

# GitHub

## 1.1 GitHub Introduction:

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.

## 1.2 Advantages of Git

- Free and open source

Git is released under GPL's open source license. It is available freely over the internet. You can use Git to manage property projects without paying a single penny. As it is an open source, you can download its source code and also perform changes according to your requirements.

- Fast and small

As most of the operations are performed locally, it gives a huge benefit in terms of speed. Git does not rely on the central server; that is why, there is no need to interact with the remote server for every operation. The core part of Git is written in C, which avoids runtime overheads associated with other high-level languages. Though Git mirrors entire repository, the size of the data on the client side is small. This illustrates the efficiency of Git at compressing and storing data on the client side.

- Implicit backup

The chances of losing data are very rare when there are multiple copies of it. Data present on any client side mirrors the repository, hence it can be used in the event of a crash or disk corruption.

- Security

Git uses a common cryptographic hash function called secure hash function (SHA1), to name and identify objects within its database. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. It implies that, it is impossible to change file, date, and commit message and any other data from the Git database without knowing Git.

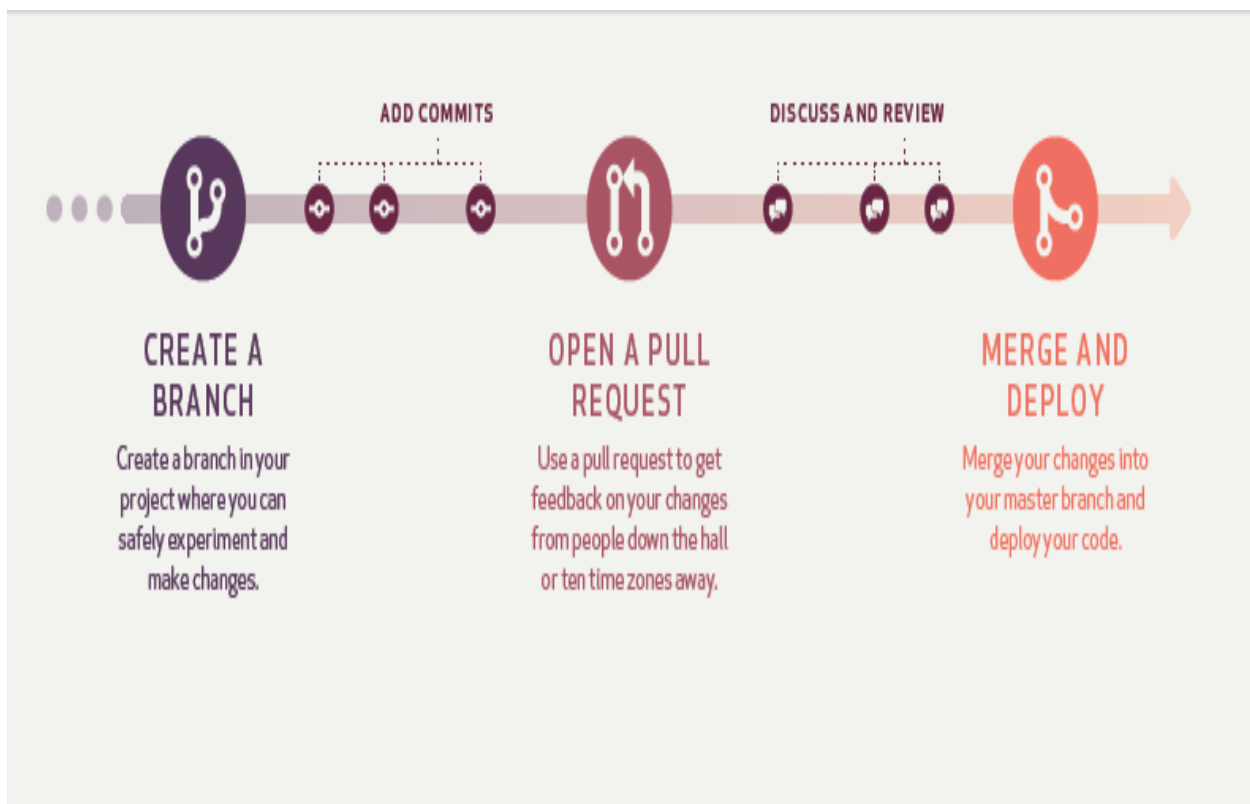
- No need of powerful hardware

In case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck. In case of DVCS, developers don't interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.

- Easier branching

CVCS uses cheap copy mechanism, If we create a new branch, it will copy all the codes to the new branch, so it is time-consuming and not efficient. Also, deletion and merging of branches in CVCS is complicated and time-consuming. But branch management with Git is very simple. It takes only a few seconds to create, delete, and merge branches.

### 1.3 Flow to create project



GitHub Flow is a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly. This guide explains how and why GitHub Flow works.

