Date __/__/____

→ After git checkout -b f1 &
→ git checkout master
→ master create new file Vim 123 than git add .
→ git commit -m " " . →
Than we do rebase
*→ git rebase master

→ Squash → Squash is done by master. In this method get data from branches without commits and versions. we get final version/commit only
→ git merge --Squash Branch1

→ If we want to Push our feature branch without commit we have to go their and write command git Push origin "Branch name"

*→ Fetch/Pull → Pull = fetch + merge
→ git fetch
→ git merge origin/master
→ git Pull origin master
→ to get data from remote repository

→ Reset/Revert → In reset commit is also deleted But in revert commit is not delete it's remain in commit area

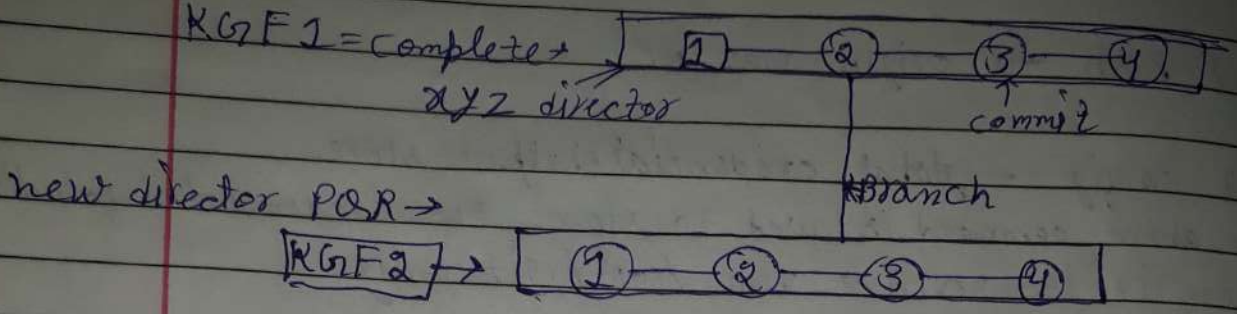
→ Fork → GitHub to Github data transfer

→ By Hooks we can automatically Push data
→ cd .git → ls
→ cd hooks

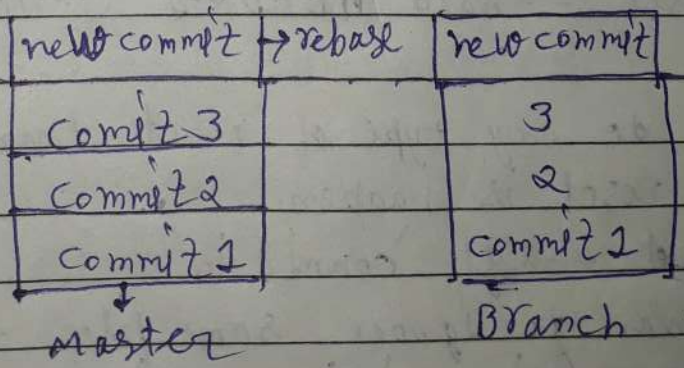
Page No.

→ * log etc

→ Branches →



→ git branch → to check branch location / git status
→ git checkout -b KGF2 → switch to new branch KGF2
→ Vim KGF2
→ git add. → git commit -m "KGF2 release"
→ git log
→ Vim KGF2 → git add. → git commit -m "kg f2 upd
→ git checkout master → to switch to branch master
→ git merge KGF2
→ Rebase → Rebase is done by branches



so after doing some changes in master branches get it by doing rebase

Date __/__/____

→ git remote -v → to check add name

→ git Push -u origin master

→ git config --global credential.helper store
the above command is used to store the credential information
where it is saved → vim /root/.gitconfig

→ Reset → In reset we simply goes to Previous version

① Soft Reset → In soft reset file is available in staging area & working area. head is only reset in commit area

② mixed reset → In mixed reset file is available in working area but commit & staging area data is not there.

