Jenkins Pipeline

JENKINS

BY ALOK SRIVASTAVA - NETWORK NUTS

CI WORKFLOW

PIPELINE

The term pipeline is used to refer to a mechanism or a system of moving something, say supplies, from one point to another. Here, we are referring to shipping an application from development to the end user.

Continuous integration and delivery pipelines help software development to move code from development to different environments such as staging and production. A **pipeline** provides a fast, repeatable, and consistent process for delivering software. While pipelines vary for different products and scenarios, there are some generic steps that are important on every pipeline.

The steps in a CI pipeline depend on the type of project you are running but, at a high level, involve pulling code from source control, preparing a test environment, and running tests.

These are mentioned in brief below:

Pulling Code from Source Control: The expectation here is that you have a central repository where every developer regularly checks in code. The pipeline is configured to pull in the latest changes from a centralized repository host such as GitHub or Bitbucket.

Preparing the Application Environment: Depending on the application, this step is optional, but most applications today rely on dependencies that have to be installed prior to running the application. For instance, a Python application could require pip packages, or a Node.js application could have some npm dependencies.

Testing: Testing is one of the more important stages in a pipeline. This stage will run all the tests in the application and ensure they pass.

Building: A build artifact is a shippable file of your application. If you are using Java, the artifact could potentially be a JAR file, and if you are using Docker, the build artifact will be a Docker image.

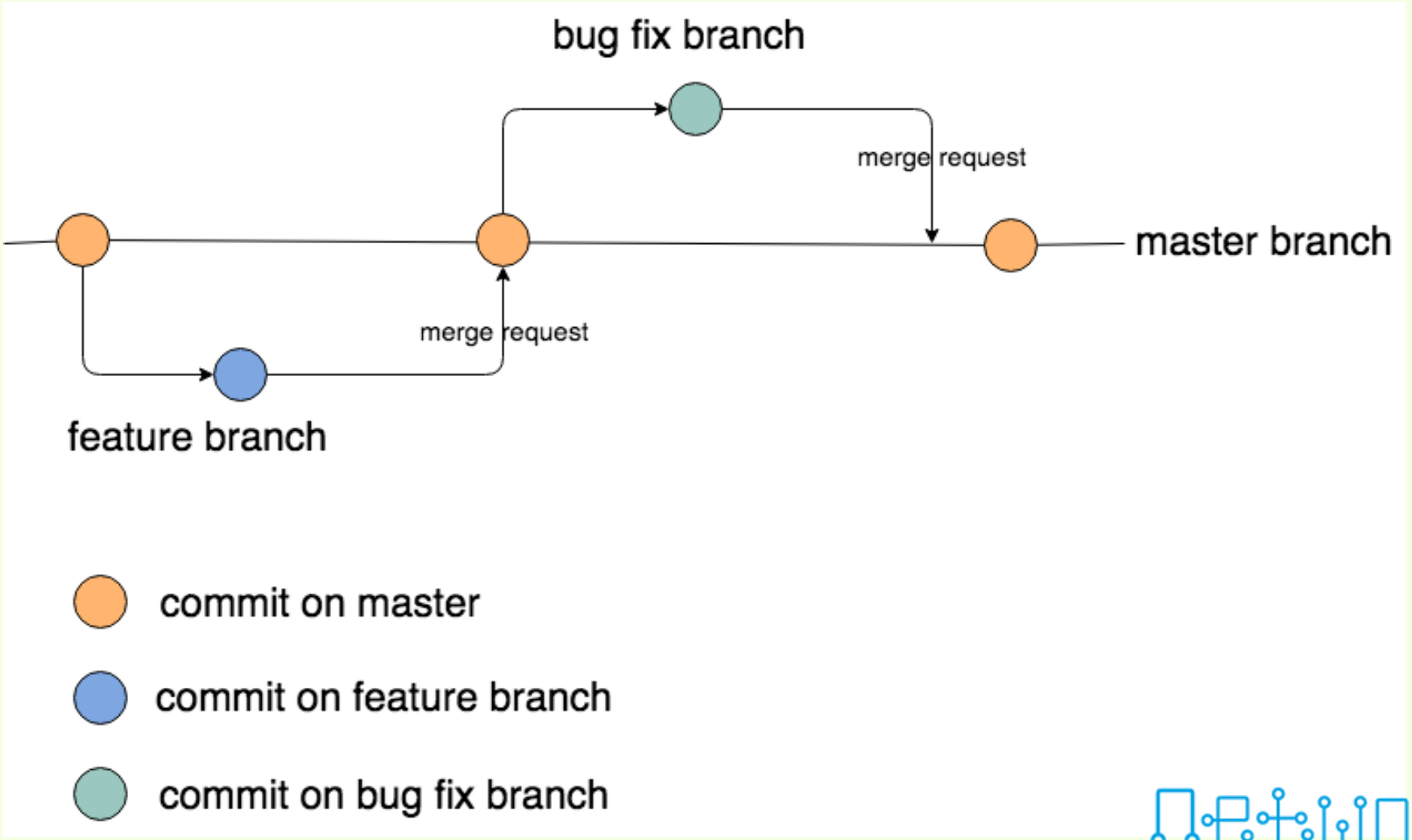
Deployment: Depending on the setup, the pipeline may go ahead and deploy the application or wait for some manual input from a member of the team to deploy the application.


