• ~~DevOps~~ DevOps:

↳ It is a process of continuous development, continuous build, continuous test, continuous release of the software.

Version Control System Tool

• Version control system is also known as sf Software Configuration management (SCM) / Source Code Management.

• Need of Version Control System?

(i.) Maintaing multiple versions manually is a very complex activity.

(ii.) Every change should be tracked,
- who did the change
- when he did the change
- which changes he did etc

(iii.) Overwriting of code shouldn't happen

(iv.) Developers have to share their code with peer developers. so that multiple developers will work in collaborative way

(v.) Parallel development must be required

Tester --> test script (need VCS) architect → documents Project Manager → Excel sheets

• How version control systems work?

Version control system always talks about files which contain source code. (it is applicable for any document.)

• Working directory/ Work Space :> Where developers are required to create/ modify files.
> Here, version control isn't applicable. Here we won't use the words like version 1, version 2 etc.

• Repository :> Where we have to store files and metadata.
> Here, version control is applicable, we can talk about versions

Good Write

Scanned by TapScanner

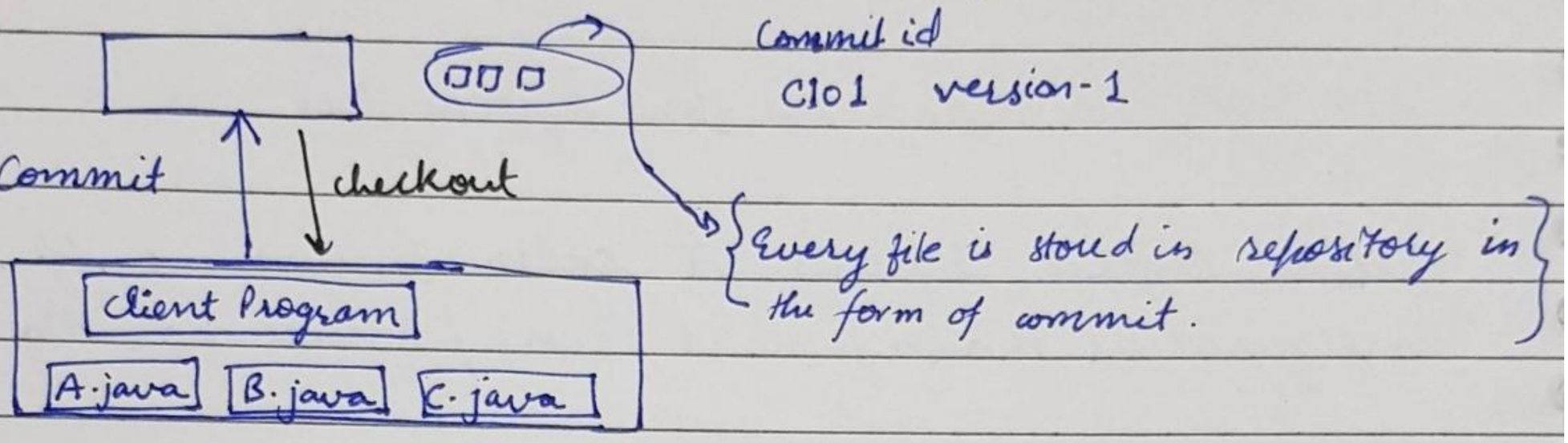
(commit is a thing which is going to hold & commit is also a process)

→ Commit means a grp of files which are stored at a particular pt-of time. It is 1 version.
· State at a particular time. (Commit Object).

DATE: __/__/____
PAGE: _____

· Commit: The process of sending files from working directory to the repository.

Every commit is associated with a unique id.



commit is the process name & group of files (both) noun & verb.

Before doing the demo the developers saved these files in the repository.

· Checkout: The process of sending files from ~~workspace~~ repository to working directory.
  · Now all these old files are going to override with this particular V file.
    version

· Benefits of Version Control System:-
  · We can maintain diff. versions & we can choose any version based on client req.
  · With every version / commit we can maintain metadata like, commit message, who did changes, when he did the change, what changes he did.
  · Developers can share the code with the peer developers in a very easy way.
  · Multiple developers can work in a collaborative way.
  · Parallel development.
  · We can provide access control like
    who can read code
    who can modify code

