GIT

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is the most commonly used version control system today and is quickly becoming *the* ***Standard for version control.***

Git is a distributed version control system, meaning your local copy of code is a complete version control repository.

Why a Version Control System like Git is needed

Real life projects generally have multiple developers working in parallel. So a version control system like Git is needed to ensure there are no code conflicts between the developers.

Additionally, the requirements in such projects change often. So a version control system allows developers to revert and go back to an older version of the code.

Finally, sometimes several projects which are being run in parallel involve the same codebase. In such a case, the concept of branching in Git is very important.

Download git

This link has details on how to install Git in multiple operating systems:
https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
Verify if Git is installed by using the following command in the command prompt:

```
git --version
```

Create your local Git repository

In your computer, create a folder for your project. Let's call the project folder simple-git-demo.
Go into your project folder and add a local Git repository to the project using the following commands:

```
cd simple-git-demo
git init
```

The git init command adds a local Git repository to the project.

Let's Add some Small Code now

Create a file called demo.txt in the project folder and add the following text into it:

Initial Content
Here we will be demoing with just plain text instead of actual code, since the main focus of this article is on Git and not on any specific programming language.

Staging and Committing the code

Committing is the process in which the code is added to the **local repository**. Before committing the code, it has to be in the **staging area**. The staging area is there to keep track of all the files which are to be committed. Any file which is not added to the staging area will not be committed. This gives the developer control over which files need to be committed.

Staging

Use the following command for staging the file:

```
git add demo.txt
```

In case you want to add multiple files you can use:

```
git add file1 file2 file3
```

If you want to add all the files inside your project folder to the staging area, use the following command:

```
git add .
```

Use this carefully since it adds all the files and folders in your project to the staging area.

Committing

Use the following command to commit the file:

```
git commit -m "Initial Commit"
```

“Initial Commit” is the commit message here. Enter a relevant commit message to indicate what code changes were done in that particular commit.

Branches

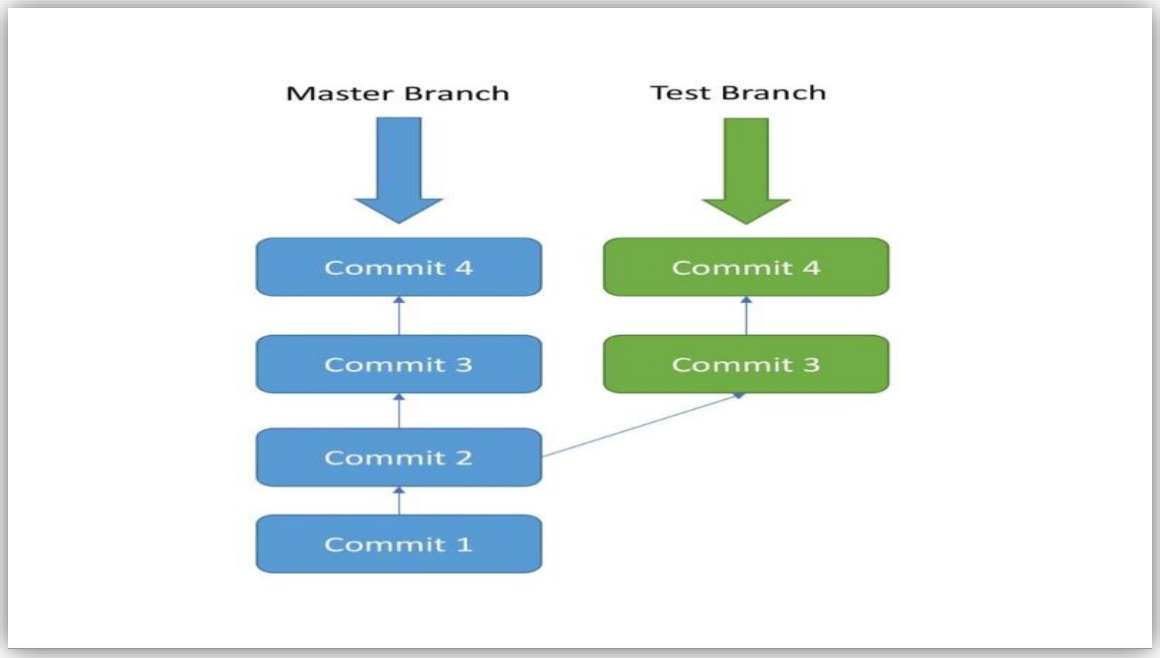
Up until now we have not created any branch in Git. By default, Git commits go into the **master** branch.

What is a branch?

A branch is nothing but a pointer to the latest commit in the Git repository. So currently our master branch is a pointer to the second commit “demo.txt file is modified”.

Why are multiple branches needed?

Multiple branches are needed to support multiple parallel developments. Refer the image below to see how branches work.



Initially, commit 1 and commit 2 were done in the master branch. After commit 2 a new Branch called as “Test” is created, and commit 3 and commit 4 are added to the test branch.

At the same time, a different commit 3 and commit 4 are added to the master branch. Here we can see that after Commit 2, two parallel developments are being done in 2 separate branches.

The Test Branch and the Master Branch have diverged here and have different code — the code from Test Branch can be merged with the Master branch using git merge. This will be covered later.

Advantage

One of the biggest advantages of Git is its branching capabilities. Unlike centralized version **control** systems, Git branches are cheap and easy to merge.
This facilitates the feature branch workflow popular with many Git users. Feature branches provide an isolated **environment** for every change to your codebase.