The Basic Elements of
GIT
BRANCHES

The main branch in Git, by default, is referred to as **master** and, ideally, this branch should have the **most stable code base** that is ready for release. As many developers work on a project, the ideal workflow is as follows: one branches off the master (hence the term branch), makes the changes they need, such as adding features and fixing bugs, and then, after they are done, they commit those changes and request a merge back to the master branch, a process referred to on GitHub as making a pull request (PR).

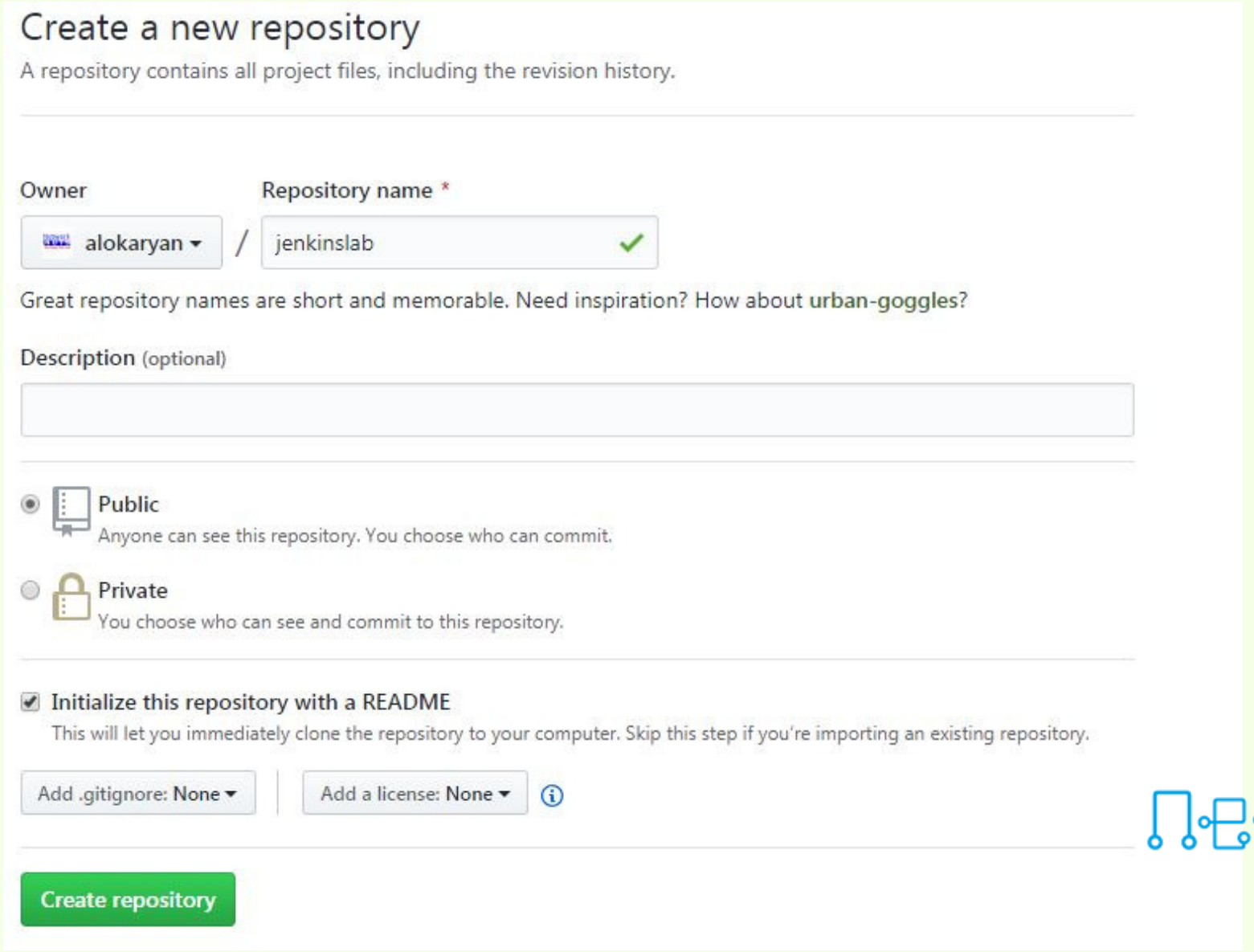
Think of a branch as a timeline. The main timeline is the master branch. When we create a branch, we create a new timeline from the master timeline and perform changes to the code base. When we want to merge back to the main timeline, we request a merge and, if our changes did not introduce any bugs or errors, then we are allowed to merge.