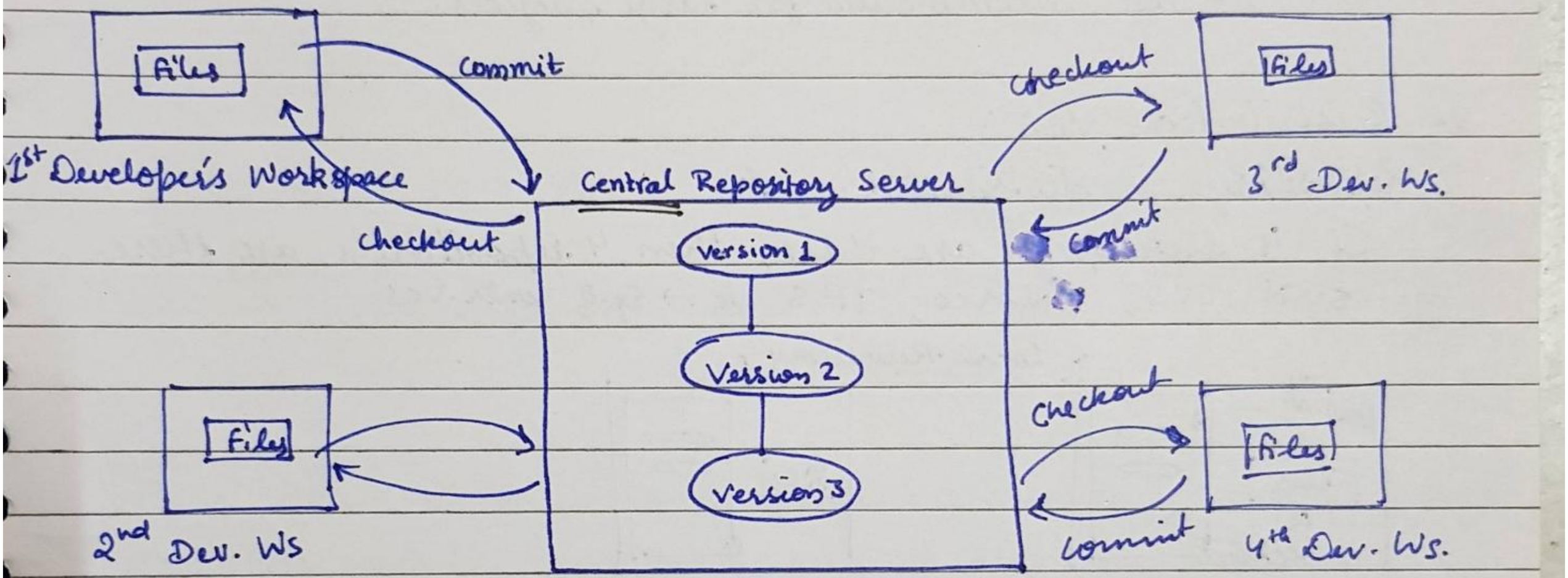
Good Write

Scanned by TapScanner

* Types of Version Control Systems :-
Two types :- (i) Centralized VCS (ii) Decentralized / Distributed VCS

(i) Centralized VCS :

- Only one repository, every developer is required to connect with that to continue his work.



- The total project code will be stored in central repository.
- This VCS is very easy to setup & easy to use.
- CVS, SVN, Perforce, Clearcase, TFS etc... all these tools are based on centralized VCS model only.

> Limitations :-

i. The total code stored at a single place (central repo).
Single pt. of failure.
If something goes wrong ... recovery is very difficult.

ii. All checkout & commit operations will be performed by connecting with remote central repository server i.e. all developers should be connected with central repository always. If n/w outage, version control

