Web Development Bootcamp

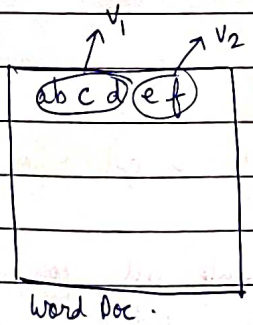
Tuesday Thursday Saturday → Love Babbar
Sunday → Lakshay

lovebabbar3@gmail.com

Pre-requisites : Git / Github / VSCode Setup.

Video 1: What is Version Control || Git vs GitHub

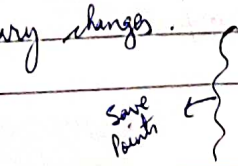
Example: When you type in Word doc, Ctrl + Z undo's your recent change. How is the OS able to do it?
Cause the OS maintains different versions as you type in word. So suppose you write 'abcdef' and want to undo typing 'ef', then OS will push the previous version where you had typed till 'abcd'. This is called version control. Had it not been in place, 'abcdef' would be the current and only version maintained by the OS, and thus the changes could not be reverted back.



When you type 'Ctrl+Z', previous version V_1 is deployed.

↳ Similarly, in development, suppose there is working code of user authentication. Now you add some code of product authentication on top of that code. So version control system will track all new changes made by you and has ability to revert it back.

↳ Version control system makes save points that saves your project. When you commit, they are committed changes, and when you don't, they are temporary changes.



GitHub – hosting service to host git repositories such that different people from all over the world can collaborate on single repository or project.

Git – Is a version control system to track changes in your source code during software development.

Video 2: Installing and configuring Git, and installing Github Desktop.

↳ Need to configure name and email id in Git. Name is the username that shows up in your Github account, and email id is the one you used to register to Github.

```
git config --global user.name "Eashan 7"
git config --global user.email "eashanroy7@gmail.com"
```

↳ To verify configurations set above:

```
git config --list
```

Video 3: Creating a Git Repository

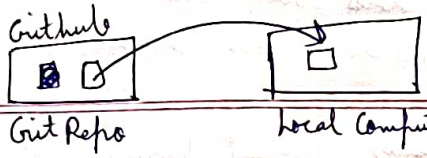
① `git init` ⇒ Converts local folder into a git repository

↳ When you convert a local folder into git repo, and cd inside the repo to see its contents, you'll see a hidden folder .git (this .git contains meta data of .git repo)

```
mkdir Desktop/dummy-git-repo      → Created a directory in Desktop
cd Desktop/dummy-git-repo         → Changed current directory
git init                          → Initialized folder as git repo
ls -al                            → To see hidden contents.
```

Output:
```
./      (denotes current directory)
../     (denotes parent directory ie. Desktop)
.git/
```

② `git clone`



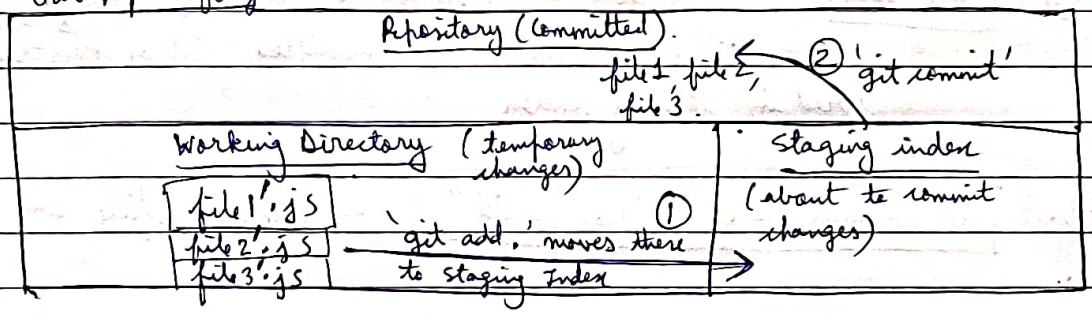
Video 4: Life Cycle of a Change

Folder { dot_dumy_repo :
Contents { ① file1.js —MODIFIED→ file1'.js
② file2.js —MODIFIED→ file2'.js
③ file3.js —MODIFIED→ file3'.js.

