Git

VCS → is a s/w that helps s/w developers to work together and maintain a complete history of their work.

→ Allows developers to work simultaneously.
→ Doesn't allow overwritting each other changes.
→ maintain history of every version.

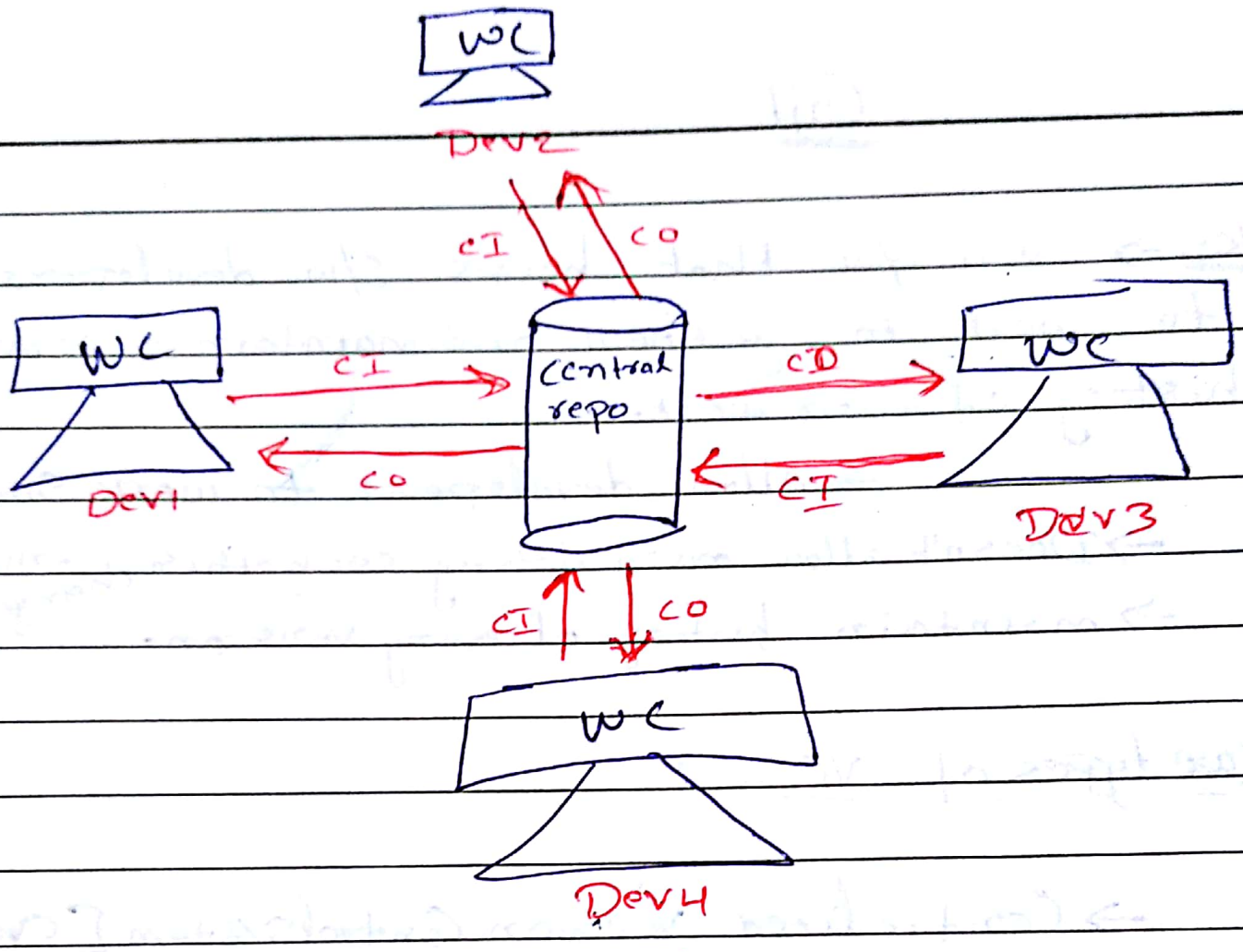
Two types of VCS's:

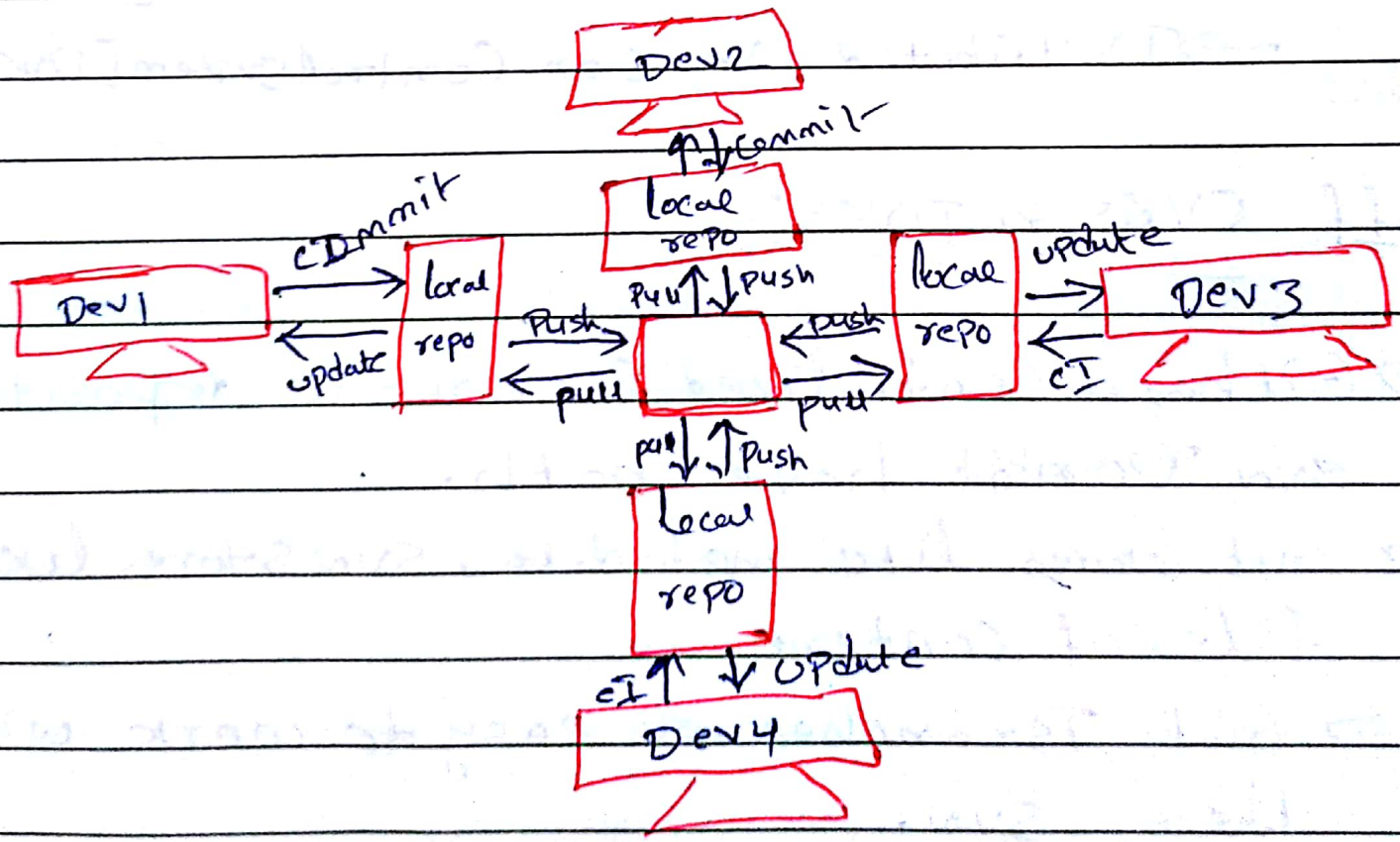
→ Centralized version Control system [CVCS]
→ Distributed Version Control system [DVCS]

Diff CVCS & DVCS:-

→ Git has a Centralized Server and repositary. SVN Doesn't have both.
→ Git stores like metadata, SVN stores like files of content.
→ Git Branches are easy to work with then SVN.
→ Git has Better Content Protection.
* → Git doesnot have a Global revision number like SVN.
→ Git, [written in 'C'] used by the linux and Ruby on Rails.

