What is Git ?

Performance Security Flexibility Version control with Git

By far, the most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments). Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.

In addition to being distributed, Git has been designed with performance, security and flexibility in mind..

What is version control?

How version control helps high performing development and DevOps teams prosper Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

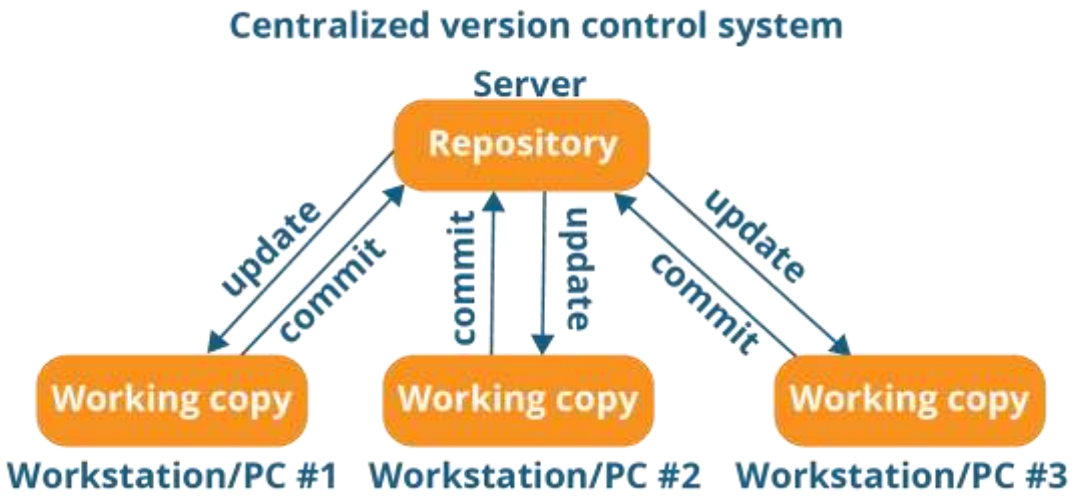
There are two types of VCS:

Centralized Version Control System (CVCS)

Distributed Version Control System (DVCS)

Centralized VCS

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server



The repository in the above diagram indicates a central server that could be local or remote which is directly connected to each of the programmer's workstation.

Every programmer can extract or update their workstations with the data present in the repository or can make changes to the data or commit in the repository. Every operation is performed directly on the repository.

Even though it seems pretty convenient to maintain a single repository, it has some major drawbacks. Some of them are:

It is not locally available; meaning you always need to be connected to a network to perform any action.

Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.

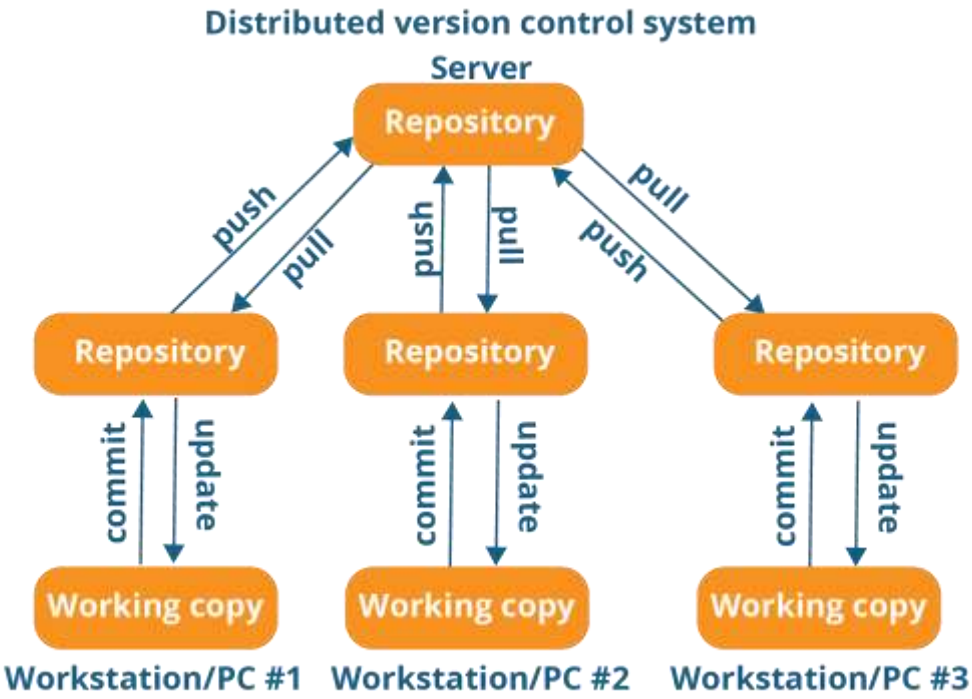
This is when Distributed VCS comes to the rescue.

