What is Git

*G - Global I - Information T - Tracker*

* It is an open- source VCS (Version control system).For tracking changes and storing a set of codes in the cloud.
* So, a Git based project data is stored in a cloud repository (A repository is a folder that includes a set of codes) hosted on a platform like Hub . (GitHub)
* It is more flexible - users can clone the main repository into their local workspace.

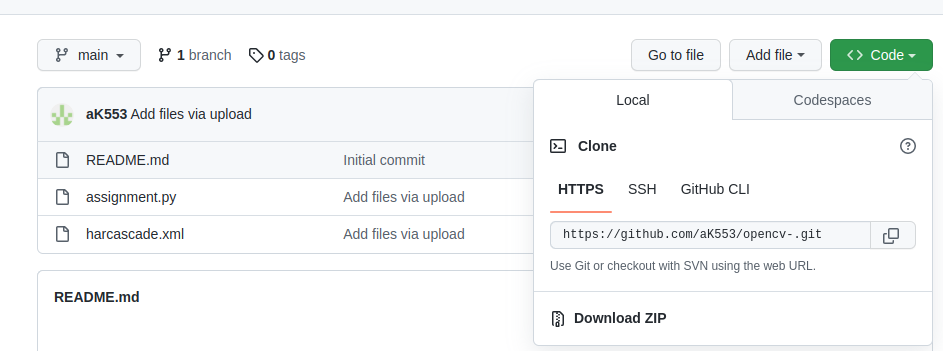
Lets see some commands used in Git

**Clone**

This command used to clone the remote repo metadata to local work machine

Command : git clone<url> // copy the url link on the github

git clone - fetch code from remote repository (in GitHub)



**Git add**

**There are four stages in GitHub workflow**

1. **Working area >> 2 .staging area >> 3. local area >> 4. Report repository**

The **git add** command adds new or modified files from the working directory to the staging area, also called **index**. First step is committing a change to a Git repository by running the **git add** command to select the files you want to commit.

*Command :* **git add.-** // add all files in the repo to the staging area

**git add\*.<file type>** // add set of files with same types example (.txt, .py , .cpp)

**git add<filename>** // add a specific file

**git add<path>** // add a specific directory of file

**Git commit**

It records the changes made to the file in repository , it submits commit message

Command :

**git commit -m “message”** - it creates a commit with an inline message as an command message

**git commit -a**  *//* creates commit includes all changes in a workspace directory

**git commit -am “message”** - // combines the function which will create a commit of all stages changes and includes an inline commit message .

**git commit –amend -** allows modification to the last commit , like last commit message without changing the snapshot

**Branch**

In Git repo there will be a main and default branch, if we create a new branch means creating a new independent line , this branch will mirror the main branch ( its a copy of main branch)

*Note : If main branch contents are changed,the copy of that branch will also change but if the contents of new branch is changed, no changes will be appear in the main branch*

*Command :* **git branch\_ branch name**

**git branch -a //** shows all available branch

**git switch -c <branch name> //**create new branch

**git branch //** will give the available branches

**git branch -d //** combines two branch

**git branch --show-current //** this command shows the current branch.

**git push**

**This command is used to push the field from terminal to remote area**

**Steps**

Edit the file in gvim or vim

**git add** // to add the command in the staging area

**git commit -m “command” file name** // staging to local repository

**git push -u origin master** // local repository to remote repository

**git remote -v //** before push check the command line is pointing to the correct address , for that this command is used

**git push <remote name> <branch name> //** this command send local commits to the remote repository

Example :

**git push origin main**

**Git pull**

Steps

**git fetch** // fetches the data form the remote repository

**git pull** // pulls the changes made to the remote repository to your local work machine

**git pull <remote repository name> //c**ommand retrieves and downloads the content of a remote repository to your local machine.

**git checkout <branch name> //** If you want to pull code from just one specific branch

**git pull <remote name>**

**Git fetch**

**Command : git fetch**

This command is used to collect the files from remote repositories, the main difference between the clone and fetch is . clone only copies the files available in the remote area but fetch will collect all the files which includes untracked file in the staging area ( untrackets files - files which are not added to the staging area )

Example

If you Want to work with code on a branch named test.cpp, and the remote repository name is the main code. The command will be as follows:

**git checkout test.cpp**

**git pull main code**

**Check out**

It's like a change we are doing in a branch , this command will save and checkout, to switch to other commands, and can be used to view old commits.

