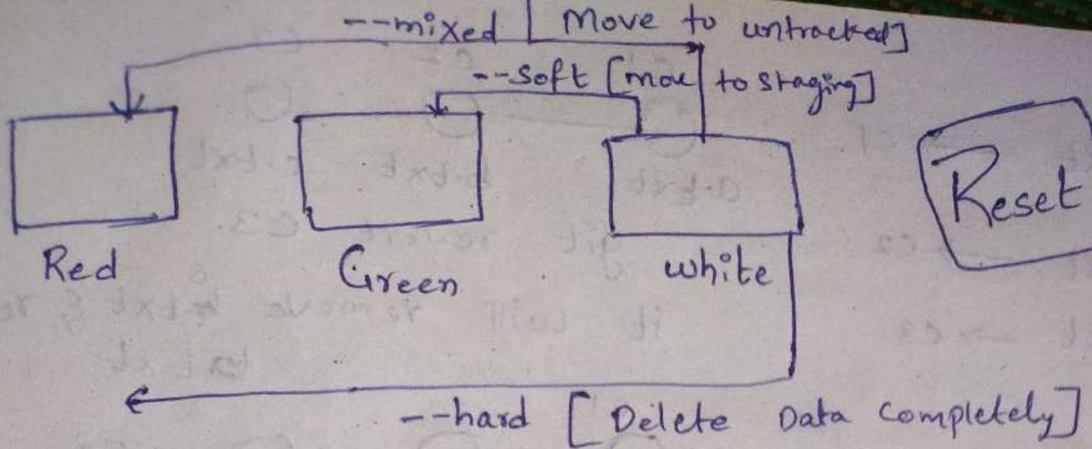
Branch Creation & Switching to that branch at a time

- git checkout -b test

- Delete a branch

  git branch < branchname> -d

Reset



Note:-

c1
c2 } c → soft
c3

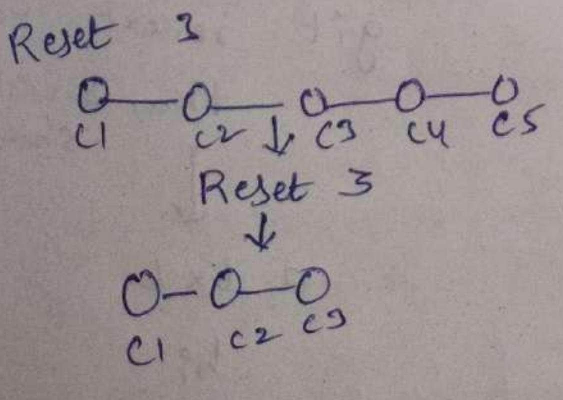
If we want to club multiple commits into a single commit Reset command is useful. [--soft]

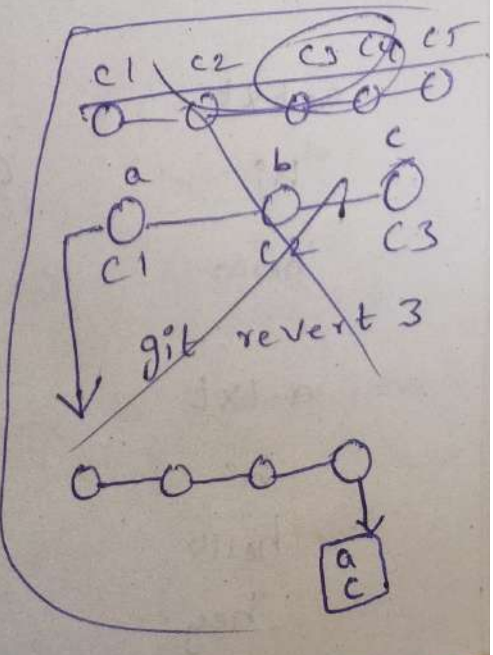
Note:-

⊡ If any file present in untracked [Red] and we wanto to lemove that [git clean -f]

Reset → Delete

Revert → X

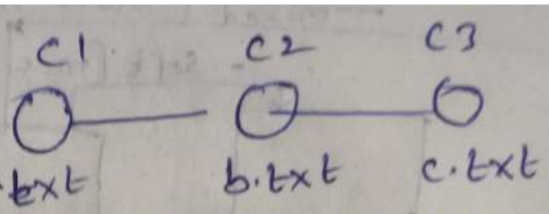




a.txt → c1

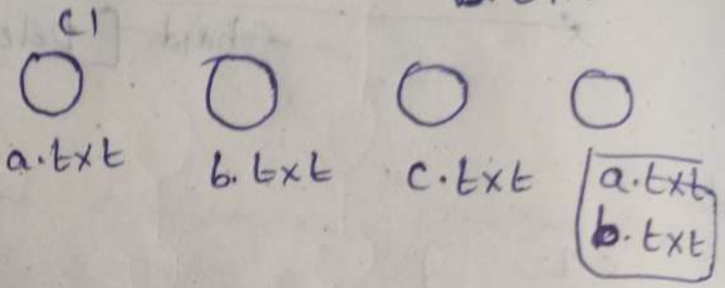
b.txt → c2

c.txt → c3



git revert C3

it will remove ~~b~~c.txt & replace ~~b~~b.txt



File	Commit ID
a.txt	m1 [16]
b.txt	m2 [17]
c.txt	m3 [18]