→ `git status` shows there modification details.

Commit → means changes are finalized

Git Repo life cycle



Every commit has a commit ID.
file1.js → Commit ID ABC
↳ file1'.js → Commit ID ABC2

VIDEO 5: Review a Repo History

③ `git log` ⇒ shows history of commits to a git repo.
↳ shows some recent commits, to get more history, keep pressing 'Enter'.
(To exit, press 'q'.)

④ `git log -3` ⇒ shows history of latest 3 commits.

⑤ `git log -p` ⇒ shows history of commits along with the actual changes in code.
(To exit, press 'q')
(To see more history, keep pressing 'Enter')

⑥. `git log --oneline` ⇒ shows just the commit messages and ids, but not the author and other details.

⑦ `git log --stat` ⇒ shows commit history along with the file names where changes have been made

⑧ `git show <paste commit ID>` ⇒ shows history of a particular commit, or what files have been changed, what lines have been changed, etc.

VIDEO 6: Doing your First Commit

⑨ `git add` – (i) to add new files to be tracked by git.
(ii) If old files are modified, it is staged i.e, stored in staging index

⑩. `git commit -m "Initial commit"` – commits file

⑪. `git diff file_name` – shows unstaged changes of your working directory w.r.t to staging area.
OR shows the changes made to the file between previous commit and current working directory (unstaged changes mean neither used git add nor git commit).

Two ways to use

Not recommended (cause changes from all directory to staging index)

`git add .`

`git add filename` Recommended (cause only adds the particular changes in the particular file to staging index).

In brief,
1. `git add .` → stages the changes made in working directory.
2. `git diff` → shows difference between previous commit and unstaged changes. If you stage the changes using `git add .` "diff" command will show you blank.
3. `git commit` → to commit the staged changes.

Head → Points to latest commit.

You make changes to a file in git repo.

↓

These are called unstaged changes in your working directory.

↓

Now git diff will show difference between staging area and current unstaged changes in working directory.

↓

Stage the changes by using git add.

↓

Now git diff shows no difference.

↓

Commit the staged changes by using git commit

(12) git restore file_name → Discards the unstaged changes made in working directory to previous HEAD i.e., previous commit

git restore --staged file_name → staged changes into unstaged. Once unstaged, you can use the above command to discard them.

(13) If you want to keep some files like .docx untracked by git, then:

→ In git repo, create file named ".gitignore"

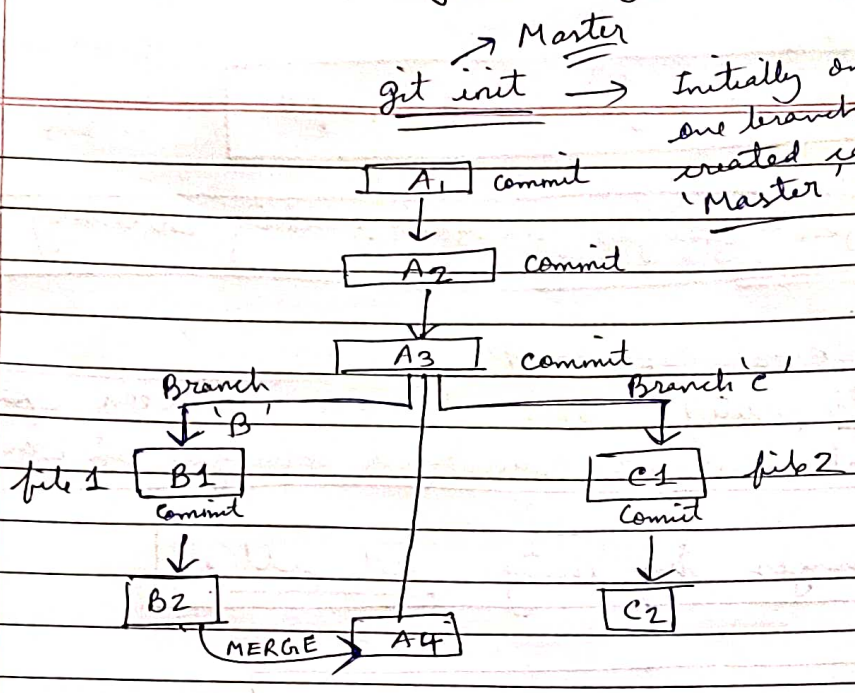
Inside the ".gitignore" file, type "*.docx" and save