Scanned by TapScanner

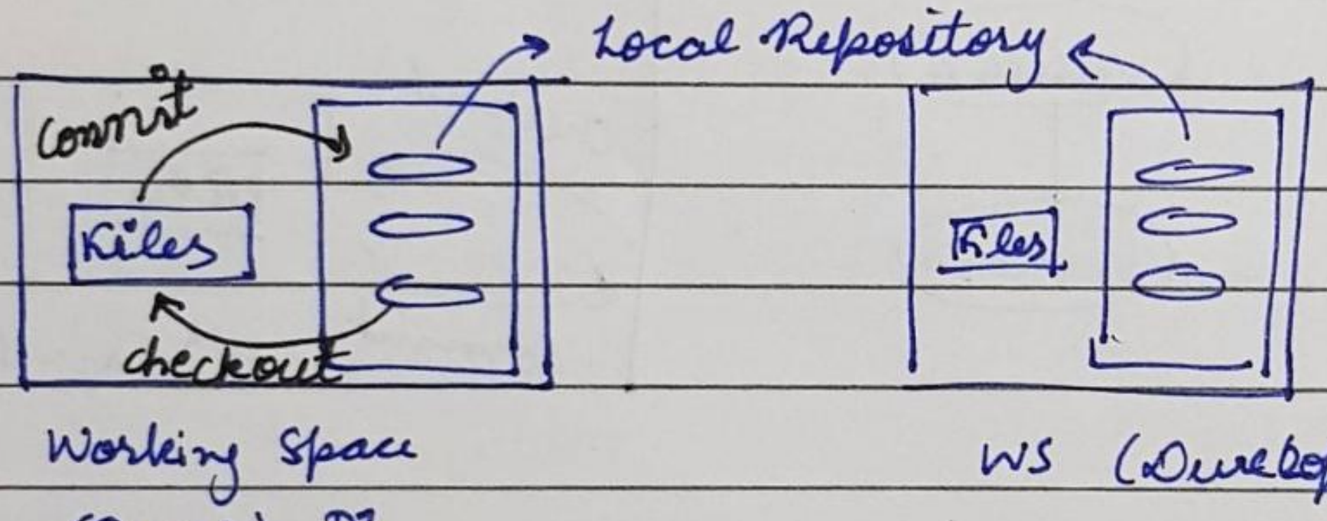
wont be available for the developers.

iii. All checkout & commit ---. server. The oper^ns will be performed over the n/w and these are not local operations. (definitely going to take some time.)
Performance is low. (Speed is low)

iv. If the no. of developers or files increases then organizations of central repo is a bit difficult.
↳ Not recommended for large projects.

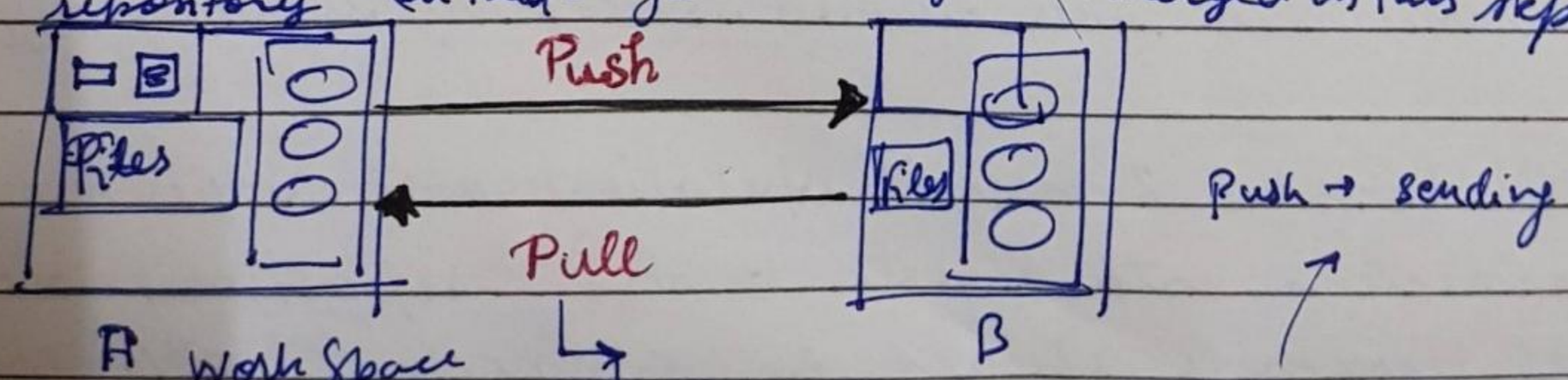
2. Distributed VCS :-

· Repository is distributed. If
If 4 developers are there, then 4 repositories are there.
Eg:- SVN, CVS, Perforce, TFS etc → Eg. of Centr. VCS



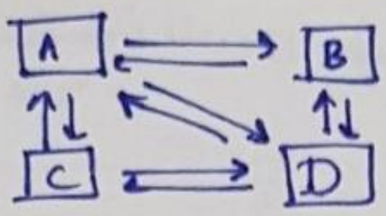
Advantages:

i. All commit & checkout operations will be performed locally and hence performance is more.
ii. Even in n/w outage, still version control is available.
Developer's machine need nood not be connected always.
(iii. There is no question of single pt. of failure, (Recycle Bin)
Every repository is available on every developer's m/c.
whatever his changes he did in repository
All these changes will be merged be merged in this repository.



↳ The process of getting files from others repository to our repository is called pull oper^n!

Scanned by TapScanner



DATE : __/__/____
PAGE : _____

- **Commit & Checkout :-**

These oper" will be performed locally b/w working directory and repository.
To performa these oper" n/w is not required.