Distributed VCS :

These systems do not necessarily rely on a central server to store all the versions of a project file.

In Distributed VCS, every contributor has a local copy or "clone" of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.

You will understand it better by referring to the diagram below:



As you can see in the above diagram, every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference.

They can update their local repositories with new data from the central server by an operation called "pull" and affect changes to the main repository by an operation called "push" from their local repository.

The act of cloning an entire repository into your workstation to get a local repository gives you the following advantages:

Install Git on Windows

Git for Windows stand-alone installer

Download the latest Git for Windows installer.

When you've successfully started the installer, you should see the Git Setup wizard screen. Follow the Next and Finish prompts to complete the installation. The default options are pretty sensible for most users.

Open a Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt).

Run the following commands to configure your Git username and email using the following commands, replacing Emma's name with your own. These details will be associated with any commits that you create:

```
$ git config --global user.name "vinay_kumbhar"
```

```
$ git config --global user.email "veenay.kumbhar@gmail.com"
```

Optional: Install the Git credential helper on Windows

Bitbucket supports pushing and pulling over HTTP to your remote Git repositories on Bitbucket. Every time you interact with the remote repository, you must supply a username/password combination. You can store these credentials, instead of supplying the combination every time, with the Git Credential Manager for Windows.

Command :

git init:-

for initialization of git folder

git status :-

current branch and changed files and updated files and new files

git add:-

1. to add only one file provide specific path

2. to add all changes for commit provide (.)

git add abc.txt (for adding a single file)

git add . (to add all changes for commit)

git reset:-

1. to remove only one file provide specific path

2. to remove all files form commit provide (.)

git reset abc.txt

git reset .

git commit -m :-

for commit message it mandatory for commit to provide message

git push :-

push changes to specific branch;

git push origin (branchname) for push the branch and changes

git pull :-

pull branch form origin

git checkout :-

1. to checkout only one file provide specific path even befor add

2. to revert all changes from all file provide (.)

git checkout abc.txt

git checkout .

git checkout -b (branchname) to create new branch

git checkout (branchname) to change branch

git stash :-

if their is situation you don't want to commit the changes and change the branch

it add files into git stage

git stash apply :-

to get back the stash files.

git log :-

commit log of the branch(give hte log file commit files time and date)