Scanned by CamScanner





Scanned by CamScanner

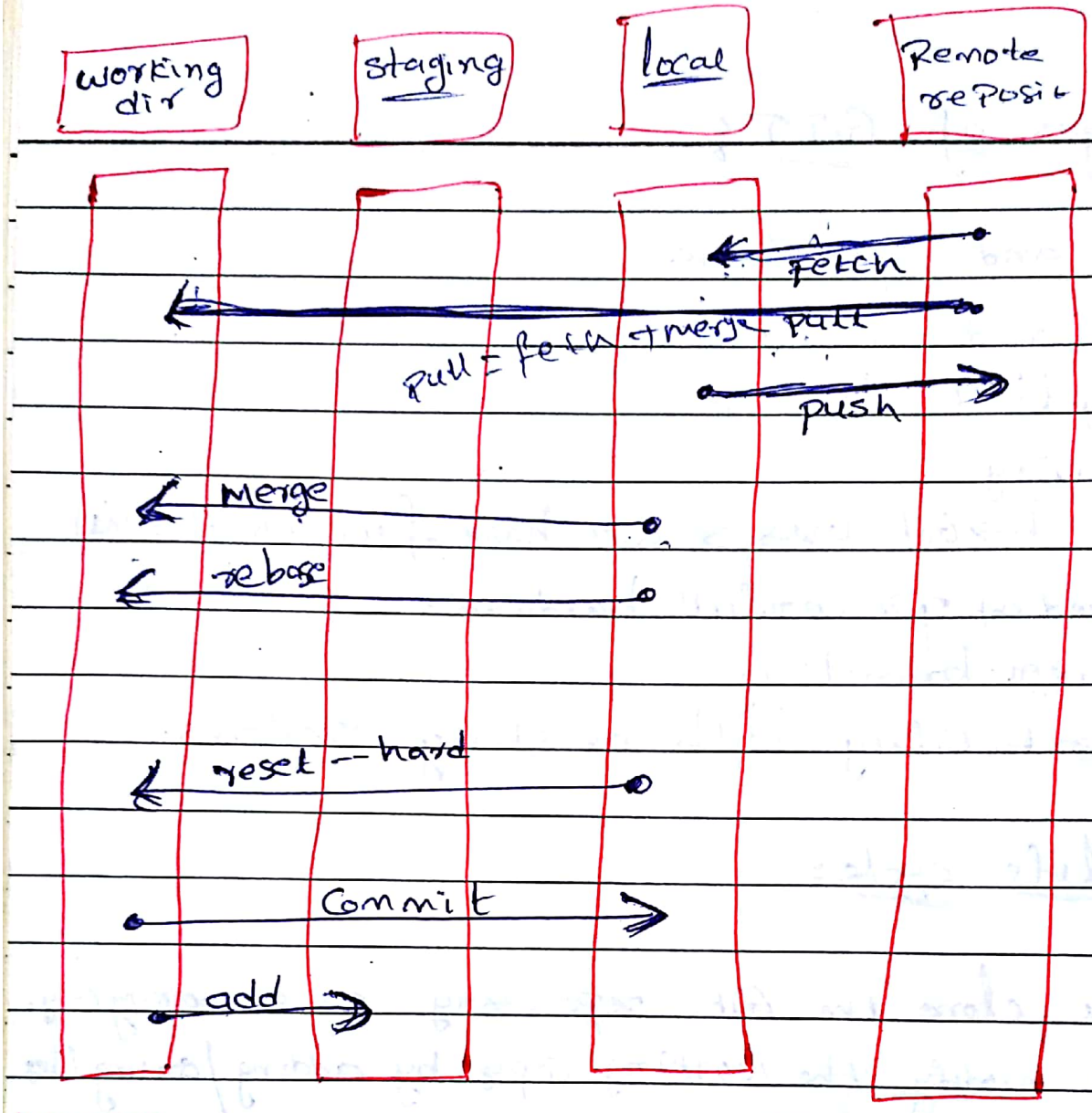
Advantages of GIT:

→ Free and open Source
→ Fast and small.
→ Implicit backup
→ Security
  ↳ Git uses secure hash function. [SHA1]
→ No need of powerfull hardware.
→ Easier branching
→ Compatability with existing systems.

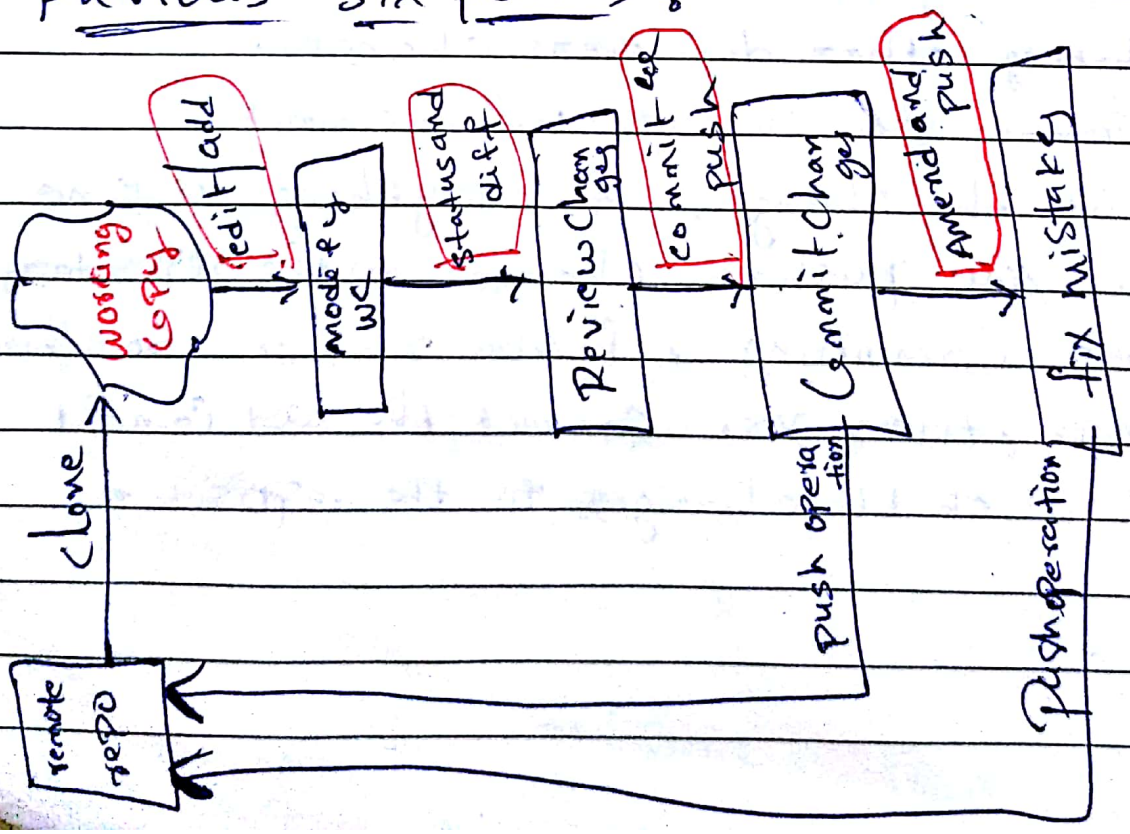
Git life cycle:

1. you clone the Git repositary as a working copy.
2. you modify the working copy by adding / editing files.
3. if necessary, you also update the working copy by taking other developers changes.
4. you review the changes before commit.
5. Commit the changes if everything is fine Then you Push the changes to the repositary.
6. After Commiting, if you realize Somthing wrong, then you Correct the last Commit and Push the changes to the repositary.

Scanned by CamScanner

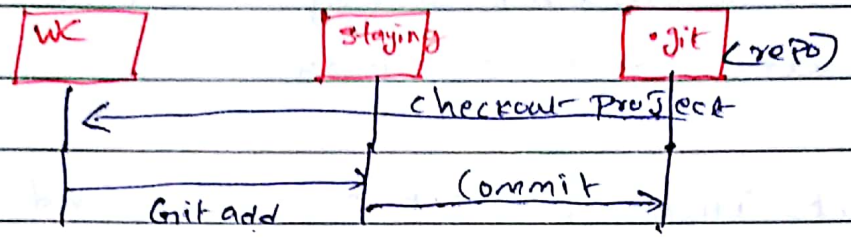


Previous six points :



Scanned by CamScanner

Git workflow :



→ Here there are thee areas, maintained by Git

(i) first one is "working Copy" which holds Actual files.

(ii) The Second one is "Index (or) stagging" acts as a staging

(iii) The Third one is ".repositary" which Points the last commit.

List of basic Commands :

1. Git Init
2. Git clone
3. Git config ✓
4. Git add
5. Git commit
6. Git status
7. Git log
8. Git blame
9. git rm
10. git mv
11. git diff
12. git show ✓
13. git branch
14. git checkout
15. git reset
16. git tag
17. git fetch ✓
18. git pull ✓
19. git push
20. git remote
21. git merge
22. git rebase ✓
23. git clean
24. git stash ✓
25. git blame ✓
26. git revert
27. git cherry-pick ✓
28. gitk

Scanned by CamScanner

① Git init :-

Initializes a Git repository and working copy in specified directory.
It creates '.git' directory

Ex: Git init [select one dir and type it through Git bash]

Git init --bare :

initialize the an empty Git repo, but omit the working directory.

difference b/w init and init --bare :

Ex: Git init var	Ex: Git init --bare var
var → .git	var → hooks, info, objects, refs, Config, Head
It has working copy	No working copy only repositary.

→ Then pushing the existing repo into Remote repositary

```
git remote add origin <url>
```

→ it synchronus local & remote repos.

Scanned by CamScanner

② Git clone :

→ Clone the remote repositary into your local working copy via HTTP (or) SSH

It is like SVN co

→ Git repositary — It has own history, and manages it's own files, and it is completly isolated environment from the original repo.

→ cloning Automatically creates a remote connection called "origin".

Ex :

```
Git clone <repo-url>
```

↳ clone the repo located at repo into local mechine.

→ `git clone <repo-url> <directory>`

↳ clonning into pertical dir location

→ `git clone <repo-url> -b <branch-name>`

we are checkout (or) clone to perticular branch and pointing out the perticular branch.

Scanned by CamScanner

③ Git Config :

→ This is used to change your Git Configuration Settings.

→ By using Git Config we can set & Configure variables [/etc/gitconfig]

Ex :

Git Config --global user.name "Narasimha"
git Config --global user.email "nrao9951@gmail.com

Git Config --list
git Config --global core.editor emacs

CHECKING :

git Config user.name
↳ checking username.
git Config user.email
↳ checking email

Scanned by CamScanner

(4) Git add :

The "git add" Command adds a change in the working directory to the staging area.

→ It tells Git that you want to include update to a particular file in the Next Commit

Usage :

$\boxed{\text{git add -A}}$

↳ Stages all

~~git add~~

$\boxed{\begin{array}{l}\text{git add .} \\ \text{git add -u}\end{array}}$

git add . → Stages New & modified [without deleted]

git add -u → Stages modified & deleted [without New]

→ It will update already tracked files.

* git add -p →

Scanned by CamScanner

⑤ Git Commit :

① Reward changes to the repositary
② It Commits the ~~changes~~ staged.

Ex: git Commit
↳ Commit the staged Snapshot
But this will launch a text editor prompting ~~ye~~ for a Commit message.