③ Hard reset → In Hard reset data is not available in working, staging and commit area

① → git reset --soft HEAD~1 (head goes 1 line below)
② git reset --mixed HEAD~1 remove from staging area
git add. → commit
③ git reset --Hard HEAD~1 (working area clean/all clear)

→ if not do any type of reset than by default mixed reset is happen
git reset --soft 'commit id'
→ If we wan to ignore some files to commit than we use → (vim .gitignore) than enter the files name that we want to ignore → (*.log for all log files)
→ ~~add~~ → git add.

→ mkdir instagram
→ cd instagram → git init
→ vim sourabh
→ touch sourabh123.txt
→ working area

→ git add .
git status
→ staging area

→ git commit -m "my 1st commit"
→ git log
→ commit area

for Particular project

→ git config user.name "Sourabh Tailor"
→ git config user.email "Sorabhtailor@gmail.com".

→ for all Project.

→ git config --global user.name "Sourabh Tailor"
→ git config --global user.email "Sorabhtailor@gmail.c

→ git log → By the we can check the commit version or commit id and about author details.

→ git clone → by using git clone command we can download any Public repository and we get all the version of commits and history.

→ git clone https://github.com/sour.git myproject
git id
want to give name to repository

→ Push data to git hub →
→ git remote add origin https://github.com/
giving name of this link to origin

Date / /

→ Git Benifits → ① work on snapshots
② work on check sum value
③ Every operation is local ④ GIT has integrity
⑤ GIT Generally only adds Data

Git like tools Aws code commit, mercurial, Helix code

→ WORK SPACE / WPI (work in progress area) → where we started work.
where we make files or directory. In this file are untracked
we can check status of tracking and where we are in branch by 'git status' command.

→ Staging area → In this area file will be tracked. after git add. command we enter in staging area. In this area file will be tracked

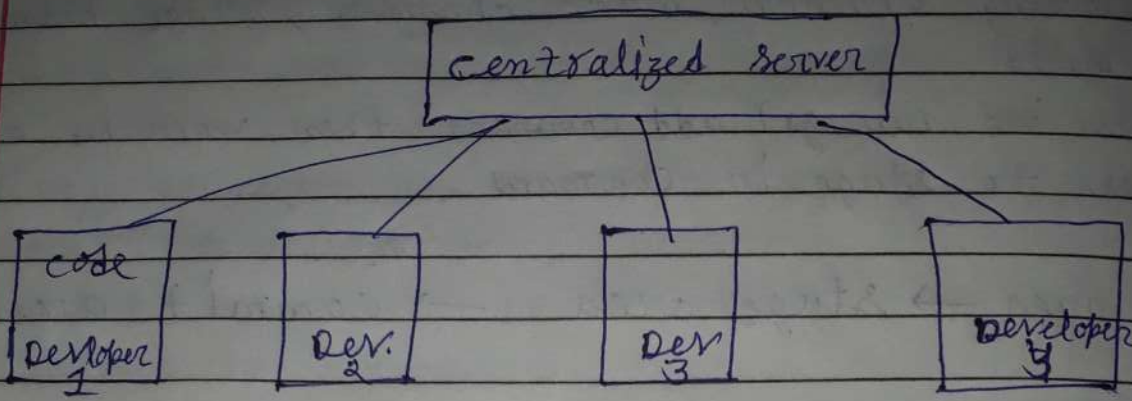
→ 'git init' → By this command we intiate empty repository. working area is initialized by git init command.

→ Commit area → After the staging area we enter in commit area. it is final area of work after 'git commit -m "commit"' we enter in commit area
→ 'git commit -m "my first commit"

→ git status → by this we get in which staging area we area and we can check tracking of file

Page No.

