Git

__/__/____

- Version control system → feature (Branching & Merging, work simultaneously, history)

→ It is a software that tracks changes to a file or set of files overtime so that you can recall specific version later.

→ Allow work together with the programmers

→ Type → (I) Central VCS ; (II) Distributed VCS

↙

These system have a single server that contained the versioned files & some clients to checkout files from central place, Administrator has control over the developer

Ex → CVS, subversion.

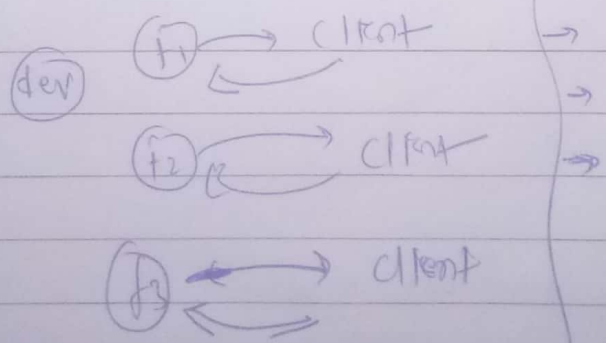
Drawback → It uses central server to store all dB, But due to single point failure, in the central server, developers do not prefer it.

(II) Distributed VCS → In this, user has a local copy of a repo. So, the clients dont just checkout the latest snapshot of the files even they can fully mirror the repo.

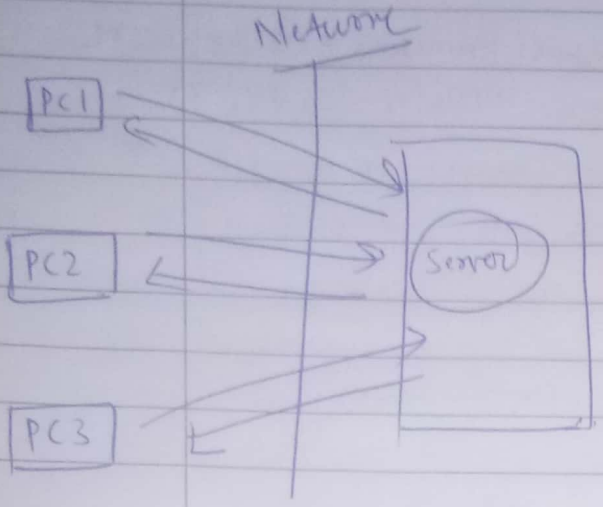
Ex → Git, Darcs.

