**GITHUB Introduction**

GitHub was created by Linus Trovalds – the same person who created Linux.

**“git” of GitHub:**

git is a version control system (VCS) for tracking changes in computer files and coordinating them with multiple people.

**“Hub” of GitHub:**

The Hub is where multiple developers / Programmers can store their projects.

**Networking and Credibility:**

The social networking aspect of GitHub is probably its most powerful feature, and is what allows projects to grow more than anything else. Each user on GitHub has their own profile, which can act like a resume of sorts, showing your past work and contributions to other projects via pull requests.

If you have identified any errors or modifications to streamline inefficiencies in any library source codes, the original authors are most likely to accept the changes and merge this into the original code files.

One can contribute to open source projects, to name a few - R Statistical library and Python, etc. If such mergers show up on one’s profile, it increases your credibility and authenticity of their knowledge.

**Repository:**

A repository is a location where all the files for a particular project are stored, usually abbreviated to “repo.”

**Forking a repo:**

“Forking” is when you create a new project based off of another project that already exists i.e. copying a different user’s project to one’s own profile. This is an amazing feature that vastly encourages the further development of programs and other projects. If you find a project on GitHub that you’d like to contribute to, you can fork the repo, make the changes you’d like, and release the revised project as a new repo.

**Branch:**

After one has forked the repository from the original author, it will be stored as the Master branch. So, if you are not sure of the changes you are doing, it is always advisable to create a branch, which is the exact copy of the Master and then do the changes to the Branch. This will help the user to compare the original author’s file to the edited document. In case the edits are not likely to succeed, you can always discard it with the Master as it is.

We can switch back and forth between the Master and the branch. Also, we can do changes and commits in both Master and the Branch.

The HEAD keeps shifting to the Master or the Branch wherever the latest commit is done.

If we create a branch, the HEAD still will be pointed towards the latest commit if in Master.

Note: It is always advisable not to make changes directly to the Master.

**Cloning:**

Once the original author’s repo is forked by one, they can perform a process called as Cloning.   
This will literally clone-copy the entire repository to the local system. Also, enables the user to use the files in his convenience offline.

**Commit / PUSH:**

This is the most important function in GitHub. When you commit, you are taking a “snapshot” of your repository at that point in time, giving us a checkpoint to which one can reevaluate or restore the project to any previous state.

**Pull requests:**

After you have done the required changes to the repository you have forked, you can deploy it on your data. If the results from the same or the new procedure seems better than the version given by the original author, you can request the author to do the specified changes through a “Pull request”.

There is an optional description text box while creating the Pull request, which is used to give any comments to the original author explaining the significance of the changes or any other comments.

**Review of code:**

Once the Pull request is initiated, both the initiator and the receiver can make comments on the edits done. So this will be useful before the author accepts any changes and are merged into the author’s Master files.

**Merge:**

If the author wants to incorporate the changes that has been made visible through the pull request, he can accept the changes and then merge it into his Master. Now, the entire team or public who has forked the author’s repo, can view the edits done.

**Readme:**

This is a file which allows the owner of the repo to give the initial instructions to the team members or the public. This might include what are the requirements that should be present in order to run the codes. Also, any other directions will be given here.

Note: The creation of this is by default asked to the user when a repository is being created.

**gitIgnore:**

This is a file in the repository named “.gitignore”. This is used to give directions to GitHub as to which files or directories has to be ignored, when a commit is made. This file can also be shared with the team or the public.

**License:**

For the repository to be truly be for public use and sharing, it has to have a correct license. Without a license, the default copyright laws will be applied, meaning the owner retains all rights to his source code and no one can reproduce, distributor create derivative works from the author’s works.

**Approaches:**

The above processes can be done in 2 ways:

1. GitHub Desktop client

This uses a Graphical User Interface which makes it very easy for non-programmers to use GitHub.

1. git Command shell

This uses a Linux like interface, which has shell commands to make all the functions work.

**References:**

1. GitHub official documentation / Guide: [https://guides.GitHub.com/activities/hello-world/](https://guides.github.com/activities/hello-world/)
2. [https://readwrite.com/2013/09/30/understanding-GitHub-a-journey-for-beginners-part-1/](https://readwrite.com/2013/09/30/understanding-github-a-journey-for-beginners-part-1/)
3. [http://rogerdudler.GitHub.io/git-guide/](http://rogerdudler.github.io/git-guide/)

**Additional Tutorials:**

1. GitHub for Noobs by DevTips, YouTube: <https://www.youtube.com/playlist?list=PLqGj3iMvMa4LFz8DZ0t-89twnelpT4Ilw>
2. [https://www.lynda.com/git-tutorials/git-Essential-Training](https://www.lynda.com/Git-tutorials/Git-Essential-Training) by Lynda.com (LinkedIn)