git branch rm (branchname) to remove the branch

git branch :-

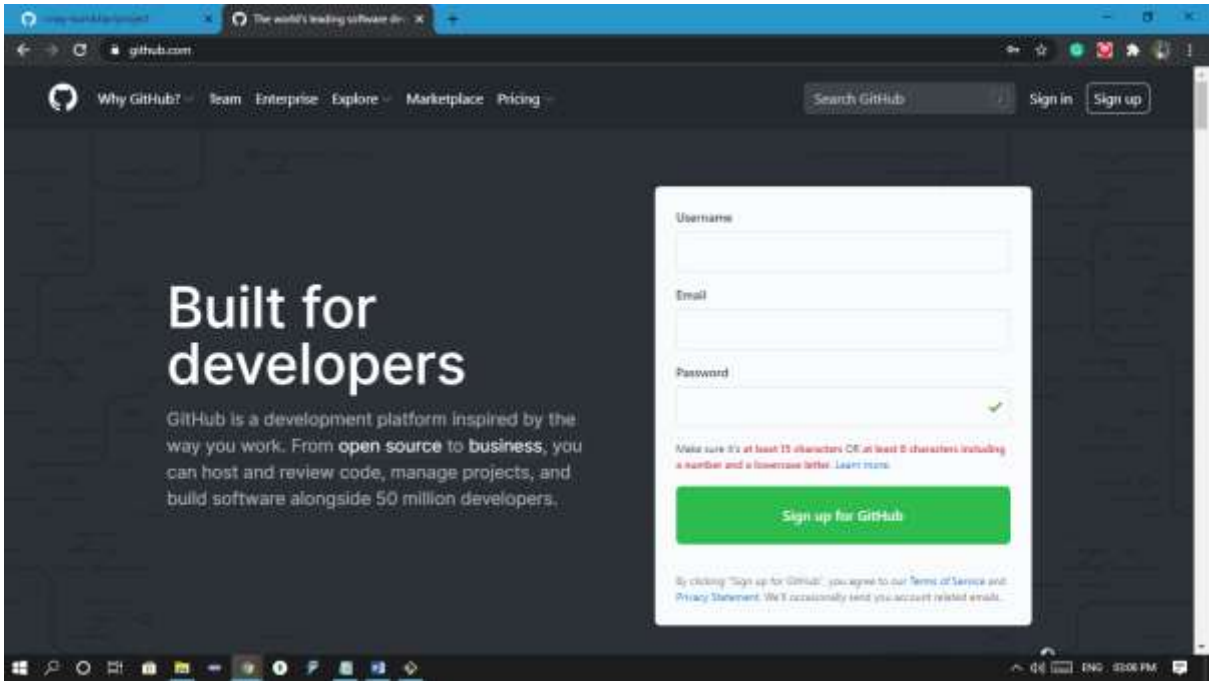
to see the list of branches in the git

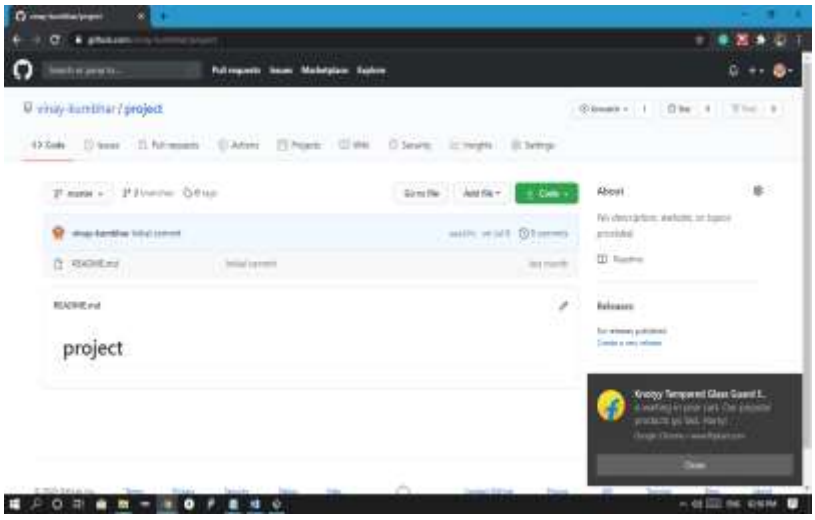
git merge <barnchName>

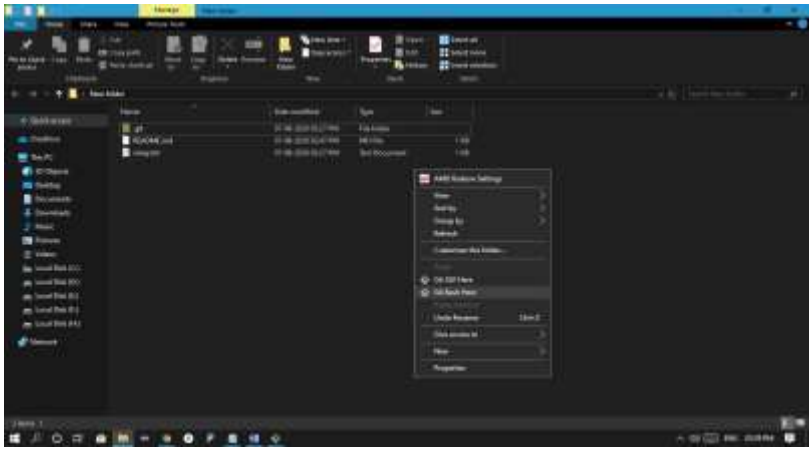
rm -f .git/index.lock :- to unlock the loked the index locked files

make marge request use the new pull request using github web site

CREATE ACCOUNT ON GITHUB:

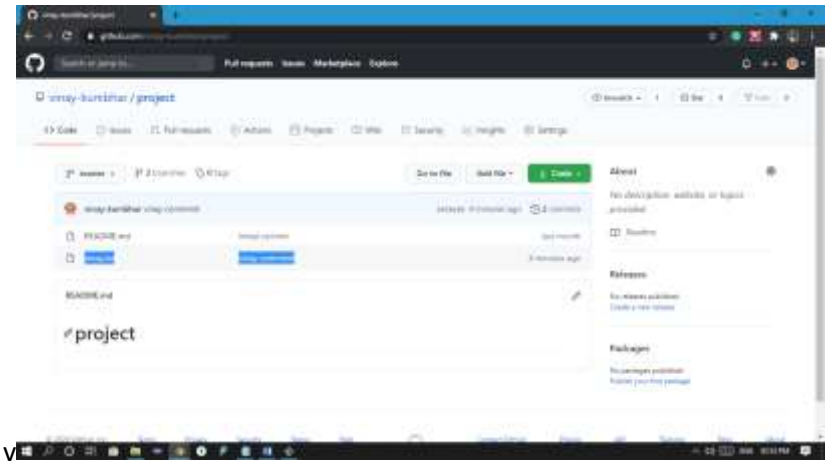






```
Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git init
Initialized empty Git repository in C:/Users/Vinay/Desktop/New folder/.git/

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git init
Reinitialized existing Git repository in C:/Users/Vinay/Desktop/New folder/.git/

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git remote add origin "https://github.com/vinay-kumbhar/project.git"

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git pull origin master
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/vinay-kumbhar/project
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git add vinay.txt

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   vinay.txt


Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git commit -m "vinay commmit"
[master b426e50] vinay commmit
 1 file changed, 1 insertion(+)
 create mode 100644 vinay.txt

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git status
On branch master
nothing to commit, working tree clean

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 283 bytes | 56.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/vinay-kumbhar/project.git
   eee23fc..b426e50  master -> master
```

```
Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git fetch
From https://github.com/vinay-kumbhar/project
 * [new branch]      vk         -> origin/vk
```

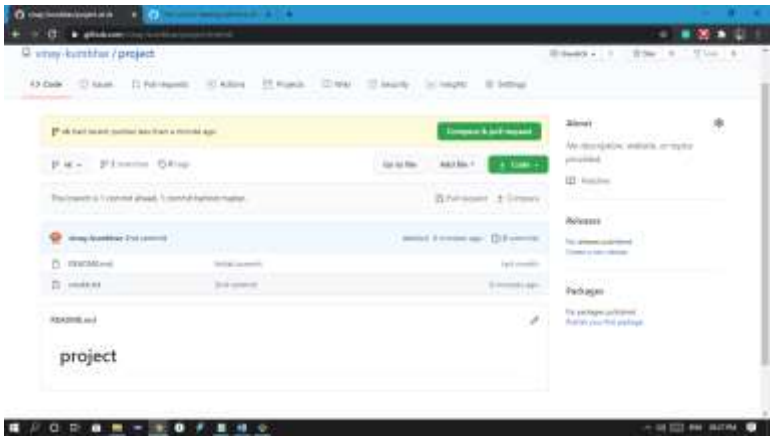


V

Commit data on other branch:

```
Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git checkout vk
Switched to a new branch 'vk'
Branch 'vk' set up to track remote branch 'vk' from 'origin'.

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git status
On branch vk
Your branch is up to date with 'origin/vk'.

nothing to commit, working tree clean

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git pull origin vk
From https://github.com/vinay-kumbhar/project
 * branch            vk         -> FETCH_HEAD
Already up to date.

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git add onvkb
fatal: pathspec 'onvkb' did not match any files

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git add onvkb.txt

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git status
On branch vk
Your branch is up to date with 'origin/vk'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   onvkb.txt


Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git commit -m "2nd commit"
[vk 00b89e9] 2nd commit
 1 file changed, 1 insertion(+)
 create mode 100644 onvkb.txt
```