Need of VCS

- → Maintain
- → version
- → Tracking
- → share
- →



Central



Disadvantage:-

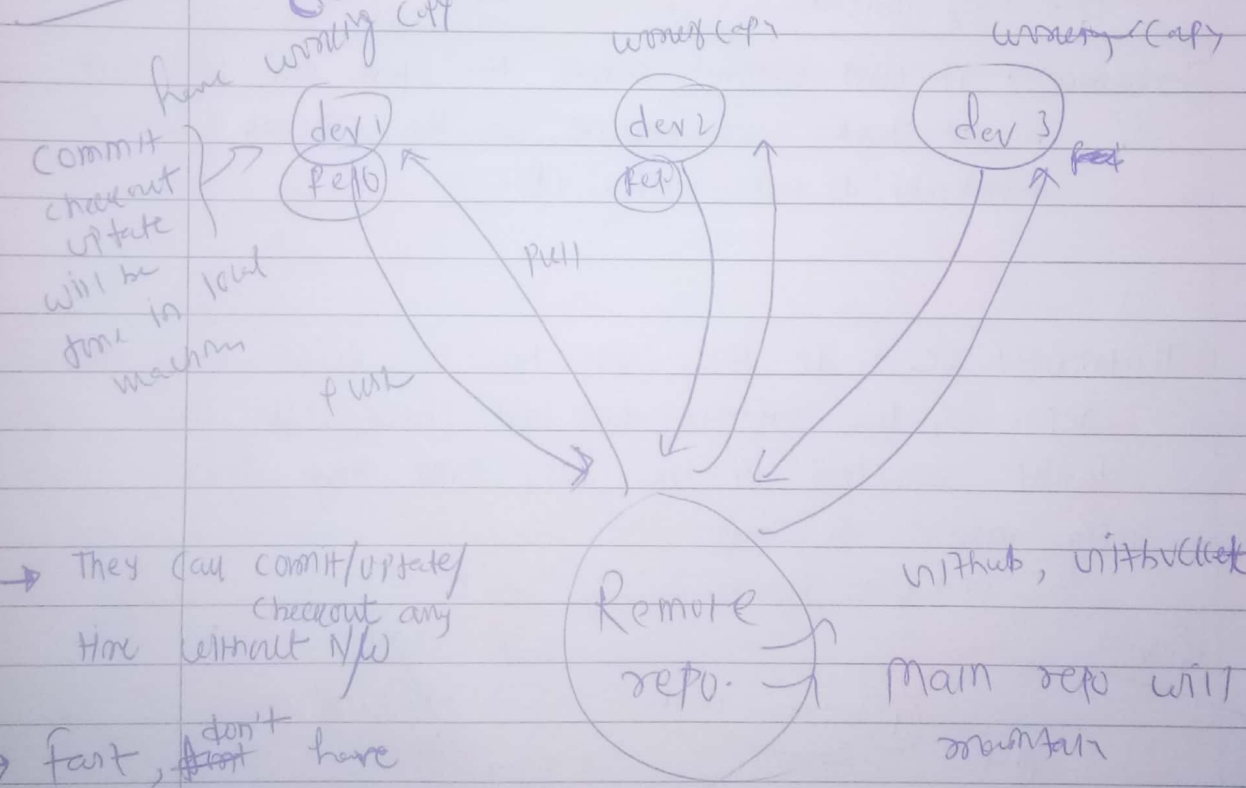
→ for push & commit we need to network to

→ If the server is down we cannt work

→ Load y the server

Ex → SVN

Distributed



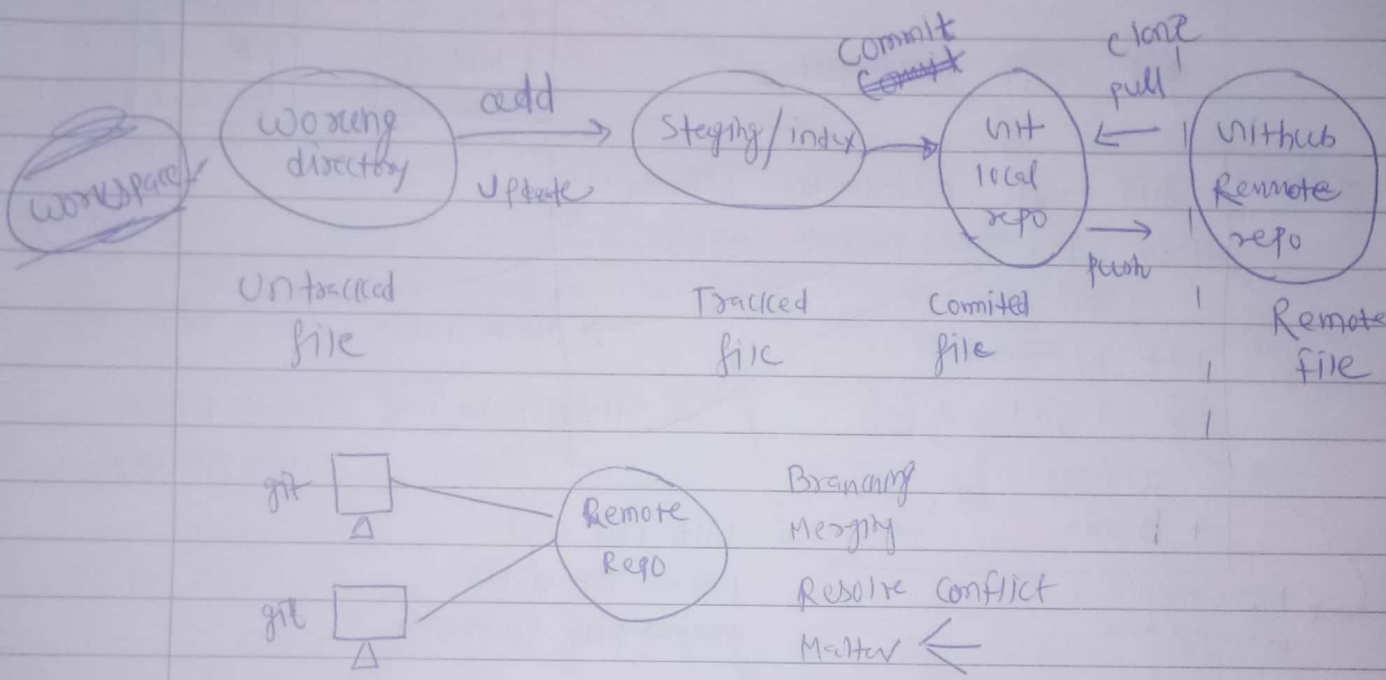
→ They can commit/update/checkout any time without N/w

