What is Git and Github -

Use Case -

- Imagine your application
- Added some **features**
- Application breaks and not running properly as before
- Now u want your past codebase where your application is running properly
- Wishing some sort of way - Picture of that codebase
- Saving history of project datewise, timewise, some files, code changes
- You want to look your code which was on 1st July -
- Imagine that 100 people are working on one project
- Assume all of them are working on one folder, how everyone is contributing to the project without changing others code - Sharing some changes.
- We can do this via **Git and Github**
- It allows us to see history of the project, date, time, which person, which change, where in the project, - **Git** helps over here
- **Git is a version control system that lets you manage and keep track of your source code history.**
- Git is - like Email Service - Server and backend - VCS
- Github is - like Gmail, Yahoo, Outlook interface - Software - hosting git repo
- **GitHub is a cloud-based hosting service that lets you manage Git repositories**.
- **Github** is platform like online website where we can host git repositories -
- What is the repository ? => It is basically a folder where all the changes are saved.

Why we are using Git and Github -

- Just now understand why we are using it..we will cover ahead in this session.
- Maintaining history of the project, when , where, which person, sharing it with people around the world.

- Look into any project on Github.com
- Open https://github.com/maheshdbabar9340/Amazing-Python-Scripts and go to commits - History of the project - Check anyone's changes - file name and which folder.
- You can see all the history - Older as well.

Downloading Git -

- Go to Git.scm.com - download from here - Do not download GUI one - Diff OS
- Easy to set up
- Open Git Bash then
- git - command for checking or git - - version - Git is installed
- Structure of the session - Used folder and Git Bash terminal.

Basic Linux Commands -

- Create folder using = **mkdir Project**
- After that list command = **ls** = You will see list of the files that are present in the folder
- How we will go inside in the folder using command line
- Change directory project = cd Project
- Now if we need to **maintain the history** of the project using Git
- Where the entire file has been stored - If we add file or deleted - So all histories are stored in another folder that git provides us =.> known as git repository => stored in .git folder which is hidden

Initializing a Git repository -

- So how we will get the folder where all the histories been saved
- Initialize an Empty git repository - **git init**
- Repository = Folder
- Check .git folder

- With the help of **ls -a** command, checking for all hidden files
- You can check what is inside the .git folder with **- ls .git**
- Don't worry about master and all files inside the git folder - We will cover
- Now any change in the project - Git will pick it up
- Hey mahesh, you have done these changes, please click a photo of it so other people can see your changes
- Lets make some changes

Making the First Change -

- Create new file using **- touch hello.py**
- Does anyone know that mahesh created hello.py, Is this history maintained in git repo - No
- How we will see that which file has been added or modified or deleted - what changes are been done and not currently saved in the history of that project
- Command for that - **git status**
- Check the output - U can see untracked files
- Currently if I share this repository to any project then no one knows that mahesh has been added to the hello.py file.
- Now we want to maintain these changes - Actually we wanted these changes in our project history
- Let's take an example - Git relates with wedding scenario
- **Untracked files** = Guests whose photo has not been taken by photographer yet
- So we need to take a photo of it so that we can save it in our git repository.

Staging the First Change and Adding data to the files-

- **git add . -** Everything in the current directory which is untracked ..put all those files in the staging area - U can write individual names as well.
- Now do **git status** - files which are added colored into green from red

- Now all are in the staging area - U can click the picture and save them in the history using
- **git commit -m “massage” -** All photos has been taken and saved in the history - Don't worry about all unknown terms
- U might get Author Identity - Please add username and email address with commands.
- Do **git status** again
- Nothing changed - No new photo has been taken - Nothing changed
- Do some changes
- **vi hello.py**
- **Esc and :wq or :x**
- **cat hello.py** = Displays all content available
- **git status**
- **git add .**
- **git status**

Remove files without committing to stage

- **git restore - -staged hello.py**
- **git status**
- **git add .**
- **git status**
- **git commit -m “message” = insertions and deletions and modifications**

View overall history of the project

- **git log**
- Watch all commits and date and all
- If I delete this file using

- **rm -rf hello.py**
- **git status = File is deleted**
- **git add .**
- **git commit =m “file deleted”**
- **git status**

Removing commit from history of the project - Imagine that file has been deleted by mistake

- You cannot remove a particular commit from the middle because as you can see each commit has a hash id and each commit is built on top of each other.
- You can unstage commit
- Copy hash id till you want all changes and it will going to remove all above commits = Using
- **git reset hash_id**

Stashing Changes -

- Imagine that like you are few people whose photo has not been taken, go to the back stage and whenever we want to come back, we will get you back
- Imagine that you are working on some feature and it’s working fine but also you don’t want to save your progress, like without commiting to git, without saving history
- So whenever you want that thing back, we will get it back
- First take that people on stage
- **git add .**
- **git status**
- Also we don't want to lose this changes
- **vi hello.py**
- **touch new.txt**
- **git add .**

- **git status**
- So we will do
- **git stash**
- **git status**
- **git log**
- You can see that all changes are not reflected in the history - it is as it is.
- **cat hello.py**