We can visualize our workflow as shown in the following diagram.

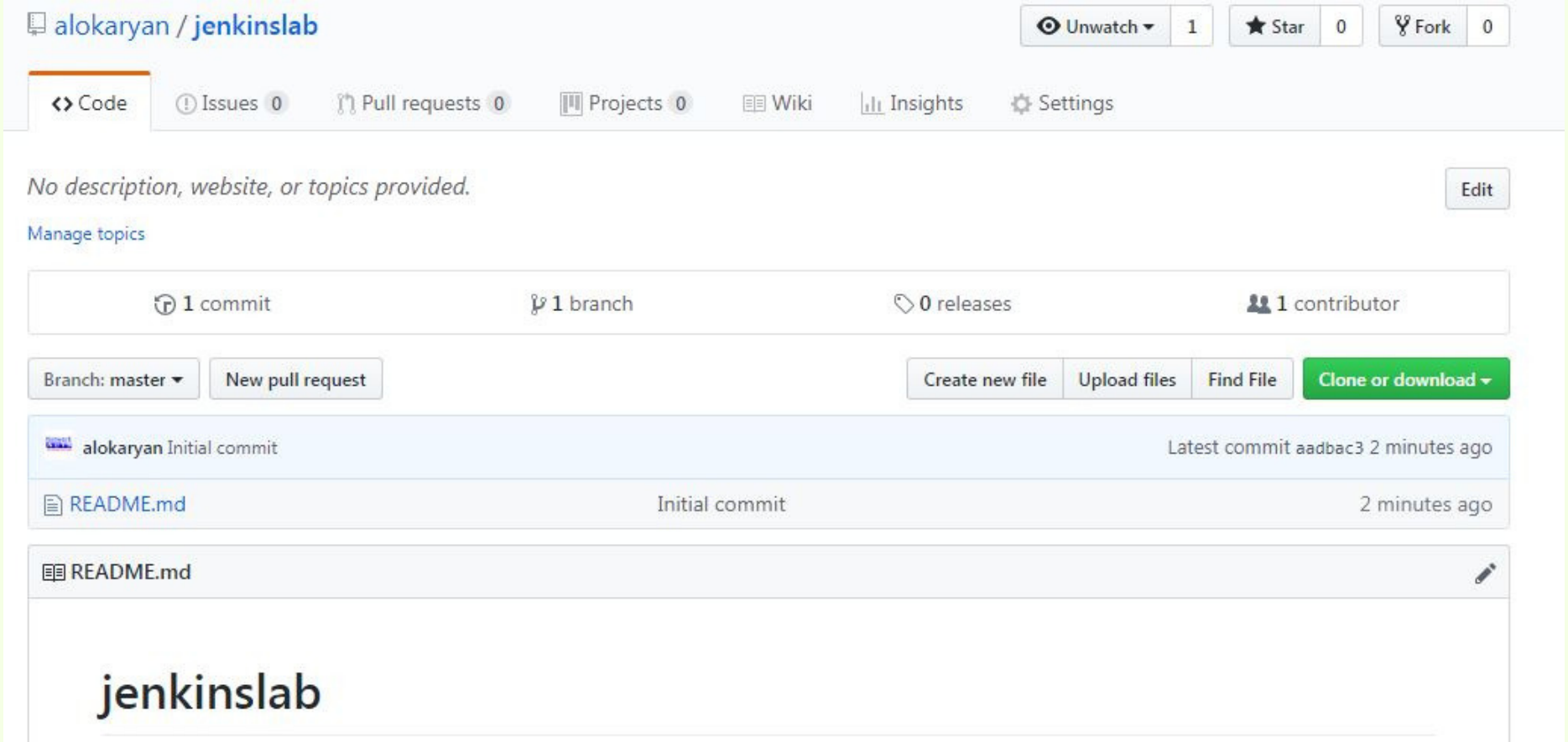


Setting up our Repository

Go to your github account and create a new repository



You will be redirected to the repository page



Set up our repository locally. On your jenkins machine. Create a folder and name it "jenkinslab" and sync with your newly created repo on github.

```
[root@jenkins jenkinslab]# pwd
/root/jenkinslab
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]# git init
Reinitialized existing Git repository in /root/jenkinslab/.git/
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]# git remote add origin https://github.com/alokaryan/jenkinslab
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]# git remote -v
origin  https://github.com/alokaryan/jenkinslab (fetch)
origin  https://github.com/alokaryan/jenkinslab (push)
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]#
```

Pull the repo from GitHub, to local repository.

```
[root@jenkins jenkinslab]# 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/alokaryan/jenkinslab
 * branch            master     -> FETCH_HEAD
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]# ll
total 4
-rw-r--r--. 1 root root 12 Apr  2 05:52 README.md