→ If you do a git status, and if previously it was showing ".docx" as a file that needs to be tracked, now it will not show it.

→ git add .gitignore

→ git commit -m "Adding git ignore file"

VIDEO 7: Branching, Tagging & Merging



(14) `git branch` → Shows you all the branches in the repo and the current branch you're on.

O/P: * master

(15) To create a new branch:

`git branch quicksort`

Note: New branch is always created based on the current branch you are on. So here quicksort is created from master

git branch

O/P: * master (current branch master)
quicksort

(16) To switch to another branch:

`git checkout quicksort`

git branch

O/P: master
* quicksort (current branch quicksort)

Note: Whatever changes you stage or commit, are always done to the current branch you are on. So before staging/adding any change, check which branch you are on by using "git branch"

Note: When you switch to a branch, the repo folder in your local PC will show you contents of that particular branch.

Note: Git branch will always create new branch based on the current branch you are on.

(17) `git checkout -b bubblesort`

↳ To create a new branch and switch to the new branch called 'bubblesort'.

(18) To merge a branch into master:

→ Switch to 'master' branch first

`git checkout master`

→ Merge that branch into 'master':

`git merge bubblesort`

(19) To delete a branch:

`git branch -d bubblesort`

(20) Merge conflict:

→ To resolve merge conflict, open the file and edit what you want to keep.

→ Then git add.

→ Then git commit.

(21) Tagging: To tag a specific commit.

`git tag -a betaV1.0 paste_the_commit_id -m "My Beta Release"`

↳ Now if you do: git log

↳ You'll see that commit will have tag: betaV1.0 written beside it.

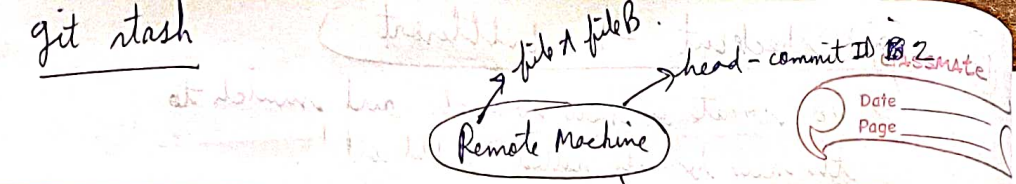
(22) `git commit -am "Merge conflict added"`

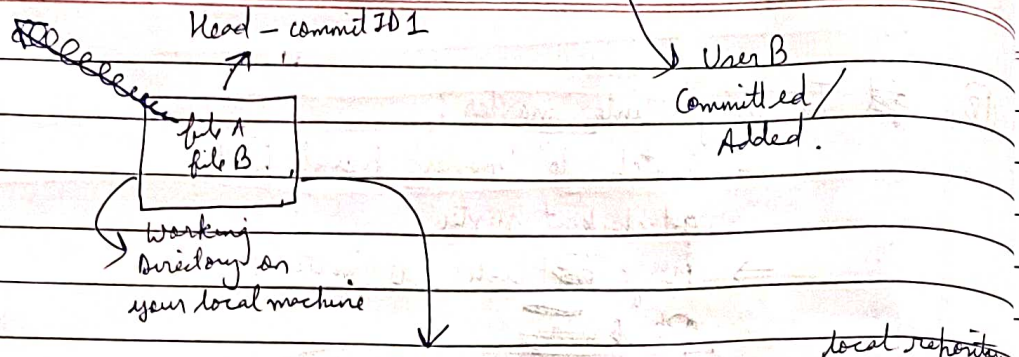
↳ To perform git add and git commit at once.