**GITHUB via COMMAND LINE**

**Command Line Basics**

“Disk / Drive letter:” : This will lead the user to the disk / drive he wants

dir: Lists down all the directories in the current location

cls : Clear screen the command prompt

. : This means all the files and folders in the current distination

.. : If this file is present in the current destination, that means that we have a parent folder above to the current destination

dir /p : This will enable the user to see only the starting page of the folders

Always the backward slash is used to separate the drives and folders and files

dir /b /p: This gives only the bare file or folder name

dir /o:s : Order by size

dir /? : Gives all the options that can be done with the dir command

dir path : gives the list of directories in the path mentioned

dir partial word\* : Gives the list of files and folders starting with partial word. \* is a wild card which can be substituted for anything

? : This is a wildcard which replaces a single character/digit

dir > Shiva.txt : This will store the output of the dir command in Shiva.txt. It will basically create a new file called Shiva.txt

cd : Redirects to your home directory

ls : Lists all the files in the directory

ls –a : Also includes all the hidden files along with the visible files in the directory..

cd “Your-Folder-Name”/

**Initializing git**

git init: This initializes a particular repository. Note: You need to be located in the particular location before running this command. A file named “.git” will be present in the location if that is a git repo.

A user can have any number of repositories required, like in the GUI

git log: Shows the user all the commits that were performed

git help: Provides a set of commands that can be used for learning / reference

git add . : Adds all the files in the working directory to the staging index

git add “Your-Folder-Name” : Adds only the particular folder to the staging index

git commit: Pushes the file/s in the staging index to the Repository.

This command will give

git status: Gives the status of the 3 divisions of the tree, namely Repository, Staging index, Working directory.

1. Tells you on which branch the user is at
2. Tells which files are on the staging index
3. Tells about the files which are not committed

pwd: Get your current working directory path. Once you come out of the path or shut down the git shell. It will help get the path again

git diff: This shows the difference between the original uncommitted files and the new changed files. Note: This command has to be executed before you transfer it to the staging index.

The unchanged part of the file will be present in white color,

The newly added part will be in green,

The deleted part will be in red.

git diff –staged: This gives the difference between the earlier staging index files and the new staging index files.

git diff --color-words “Your-File-Name” : This will also the difference of the original file and the changed file, but in a different format. For egs. this will not show the entire line, but only the changed word in the line in green, red terminologies

git rm “Your-File-Name”: Removes the file permanently from the working directory. It will also not be present in recycle bin / Trash.

Note: It is better to delete the files manually if you want to access it in case if needed later (from Recycle Bin)

Renaming of files:

git mv “Your-File-Old-Name” “Your-File-New-Name”

Moving of files:

git mv “Your-File-Name” “New-Folder-Name/Your-File-Name”

For git, moving and renaming are synonymous.

The user should commit after renaming or moving the files.

**git Commit**

Got commit –am “Message for commit”: This command will push everything from working directory to the staging index and then commit

git add “Your-directory-Name/\*” or git add “Your-Directory-Name” : Will add everything inside this directory into the staging index area

git checkout “Your-File-Name” : This will replace your file in the repository (latest commit) with the file in the working directory. So the changes in the working directory will be lost.

If there is a situation of the file and the branch name are same, git will take the branch and replace it in the working directory

git checkout – “Your-File-Name” : This will just replace the files in the current branch

git reset HEAD “Your-File-Name” : This command will remove your file from the staging index

git commit –amend –m “Your-Message” : Only the latest commit can be changed. This command will replace the previous commit which the user has done with the new commit. The HEAD will always be pointing at the latest commit.

We cannot change the older commits since the data integrity is affected.

git checkout “SHA number” – “Your-File-Name” : This will replace the particular file from the older commit version to the current working directory

git revert “SHA Number” : This reverts the last commit to the state it was before it. It flips all the actions done for the last commit. This can be done only for the last commit.

This takes care of the staging index and the committing step.

git reset –soft “SHA Number”:

This does not change the staging index or working directory

git reset –mixed “SHA Number”: This changes working directory to match repository.

Does not change the staging index. We need to restage them and recommit them

git reset –hard “SHA Number”:

Changes the staging index and working directory to match repository. When we do the hard reset to the 6th commit in a set of 10 commits, the 7th to 10 th commits info is lost. Now the head will b3e pointing to the 6th commit.

.git/refs/heads/master : This will give me the location of the commit where the head is present

working directory

git clean –n : This gives the information about which files and folders can be removed by using this command

git clean –f : This command runs the git clean and removes the the junk files which are not tracked i.e. not in our repository

This clean command touches only the files that are present in the working directory and not the staging index. Also, it does not remove the folders.

**git Ignore**

.gitignore files: These are the files which will not be tracked by git and cannot be tracked. We should not ignore the .gitignore file, and we need to commit it.

Note: Comments can be given starting with #

The below is the link to find all the gitignore files in each language:

[https://GitHub.com/GitHub/gitignore](https://github.com/github/gitignore)

Global ignored files: Some files can be always ignored by the user. So all the repositories will ignore these files. This is user specific rather than repository specific.