→ fast, don't have to interact with server each time

→ push, pull will be done via N/w.

Unit → Distributed version control system

Architecture of Unit



Branching
Merging
Resolve conflict
Master

→ Initialise git local repo in machine

→ Goto folder
→ git init
→ git add -A // To add all files in staging Area
→ git commit -m "Message" // To add files in local git repo.
→ git status // To show all status of file
→ git log → // Commit history

workspace from

Git Commands

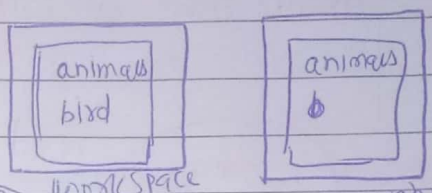
__/__/____

Git Structure Area

working directory → Staging → local repo → Remote repo

Task → 1 To see the difference in file content b/w working directory and staging area

git diff index.txt



Working Space (Destination)

Staging Area (source)

output →

```
diff --git a/index.txt b/index.txt
index fcb5845.. b5bf6bc 100644
--- a/index.txt
+++ b/index.txt
@@ -1 , +1 , 2 @@
 animals
+ birds
```

2 line has

Hash of file content destination

Hash of the file content from source/staging

1 line missing in staging Area

Unit file mod

100 → Type of file

644 → File permissions

permission → rw - r - r

owner, groups

4 → r
2 → w
1 - e

--- → a/index.txt → source file missing some lines (staging)

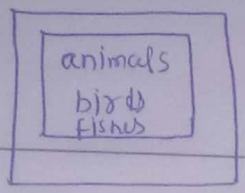
+++ b/index.txt → New line added in the workspace.

If anyline prefixed with space means it is unchanged

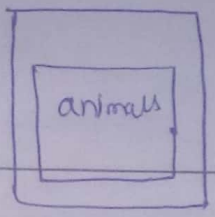
If anyline " " + means it is added in destination copy

If anyline " " - means it is removed from destination copy

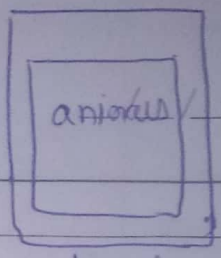
Note → last Commit is represented by HEAD in local repo



working directory



Staging



local repo. first commit

Task 2 →

To see the difference in file content b/w working directory & last commit (local repo). or latest

git diff HEAD index.txt

content of last commit of index.txt of local repo

content of workspace

output → animals → No change in this line in both
+birds → change in destination index.txt
+fishes

terms → diff b/w specific commit & staging Area → git diff --staged id

Task 3 → To see the difference in file content content b/w staging area and last commit of the specific file

destination source

git diff --staged HEAD index.txt
--chached

If we skip this it will compare all files.

Task 4 → Difference in file content b/w specific commit and working directory copy

first we need id of that specific commit → git log --oneline

git diff id index.txt

Commit id

Task→ To see the content diff ~~in file~~ between 2 specified commits

git diff id1 id2.

Task→ To see differences ~~b/w~~ in content b/w 2 branches

git diff branch1 branch2

~~Keno~~

Removing files

→ git ls-files → Files present in staging area

→ Remove files from both staging area & working directory
git rm filename , git rm -r .
↳ To remove all files

→ git rm --cached filename → To remove file from staging area only

→ git rm -f filename → To delete from stg and W.D.