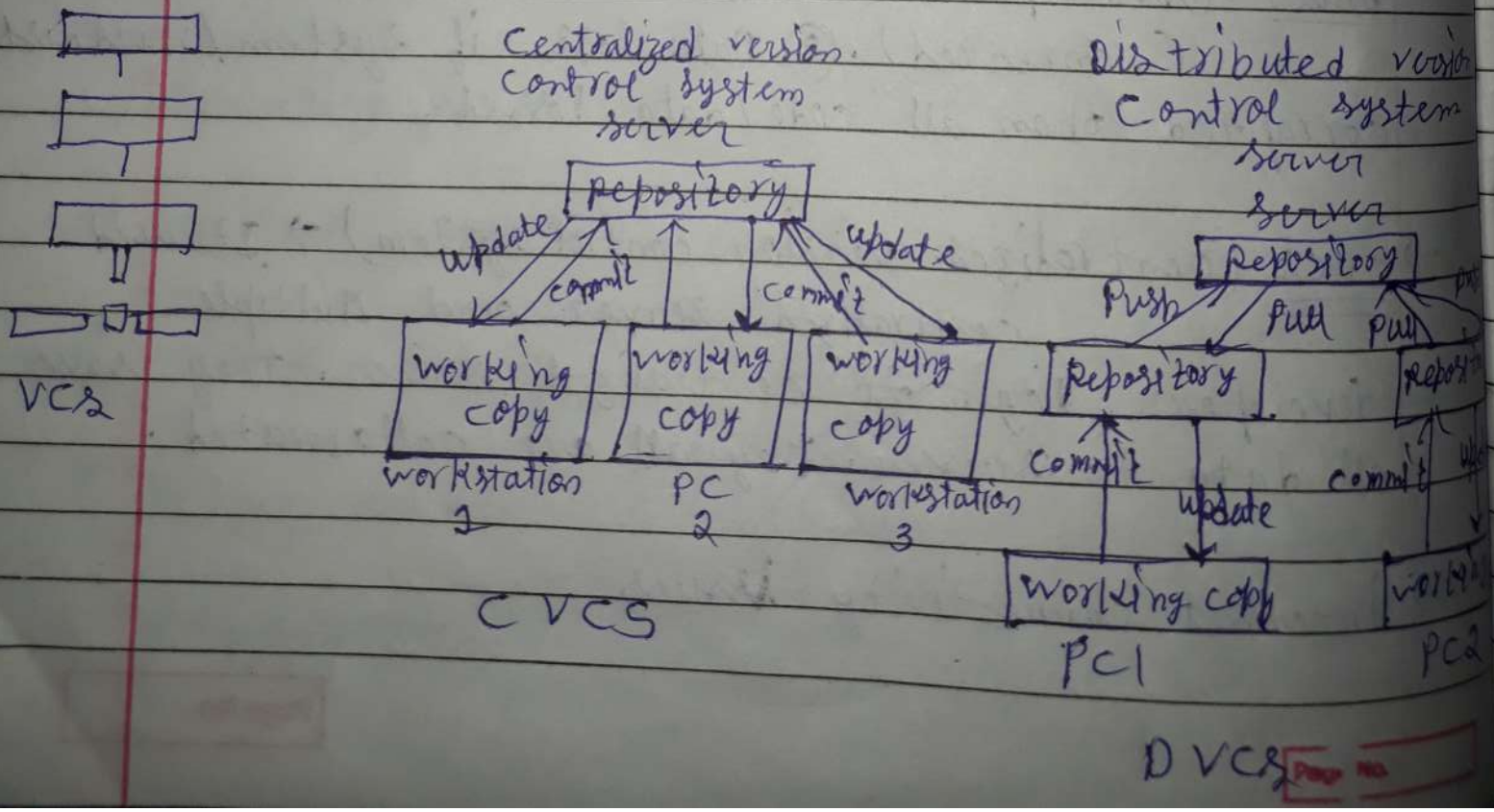
③ DVCS (Distributed version controle system) → GIT is an example of DVCS:



→ In DVCS bothe developer and server have the copy of data

→ Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

→ Eg:→ Software code use by developers, Images by graphic designer.



