**GITHUB**

GitHub is **a code hosting platform for version control and collaboration**. It lets you and others work together on projects from anywhere. This tutorial teaches you GitHub essentials like repositories, branches, commits, and pull requests.

Git is a distributed revision control and source code management system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development. Git is a free software distributed under the terms of the GNU General Public License version 2. This tutorial explains how to use Git for project version control in a distributed environment while working on web-based and non web-based applications development.

**What are GitHub’s Features**

**1. Easy Project Management**

GitHub is a place where project managers and developers come together to coordinate, track, and update their work so that projects are transparent and stay on schedule.

**2. Increased Safety With Packages**

Packages can be published privately, within the team, or publicly to the open-source community. The packages can be used or reused by downloading them from GitHub.

**3. Effective Team Management**

GitHub helps all the team members stay on the same page and organized. Moderation tools like Issue and Pull Request Locking help the team to focus on the code.

**4. Improved Code Writing**

[Pull requests](https://www.simplilearn.com/tutorials/git-tutorial/git-pull-request" \t "_blank" \o "Pull requests) help the organizations to review, develop, and propose new code. Team members can discuss any implementations and proposals through these before changing the source code.

**5. Increased Code Safety**

GitHub uses dedicated tools to identify and analyze vulnerabilities to the code that other tools tend to miss. Development teams everywhere work together to secure the software supply chain, from start to finish.

**6. Easy Code Hosting**

All the code and documentation are in one place. There are millions of repositories on GitHub, and each repository has its own tools to help you host and release code.

**Github life cycle:**

In this chapter, we will discuss the life cycle of Git. In later chapters, we will cover the Git commands for each operation.

General workflow is as follows −

* You clone the Git repository as a working copy.
* You modify the working copy by adding/editing files.
* If necessary, you also update the working copy by taking other developer's changes.
* You review the changes before commit.
* You commit changes. If everything is fine, then you push the changes to the repository.
* After committing, if you realize something is wrong, then you correct the last commit and push the changes to the repository.

**Flowchart:**



**List of useful Github Commands**

Github is a distributed version control system which helps to manage the repositories.

These are a list of few commands that you can use frequently on github(git bash) 

1.git help

Take help from github help section for different commands and other errors 

2.git config

To set the basic configurations on github like your name and email. 

3.git config –-global user.name “Ashish Madaan”

Sets configuration values for your user name on git. 

4.git config –-global user.email ashishmadaan6@gmail.com

Sets configuration values for your user email on git. 

5.git config –-global color.ui true

To see different colours on command line for different outputs. 

6.mkdir store

Create a directory if not created initially. 

7.cd store

To go inside the directory and work upon its contents. 

8.git init

To create a local git repository for us in our store folder.This will help to manage the git commands for that particular repository. 

9.git status

To see whats changed since last commit.It shows all the files that have been added and modified and ready to be commmitted and files which are untracked 

10.git add Readme.txt

To add a file Readme.txt to the staging area to track its changes. 

11.git commit -m “Created a Readme.txt”

To commit our changes(taking a snapshot) and providing a message to remember for future reference. 

12.git log

To check the history of commits for our reference.

**Different ways to use add command:**

13.git add

To add a specific list of files to staging area. 

14.git add --all

To add all files of current directory to staging area. 

15.git add \*.txt

To add all text files of the current directory to staging area. 

16.git add docs/\*.txt

To add all text files of a particular directory(docs) to staging area. 

17.git add docs/

To add all files in a particular directory(docs) to staging area. 

18.git add “\*.txt”

To add text files of entire project to staging area.   
**More Commands:** 

19.git diff

To figure out what changes you made since last commit. 

20.git reset head license

To undo staging of the file that was added in the staging area. 

21.git checkout –license

To Blow away all changes since the last commit of the file. 

22.git commit -a -m “Readme.md”

To add any of our tracked files to staging area and commit them by providing a message to remember. 

23.git reset –soft HEAD^

To undo last commit and bring file to staging area. 

24.git reset –hard HEAD^

To undo last commit and remove file from the staging area as well(In case we went horribly wrong). 

25.git reset –hard HEAD^^

To undo last 2 commits and all changes. 

26.git remote add origin https://github.com/madaan123/MyAlgorithms.git

This commands make a bookmark which signifies that this particular remote refers to this URL.   
This remote will be used to pull any content from the directory and push our local content to the global server. 

27.git remote add <address>

To add new remotes to our local repository for a particular git address. 

28.git remove rm

To remove a remote from our local repository. 

29.git push -u origin master

To push all the contents of our local repository that belong to master branch to the server(Global repository). 

30.git clone https://github.com/bejjala/MyAlgorithms.git

To clone or make a local copy of the global repository in your system   
(git clone command downloads the repository and creates a remote named as origin which can be checked by command – git remote -v). 

31.git branch Testing

To create a new branch named as Testing. 

32.git branch

To see all the branches present and current branch that we are working on. 

33.git checkout Testing

To switch to branch Testing from master branch. 

34.ls

To see directories and files in the current directory. 

35.git merge Testing

To merge Testing branch with master branch. 

36.git branch -d Testing

To delete Testing branch. 

37.git checkout -b admin

To create a new branch admin and set it as current branch. 

38.git branch -r

To look at all the remote branches. 

39.git branch -D Testing

To forcefully delete a branch without making commmits. 

40.git tag

To see the list of available tags. 

41.git checkout v0.0.1

To set the current tag to v0.0.1. 

42.git tag -a v0.0.3 -m “version 0.0.3”

To create a new tag. 

43.git push –tags

To push the tags to remote repository. 

44.git fetch

To fetch down any changes from global repository to current repository 

45.git rebase

Three tasks are performed by git rebase 

1. Move all changes to master which are not in origin/master to a temporary area.
2. Run all origin master commits.
3. Run all commits in the temporary area on top of our master one at a time, so it avoids merge commits.