→ rm filename → To remove file from working Area

→ Cat filename → To see content of file

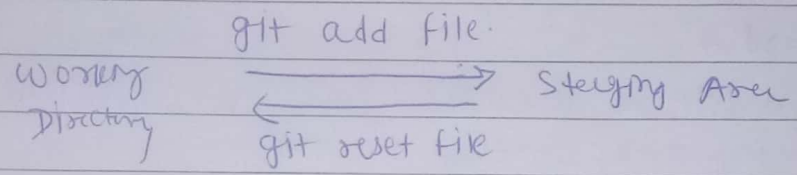
→ git checkout -- filename → To undo the content of working directory as Staging Area content of the file

Git reset command

1) To remove changes from staging area
2) To undo commits at repo level.
→ (git local repo)

→ git reset is just opposite of git add command.

1) → git reset filename → filename file will be removed from staging area.
It will not affect working directory file.

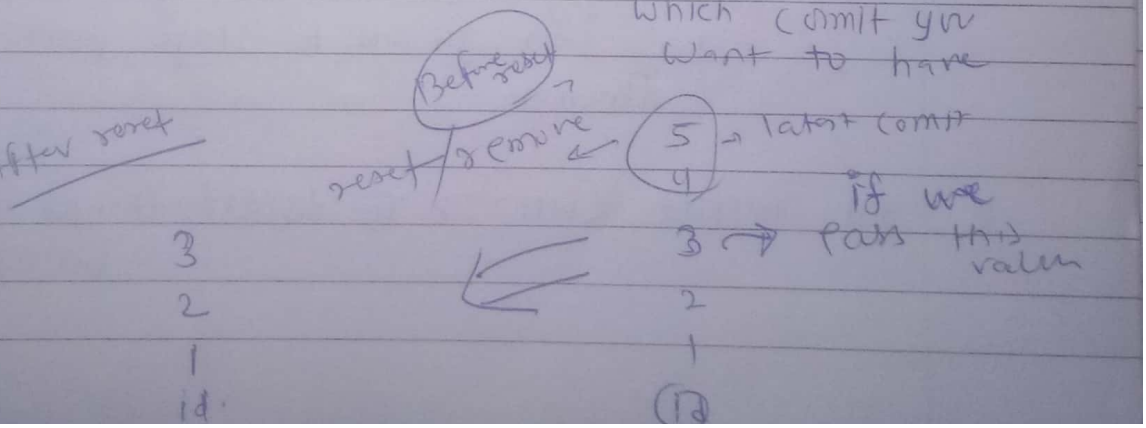


2) To undo commits → git reset <mode> <commit id>
-- mixed
-- soft
-- hard

→ mode will decide whether these changes are going to remove from staging area and working dir. or not.

a) reset with -- mixed mode → (Default mode)
→ To discard commits Discard/Undo from git local repo.
& also staging area. working dir. will not affected.
& file will be removed from staging Area.

Ex→ git reset --mixed id → push the id of which commit ~~chose~~ up to which commit you want to have



b) reset with --soft mode → git reset --soft id

→ It is exactly same as --mixed mode, but changes are available in working directory as well as in staging area

→ Remove commit from local repo only

** → It won't touch staging area & working directory

→ As changes already present in staging area, just we have to use commit to revert back

→ we can revert with git commit

c) --hard →

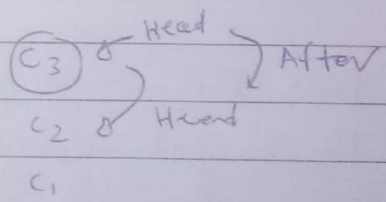
→ Changes will be discarded from everywhere (git local repo, staging area, working directory)

→ No way to revert.