- **Push & Pull :-**

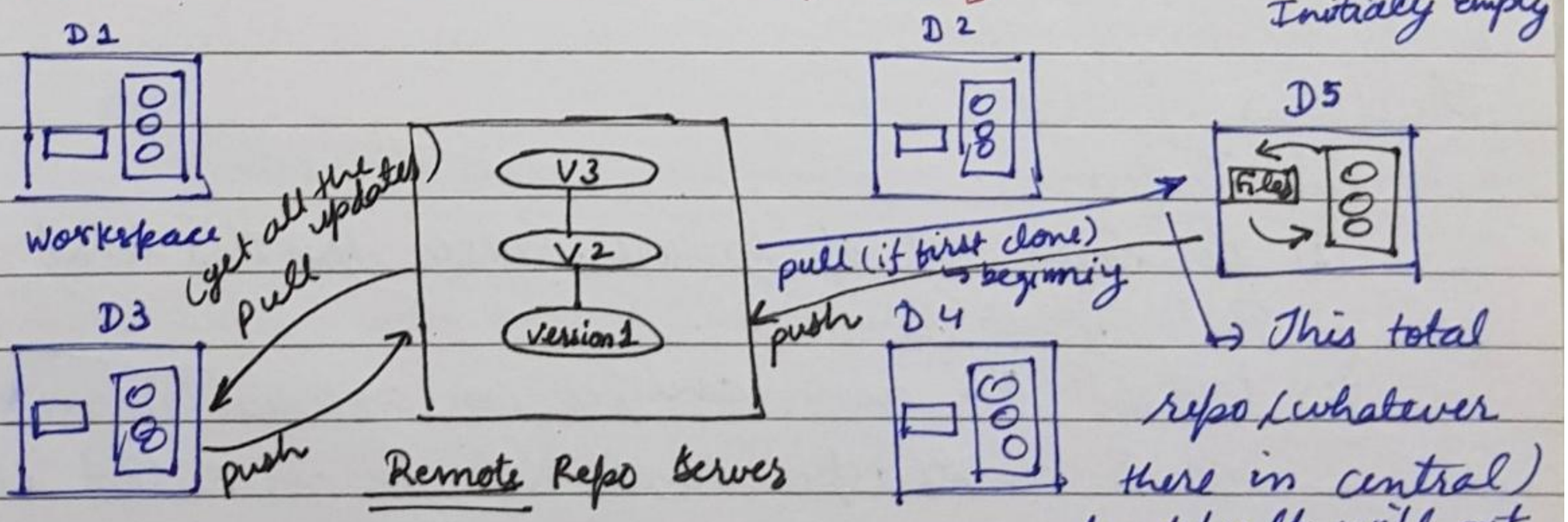
These oper" will be performed b/w 2 repositories. These are remote oper"s.
Compulsory n/w must be there.

Push & pull → Rare oper" (no. of commits & checkouts) >>>> (No. of push & pull oper")

- GIT → belongs to Distributed VCS model
  Tools ↳ Eg: GIT, Mercurial, Fossil

- Once our total work is completed then only we are going to share to (not everytime) our developers.

* Distributed VCS with Remote Repository :-



- **Remote Repo Vs Centralized Repo :-**

1. Commit & checkout are performed on local repo but not on remote repo. (you don't commit any changes here)
2. Every dev. has his own local repo. Main job of remote repo is to share changes to other developers.
   Good Write (to share developers code to peer dev.)

Scanned by TapScanner

diff. b/w rep remote & central repo.

DATE: ___/___/____
PAGE: _____

• Finally delivery is going to be happen from remote repo.
→ Just to match local repo with remote repo push & pull oper" is performed.
To match this remote repo to our local repo pull oper" we are going to use.

Eg:

Examples of Remote Repository Server:
↳ GitHub ↳ Gitlab ↳ BitBucket ↳ Cloud Platform

⭐ GIT:

• GIT is Distributed VCS tool.
• Linux → not an OS.
  • It is just a kernel. (kernel which is a part of our OS)
  • GNU → OS; the kernel is Linux ∴ GNU/Linux
    ↳ Project which was initiated to develop OS name.
    develop UNIX-based os.
    Everything they developed except the kernel.

• GIT → also developed by Linus Torvald.

