**Git Tutorials #1**

What is Git/GitHub & Why do we need it? |

What's Up everyone. Today I am going to discuss Git and Github . So today i am going to talk about what is Git?, Why is it around? and What is Github?. So Let’s get started.

**Git**: So Git is a version control system. Let’s assume I have made an AI desktop assistant. Now I was working on version 1 which has 3 files, f1, f2, and data.csv. After a few days I have changed the code and made version 3 which also has 3 files, but this time f1, f2 file doesn't exist, instead it has f3, f4 and data.csv. So now f1, f2 files are not around. If in the future we get into a situation where we need those files there will be no option, but if we use a version control system like Git then we can go back and see those files and work with it

**Pros of Version Control System:**

1. Easy File Recovery: You can easily recover files from the version control system.
2. Issue: If more than 1 person is working on a project and the project stops working after someone’s commit then you can easily check who’s commit it was and rollback.
3. Rollback: You can easily rollback to a previously working state

**History of Version Control Systems:**

**Local Version Control System:**So the first version control system made by programmers was a Local Version Control System, which used Database to keep track of files. But the major drawback was all the changes were saved in the computer. So if the computer gets corrupted then all of the data will be gone.

**Pros:**

* You can track files.
* You can Rollback.

**Cons:**

* If the local Computer/ Laptop gets corrupted then all of the data will be lost.

**Centralised Version Control System:**So after Local Version System the programmers made Centralised Version Control System. In the Centralised Version Control System the changes were saved in a server, so there is no chance of losing all the data and the changes due to data corruption. The programmer needs to pull the file from the server, make his changes and then push that modified file to the server. Now there is a chance that if the server gets damaged then the files can be lost forever.

**Pros:**

* You can track files.
* You can Rollback to a specific version of the file.
* Changes were saved in a server instead of a local computer.

**Cons:**

* If somehow the server gets damaged then the files and changes will be gone.

**Distributed Version Control System:**This is almost like the Centralised Version Control System. The only difference is when the programmer pulls the file he gets the complete changes of that project and each programmer has a complete backup of the server. So if the server gets damaged the programmers can fix it easily without losing any files.

**How was Git Started:**In 2002 the programmers who were working for development of Linux started using Bitkeeper VCS as their version control system. But in 2005 Bitkeeper removed their free of charge status from them and asked to pay the Linux Development team a huge amount of money. After this**Linus Torvalds**, the creator of **Linux** started making his own version control system and then he made Git, a free Centralised Version Control System.  And everyone started using Git.

**Features of Git:**

* It stores the snapshot not the differences so the only file which is changed will be stored and disk space will be saved.
* Almost every operation is local. You can work on your local machine and then push those changes to the centralised repository.
* Git has integrity. Git generates SHA-1 Checksum for each and everychange, so only those who have access can modify files.
* Git generally adds data. So as you change the versions it stores them.

**Github:** Github is a hosting website which hosts Git repositories. Git is free to use but Github charges for managing those files. There are also alternative websites which host git repositories like BitBucket, Gitlab etc. Among these Github is most popular.

### Git Tutorials #2

### Installing Git + Initial Setup? |

Hey everyone welcome, In this tutorial I am going to teach you about installing Git and its Initial setup.

#### ****Installation:****

* Navigate to [Git’s](https://git-scm.com/)homepage and download the installer.
* Open the Installer and follow the simple steps to install it.
* Done.
* Now Search for Git Bash on the search bar and open the application.
* To check if git is installed correctly, type git and press enter in git bash. If you don’t get an error then it is installed correctly.
* You can run git from powershell too. Just type git there to verify it.