Date / /

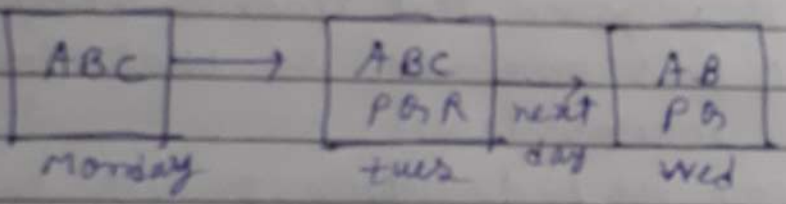
(3) Git directory (Repository) → Perform a commit that stores the snapshots permanently to your Git directory. checkout any existing version, make changes, stage them and commit.

→ When we run git add command then work in command goes to stage in command

working area, → stage area → commit area

→ storage problem →

Local version control system → (VCS) To save only changes in source code.

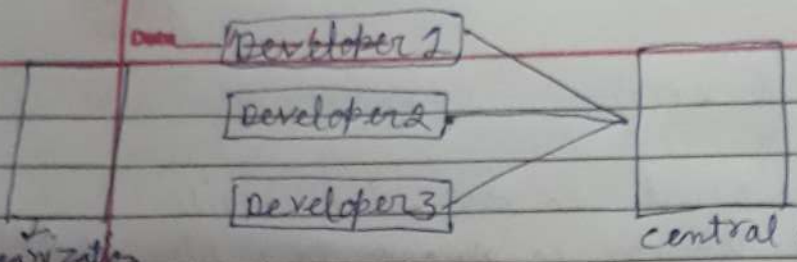


① → VCS solve the problem of storage issues → ① Developers can not collaborate with each other (connected). ② Data loss - if system / Hard disk corrupted then all code / data lossed.

② → CVCS (Centralized version control system) → It used centralized server and Multiple developers login to centralized server. They save all data to server. They all are collaborated.

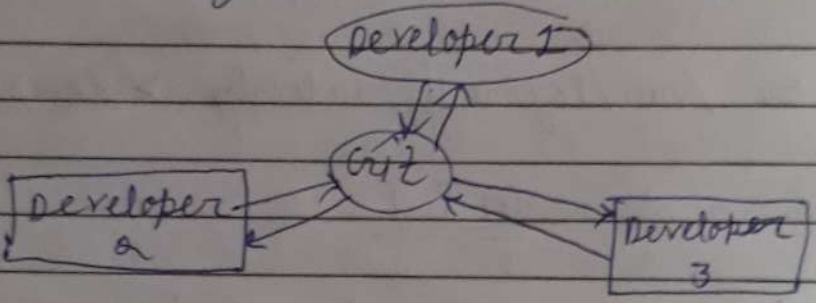
→ internet connectivity issues

Page No.



Now let's look at the scenario after Git →

→ Every developer has an entire copy of the code on their local system
→ Any changes made to the source code can be tracked by others
→ There is regular communication blw the developers



→ Features of Git → (1) Tracks history
(2) Free and open source
(3) Supports non-linear development (4) Creates backups
(5) Scalable (6) Supports collaboration
(7) Branching is easier (8) Distributed development

→ The Git workflow is divided into three states

① Working directory/area → Modify files in your working directory
② Staging area (Index) → Stage the files and add snapshots of them to your staging area

→ Standard I/P, o/P & error →

→ >file = redirect stout to overwrite a file
→ >>file = redirect stout to append to a file
→ 2>file = redirect stderr to overwrite a file
→ 2>/dev/null = discard stderr message by redirecting to /dev/null
→ >file 2>&1 = redirect stout and sterr to overwrite
&>file the same file
→ >>file 2>&1 = redirect stout and stderr to append
&>>file to the same file

→ tail -n 50 /var/log/audit/audit.log > /tmp/last50file

→ GIT

→ GIT → Grit is a Devops tool used for source code Management. It is a free and open-source version control system used to handle small to very large projects efficiently. Grit is used to tracking changes in the source, enable multiple developers to work together on non-linear development.
Linus Torvalds created Grit in 2005 for the development of Linux Kernel.

→ Before Grit tool developers used to submit their codes to the central server without having copies of their own. Any changes made to the source code were unknown to the other developers.
→ There was no communication b/w any of the developers

Page No.