git revert 17 → it will ~~remove~~ delete b.txt and add ~~new~~extra commit. [m1, m3]
a.txt c.txt

a.txt

hi C1

a.txt

hi C2

hello

a.txt

hi

hello C3

git revert C2

then it will show

↓

a.txt

hi

⊡ git commit -am "ca"

⊡ RM Command

⊡ git rm -f <filename> [Removes the file from git & file system.]

⊡ git rm --cached <filename> [Remove the f
from git but file is still there in project]

caching:- Removing the file from the records but it will move to untracked area after Cacheing and after moving to untracking area we will add that file to .gitignore file.

⊡ Reset

⊡ git reset --soft <commit-ID> [Move the file to Staging area]

⊡ git reset --mixed <commit-ID> [Move the file t unstaged area]

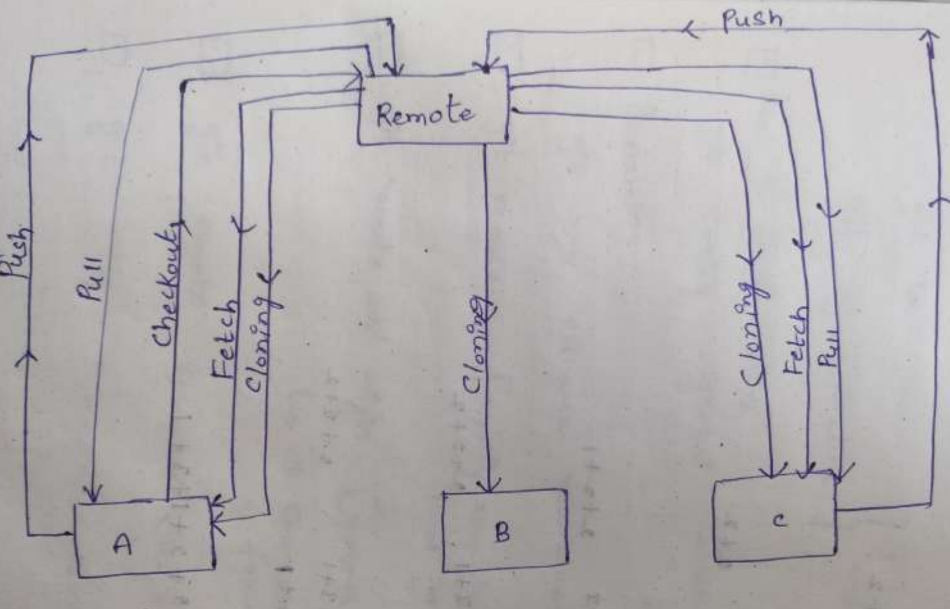
02:23

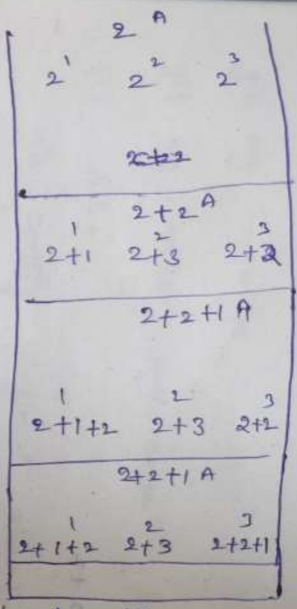
Soft
hard
Mixed } Reset

Reset - Manipulate Version history

Revert - To move to a previous change state

Clone
Fetch
Checkout
Pull
push





on Remote machine Pm will Create Project

Checkout: Analyze the commits on the remote repo through our local machine.
[what is the stage of the project in Remote Repo]

Fetch: will tell about what are the updates on Remote machine.
checks only what is there in remote machine and brings the history to us.

Pull: If any new commits are there in Remote Repo those will be pulled to our machine.

Push: Moves the new commits from our local to Remote.

①

- Introduction
- system Setup and Configuration.
- Basics of git Architecture
- Basic Commands of Git local
- Introduction GitHub
- Account Setup
- SSH Introduction and Configuration.

②

- Data movement from local to remote.
- Fetch, Pull, push
- cloning
- Log
- Diff Commands
- Alias
- rename.

③

- Amend
- Tags
- Stash
- ignore
- checkout

④

- Clean = Untracked
- Rm = Tracked
- Reset = Manupulate Version History.
- Revert = To move to a previous change state.
- Fork
- Pull request