#### ****Setup:****

* You can use the CD command to navigate through folders.
* Create a folder where you want to run git and open the folder
* Now right-click while pressing shift and choose **Open git bash here.**
* Let’s setup the username and email by running these codes:
  + git config --global user.name “Username” then press enter.
  + git config --global user.email “[UserEmail@useremail.com](mailto:UserEmail@useremail.com)” and press enter again.
* Now type git config --list to show the configuration and verify that you have entered the correct name and email
* You can also change the editor by entering the command git config --global core.editor vim to use vim as editor

### Git Tutorials #3

### Git: Three Stage Architecture |

Hey everyone welcome, We are going to learn about Three-stage architecture of git. So let’s get started

Let’s assume we are working on a project which has an index.html file, a folder named static and engine.js file. Now you have completed this and made it version 1 of this project. Now you want to add more features into it and if anything goes wrong, you could roll back to version 1 which runs perfectly. In this case, you will take a snapshot of the project. Which means you will save a state of this project where it is running without any errors. This process is called “Commit”. Now you have committed this project as C1 and continued to work. Now You have changed index.html and also the other files, but somehow engine.js file is not working anymore. So, in this case, you will just put index.html file on the second snapshot and pull the other working files from the first snapshot. This is where Three-stage architecture comes.

#### We have 3 areas in the Three-stage Architecture:

##### **Working Directory:**

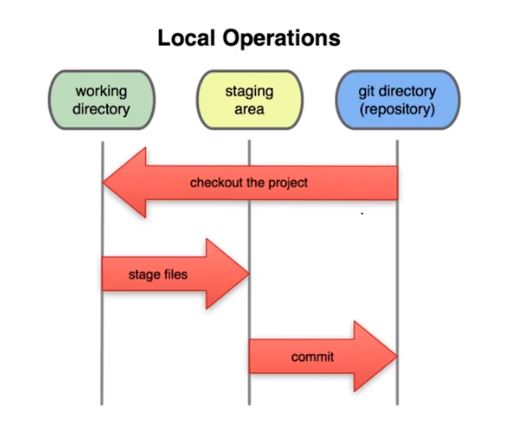
The working directory is the folder in your local computer where the project files and folders are stored.

##### **Staging Area:**

The staging area has those files who are supposed to go to the next commit. Only those files which are needed to go to the next commit stay in the staging area. Like, suppose you are working on that previous project and you have upgraded the index.html file but somehow you broke the engine.js file, so now you will just add the index.html file to stage area so it can be added to the next commit and the engine.js will be used from a previous version. And as you have not staged the broken engine.js file, the broken fill will not be committed.

##### **Git Repository:**

Git repo is a hidden file named .git. It stores all the commits and compresses them. So when you need a specific commit it can present that to you.



**Fig: Git Three Stage Architecture**

### Git Tutorials #4

### Tracking Our first Git Project |

Hey everyone welcome, We are going to learn about Tracking a git project. So let’s get started.

Let’s say we have 3 files in a folder for which we want to create a git repository. So follow these steps to track these files using git.

* Open the folder, Right-Click anywhere while shifting and choose “Git Bash Here”.
* After Git bash is opened type “git status”.
* It will give you a message that “fatal: not a git repository (or any of the parent directories): .git”.
* It is a message that says these files are not considered as a git repository.
* Now we will type “git init” to initialize this folder as a repository.
* Now if we type “git status” then we will get the files which are present in the folder. Also, it says there that the files are not tracked.
* Now to track all the files inside this got repository we will type “git add --a”, which will add all these files to the staging area.
* Now we have to commit using this command:
* “git commit -m “Initial Commit”
* Now if we do git status again it will say There’s nothing to commit, working tree clean. Which means we have successfully tracked our files.
* Now to see the commits we have made, we will use “git log” command.
* Now if we modify a single file and want to stage that only file we will use
* “git add file.ext”(Where file.ext is the filename and extension).
* Now we will commit with a message by typing
* git commit -m “Your commit message”
* Done Now You have successfully tracked your files.

### Git Tutorials #5

### **Cloning a Remote Git Repository from GitHub** |

Hey everyone welcome, We are going to learn about Cloning a Git Repository from Github. So let’s get started.

#### ****Steps:****

* Open Browser and go to the GitHub page of that git repository.
* Let’s say for an example we are cloning Tensorflow’s repo.
* Let’s navigate to the official Github repo of [Tensorflow.](https://github.com/tensorflow/tensorflow" \t "_blank)
* Now click on code or download and copy the URL.
* Open git bash and type “git clone (copied URL)”
* Example: “git clone <https://github.com/tensorflow/tensorflow.git>”
* Wait for the git clone to complete, this will only take a little bit of time for the first time.
* Now you can change the files locally, track them also commit them locally.