Why GIT is very popular

Features of GIT :-

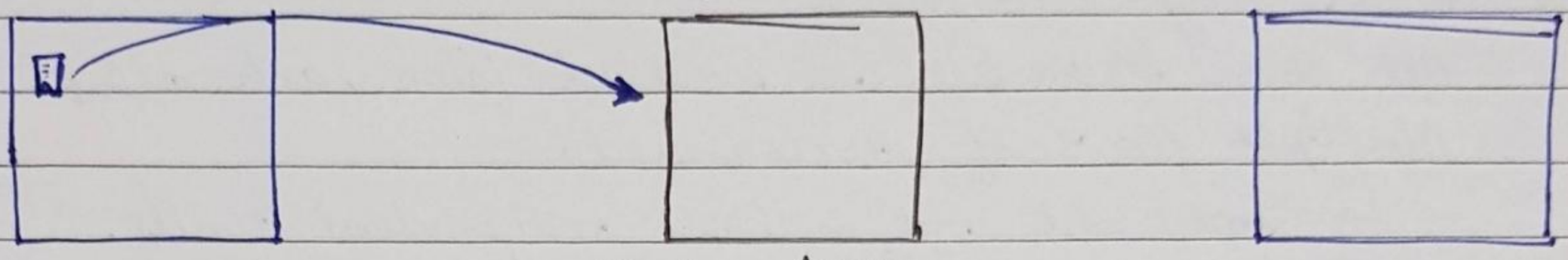
1. Distributed.
   (i) No single pt. of failure. Every dev. has local repo.
   (ii) Performance is more.
   (iii) Without n/w also, developer can continue his work.
   WS & remote repo need not be connected always.
   → logical layer b/w WD & local Repo.
2. Staging Area (or Index Area)
   (i) Biggest strength to our GIT VCS.
   (ii) This type of arrangement is not there in other VCS.
   (iii) Job of this area? (on next page)

3. Good Write Branching and Merging (next to next page)
4. Freeware and Open Source. 5. Provides support for multiple platforms.

Scanned by TapScanner

. Only files will be there.
. Middle place to hold our files before commit

. Versions will be maintained only in the local repository



Working Directory
↳ Physical

Staging Area
↳ Virtual

Local Repo
↳ Physical

* In GIT, commit is a 2-step process:
  i. First we have to add files to the staging area and then we have to commit from that staging area.

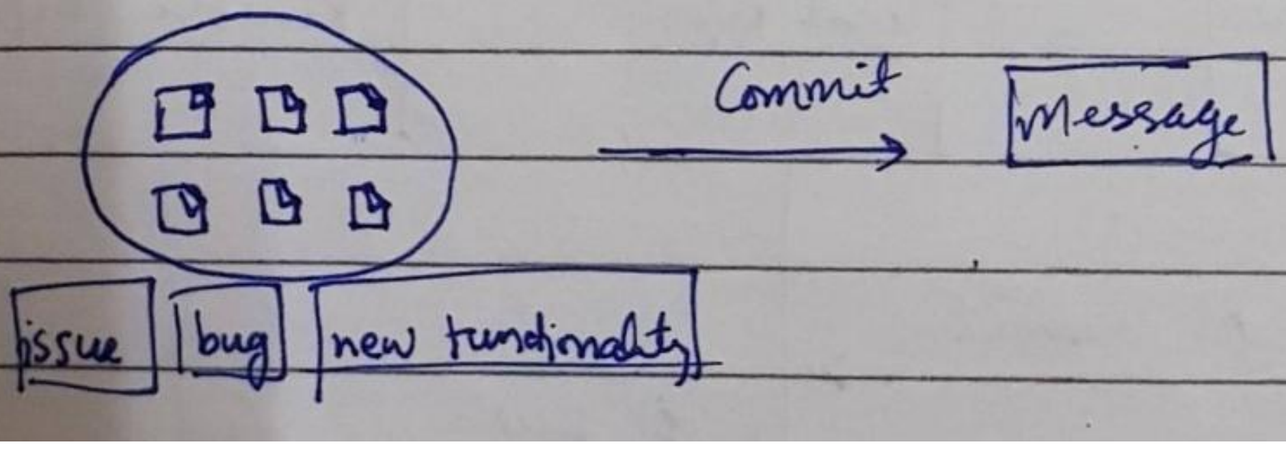
Advantage of Staging Area:-
↳ We can double check our changes before commit. If everything is fine then we can commit.

↳ · 12GB files we have to store in SVN.
  · GIT requires only 420 MB.
  · hashing
  · Snapshots of our data. Only SS will be stored inside GIT.
    ↳ internally server mechanism, hashing mech., snapshot mech. is there.

↳ So that we can add related files together & commit them.