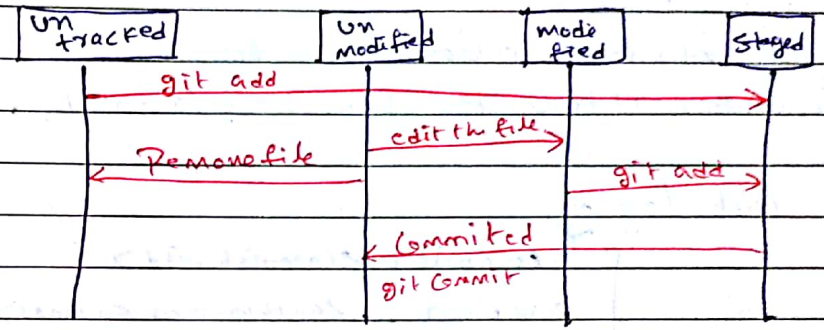
git Commit -m " message "
↳ Instead of text editor we can give message directly.

git Commit -a -m " message "
[-am]

↳ git add .
git Commit -m "msg"

(But not new files)

lifecycle In local



⑥ Git status :

→ Git status Command displays the state of the working directory and the staging area.

→ we can see which files are changed, deleted, staged, un tracked.

Ex: Git status
new file ok.txt

Git status -s
A ok.txt

Scanned by CamScanner

⑦ Git log :-

It is tool allows you to view Information about the Previous commits.

Ex:- Git log ⇒

```
commit : <commit_Id>
author : <author name email>
Date   : < Date of the commit>
```

→ git log -2
↳ It will display the last two commits

→ git log --oneline :- viewing oneline commit

→ git log --author <name>
↳ Display commits based on the author name

→ git log --before <date>

→ git log --after <date>

→ git log --after <date> --before <date>
Ex:- git log --after "2016-04-01" --before "2016-04-29"

→ git log -p ⇒ view the entire diff of changes for each commit found

→ git log --stat ⇒ show the sumary of changes in git

⑧ git blame :-

git blame Command can be used to determind when a certain changes was made

Ex:- git blame <filename>

Process:-

```
Touch <filename>
git add .
git commit -m "mesg"
vi <filename>
or, we can,
save
Then again git commit -am " mesg"
git blame <filename>
```

Scanned by CamScanner

9) git rm :

Remove files from the working tre and index

Ex: git rm <filename>

git rm --cached <filename>

↳ Remove the file from the index but keep your copy in the local file system.

10) git mv :

move or rename a file, a directory

Ex: Git mv ok.txt welcome.txt

Git status

↳ Here ok.txt was deleted. welcome.txt is created.

Git mv -f k.txt kk.txt

↳ forcefully that is if already kk.txt exist forcefully replaced.

Git add -u

↳ It is Importent here to It will update already tracked files.

11) git diff :

show the changes b/w Commits, Commit ce working tree.

Ex: Git diff

↳ Compare modified files

git diff --color-words

↳ Compare modified files and highlights changes only

git diff --staged

↳ Compare modified files, within the staging area.

git diff master <another Baranch-name>

↳ Compare Branchs

git diff <commit-id> <Commit-id>

↳ Compare commits.

12) Git show :

show the various types of objects.

Syntax: git show <option> <objects>

Description :-

① shows one or more objects [blobs, trees, tags & Commits]

② For commit it shows the log mesg and textual diff

③ For tags, it shows the tag mesg.

④ For trees, it shows the names

⑤ For blobs, it shows the Plain Context

Scanned by CamScanner

Ex: git show → It will display the information about latest commit

git show <commit-id>
↳ It will display particular commit info

git show v1.0.0
↳ It will display the tag v1.0.0. along with the object

git show master → try your own

13) Git branch:

git branch command used to create branches, listing branches, deleting branches.

Creating Branch: $\boxed{\text{git branch <Branch-name>}}$

Listing Branch: git branch
Listing remote branch only ↑ git branch -r

Listing all branches: git branch -a
↳ listing local as well as remote

Delete Branch:
git branch -d <Branch-name>
git branch -D <Branch-name> → forcably

Rename current Branch
git branch -m <Branch-name>

14) git checkout:

git checkout command is used to switch one branch to another branch and restoring the changes of working tree files.

syntax:
git checkout <options>

git checkout ABC → switched to ABC Branch

git checkout -b xyz → It will create & switched to xyz Branch.

git checkout <commit-id> <filename> → To revert the local changes
(first 8 letters)

git checkout master -- <filename>
↳ It will update our file with master branch file data in current branch.

Note:
pushing to remote
git push origin <Branch-name>
git push origin <branch-name> : <another name>
↳ pushing local branch to remote with other name.

git branch -d <branch-name>
↳ delete local branch

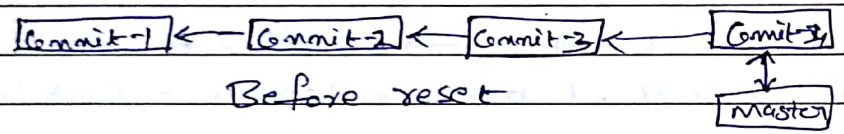
git push origin : <branch-name>
↳ delete remote as well

Scanned by CamScanner

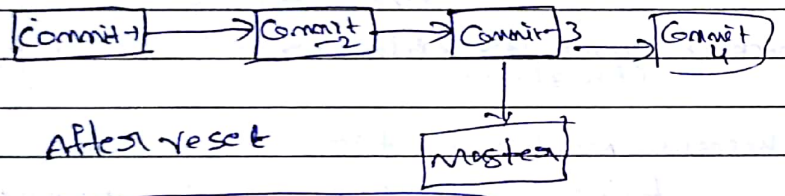
15) Git reset :-

Move HEAD Pointer with Git Reset

→ After doing few changes, you may decide to remove these changes, The git reset command is used to reset ~~the~~ (or) revert changes.