### Git Tutorials #6

### Git: File Status Lifecycle |

Hey everyone welcome, Today we are going to learn about File status life cycle. So let’s get started. When you start to track files on an empty repository every file stays in the untracked stage. Which means we have not staged these files. So after adding the files by typing “git add --a”, every file moves into the unmodified section. Now if we change a few files they will be moved to modified and staged. Now what happened here is, we have added every file to the tracker and now as we modify files will be moved to the modified phase. Now if we run the add command then the files which were in the modified stage will be staged.

* Let’s open a new folder and initialize as a git repo by opening Git bash and typing “git init”
* Now lets add the files by using “git add --a”.
* Now every file has been staged.
* Now if we change 1 file and do “git status” then you will see that the file we changed is present both on staging area and untracked area. Because you have staged the file and those are meant to go to the commit until the user stages other changes. So only those will go to the next commit which are staged. The untracked files will be ignored unless the user adds them to stage.

### Git Tutorials #7

### .gitignore: Ignoring Files in Git |

Hey everyone welcome, Today we are going to learn about gitignore. So let’s get started.

#### ****What is gitignore?****

.gitignore is a file which tells git which files (or patterns) it should ignore. It's usually used to avoid committing unnecessary files from your working directory that aren't useful such as compilation products, temporary files etc.

#### ****Now Let’s see how****

#### ****to add .gitignore to your repo and ignore files****

* Let’s start by creating an unnecessary file.
* Open git bash on that folder and type “touch file.log”
* Now you will see file.log will be created.
* Now we will gitignore this file.
* Again open git bash and type touch .gitignore.
* Now if you do “git status” it will return you 2 untracked files, .gitignore and file.log.
* Now let’s open .gitignore file using notepad, type there file.log and save and close it.
* Now if we do “git status”, it will return only 1 file, which is .gitignore. Because the other one will be ignored.
* Now type “git add --a” to add gitignore to the staging area.
* Now let’s commit by typing “git commit -m “added .gitignore””.
* Done! We have successfully created .gitignore for our repo.

#### ****Ignoring Specific Extension files:****

* Create .gitignore the same as the last one.
* Now open it using notepad and type there “\*.extension” to ignore. (example for ignoring .log files : “\*.log”)
* Now save and exit the notepad and commit the gitignore to your repo.

#### ****Ignoring Folders:****

* Now we will ignore a whole directory.
* Create and open .gitignore and type there the directory name followed by “\”. (example: dir/ )
* If you create a blank folder, git will automatically ignore it.

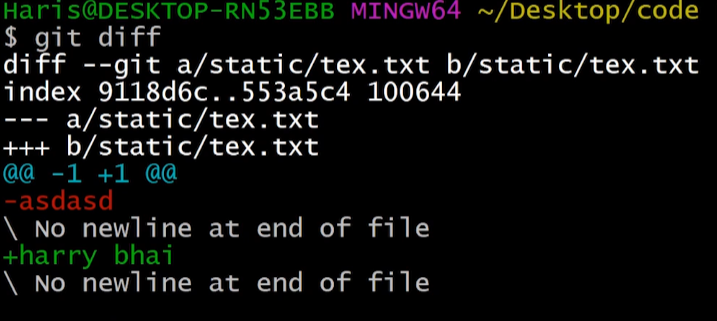
### Git Tutorials #8

### Git Diff: Showing Changes Between Commits/Staging Area & Working Directory | .

#### ****What is Git Diff?****

Diff command is used in git to track the difference between the changes made on a file.. Diff command takes two inputs and shows the differences between them. These inputs can be branches, working trees, commits and more

* Let’s start by staging one file.
* Open git bash on that folder and add that file into the staging area by typing “git add --a‘.
* Now if you do “git status” it will show that the file has been staged
* Now if you modify the file and do “git status”, it will show that the file has been modified and not staged for commit and also it will show that the file is ready to be committed. Why does it happen? It happens only because we have staged the file earlier so it has moved to the staging area and is ready to be committed, and when we have modified it, it also shows up on the modified area. We have seen this happen earlier too.
* Now let’s compare the working directory with the staging area so we can see those changes.
* Type **“git diff”** and see the output.
* The output should look like this.



* The red line shows that the lines are modified or deleted on the staging area and the green shows things added on the working directory.
* Now if we do **“git add --a”** and do git diff it will show nothing because we have staged every change.
* compare every commit with your working directory.

