GIT (Version Control System) Free and open scalable. It’s a tool locally

Its nothing that but stores data as version control system -- bank statement/project history.

To track history / collaborate (different developers change)

GITHUB.com is a site where code can be saved as repository under github userid.

If repo can be shared then public can be viewed.

Login with github userid on github.com

Create a repo and provide a name, public and readme.md file.

Download and install git to local machine by vscode. And check with

git –version

Configure git using ------git config------- ONE TIME SETUP

git config --global user.name “Riyaz”

git config –global user.email “[riyaz75@yahoo.com](mailto:riyaz75@yahoo.com)”

git config –list (to check what is configured in git config)

**FIRST WAY to copy projects which is available on github so call this as GIT CLONE. (Github to local)**

For cloning need to provide a code available on github https link.

At local machine, type **git clone <https/ssh/ ---link>**

Git clone <https://github.com/riyaz75/sch_mgmt_proj.git> (link can get from github repo of code https)

Whatever code or file available on repo it will be copied to local machine.

**Git status to know the status of the folder.**

modified (any change in previous file need to commit in git),

untracked files (new file need to add in git),

So use **add** to stage in git, ready to **commit** will be next step

**git add . or got add filename** (This will bring to staged status means ready to commit)

commit means to make a record of changes like what, when such details to be documented.

**git commit – m “Added docs folder and files”**

After committing status will tell you that your local is ahead with extra commit and your github is pending the latest commit so need to **git push at remote (guthub repo)**

**git push origin main (Copy from local to github)**

**git fork** is a rough copy of repository for others into your github. Copy express code to your github repo) for contributing to others code. OPEN SOURCE

**\*\*\*\*\*NEXT is GIT LOCAL\*\*\*\*\***

**On local machine create a folder xyz**

Cd xyz and run **git init** (to add folder into git environment)

Change or add files/folders in the folder then check

**git status** (Found untracked files since it is new)

**git add .**

**git commit -m “first commit with initial files and folder”**

Now create a **repo in github** and copy the https link as origin

Locally **git remote add origin <https link>**

**git remote -v (to verify remote origin)**

if number of teams are working on same project such as frontend, backend and bugfix.

So we create branches to know branch

**git checkout -b <feature1>** (To create new branch as <br name/feature1 >)

**git checkout -d feature2** (To delete a branch as feature1)

**git branch -M main (rename branch)**

**git checkout <branchname>** to shift to other branch

**git branch** (to list the branches) and \* shows current branch

**git push -u origin main** (-u indicates that **-u is for origin main** for future use Ex: **git push** ) (Upstream)

**git push origin feature1** (push the code to feature1 branch)

Now the difference wants to know in branches , files … So

Let say you are in feature1 branch and want tto copmpare the difference with main1

**git diff <br name> or git diff main**

**merge the branch with main**

**git merge main** (feature 1 will merge with main br)

**or**

**using pull request from github for this they create PR is PULL REQUEST**

PR will be reviewed by Senior dev who is incharge of main branch. (accept or not with comments)

From **github Compare and pull request** then **merge pull request** then **confirm merge then done.**

**AND FROM REMOTE TO LOCAL Transfers use**

**git pull origin main** (Bring code from Github/Remotte to local repo)

**RESOLVING MERGE CONFLICTS Manually it can be done seeing all**

**UNDOING CHANGES**

**git log** to see the hash values

Staged changes (add stage)

**Git reset <filename>**

**Git reset** for all

COMMITED CHANGES (for one commit/head)

**Git reset HEAD~1**

COMMITED CHANGES (for many heads/commits)

**Git reset HEAD~1 (for many commits)**

**Got reset <commit hash>**

**Git reset –hard <commit hash>**