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]#
```

Now create a new branch called test. Use the **git checkout command and the -b flag**, which will create a new branch if one does not exist.

```
[root@jenkins jenkinslab]# ll
total 4
-rw-r--r--. 1 root root 12 Apr  2 05:52 README.md
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]# git checkout -b test
Switched to a new branch 'test'
[root@jenkins jenkinslab]#
```

While on the same branch. Create a sample python file. Use git status to view the files that haven't been tracked, followed by git add. to start tracking the files.

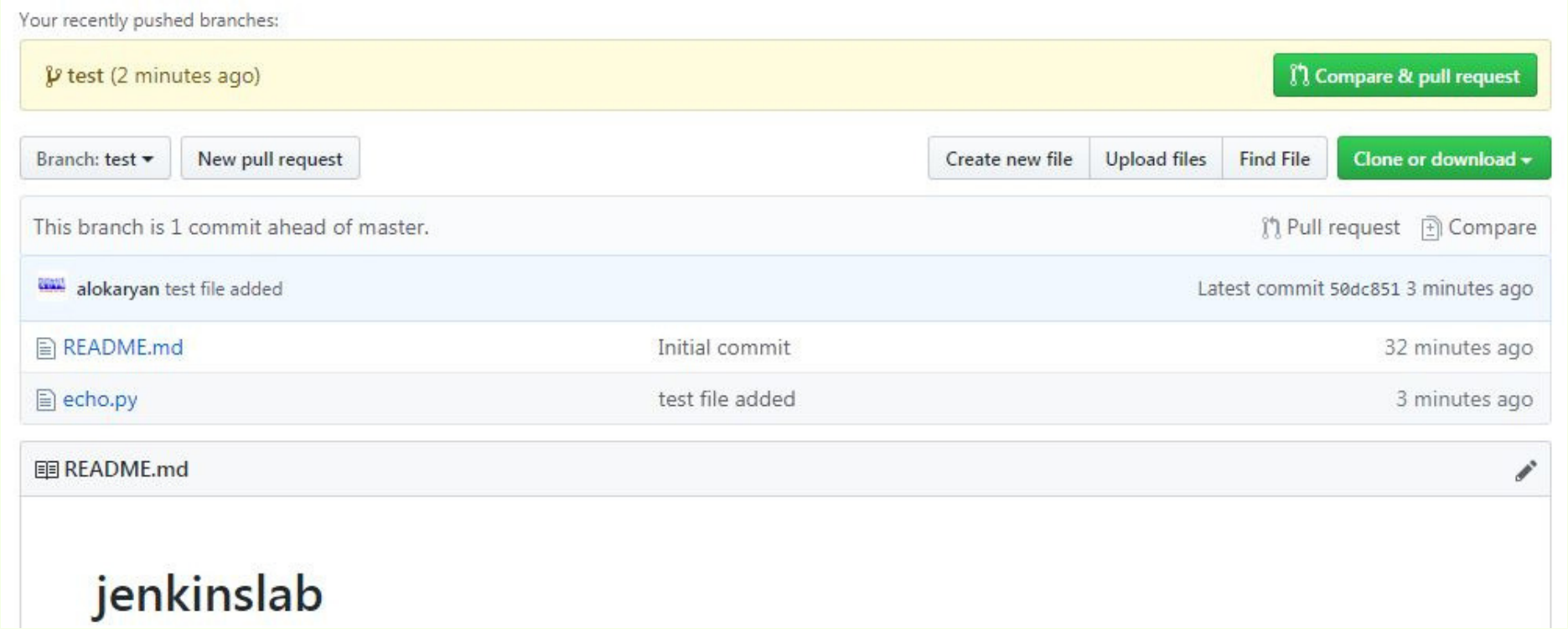
```
[root@jenkins jenkinslab]# cat echo.py
#!/usr/bin/env python
print("simple hello from python and git")

[root@jenkins jenkinslab]# git status
# On branch test
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       echo.py
nothing added to commit but untracked files present (use "git add" to track)
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]# git add .
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]# git status
# On branch test
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   echo.py
#
[root@jenkins jenkinslab]#
```