→ Working tree wont be clean

git reset --hard commitId

git reset --hard HEAD~1 // Move head one step backward



git reset filename

* ~~git reset --staged .~~ → To remove ~~all~~ specific files from staging area to working directory. Again we can git add to stage from working directory

* git checkout <file> → To discard changes in working directory

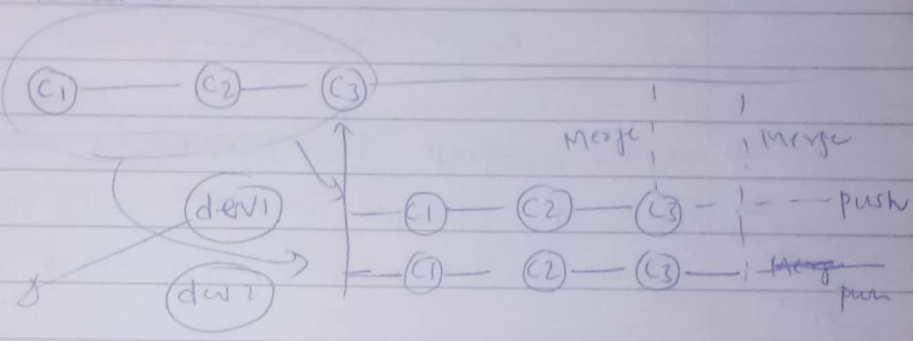
Git branching

__/__/__

Branching → / Without disturbing main/master branch we can get a copy of master branch and then, we can develop new feature. in parallel other dev will also work to develop another feature

→ Purpose → To work parallel along with teammate.
→ Code of master will be unaffected / clean.
→ All branch are isolated

Ex →



In this branch all commit file present until C3 are present but not in master

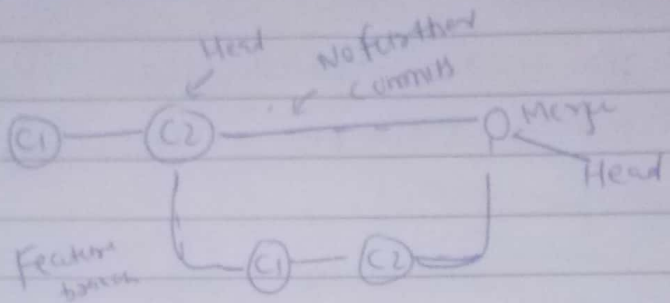
→ Whatever changes will have child branch does not impact master branch

Git branching commands

i) git branch →
ii) git branch branchname →
iii) ~~git branch~~ → git checkout -b branchname

Merging

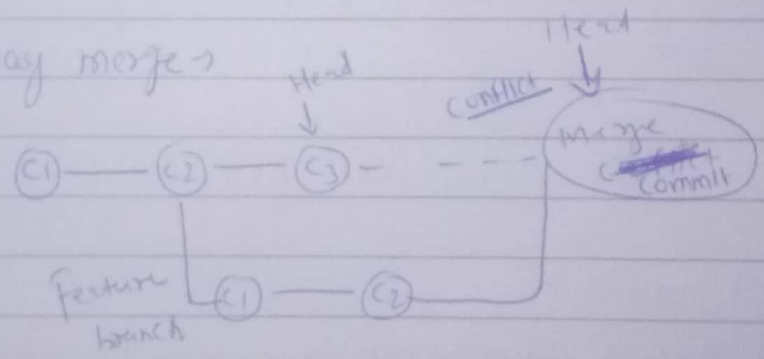
- Merging → When we do parallel development, when you complete the feature/ bug fixes that code will be merge in to masterbranch.
- → Creates graphical history,
- (I) Fast-forward merge → git checkout master; git merge feature



- → After merging there will be 4 commits total
- → Without interfering main branch → we do changes only in child branches we dont touch the master branch. So, there will be no conflict)

Note → After creating child branch ~~we~~ further we dont have to do further commit in master branch.

(II) Three way merge →



there will be chance of conflict

- → Changes happen in master and child branch parallelly - So
- → Merge commit happen when we do changes in master branch after creation of ~~the~~ child branch

- → Total 6 commit (3 master + 2 child + 1 merge commit)