**“git diff --staged”**

### Git Tutorials #9

### Git: Skipping The Staging Area |

Hey everyone welcome, Today we are going to learn how to skip the staging area. So let’s get started.

* Let’s modify a file in our directory.
* Now open git bash in that directory.
* If we do “git status” it will show us that the file has been modified but not has been staged.
* Let’s create a new file on that directory and do “git status”
* It will show us that the tracked file (the file we modified earlier) has been modified but not added to the staging area and the new file is untracked.
* Now let’s skip the staging and directly commit.
* NOTE: Only tracked files can skip the staging area, to add your file to tracker type “git add --a” or “git add filename.extension”.
* Now let's skip the staging by typing   
  **“git commit -a -m “Commit Message””.**
* Now if we do “git status” it will show that the working directory is clean. Which means we have successfully skipped the staging area and committed the changes.

### Git Tutorials #10

### Moving and Renaming Files In Git |

Hey everyone welcome, Today we are going to learn how to move and rename files in git. We can also do it manually but then, we need to stage those changes using git bash. That is why we are going to do it inside git. Git will automatically stage it after moving/deleting/renaming that file. So let’s get started.

##### **Deleting Files:**

To delete files using git we need use this command: “”**git rm filename.extension**”. It will delete the file and if you do “git status” now it will say that the file has been deleted and staged. Now we can commit it by using the “**git commit -m “commit message”**” command.

##### **Renaming Files:**

To rename file we need to use this command : “**git mv filename.extension renamefile.extension**”. It will rename the file from filename.extension to renamefile.extension. And it will also get automatically staged by git. All we need to do is commit.

##### **Untracking Tracked Files:**

To untrack a specific file we need to use “**git rm --cached file.extension**” command. It will remove that file from the tracker and it will become an untracked file

### Git Tutorials #11

### Git Log: Viewing & Changing Commits In Git |

##### **Commits History:**

To see the commits made on the git repo, we need to type “**git log**”. After typing this you can see the commits that have been made on the repo. To exit we need to type “q” on our keyboard and press enter.

##### **Diff in Commits:**

To see the Diff in a commit we need to use “**git log -p**”. It will show what has been changed on a commit. To see specific no. of commits with changes we need to use “**git log -p -2**”(for seeing the last 2 changes).

##### **Brief Summary:**

Also we can get a brief summary of commits by typing “**git log --stat**”.

If you want to see the commits on one line then type “**git log --pretty=oneline** ”.

##### **Customized Commit Output:**

If you just want to see the commits and The author then use “ **git log --pretty=short**”. And if you want a little bit more info like who is the committer, commit message, then use “**git log --pretty=full**”.

If you want to filter commits by time, then use this command,   
“**git log --since=2.weeks**”. To see the last 2 months of data type “**git log --since=2.months”.**

We can format the output by using the formatting  codes provided in [Git’s](https://git-scm.com/) website. For an example we can use this format to print just the author name and the hash : “**git log --pretty=format:“%h --%an”**email=>**git log --pretty=format:“%h --%ae”**.

##### **Changing A Commit Message:**

Now to change a commit. So, to change the most recent commit we need to type “**git commit --amend**” and press enter. Then an editor will open where you can change the commit message by pressing I .now all you need to do is change the message after pressing “i” and then you need to close it by pressing esc key then “i” type “**:wq**” to exit the editor. Now you have successfully edited the commit.

**Git Tutorials #12**

Unstaging & Unmodifying Files In Git |

Hey everyone welcome, Today we are going to learn how to unstage files in Git. So let’s get started.

If we have few files in our git repository and we have staged them. Now for some reason we want to unstage a file. Now how to do that? It is super easy. Let’s see.

* To unstage a file use “**git restore --staged file.ext**”.

It will unstage the file and you can verify it by using “**git status**”.

* Now let’s say we have modified the unstaged file and made some changes which were not necessary. Now the program is not working. So we have to restore that file to its previous state where it was working. To do it use “
* **git checkout -- file.ext”.** Now git will restore that file to its last commit state.

To restore your entire working directory to the previous commit use

“**git checkout -f”.** It will restore your entire directory to the last commit