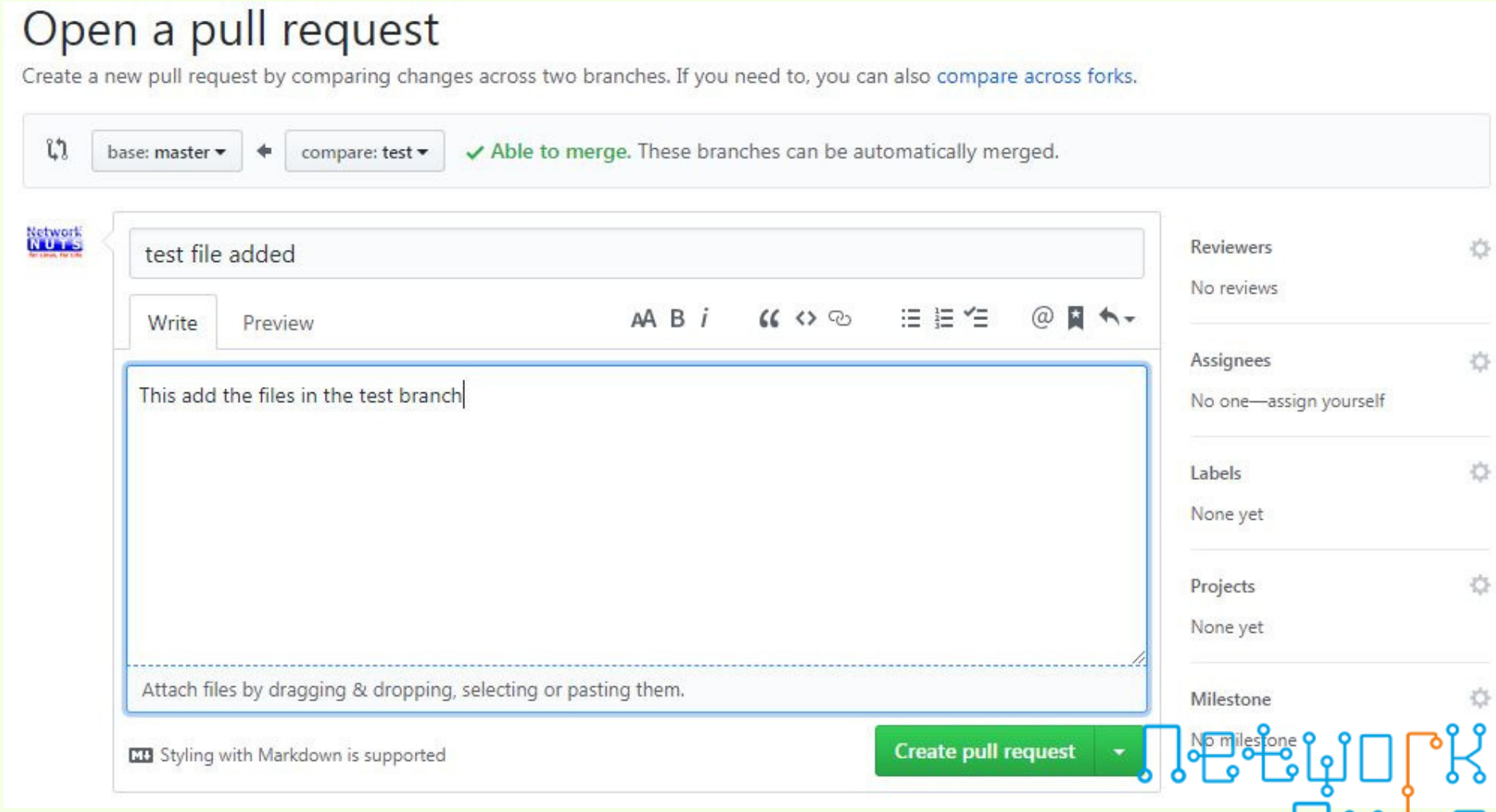
Finally commit our changes and push them to the remote repository. Use **git commit** command flag with the -m flag to add a descriptive comment.

```
[root@jenkins jenkinslab]# git commit -m "test file added"
[test 50dc851] test file added
 1 file changed, 3 insertions(+)
 create mode 100644 echo.py
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]# git push origin test
Username for 'https://github.com': alokaryan
Password for 'https://alokaryan@github.com':
Counting objects: 4, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 330 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'test' on GitHub by visiting:
remote:      https://github.com/alokaryan/jenkinslab/pull/new/test
remote:
To https://github.com/alokaryan/jenkinslab
 * [new branch]      test -> test
[root@jenkins jenkinslab]#
[root@jenkins jenkinslab]#
```