Rebase squash

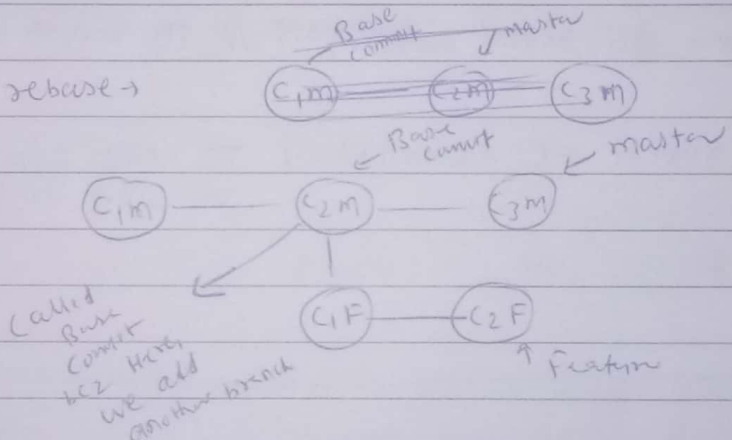
Unit rebase

__/__/____

- Unit rebase → It is the process of moving or combining a sequance of commits to a new base commit.
- → Rebasing replays changes from one line of work onto another in the order they were introduced.
- → Linear process of merging, each commit has only one parent.
- → Creates a linear trace of commits, linear history that can be easily understood.

Need → To avoid conflict, and want to follow ff-merge, To avoid multiple parent for specific commit.

Process → Before rebase →

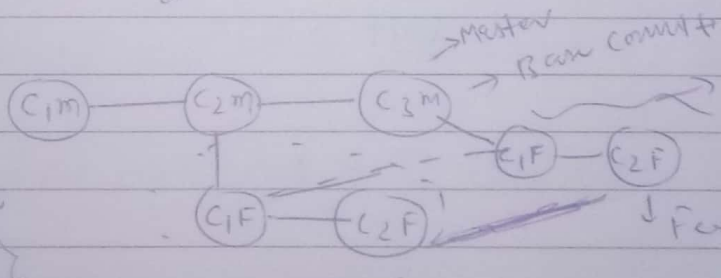


Rebase Step I →

Rebase feature branch on top of master branch

git rebase master

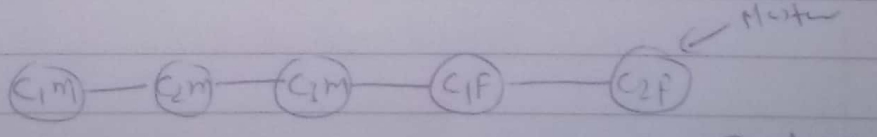
Note → Run in feature branch



Rebase step 2 →

Merge feature branch to master

git merge feature



Git revert

__/__/____

- Revert → is used to revert some changes. It is an Undo type command.
- → Useful to revert when our code is used by other dev in remote also.
- → It does not delete any data in this process instead it will create a new change with the opposite effect and thereby undo the specified commit.
- *. → It is undo operation that offers a safe method of undoing changes.

Use
- → Useful for tracking bugs
- → git revert records some new changes that are just opposite to previously made commits.

Syntax → git revert commitid // TO revert commit with specific Id.

default option ↑ git revert -e CommitId // TO edit commit msg before reverting the commit

gi

Git Squash

__/__/____

Squashing → combining multiple commits in to one

→ can help to keep your git history clean & easy to read.

→ Squashing retains all the changes you made but condenses them down in to single commit.

git rebase -i CommitId p → pick

git merge --squash branchName

Git Clean

__/__/____

* Clean → Used to remove/deleting untracked file in a repo working directory. As well as removing temporary build artifacts or merge conflict files.

git clean -f → To delete remove all untracked files from working directory

git clean -df → To remove/delete all directory

Git amend

→ It allows you to ~~change~~ modify your last commit. We can change your log message and the files that appear in the commit.

→ The old commit is replaced with a new commit which means that when you amend your old commit it will no longer be visible in the project history.

git commit --amend → To edit/modify last Commit.

Unit cherry pick

- git cherry-pick → is used if you want to apply particular commit from one branch to another branch
- → Cherry-pick is mainly used if you don't want to merge the whole branch and you want some of the commits.

used

- → Mainly used for the bug fixes where you want to place that bugfix commit in all the version branches.
- → Used when we accidentaly make a commit in wrong branch

Example → We have code version 1.0, 2.0, 3.0 (Now working)

We observe some bug in this version & also in all the previous version instead of fixing bug in all version we can fix in one version & then, we cherry-pick that fix in all version

git cherry-pick commitId° → This will append the commitId in the branch where you run this.