**Basic Git Commands**

**Git clone command**

This command is used to make a copy of a repository from an existing URL. If I want a local copy of my repository from GitHub, this command allows creating a local copy of that repository on your local directory from the repository URL.

**Syntax**

$ git clone URL

**Git add command**

This command is used to add one or more files to staging (Index) area.

**Syntax**

To add one file

$ git add Filename

To add more than one file

$ git add\*

**Git commit command**

Commit command is used in two scenarios. They are as follows.

Git commit -m

This command changes the head. It records or snapshots the file permanently in the version history with a message.

**Syntax**

$ git commit -m " Commit Message"

Git commit -a

This command commits any files added in the repository with git add and also commits any files you've changed since then.

**Syntax**

$ git commit -a

**Git status command**

The status command is used to display the state of the working directory and the staging area. It allows you to see which changes have been staged, which haven't, and which files aren?t being tracked by Git. It does not show you any information about the committed project history. For this, you need to use the git log. It also lists the files that you've changed and those you still need to add or commit.

**Syntax**

$ git status

**Git push Command**

It is used to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repo. It's the complement to git fetch, but whereas fetching imports commits to local branches on comparatively pushing exports commits to remote branches. Remote branches are configured by using the git remote command. Pushing is capable of overwriting changes, and caution should be taken when pushing.

Git push command can be used as follows.

Git push origin master

This command sends the changes made on the master branch, to your remote repository.

**Syntax**

$ git push [variable name] master

**Git push -all**

This command pushes all the branches to the server repository.

**Syntax**

$ git push --all

**Git pull command**

Pull command is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory.

**Syntax**

$ git pull URL

**Git Branch Command**

This command lists all the branches available in the repository.

**Syntax**

$ git branch

**Git Merge Command**

This command is used to merge the specified branch?s history into the current branch.

**Syntax**

$ git merge BranchName

**Git log Command**

This command is used to check the commit history.

**Syntax**

$ git log

By default, if no argument passed, Git log shows the most recent commits first. We can limit the number of log entries displayed by passing a number as an option, such as -3 to show only the last three entries.

$ git log -3

**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. It also prepares the staged content for the next commit. Every time we add or update any file in our project, it is required to forward updates to the staging area.

The git add command is a core part of Git technology. It typically adds one file at a time, but there some options are available that can add more than one file at once.

The "index" contains a snapshot of the working tree data. This snapshot will be forwarded for the next commit.

The git add command can be run many times before making a commit. These all add operations can be put under one commit. The add command adds the files that are specified on command line.

Git add command is a straight forward command. It adds files to the staging area. We can add single or multiple files at once in the staging area. It will be run as:

$ git add <File name>

The above command is added to the git staging area, but yet it cannot be shared on the version control system. A commit operation is needed to share it.

Git Add All

We can add more than one files in Git, but we have to run the add command repeatedly. Git facilitates us with a unique option of the add command by which we can add all the available files at once. To add all the files from the repository, run the add command with -A option. We can use '.' Instead of -A option. This command will stage all the files at a time. It will run as follows:

$ git add -A Or $ git add.

Add all New and Updated Files Only:

Git allows us to stage only updated and newly created files at once. We will use the ignore removal option to do so. It will be used as follows:

$ git add --ignore-removal

Add all Modified and Deleted Files

Git add facilitates us with a variety of options. There is another option that is available in Git, which allows us to stage only the modified and deleted files. It will not stage the newly created file. To stage all modified and deleted files only, run the below command:

$ git add -u

Add Files by Wildcard

Git allows us to add all the same pattern files at once. It is another way to add multiple files together. Suppose I want to add all java files or text files, then we can use pattern .java or .txt. To do so, we will run the command as follows, The below command will stage all the Java files

$ git add \*.java

**Git Commit**

It is used to record the changes in the repository. It is the next command after the git add. Every commit contains the index data and the commit message. Every commit forms a parent-child relationship. When we add a file in Git, it will take place in the staging area. A commit command is used to fetch updates from the staging area to the repository.

The staging and committing are co-related to each other. Staging allows us to continue in making changes to the repository, and when we want to share these changes to the version control system, committing allows us to record these changes.

Commits are the snapshots of the project. Every commit is recorded in the master branch of the repository. We can recall the commits or revert it to the older version. Two different commits will never overwrite because each commit has its own commit-id. This commit-id is a cryptographic number created by SHA (Secure Hash Algorithm) algorithm

The commit command will commit the changes and generate a commit-id. The commit command without any argument will open the default text editor and ask for the commit message. We can specify our commit message in this text editor. It will run as follows:

$ git commit

The above command will prompt a default editor and ask for a commit message.

Git commit -a

The commit command also provides -a option to specify some commits. It is used to commit the snapshots of all changes. This option only considers already added files in Git. It will not commit the newly created files. Consider below scenario:

We have made some updates to our already staged file newfile3 and create a file newfile4.txt. Check the status of the repository and run the commit command as follows:

$ git commit -a

Git commit -m

The -m option of commit command lets you to write the commit message on the command line. This command will not prompt the text editor. It will run as follows:

$ git commit -m "Commit message."

Git Commit Amend (Change commit message)

The amend option lets us to edit the last commit. If accidentally, we have committed a wrong commit message, then this feature is a savage option for us. It will run as follows:

$ git commit -amend

**Git Clone**

In Git, cloning is the act of making a copy of any target repository. The target repository can be remote or local. You can clone your repository from the remote repository to create a local copy on your system. Also, you can sync between the two locations.

The git clone is a command-line utility which is used to make a local copy of a remote repository. It accesses the repository through a remote URL.

Usually, the original repository is located on a remote server, often from a Git service like GitHub, Bitbucket, or GitLab. The remote repository URL is referred to the origin.

$ git clone <repository URL>

**Git Clone Branch**

Git allows making a copy of only a particular branch from a repository. You can make a directory for the individual branch by using the git clone command. To make a clone branch, you need to specify the branch name with -b command. Below is the syntax of the command to clone the specific git branch:

$ git clone -b <Branch name><Repository URL>

**Git Fork**

A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project. One of the excessive use of forking is to propose changes for bug fixing. To resolve an issue for a bug that you found, you can:

* Fork the repository.
* Make the fix.
* Forward a pull request to the project owner.

Forking is not a Git function; it is a feature of Git service like GitHub.

Generally, forking a repository allows us to experiment on the project without affecting the original project. Following are the reasons for forking the repository:

**How to Fork a Repository on GitHub?**

The forking and branching are excellent ways to contribute to an open-source project. These two features of Git allow the enhanced collaboration on the projects.

Forking is a safe way to contribute. It allows us to make a rough copy of the project. We can freely experiment on the project. After the final version of the project, we can create a pull request for merging.

It is a straight-forward process. Steps for forking the repository are as follows:

* Login to the GitHub account.
* Find the GitHub repository which you want to fork.
* Click the Fork button on the upper right side of the repository's page.

We can't fork our own repository. Only shared repositories can be fork. If someone wants to fork the repository, then he must log in with his account.