**Git vs GitHub**

| **S.No.** | **Git** | **GitHub** |
| --- | --- | --- |
| 1. | Git is a software. | GitHub is a service. |
| 2. | Git is a command-line tool | GitHub is a graphical user interface |
| 3. | Git is installed locally on the system | GitHub is hosted on the web |
| 4. | Git is maintained by linux. | GitHub is maintained by microsoft. |
| 5. | Git is focused on version control and code sharing. | GitHub is focused on centralized source code hosting. |
| 6. | Git is a version control system to manage source code history. | GitHub is a hosting service for Git repositories. |
| 7. | Git was first released in 2005. | GitHub was launched in 2008. |
| 8. | Git has no user management feature. | GitHub has built-in user management feature. |

**Configuring user name and email:**

* git config --global user.name "user name" --- configuring the user name
* git config user.name --- to check if user is configured
* git config user.email --- to if email configured
* git config --global user.email --- to configure email

**Repository:**

A git repo is workspace which tracks and manages files.

We can have any number of repo in our machines as needed, all with seperate histories and contents

**Initializing the repository**

git init

Create a new git repo in the path where we ran this command.

**Keeping git commit atomic**

We should try to make our git commit atomic. It means a commit should encompass a single feature, change or fix. In other words, try to keep each commit focused on a single thing.

**# .git hidden folder#**

It is hidden directory which contains all the logs about the repository.

if we delete this hidden directory then it will delete all logs

Note:

Don't initialize any folder or repo inside another repo, i.e don't know nesting of git init

**#git add#**

The git add command is used to add file contents to the Index (Staging Area). This command updates the current content of the working tree to the staging area.

Files which are not added are called untracked file.

git add -A or git add . ----- to add all files from repository into stating/index are

git reset <filename> ------- to undo the git add

git add <filename(s)> ----- to add file into staging area  
It add(git add .) all untracked file to staging area or repo doesn’t matter from which branch it ran.

**git status:**

It is used to check for untracked/stages files. When we run(from any branch), it shows all untracked/staged files in repo.

**#git commit#**

this basically updates .git folder, add all message or info.

git commit -m "my message"

git commit -amend ---- to Change commit message. if ran then it prompt for entering message

Git Reset ---- Git undo commit

Git Revert -- git revert commit

Git Rm ---- git remove commit

Working Area ----(git add)-----> Staging area----(git commit)-->repository

**Git Ignore**

This is used to tell git to INGORE/untrack some file from repo.

Create a file by name '.gitignore' and add all files/folder that you want to untract/ignore in '.gitignore'.

'.gitignore' should be created in project/repo root directed.

**Main branch vs Master branch**

In 2020, GitHub renamed the default branch from master to main. The default git brac=nch name is still master, though the git team is exploring a potential change.

**What is HEAD in any branch name?**

It refers to the branch name and commit point where you are currently.

**Operations on Branches**

We can perform various operations on Git branches. The git branch command allows you to create, list, rename and delete branches

**list all branch**

We can run below command to list out all branch from current repo.

**git branch** **/ git branch --list**

------ list all branch name. where ever it shows \*, that refers the branch you are.

$ git branch

\* master -- \* indicates that we are on master branch and there are two master and oldies branch

oldies

**Create new branch**

We can use below command to create new branch.

git branch <branch name> ---- to create new branch by <branch name>

rakes@LAPTOP-IK629TG3 MINGW64 /d/gitlearn (master)

$ git branch ---- only master branch here

\* master

rakes@LAPTOP-IK629TG3 MINGW64 /d/gitlearn (master)

$ git branch oldies --- created oldies branch

rakes@LAPTOP-IK629TG3 MINGW64 /d/gitlearn (master)

$ *git branch -------- now showing two branch*

*\* master*

*oldies*

**Delete branch**

We can use below command to delete branch

git branch -d<branch name> --- to delete branch <branch name>

rakes@LAPTOP-IK629TG3 MINGW64 /d/gitlearn (master)

$ git branch --- two branch here

\* master

oldies

rakes@LAPTOP-IK629TG3 MINGW64 /d/gitlearn (master)

$ git branch -d oldies

Deleted branch oldies (was 81496ed).

rakes@LAPTOP-IK629TG3 MINGW64 /d/gitlearn (master)

$ git branch --- now only master branch b/c deleted oldies branch above

\* master

rakes@LAPTOP-IK629TG3 MINGW64 /d/gitlearn (master)

**Delete a Remote Branch**

We can delete a remote branch from Git desktop application. Below command is used to delete a remote branch:

git push origin -delete <branch name>

**Switching to any branch**

We can use switch/checkout command to switch the branches.

git branch <branch name> --- switch to branch name

git checkout <branch name> --- switch to checkout branch

Note:

Checkout is old command for switching newer is switch

**Rename branch**

We can use below command to rename branch

git branch -m <old branch name> <new branch name> --- rename old branch name to new branch name

$ git branch

master

\* oldies

rakes@LAPTOP-IK629TG3 MINGW64 /d/gitlearn (oldies)

$ git branch -m oldies retro --- renaming oldies to retro

rakes@LAPTOP-IK629TG3 MINGW64 /d/gitlearn (retro)

$ git branch

master

\* retro

**Merge Branch**

Git allows you to merge the other branch with the currently active branch. You can merge two branches with the help of git merge command.

git merge <branch name>

**Git diff**

It's a multiuse Git command. When it is executed, it runs a diff function on Git data sources.

It list all the changes in our working directory that are NOT staged for the next commit.

Example –

Scenerio1: Track the changes that have not been staged. ---- git diff

Scenerio2: Track the changes that have staged but not committed --- git diff --staged

Scenerio3: Track all the changes stagged/unstagged in a file --- git diff HEAD

Scererio4: Trach changes b/w two commits --- git diff <commit1-sha> <commit2-sha>

Scenerio5:Comparing branched --- git diff branch1..branch2

Scenerios6: Diff-ing specific file --- git diff –staged filename (diffing stagged <filename>)

git diff ----- compared Staging area and working directory