↳ ~~Before commit~~,

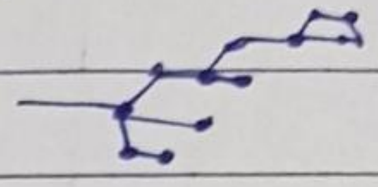
↳ In local repo, file size will be less compared to our workspace.



Scanned by TapScanner

* Branching and Merge :-

- All these branches are isolated from each other.
- Multiple flows of development.
  ↳ Same work but multiple independent units.



- We can create & work on multiple branches simultaneously and all these branches are isolated from each other.
  It enables multiple work flows.
  We can merge multiple branches on a single branch

⊛ GIT Architecture :-

~~Staging~~ GIT has 2 types of repositories :-
  i. local Repository
  ii. Remote "

- Usually the total project code will be available in remote repository.
- The current work of developer will be stored in local repo
- New files will be created in working directory.
- Once work completed, we have to add these files to the staging area. For this we have to use
- "git add" command.
  ↳ Send files WD → Staging Area

- "git commit" → Staged changes will be moved to local repo.
- "git push" → Move files from local repo to remote repo.

Scanned by TapScanner

- git clone / git pull : To bring files from ~~local~~ remote repo to local repo.

↳ git clone : creating exactly duplicate copy.

If you want complete copy of repo in your local repo.

↳ To create a new local repo from the remote repo, (mostly used when there's new dev. in team) which is the exact copy of remote repo.

↳ git pull : Only updates.

↳ To get updated files from remote repo to local repo.

⊛ Life Cycle of File in GIT :-

- Every file in the GIT, is in one of the following 4 states :-

1. Untracked 4. Modified
2. Staged 3. In Repository / Committed

1. Untracked :-

Every new file will be created in working directory. GIT isn't aware of these new files. Such files are said to be in "untracked" state.

↳ Only new files not the modified ones.

(wd. staging area, local repo)

* git status : To know status of files in all areas.

2. Staged :

The files which are added to staging area are said to be in staged state.

git add a.txt

" " . → to add all the files

" " a.txt b.txt c.txt

" " *.txt

Good Write

Scanned by TapScanner

3. Any file which is committed is said to be in repo state or committed state.

We can commit staged changes by using git commit command.

↳ git config --global user.email "email add"

↳ git config --global user.name "Name"

*** ↳ commit. message → mandatory

git commit -m "Write your message here".

4. Modified :

· If the file added to staging area. or commit, now it is tracked by git.

↳ If you create a new file → untracked

↳ staging area → tracked by git

↳ if restored back from staging area → untracked

↳ Once committed only then it will show modified & not untracked.

* → ls : which files are there in current working directory

~~git ls~~ git ls-files : which files are tracked by git & it is git command.

→ linux command

* · Any file which is already tracked by git, but is modified in wd. is said to be in modified state.

WD		Staging Area		Local Repo

1. untracked → 2. staged → 3 Committed / In Repo

4. modified

Repititions

working directory 2

Untracked → Staged → Committed / In Repo

working directory 1

Modified

Good Write

Scanned by TapScanner

- git init :- Converts the current ws/wd into git repo (or local repo)
  - To provide/create an empty repo to our workspace so that version control is available to our working directory.
  - Immediately an empty repo will be provided. & it's name is (".git") → nothing but local Repository.
  - → Create an empty git repo ".git" or reinitialize an existing one.

everything same but ↓

* Configuration
  - git config --global user.email "..."
  - " " " " user.name "..."

(*) Combine adding to SA & then committing :-

git add a.txt b.txt
git commit -m "commit message"

} → git commit -a -m "commess"
↳ Not applicable for newly created files but only for the files that are being tracked. (modified files)

git status -s → Concised o/p.

* Staging Area → Index area or Cache area.
- Commit id is also known as hash which is a hexadecimal no. of 40 characters.
- Commit id is unique which can be used to identify commit The first 7 characters are also unique.
- SHA-1 hash f" is used in GIT.