Before reset



After reset

Ext git reset <filename>

Note: where head Pointer location in various branch

- .git/refs/heads/master

Soft:

It will reset the Head Pointer only

git reset --soft HEAD

Mixed:

revert the those changes from the staging area that have not been commited yet. It reverts changes from the staging area only.

hard:

It will clear the staging area, it will reset the HEAD pointer to the latest commit of specific commit ID.

16) git tag :

Git Tag are used for creating "Stable release", By using git tag command we can create tags, delete tags and list tags.

Syntax:

git tag <tag-name>

→ create a tag

git tag release 1.0

→ Push into remote

git push origin release 1.0

→ listing tags

git tag --list

→ git tag -d release 1.0

↳ Delete tag

git push origin :release 1.0

delete remote repository.

Git Pull --tags

Fetch and update local branch list in the local

Scanned by CamScanner

17) Git fetch :

git fetch command is used to gathers any commits from the target branch that doesnot exist in your current branch and stores them in you local repo.

But it doesnot merge into your current branch,

Ex:- git fetch origin

git fetch develop : develop

18) Git Pull :

git Pull command used to Fetch changes from remote branch and intigrate with local branchs.

$\boxed{\text{git pull origin}}$

Git pull = git fetch + git merge

git pull origin master → updating only master branch.

19) Git Push :

git push command is used to push your local repositary changes to remote repositary like [new branches, new commits, tags --- etc]

Syntax :

git push origin <options>

Ex:

git push origin master
git push origin --all
git push origin --tags
git push origin --force.

20) git remote :

Manage set of tracked repositaries.

→ git remote is command used to list, add and delete remote repositari alias.

syntax:

git remote <options>

Ex: git remote ↳ It will list all your remote alias

git remote -v
↳ display like aliases and fetch and push urls

gitremote add <url>
↳ It will add new remote repo of your project

Scanned by CamScanner

→ git remote rm origin
↳ removing the existing alias

→ git remote rename <old-alias> <new-alias>
↳ rename remote aliages names.

→ git remote set-url origin <new-url>
↳ It will update an existing remote url with specified.

→ git remote show <url>
↳ It will show information about remote url.

21) git merge :

It is a Command used to merge the Content of another branch into current branch

syntax: git Merge <options> <branch-name>

Ex: git merge master [But we are at slave]
↳ This Command will merge Concept of master Command into Current branch

git merge --no-off --no-Commit test
↳ It will merge test branch into our branch, but It will not "Commited."

22) git rebase :

In Git, there are two main ways to intigrate changes from one branch to another the MERGE and REBASE.

Rebase is a processes of moving a branch to a new base Commit.

syntax: git rebase <branch name>

{ git checkout feture
git rebase master } → you can rebase the future branch onto master branch

git rebase -i master.

23) Git clean

This Command is used to Remove untracked files from the working tree.

Ex: git clean -f -d (or) git clean -fd
↳ To remove directories which are untracked

git clean -fx (or) git clean -fx
↳ To remove files which are untracked

Scanned by CamScanner

24) Git Stash:

git stash Command is used to stash the changes in current working directory.

→ stash temporarily saves your changes, you have made to your working copy so you can work on something else and then come back and re-apply them later on.

Syntax :- → git stash <options>

Ex git stash
↳ It will save our local changes.

→ git stash list
↳ It will display the list of stashes

→ git stash pop <stash name>
↳ It will re-apply the stash in our local working directory

→ git stash show
↳ It will display changes in stash

→ git stash drop <stash name>
↳ It will display changes in stash

25) git blame :-

It will display what revision (commit id) and author last modified each line of a file.

Ex git blame <filename>

26) git revert :-

revert some existing commits (or) undoing changes.

Ex git revert <commit_id>

ls :- ok.txt, wel.txt
git rm ok.txt
git commit -m "ok.txt deleted"
git status
ls → wel.txt
git revert <commit_Id>
ls → ok.txt, wel.txt

Scanned by CamScanner

27) git-cherry-pick :-

Apply the changes introduced by some existing commits.

Ex: git cherry-pick -x <commit_hash>

Note:

It may avoid merge conflicts in the future

28) gitk :-

It is a git repositary browser

Ex: gitk --all

→ It is a viewing repo histories and understanding the relationship b/w different branches.

Branching strategy :-

fork strategy :-

① fork is a copy of a repositary

② Forking a repositary allows you to freely experiment with changes without effecting the original project.