#### ➤ Creating Repository

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: ajaypp123 / Repository name:

Great repository names are short and memorable. Need inspiration? How about sturdy-palm-tree.

Description (optional):

☒ Public  
Anyone can see this repository. You choose who can commit.

☐ Private  
You choose who can see and commit to this repository.

☐ Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None ⓘ

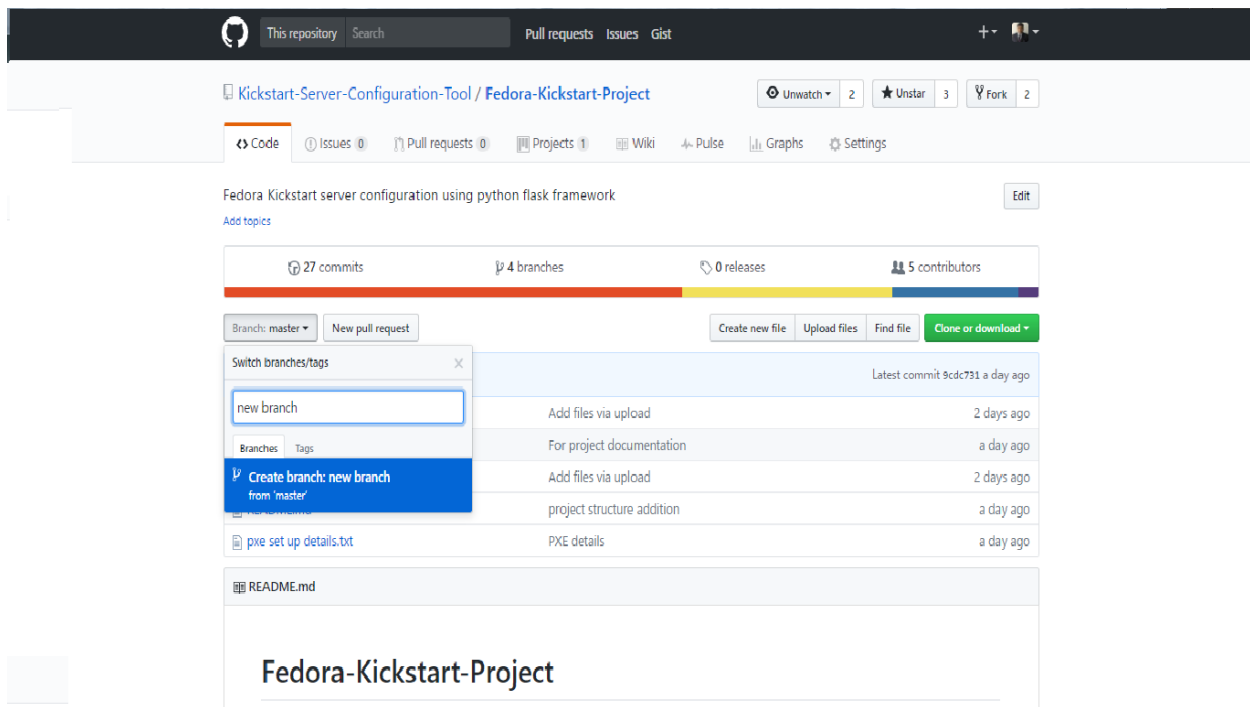
Create repository

Figure 1

Log into your GitHub account (or create one) and then click the icon on the top right that shows the tool tip "Create a new repo". This should open a page similar to the one shown in Figure 1, with your GitHub user name instead of "ajaypp123." Now enter a repository name such as "test" and optionally a description such as "test repository writing project" in our example. We selected this to be a public repository as we do not have the paid-for service of private repositories. Finally, click the button "Create repository" and proceed.

## ➤ Create a branch

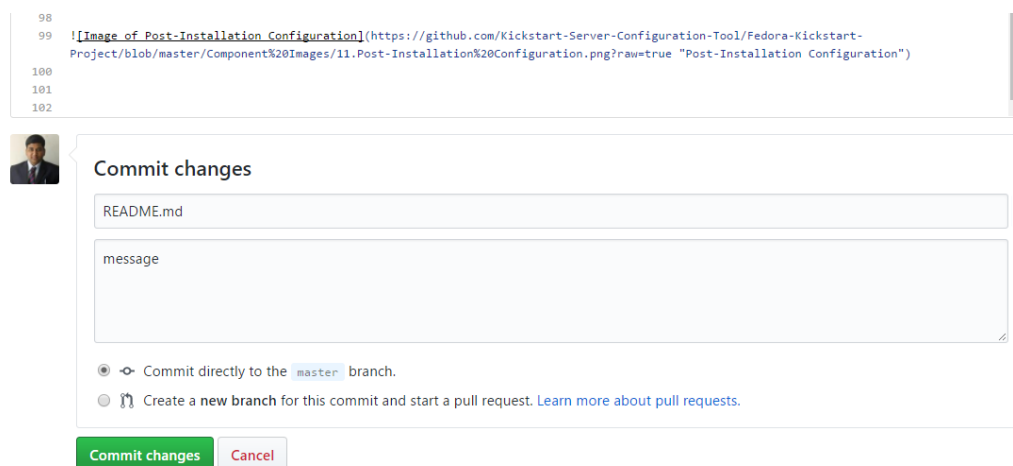
When you create a branch in your project, you're creating an environment where you can try out new ideas. Changes you make on a branch don't affect the master branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.