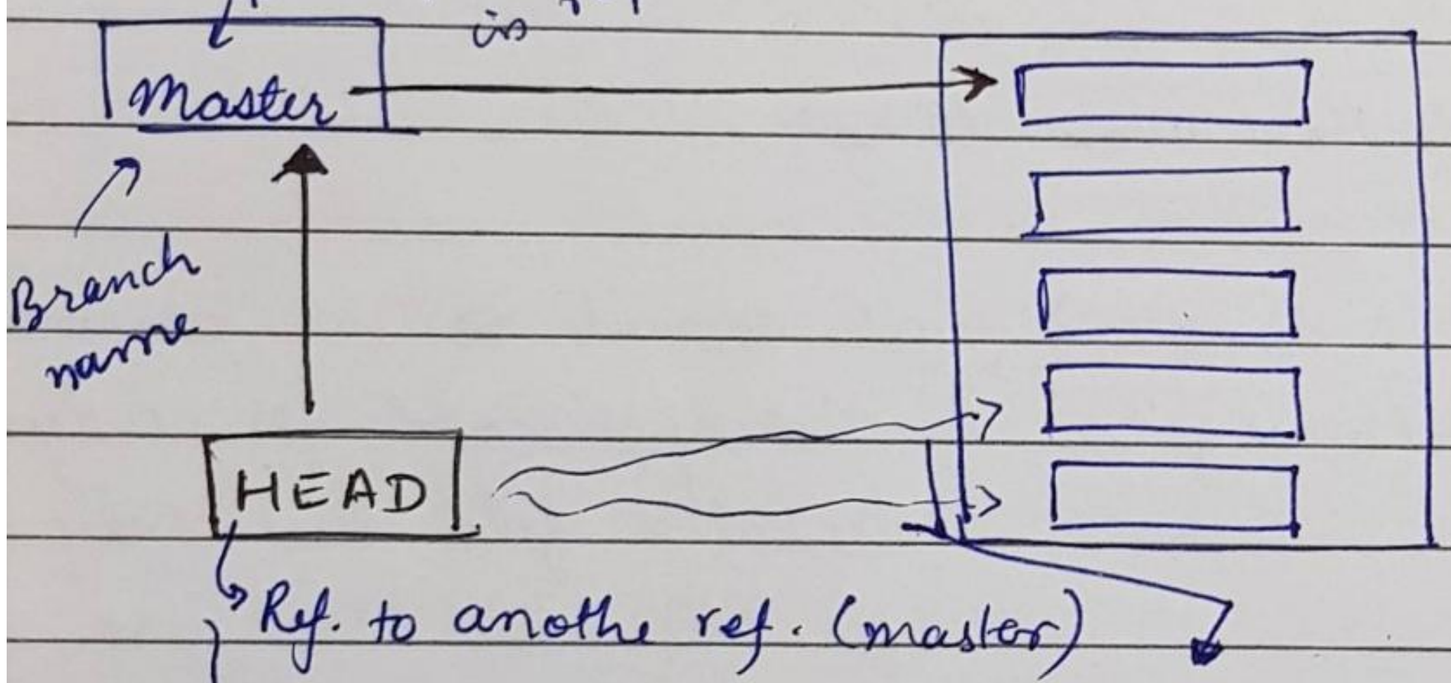
Scanned by TapScanner

- create mode 100644 file2.txt
  - 100 → The type of file (first 3 bits)
    - 100 → means ascii text data.
  - 644 → file permissions

- unique id
- Our data is available in hash form.
- hash; Adv- of uniq-id & hash :-
  1) i) Security
  2) ii) Less Space

⊙ HEAD → The most recent commit.

internally it is a reference pointing to most recent commit in repo



- Ref. to another ref. (master)
- Change HEAD reference to somewhere also → Known as "Ditached Head".

* git config :
  - ↳ used for git configs. like user name, mail id, etc.

- git config -l
  - ↳ To list out all git configurations.