(23) To delete a tag:

`git tag -d betaV1.0`

(24.) # git stash

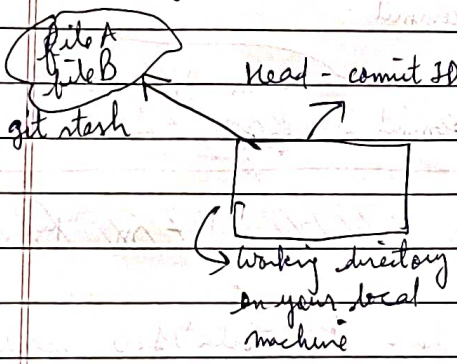




git pull → will sync your ~~working directory~~ local repository with the remote repository

But when you try to git pull from remote repo you'll get message that remote repo has latest changes to file A and file B, and you are making changes to an old version of ~~files~~ files. So you can't pull rightaway.

↳ To avoid this issue we use git stash.
Git stash will ~~copy~~ move all your ~~do~~ local changes in working directory to a separate stash area.



↳ Now do git pull to get latest version from remote repo.
↳ Now from git stash, pull your own local changes as well,
↳ There will be merge conflict, resolve it, and then commit.

Your friend added a new commit from his machine to github which is remote repo for you. (Github is remote repo for you)

↓

In your local pc, if you do git log, you won't see that commit

↓

Now you make unstaged changes in the same file

↓

You do git pull to sync with remote repo

↓

You'll get error: your local changes will be overwritten by merge, please stash them before you merge

This error means your remote repo and local repo have different versions for the same files.

↓

git stash

↓

Your local changes will be moved to stash area.
Can check with git stash list

↓

git pull

↓

git stash apply
// to bring back changes from you pushed to stash to your working directory.

↓

You'll get merge conflict

↓

Open the file: >>> Updated upstream (means remote repo change).
>>> stashed changes (means your working directory change)
So you decide which change you want to keep and edit file accordingly

↓

git add. → git commit -m "Correct"

(25). `git push -u origin master` → place any branch name you want to push into
↳ to push to git repo

VIDEO 8 : Undoing changes in Git

git commit --amend → amend the most recent commit
git revert → revert given commit
git reset → delete commit (dangerous command)

(26) `git revert commit_ID`
↳ After this cmd, press Esc :wq! then Enter.
↳ If you do git log, you'll see additional revert commit

(27). `git reset --soft commit-id`
↳ Deletes the commits above given commit-id, such that HEAD points to the given commit-id.

(28). `git reset --hard`
↳ discards unstaged changes in working directory and points head to the sha id (ie., commit ID) you mention in command.

(29). `git reset --mixed`
↳ Keeps unstaged changes intact, and head points to mentioned sha ID.

(30). `git reset --soft`
↳ Unstaged changes will be staged, and head points to mentioned sha ID.

(31) `git commit --amend`
↳ to change commit message of the latest commit.

(32) git diff --staged

↳ shows staged changes that have been made with respect to previous commit.

VIDEO 9: Git GUI

If in your local, you're creating a git repo and pushing it to Github, you need to sync your local repo and remote repo (ie., the Github repo). So for that use the command:

git remote add origin paste_github_repo_url

Steps:

1. Create a Git repo with name "myrepo.git" in Github
2. In your local, create folder "myrepo", do git init, add README.md file, commit.
3. git remote add origin paste-github_repo_url
4. git push -u origin master (to push from local to Github repo's master branch)

↳ Alternatively can use Github Desktop which is GUI tool compared to using git bash.

↳ Pull requests → merging your branch to master or other branch.

(same like merge, just more interactive and has more functionality).