Popping Stash -

- How to take all people from unstage area to the stage area
- **git stash pop**
- We can all changes that are made previously

Remove stash changes - which are not committed

- **git add .**
- **git stash**
- **git stash clear**

GitHub

- U can your project on github - if u want to share it with other people
- Go to github.com
- Create new repository
- So you will get URL of your repository
- And we want this url to be attached to the project which we have created earlier

Connecting Remote repository to Local Repository

- **git remote add origin URL_which_copied_from_github**

- Git - It is just a git command
- Remote - U r working with URLs
- Add - Adding new URL
- Origin - What is the name of the URL that you are going to add
- So name of the copied url is origin - It can be anything - origin is by convention
- Origin - All the repo which are inside in your personal account they have name as origin
- **git remote -v**
- This will show all the repositories that are attached to Project folder
- Now this URL is connected with folder now

Pushing Local changes to Remote repository

- Refresh github - Nothing changed
- Because we have not shared the changes on this URL
- Command for that is - **git push origin master**
- **Origin - which url u want to share it**
- **Master - which branch u want share it**
- It may ask you to sign in with your browser. - Click on Sign In
- Refresh Github - Click on the commits

What are the Branches -

- Master - It is by default
- Let's create few commits
- touch hotel.txt
- git add .
- git commit -m “hotel.txt added”

- Create 2-3 more commits like these
- What is the branch ? Learn Git Branching - https://learngitbranching.js.org/ - visualize
- 4 commits
- Can u see branch structure - Linked to each other
- It’s like Directed Acyclic Graph

What is the use of Branch -

- New Feature
- Resolving Bug - Create new branch
- What is this head ? visualize
- U should never commit on master branch
- Check a look of any open source project
- Master branch - Code is used actual by people - Users and developers
- Because our code which is not finalized might contains errors
- I.e. all the code should go in another branch so that user should not affected

Making new branch and Switching to it -

- Visualization
- **git branch addition**
- **git checkout addition**
- What is a head? * thing - All the commits that u will made - it will added to head -
- We have changed head to addition
- So all commits will go to addition branch - Main branches as it is
- Another thing is U r not only one which is contributing to this project - Many peoples
- While u r working on your branch
- Someone else did some changes

- git checkout master
- git commit
- It will go to master branch
- Anyone can work - Use of branches

Merging branching to master -

- git merge addition - It will be merged to head branch
- git checkout c3
- git checkout master
- How merging will happen - Via Pull request
- Why should everyone work on a new branch? Not to commit to master ? How do we get a new branch merged? What is a Pull Request?
- Go to **git bash**
- **git log**
- **git remote -v**
- Push all commits to origin branch
- git push origin master
- Refresh github … go to commits

Working with existing projects on Github -

- Select any project, what do we need to do inorder to make changes
- Should anyone in the world be able to make the changes to a specific organization?
- Should they get access to a specific organization?
- How dangerous would that be? So if you tried it right now …you'll be not able change directly any file in this folder - Commclassroom
- Because you don't have access to this account

Why Fork and How to Fork -

- Then How we will contribute to the commclassroom organization
- Create a copy of that project in your own account - Click on **Fork** in my own account
- mahesh/commclasroom op project - It would be like these
- You will have 2 (main and forked) repositories

Cloning the Forked Project to local -

- From your personal git repository - it means from origin - Any folder which starts with your personal account - Origin
- **git clone URL_Personal_Account_Which_Is_forked**
- **cd commclassroomOP**
- **cat README.md**
- U cannot make changes directly to anyone's account
- U need to fork it
- How someone will feel that u r changing code without his permission
- Take a copy ..do whatever u want - it will not reflect to others account
- Until people have a approval for this project can merge ur code , approve your changes via pull request

What is upstream and adding it to local -

- U know that origin means it's your own account
- From where u have forked this project - organization URL - that is known as upstream url by convention
- I can add one more url **git remote add upstream URL_Organization**
- **git remote -v**

What is Pull Request -

- Let's make few commits
- In the beginning I mentioned to you that never commit on master branch
- Always create new branch for whatever u are working on

- Lets create new branch
- **git branch mahesh**
- **git checkout mahesh = head will come on mahesh branch = main will be changed to kunal**
- **git add .**
- **git commit -m “message added”**
- **git log**
- So whatever code I have in Mahesh branch please merge it with main branch
-
- Given any example of open source like Kubernetes - Show Closed PR’s
- Merging feature branch to main branch so that other people can see that changes
- They will give some suggestions - We need to make those changes as well - After approval our PR will be merged.
- https://github.com/tensorflow/tensorflow/pulls?q=is%3Apr+is%3Aclosed 4 th tab - Fix year in sqlite

- Completed changes - But how those changes will be visible in the main branch - So for we will request it via pull request
- People review your code, suggest some changes ..u will make those changes.. then they will approve via pull request ..so those changes will be merged then your code sample will be in main projects main branch
- Below code will be in main website/application