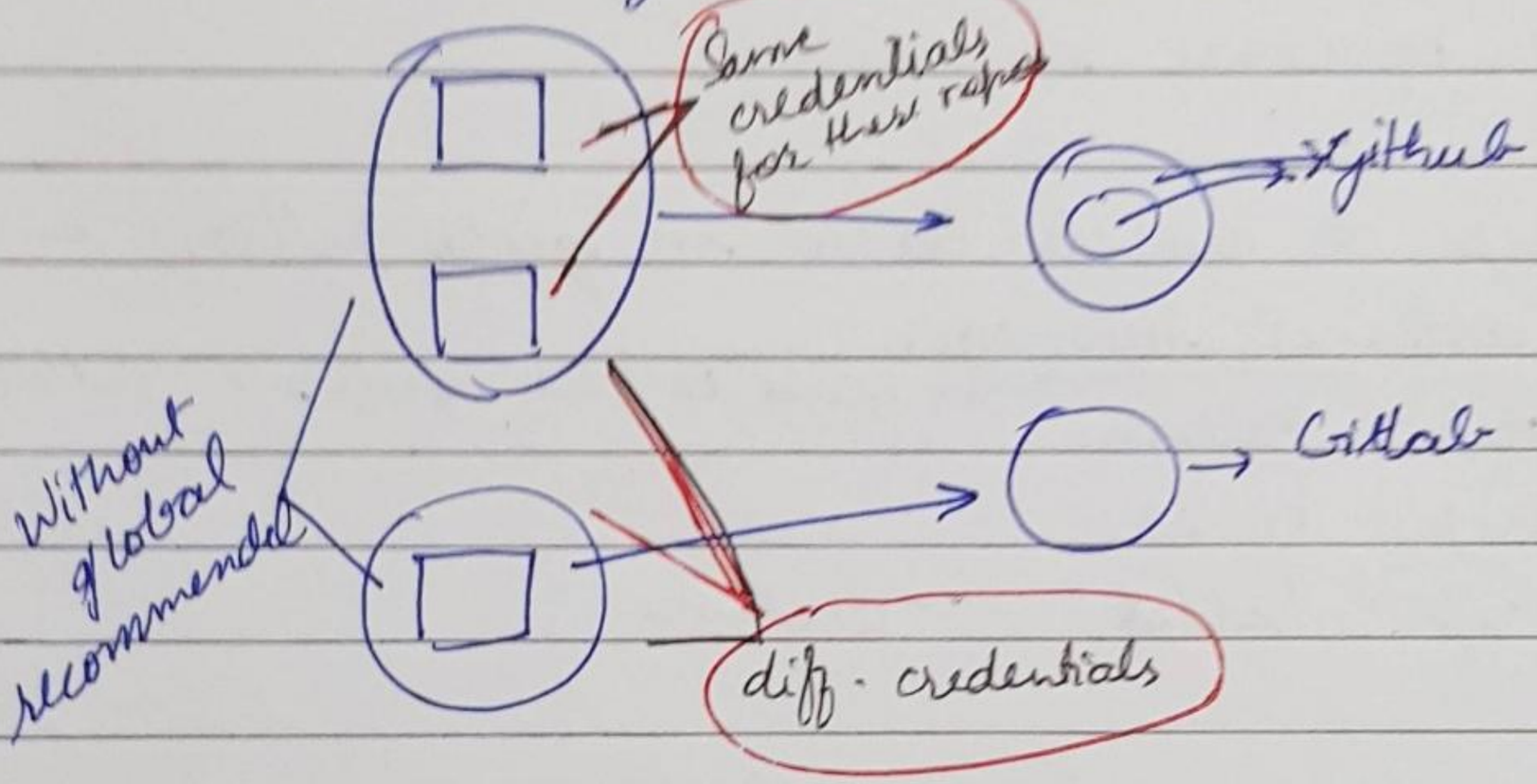
- git config user.naml
  - ↳ Change user name & mail.

git config user.name "Sans"
" " user.email "_@gmail.com" } But you used these commands with global options

Scanned by TapScanner

- Config with & without --global :-
  - For all repositories (managed by current git) these config. are applicable → global
- Config applicable only for current repo → without global
  " " for all repo → with global

If you are working on multiple repositories, recommended to use without global in real time.



- git log :-
  ↳ provides detailed log info
  - commit id, author (Name, email), date.

@ How to see log info of a particular file?
↳ git log file1.txt

GIT log :-

1. git log --oneline :- To get brief info; one line per commit.
   ↳ commit id + commid message (7 chars) → first 7 chars. of commit id
   ↳ Helpful if we have lots of commits & to identify commit based on message.

Good Write

Scanned by TapScanner

2. Option-2 : -n option to limit the no. of commits to display.

git log -n 2 (latest 2 commits)

or

git log -2 (or) git log --max-count = 2

↳ Q. Can we use tail commands of linux?

↳ No, ∵ head/tail command blindly consider lines whereas '-n' considers commits.

3. Option-3 : --grep to search based on given pattern in the commit message.

git log --grep = Pattern → no spaces in b/w "grep" = & "Pattern"

Eg:- git log --grep = sr-231

git log --grep = "added" --oneline

4. Option-4 : Show commits more recent than a specific date or time.

--since = "2020-04-25"

--after = " " "

"10 days ago"

"3 hours ago"

5. Option-5 : Show commits older than a specific time.

--until = "17-05-2020" 17 or before

--before = "16-05-2020"

"5 min ago"

"10 days ago"

6. Option-6 : Show commits based on author.

--author = Bhairavi

Scanned by TapScanner

7. Option 7 : To display extra info. branch name, HEAD, tags info etc
--decorate (You may think that the result is same as git log;
but with 'decorate' opt" you'll have branch name,
HEAD & tag info displayed too).

much more options left.

Scanned by TapScanner