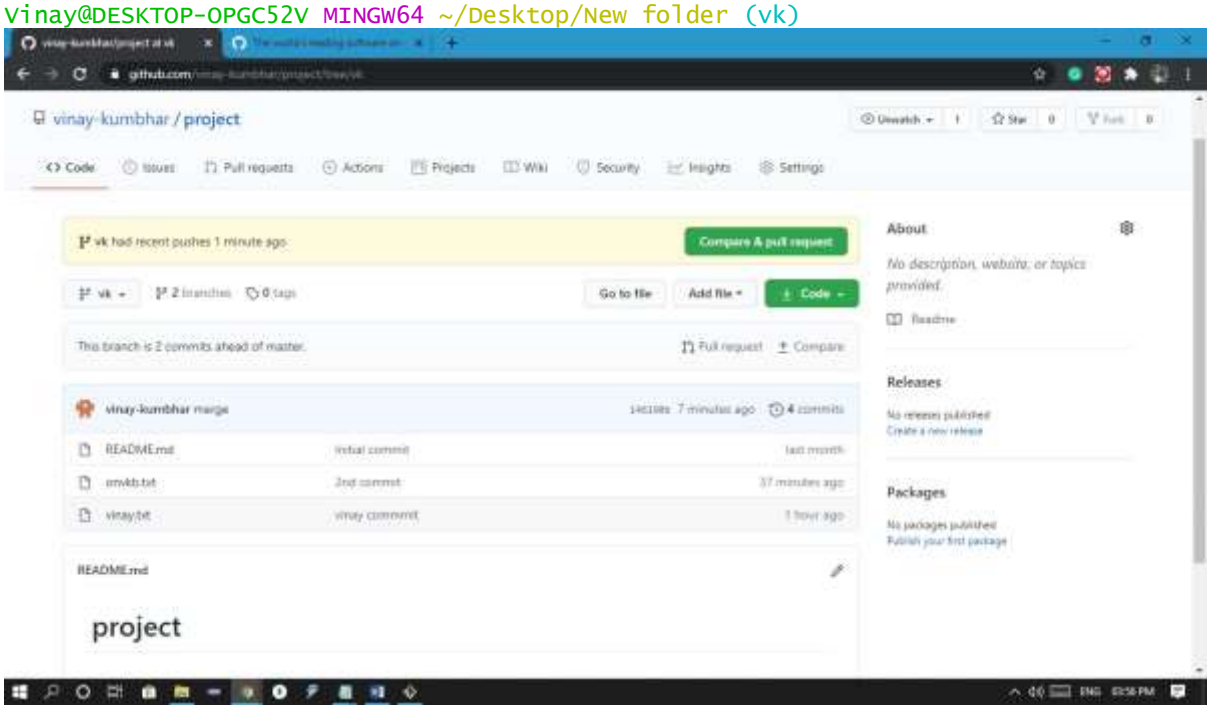
```
Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git status
On branch vk
Your branch is ahead of 'origin/vk' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git pust origin vk
git: 'pust' is not a git command. See 'git --help'.

The most similar command is
        push

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git push origin vk
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 283 bytes | 283.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/vinay-kumbhar/project.git
   eee23fc..00b89e9  vk -> vk

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git fetch

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git status
On branch vk
Your branch is up to date with 'origin/vk'.

nothing to commit, working tree clean
```



Merging :

```
Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (master)
$ git checkout vk
Switched to branch
'vk'
Your branch is up to date with 'origin/vk'.

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git merge master -m "merge"
Merge made by the 'recursive'
 strategy. vinay.txt | 1 +
 1 file changed, 1
 insertion(+) create mode
 100644 vinay.txt
```

```
Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git pull origin vk
From https://github.com/vinay-kumbhar/project
 * branch            vk         ->
FETCH_HEAD Already up to date.

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
$ git push origin vk
Enumerating objects: 4,
done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4
threads Compressing objects: 100%
(2/2), done.
Writing objects: 100% (2/2), 322 bytes | 322.00 KiB/s,
done. Total 2 (delta 0), reused 0 (delta 0)
To https://github.com/vinay-
   kumbhar/project.git 00b89e9..146190b
                       vk -> vk

Vinay@DESKTOP-OPGC52V MINGW64 ~/Desktop/New folder (vk)
```



