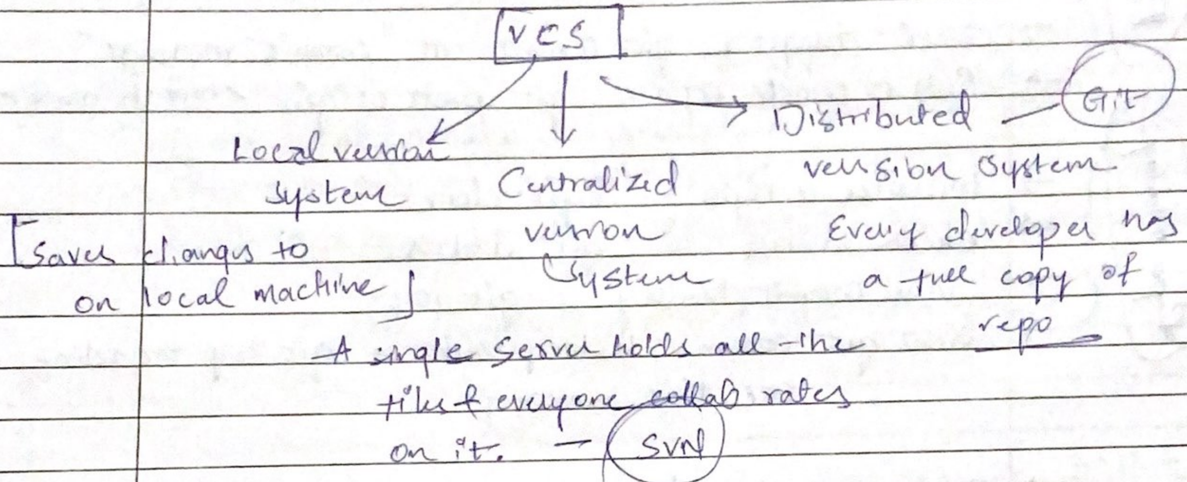
Git + Github

① Version Control System:-

→ It is a system that records changes to a file or set of files over time so you can overall track & recall specific versions later. Git is a Distributed version control system (D.V.C.S)



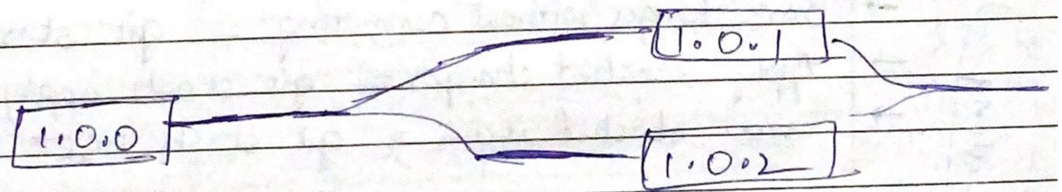
[Saves changes to on local machine]

A single server holds all the files & everyone collaborates on it.

Every developer has a full copy of repo

Git terminology:-

1) Repository:- A directory that holds your project code and its history
2) Working directory:- The current state of your project
3) Staging Area (Index):- The area where changes are staged before committing
4) Commit:- A snapshot of your repository at a certain point in time.
5) Branch:- An independent line of development



branching

Quick Work
Page No.:
Date:
M T W T F S S

Git Workflow: —

BASIC
- → Clone a repo = git clone <url>
- → Stage changes = git add <filename>
  to stage all changes [git add .]
- → Commit changes = git commit -m "commit message"
- → Push to remote repo = git push origin <branch-name>

Repo managmt
- → Initialize a repo = git clone
- → Check status = git status
- → view commit history = git log
- → view git commit history with one line summary = git log --oneline

Branching + Merging
- → Create new branch → git branch <new-branch>
- → Switch to a branch → git checkout <branch name> or
  git switch <branch name>
- →
  git checkout -b <branch-name>
  This command lets you create + switch to a new branch
- → Merge a branch = git merge <branch name>
- → Delete a branch = git branch -d <branch name>

Stashing
- → Save changes without committing = git stash
- → Apply stashed changes = git stash apply
- → view stashed items = git stash list