## ➤ Add commits

Once your branch has been created, it's time to start making changes. Whenever you add, edit, or delete a file, you're making a commit, and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.

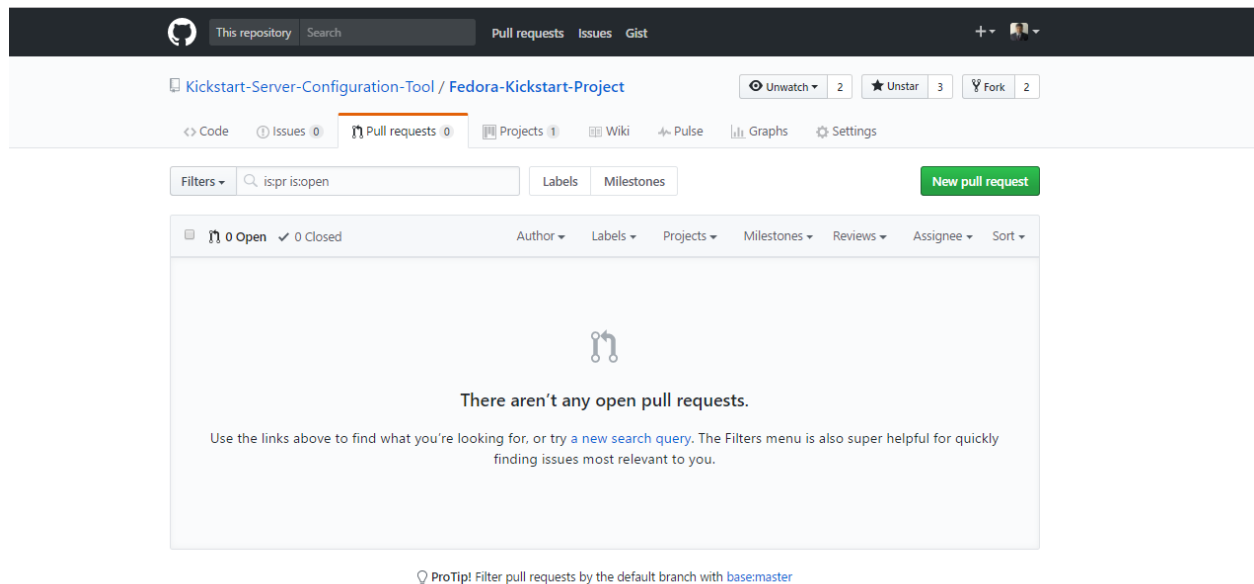
Commits also create a transparent history of your work that others can follow to understand what you've done and why. Each commit has an associated commit message, which is a description explaining why a particular change was made. Furthermore, each commit is considered a separate unit of change. This lets you roll back changes if a bug is found, or if you decide to head in a different direction.



## ➤ Open a Pull Request

Pull Requests initiate discussion about your commits. Because they're tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.

You can open a Pull Request at any point during the development process: when you have little or no code but want to share some screenshots or general ideas, when you're stuck and need help or advice, or when you're ready for someone to review your work. By using GitHub's @mention system in your Pull Request message, you can ask for feedback from specific people or teams, whether they're down the hall or ten time zones away.



## ➤ Discuss and review your code

Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments. Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage and capture this type of conversation.

You can also continue to push to your branch in light of discussion and feedback about your commits. If someone comments that you forgot to do something or if there is a bug in the code, you can fix it in your branch and push up the change. GitHub will show your new commits and any additional feedback you may receive in the unified Pull Request view.


## ➤ Deploy


Once your pull request has been reviewed and the branch passes your tests, you can deploy your changes to verify them in production. If your branch causes issues, you can roll it back by deploying the existing master into production.

## ➤ Merge


Now that your changes have been verified in production, it is time to merge your code into the master branch.

Once merged, Pull Requests preserve a record of the historical changes to your code. Because they're searchable, they let anyone go back in time to understand why and how a decision was made.




**This branch has no conflicts with the base branch**

Merging can be performed automatically.


 **Merge pull request**

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).



**Pull request successfully merged and closed**

You're all set—the `readme-edits` branch can be safely deleted.

 **Delete branch**