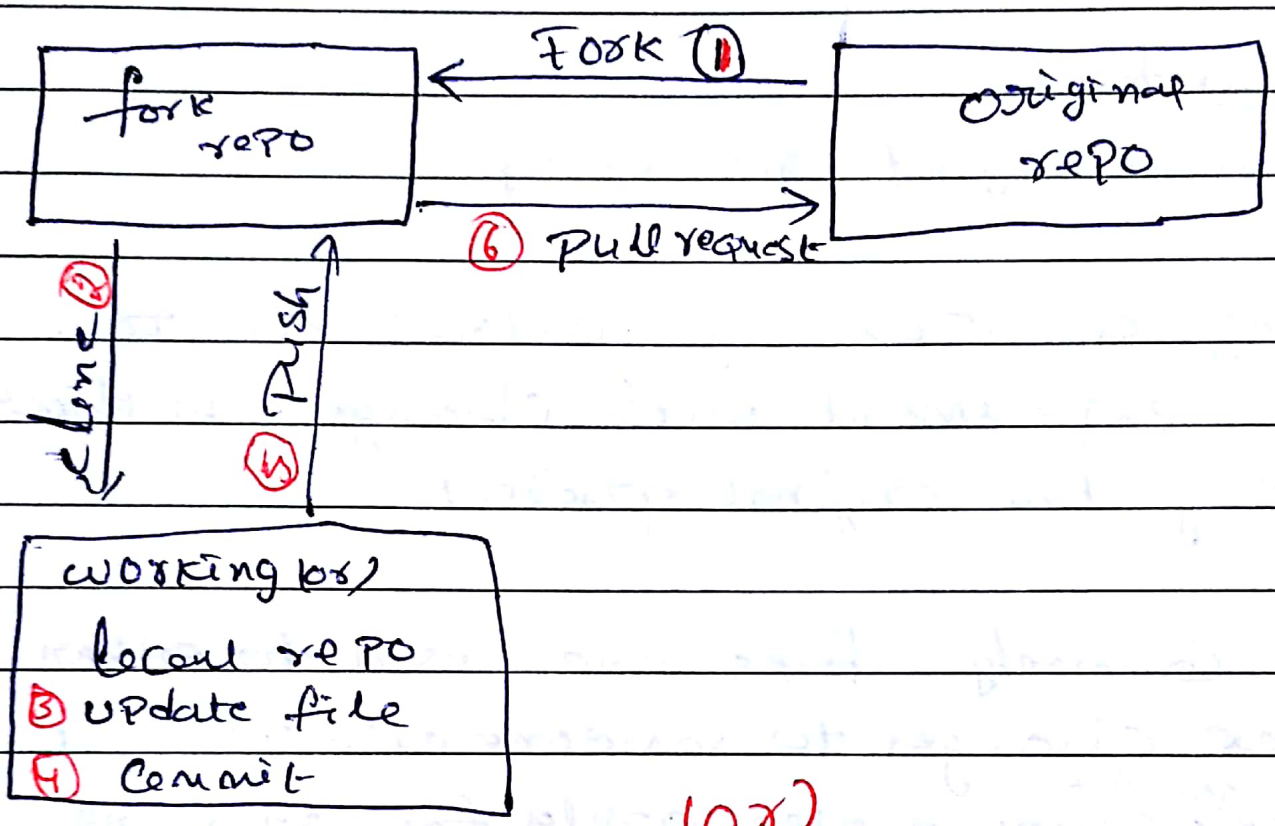
③ most commonly forks are used to either propose changes to someone else project (or) use project someone else apply for starting point for your own idea.

Ex: we have a great example of using forks to propose changes is for "Bug fixes". Rather than logging an issue for a bug you have found. you can.