git init <directory>	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
git clone <repo>	Clone repo located at <***repo***> onto local machine. Original repo can be located on the local filesystem or on a remote machine via ***HTTP*** or ***SSH***.
git config user.name <name>	Define author name to be used for all commits in current repo. Devs commonly use ***--global*** flag to set config options for current user.
git add <directory>	Stage all changes in <***directory***> for the next commit. Replace <***directory***> with a <***file***> to change a specific file.
git commit -m "<message>"	Commit the staged snapshot, but instead of launching a text editor, use <***message***> as the commit message.
git status	List which files are staged, unstaged, and untracked.
git log	Display the entire commit history using the default format. For customization see additional options.
git diff	Show unstaged changes between your index and working directory.

git revert <commit>	Create new commit that undoes all of the changes made in <***commit***>, then apply it to the current branch.
git reset <file>	Remove <***file***> from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
git clean -n	Shows which files would be removed from working directory. Use the ***-f*** flag in place of the ***-n*** flag to execute the clean.

git commit --amend	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
git rebase <base>	Rebase the current branch onto <***base***>. <***base***> can be a commit ID, branch name, a tag, or a relative reference to ***HEAD***.
git reflog	Show a log of changes to the local repository's ***HEAD***. Add ***--relative-date*** flag to show date info or ***--all*** to show all refs.

git branch	List all of the branches in your repo. Add a <***branch***> argument to create a new branch with the name <***branch***>.
git checkout -b <branch>	Create and check out a new branch named <***branch***>. Drop the ***-b*** flag to checkout an existing branch.
git merge <branch>	Merge <***branch***> into the current branch.

git remote add <name> <url>	Create a new connection to a remote repo. After adding a remote, you can use <***name***> as a shortcut for <***url***> in other commands.
git fetch <remote> <branch>	Fetches a specific <***branch***>, from the repo. Leave off <***branch***> to fetch all remote refs.
git pull <remote>	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
git push <remote> <branch>	Push the branch to <***remote***>, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

git config --global user.name <name>	Define the author name to be used for all commits by the current user.
git config --global user.email <email>	Define the author email to be used for all commits by the current user.
git config --global alias. <alias-name> <git-command>	Create shortcut for a Git command. E.g. ***alias.glog "log --graph --oneline"*** will set ***"git glog"*** equivalent to ***"git log --graph--oneline.***
git config --system core.editor <editor>	Set text editor used by commands for all users on the machine. <***editor***> arg should be the command that launches the desired editor (e.g., vi).
git config --global --edit	Open the global configuration file in a text editor for manual editing.

git reset	Reset staging area to match most recent commit, but leave the working directory unchanged.
git reset --hard	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
git reset <commit>	Move the current branch tip backward to *<commit>*, reset the staging area to match, but leave the working directory alone.
git reset --hard <commit>	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** *<commit>*.

git log -<limit>	Limit number of commits by *<limit>*. E.g. *"git log -5"* will limit to 5 commits.
git log --oneline	Condense each commit to a single line.
git log -p	Display the full diff of each commit.
git log --stat	Include which files were altered and the relative number of lines that were added or deleted from each of them.
git log --author= "<pattern>"	Search for commits by a particular author.
git log --grep="<pattern>"	Search for commits with a commit message that matches *<pattern>*.
git log <since>..<until>	Show commits that occur between *<since>* and *<until>*. Args can be a commit ID, branch name, *HEAD*, or any other kind of revision reference.
git log -- <file>	Only display commits that have the specified file.
git log --graph --decorate	*--graph* flag draws a text based graph of commits on left side of commit msgs. *--decorate* adds names of branches or tags of commits shown.

git rebase -i <base>	Interactively rebase current branch onto *<base>*. Launches editor to enter commands for how each commit will be transferred to the new base.

git pull --rebase <remote>	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

git push <remote> --force	Forces the *git push* even if it results in a non-fast-forward merge. Do not use the *--force* flag unless you're absolutely sure you know what you're doing.
git push <remote> --all	Push all of your local branches to the specified remote.
git push <remote> --tags	Tags aren't automatically pushed when you push a branch or use the *--all* flag. The *--tags* flag sends all of your local tags to the remote repo.