Never commit on main branch & creating our first pull request -

- New feature - New Branch
- U don't have access to upstream - U r unable to push it to upstream directly
- But u have access to your personal forked repository - which is origin url

- **git log**
- **git push origin mahesh**
- Goto github to your personal forked one
- **Click on Compare and Pull request - Automatically**
- Describe that someone is trying to merge changes to main branch of the project
- **Before merging I want to show you something**
- If you have created few more commits - and then push it to mahesh branch -
- touch new.txt
- git add .
- git commit -m “new added”
- git push origin mahesh
- Check github UI
- Now u will notice that u don’t need to create new PR
- It will add this commits in the previous PR only
- Commits will be increased
- So these is the reason you should never commit on main branch
- Why ?
- Imagine u are working on 10 projects, for every feature u created only one PR
- How difficult to review your code
- All 10 diff things only on 1 Pull Request
- I.e. why for every new feature, every new bug - create a new Pull Request
- From here u can see, it will not allow us to create new Pull Request - All commits will go into one PR of 10 diff project
- In simple language, 1 PR = 1 Branch = 1 Feature
- Create a new branch in your local folder, make a PR from that.

- Give open source example - All PR in on one PR - All code base will be mixed

Removing a commit from the pull request by force pushing to it -

- Let's assume I want to remove last commit
- **git status**
- **git log**
- Copy hash id of second last commit - so that all changes will be as it is till that commit
- **git reset copied_hash_id**
- Now file will be unstaged
- **git add .**
- **git stash**
- **git log**
- File will be deleted from folder
- Now i have done some changes and I want to push it but forcefully push it because online repository contains 2 commits and my local contains only 1 commit
- **git push origin mahesh -f**
- Let’s refresh github
- Check github and all commits -

Merging Pull Request -

- Changes made in mahesh branch will be seen in master branch, main projects main branch,

Making forked project even with main project -

- If u r not able to change others account, how others will change your account

- 2 ways - You can click on fetch upstream - **Sync Fork**
- **Describe your commands only - not in practically**
- Branch is behind 2 commits of upstream branch - Organization one
- Check the commits
- Assume that someone merged their pull request, so how you will be able to see what changes have been made, your branch should be up to date. - With the help of fetch upstream button - **Sync Fork button**
- **git status**
- **git log**
- First I need to fetch all the changes
- **git fetch - -all - - prune**
- All branches - Deleted as well
- Fetched all changes
- Reset the main branch of my origin to the main branch of upstream
- **git reset –hard upstream/main**
- **git log**
- Now the main branch of the personal account is exactly the same as the upstream branch.
- **git push origin main**
- Sync with the main upstream branch

- Lets create new commit in main project - Commclassroom
- Do it via github only
- But now your personal account is 1 commit behind of upstream
- **git pull upstream main**
- **git log**

- **git push origin main**

Squashing commits -

- **"squash" in Git means to combine multiple commits into one.**
- Create new branch and checkout to that branch
- **git branch temp**
- **git checkout temp**
- Make sure that your main branch is up to date before creating new branch
- In the temp branch- touch 1
- git add .
- git commit -m “1”
- touch 2
- git add .
- git commit -m “2”
- Just creating random files till 4
- Let’s assume that I want to merge all commits to one commit

Using Rebase Command -

- U can do these by resetting with the help of hash id
- All the commits will be in the unstage area and get them in the staging area and do 1 single commit at a time.
- Copy below hash id commit 1
- **git rebase -i hash_id = i interactive environment**
- All the commits above it I can pick or squash by going to pick and convert it into s
- Squash 2 , 3 into 1 single commit
- Main commit we have 1, ==== commits 2 and 3 will be squashed.
- U can merge middle s 2, s 3 and to pick 1

- Whatever squash you have, merge it to the above pick of that.
- Esc and :x
- Now you need to add new message after 3
- Esc and :x

Using the Hard flag to reset -

- **git log**
- Check all commits because all commits are on the same commit

What are merge conflicts and how we will resolve them -

- Let's assume that u made a change on line 3 and someone else also made a change on line no. 3 = Now git will get confused - should i take your changes or should i take other person's changes - which change did you want to take
- So git will ask for help - tell me which changes you want to take
- You want to take this change or that change because both of u people modified same line no.
- Let's try to get a merge conflicts first

- Create a new pull request in the main account by github UI - Click on compare Pull Request
- Now create new branch as well from git bash UI
- Let's take a new_mg branch only.
- Let me change line no. 4 from same file by removing first line
- git add .
- git commit -m "new merged message"
- git push origin new_mg
- Go to Github UI
- See Compare & Pull Request

- Click on Pull request
- Then you will be able to **Resolve Conflicts** button - It will give merge conflicts
- Do some changes and remove red lines
- And Mark as resolved
- Now we will see both line 4 has been changed
- Click on Merge Pull Request
- This will be merged and it will give merge conflicts
- So over here overriding will happen
- Do Fetch and Merge and all.

- Get some hands on practice then only you will be able to see the intuition behind it.