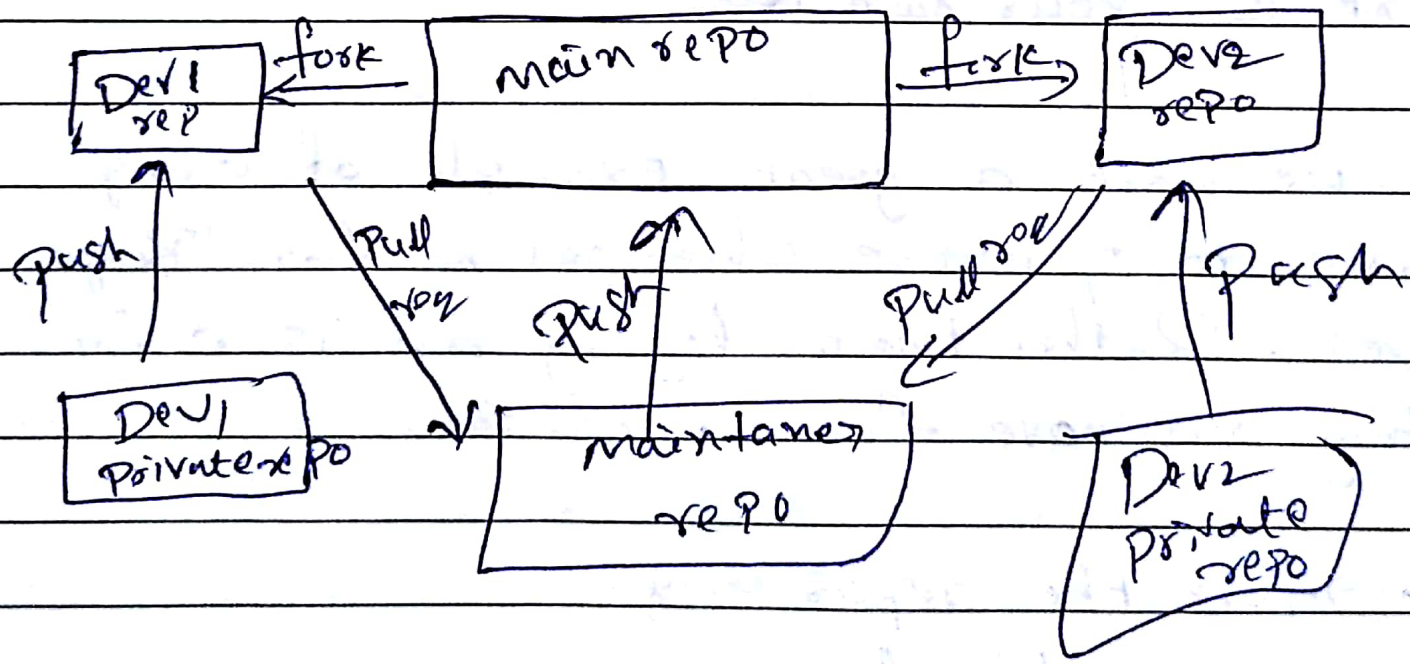
→ fork the repositary.
→ make the fix
→ Submit pullrequest to the project manager

If the project manager likes your work, they might pull your fix in to original repo.

Scanned by CamScanner

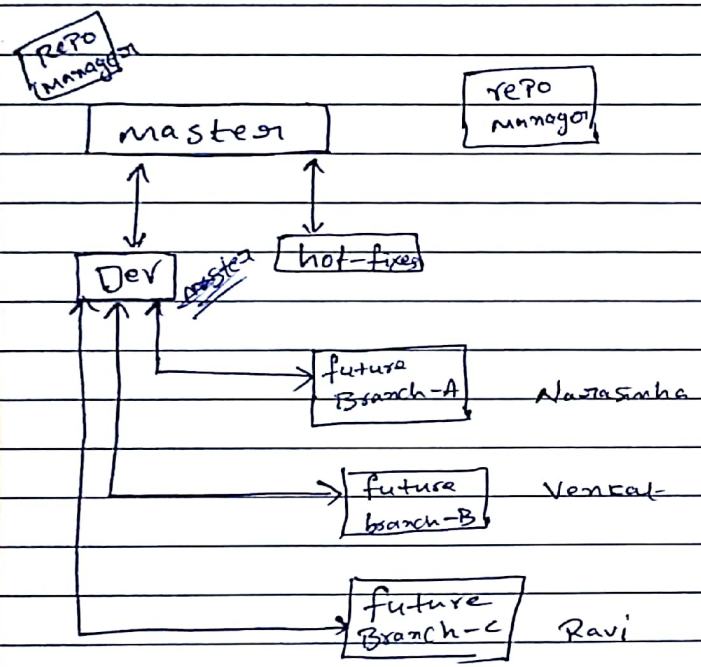


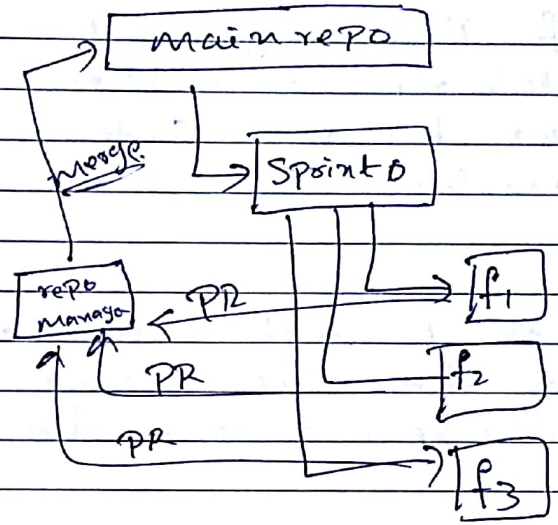
(or)



Scanned by CamScanner

Future Branching:-





① what is pull request?

Ans: It is usually asking that changes from a branch to your repositary be merge with a branch on another repositary.

② what are the objects in git?

Ans: git contain four(4) objects

① Blobs
② Trees
③ Commits
④ Branches

Scanned by CamScanner

① Blobs:-

Blob stands for Binary Large OBJect, Each version of a file is represented by blob. It holds the file data but doesn't contain any metadata about the file.

② Trees:-

Tree is an object, which represents a directory. It holds blobs as well as other sub-dirs. It stores references of blobs.

③ Commits:-

It holds the current state of repositary. It is also named by SHA hash.

④ Branches:-

Branches are used to create another line of development.

⑤ Tag:-

Scanned by CamScanner