The user can keep this file in the location of the .gitconfig file

git config –global core.excludesfile ~/.”Users-Global-Ignoring-File-Names”

Note:

Ctrl + c is the command to stop the executions that is happening on the powershell

q is the command to stop in command line

**git Log**

git ls-tree HEAD : This command will give you the files / folders present in the repository

HEAD is a pointer/label to the most recent checkout of the branch you are currently on. master is the default branch created when you initialized a git repository (e.g. git init).

git ls-tree master^ : To reference the commit before master

git log –oneline : This will give the log of the commits done, each one in a line

git log –-oneline -3 : This will give latest 3 commits done

git log –since=2.weeks –until=3.days

git log –since=1.weeks

git log “Commit’s SHA#” –oneline

git log –stat : Gives the statistics in terms of what are the changes done during in the commits

git log –summary : Gives the summary of the commits

git log –format=oneline

git log –format=email : This gives the version of the commits which can be emailed

git log –format=short: This gives the short version of commits

git log –format: This is the default one which displays the medium format

git log –graph : This displays the commits in a graph format, this displays the branches and the merges

git show “SHA#” : Shows the the diff of the commit before and after the particular commit

If null is present in the git show output, then that means a new file was being created

**Comparing in GitHub**

git diff “SHA#” : Shows the details of all the additions and deletions done since the particular SHA# commit. In short it is, (Current state – After The SHA# commit state)

git diff “SHA#” “Your-File-Name” : This gives the changes done to a particular file since the commit SHA#

git diff “SHA#1”..”SHA#2” : This will tell you all the changes done between the SHA#1 commit and SHA#2 commit

git diff “SHAH#”..HEAD

git diff “SHAH#”..HEAD^^

The user can know where the HEAD is pointed to from the .git/HEAD folder

.git/refs/heads is a folder of all the branches that are present

cat git/refs/heads/master will display the commit at which our HEAD is present

This will also be the latest commit SHA#

**Branching in GitHub**

git branch “new-brach-name”: Creates a new branch

git checkout “Your-Branch-Name”: This will switch from the previous branh to the branch you mentioned

git commit –am”Your-Message” : This will make a commit directly from the working directory to the repository bypassing the staging index

git checkout –b “Yout-New-Branch-Name” : This will create a new branch and also check out in the same time

Note: git log will always give al the commits from the branch where in we run this command

git log -–graph –-oneline -–decorate –-all : gives the commits of all the branches present

Note: You can checkout only if you have your working directory clean. Since the uncommitted changes will be lost if I check out, it will abort the check out command

git diff master..”Branch-Name” : Gives the difference between the Master branch and the new Branch Name

git diff –color –words Master..”Branch-name” : This will give the difference in one line but with different colors

**Renaming and Deletion of Branches**

git branch –m “Old-Branch-Name” “New-Branch-Name” : This will rename the branch

git branch –d “Branch-To-Delete” : this will delete the particular branch

Note: You cannot delete the branch if the user is on it or if the user is sure to delete it, use

git branch –D “Branch-To-Delete” : This will force delete it and then move to the Master branch

**Merging of Branches**

Note: The user has to be present in the branch he wants to merge into (The receiver) before running the merge command.

git merge “Branch-Name” : This will merge the branch mentioned into the receiver

The user can delete the Branch after merging into the receiver.

Fast Forward merge: If the head is on the top of the Master and there are no other commits on it apart from a branch coming out from it. git will know that it is safe to merge it, by just pushing it to the branch as a new commit to the Master

git merge -–no--ff branch : Do not do a fast forward merge

git merge –-ff--only branch : Do only fast forward merge

Recursive strategy: It checks if the branch and head of the receiver branch and performs the merge

**Merging conflicts:**

If in the receiver branch, Line 3 of file A is different from the sender branch Line

For example, let's say that we've a “A” file and in the master branch we make a change to the top of file “A”. Then we have another branch that we are merging in and we made a change near the bottom of the “A” file. That's not a problem for git. git sees those, it recognizes the line numbers and realizes that those are not next to each other, and it takes both of those changes and incorporates them into one composite document.

If there is a difference between the same line numbers in both the receiver and the sender branch file, then the git will not know how to merge and it raises a conflict asking the user to solve it.

git merge –abort: This will abort the operation of merging when it throws up a conflict, since git will consider that you are still in the process of merging

The user can use a diff command for both the receiver and the sender files. This will help to resolve the conflicts.

**Best practices for Merging:**

1. Keep Lines short
2. Keep commits small
3. Do not do stray edits to whitespace (Spaces, tabs, line returns)
4. Merge often. If there is an accepted addition, one can go ahead and merge it, rather than waiting to do a lot of changes
5. Track changes to master. In the case of multiple branches, keep all the branches updated as per the master

**Stashing:**

This is the 4th area apart from Repository, Staging index and the Working directory.

git stash save “Message” : This will save the non-committed files to the stash

We can include the untracked files too by giving an extra command of –include untracked

git stash show “Stash#” : This shows the stash and what is present

git stash apply : This will leave a copy of the stash and also gets whatever present in the stash to the working repository

git stash pop : This will remove the items in the stash and also gets whatever present in the stash to the working repository

git stash pop”Stash#”

git stash list: Will give the list of all the items in the stash

git stash drop “Stash#” : Removes the stash with that particular Stash# from the stash

git stash clear: This will drop all the items in the stash. Note: Check before you clear since it can be destructive

**Remotes and Cloud:**

All the changes can be synced to the remote server or cloud.