Quick Work
Page No.:
Date:
M T W T F S S

>

me>

[git add .]

it message"

, <branch-name>

log --oneline

-branch>

ranch name> or

anch name>

o a new branch

me>

nch name>

ish

y

Quick Work
Page No.:
Date:
M T W T F S S

Remote manag

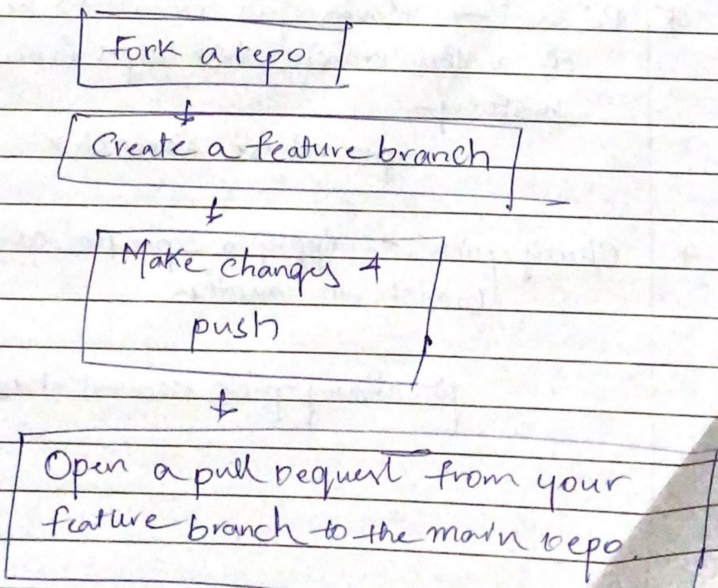
→ Add a remote repo = git remote add origin < >
→ view remote repo = git remote -v
→ fetch changes from repo = git fetch origin
→ push changes to remote = git push origin <branch->
→ pull changes from remote
  (~~pull~~ fetch + merge) → git pull origin <branch>

Undoing changes

→ Unstage a file == git reset <file name>
→ Undo local changes = git checkout -- <file name>
→ Revert a commit = git revert <commit-hash>
→ Reset to a prev commit = git reset --hard <commit hash>

* Be cautious, this delete changes.*

→ Forking + Pull requests
forking is copying someone's repo to your GitHub, making changes, + then submitting a pull request for them to merge your changes to current repo.

→

Fork a repo
Create a feature branch
Make changes + push
Open a pull request from your feature branch to the main repo.

Quick Work
Page No.:
Date:
M T W T F S S

Git Conflicts :-

Merge conflict occurs when git can't automaticalley merge files due to ovelapping changes in the same part of a file

- How to resolve?

1) Git marks conflicts in the file.
2) Edit the conflicting file + decide which code to keep.
3) After resolving, you need to add the file + commit the changes.
4) commands to handle :-

    i) git status (to see conflicts)
    ii) git add <conflicted file> after fixing
    iii) git commit (to finalize merge)

* Rebase :- Moves your commit to be applied on top of another branch, which helps in Keeping a linear history

```
git rebase <branch>
```

* Cherry-pick :- Apply a specific commit from one branch to another

```
git cherry-pick <commit-hash>
```

Quick Work
Page No.:
Date:
M T W T F S S

Advanced commands

→ Amending a commit = ~~git commit --head~~
git commit --amend.
" Useful for modifying the most recent commit's message or changes "

→ Squash multiple commits into one - git rebase -i <branch>
→ show differences between files :- git diff <filename>
→ Blame a file = git blame <file name>
" shows who changed each line "

Tags

→ Create a tag = git tag <tag-name>
→ push tags to remote = git push origin --tags
→ Delete a tag locally = git tag -d <tag_name>
remotely git push origin :refs/tags/<tag-name>

→ Best practices :-

* Commit frequently - (smaller, frequent commits make it easier to manage & debug.
* Write meaningful commit messages.
* Use branches for features
* Keep main branch stable
* Code reviews via pull requests (before merging new branches them into main)