Note : git clone and git branch will be confusing

Difference is

git checkout - used to switch between branches in local repository (in workspace)

*Command :* g**it checkout < branch name > *//*** *switch to other branch in local repo*

**git checkout -b<branch name>** // *to create new branch in local repo*

**git checkout -b test.cpp**  // thai command checks out and allows to switch it immediately

If we need to switch between remote branch in terminal (work space )

*Commands :*

**git fetch --all (or) git fetch --a** *// In order to checkout a remote branch you have to first fetch the contents of the branch.*

**git checkout < remote branch name>** *// then checkout the remote branch like a local branch.*

Additionally you can checkout a new local branch and reset it to the remote branch last commit.

**git checkout -b ＜branchname＞**

**git reset --hard origin/＜branchname＞**

**Git merge**

It’s nothing but **combining** a two independent branch in a repo

For example : take a scenario Branch 1 have main source code (main branch)

Branch 2 have sub codes ( target or staging branch

The missing codes in branch 1 will be updated with history.

*Note* : once the target branch is merged with the main branch the content of the main branch will only reflect in the main branch , there will be no changes done in the target branch.

*Command* : **git merge <main branch name> <target branch name>**

**git init**

The first step in creating a new repository is running the **git init** command

*Command :*

**git init** //It sets up a new directory in a **.git** folder in your working tree, it contains Git metadata .

**git init <folder>** // this command specifies the directory you want to initialize a new repository

Git configuration setup

git config --global user.name

Git config --global user.email

**Git status**

shows the current state of the working directory and the staging area.

command : **git status**

This command checks which local changes have committed , if there is no changes to commit it indicates there is nothing to commit and up to date

**Git rm –cached**

It removes a file from the index, but it will remain intact in the working directory.

There are two ways to remove

1. Remove specific files only
2. clear an entire directory.

Command : **git rm --cached <file name with extension>** // this command will remove a cache in specific file

**Git rm -r --cached <directory name>** // entire directory from index

**Gitignore**

It is a text file that specifies which files and folders Git should ignore in your working tree. This command will remove files which you don’t want to include in a remote repository but that files will be in local repositories.

So there are many files types to be ignored

* Hidden system files - DS\_Store and Thumbs.db
* Files generated during runtime log and .temp

To create a .gitignore file,

Create a normal txt file and name it as .gitignore and add the target files as need

Command :

**File.txt** // ignore a specific file name (file.txt)

**directory/** // ignore entire directory by addin (/)

**\*.html** // it igoners a set of same extension files

**\*~**  // it ignores files ending with “~”

**Git log**

commit history of a repository

Commands : **git log -n 5** // -n flag helps you see a number of your most recent logs.

return a list of the five most recent commits made to a repository

**git log --author="ashok.k"** // return a list of commits that were changed by Ashok.k.

**git log --before="yesterday"** // command will retrieve a list of commits from the previous day.

**git log -- index.py** // this command will list a specific files you want

**git log -S"# Hello"** // this command has s flag which enables to search all commit that includes particular line

**Git remote**

**git remote** // it shows the remote configuration ,list of remote connection that are in repositorie**s**

**git remote add <name> <url> //** to add a new remote repository connection

For example after adding the url

**git remote add new-remote https:/github.com/user/new-remote.git**

**git remote rename <old-name> <new-name> //** this command used to rename the remote connection

**git remote rm <name> //** it i used to delete the remote connection

# **Forks**

A fork is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository. You can fetch updates from or submit changes to the original repository with pull requests.

Forking a repository is similar to copying a repository, with two major differences:

* You can use a pull request to suggest changes from your user-owned fork to the original repository in its GitHub instance, also known as the *upstream* repository.
* You can bring changes from the upstream repository to your local fork by synchronizing your fork with the upstream repository.

***Roadmap for Git workflow in relation to a remote repository***