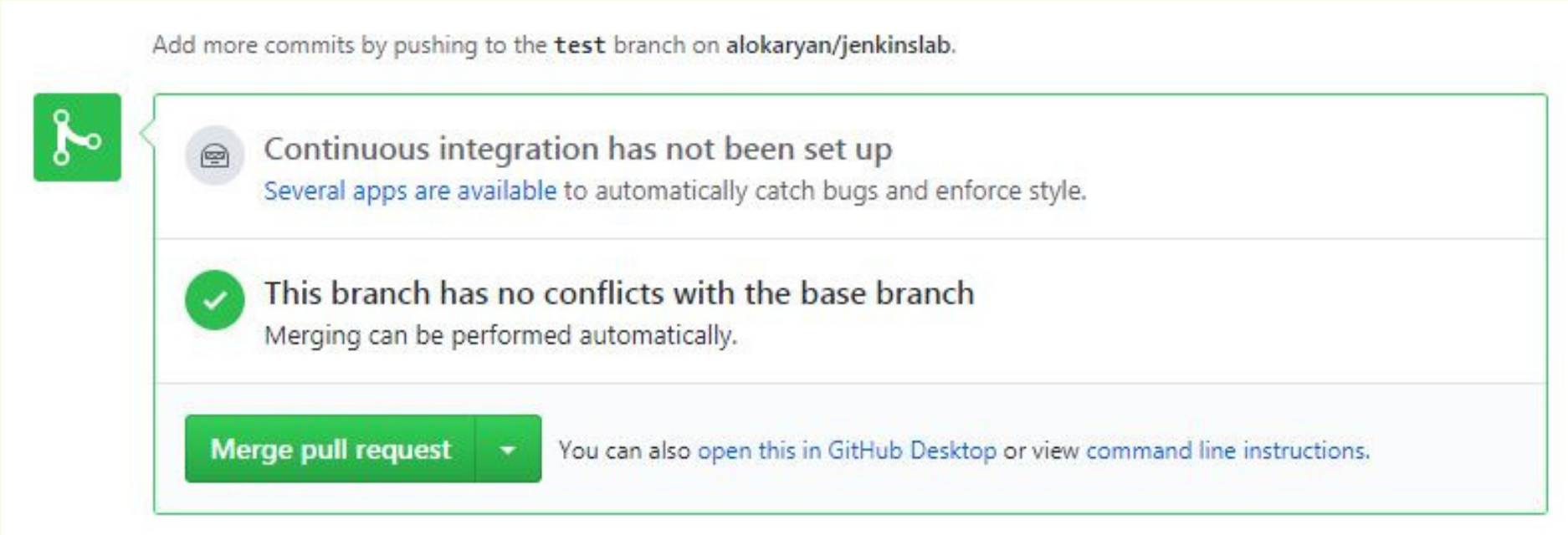
Going back to our project dashboard on GitHub, we can see that a new branch has been added.



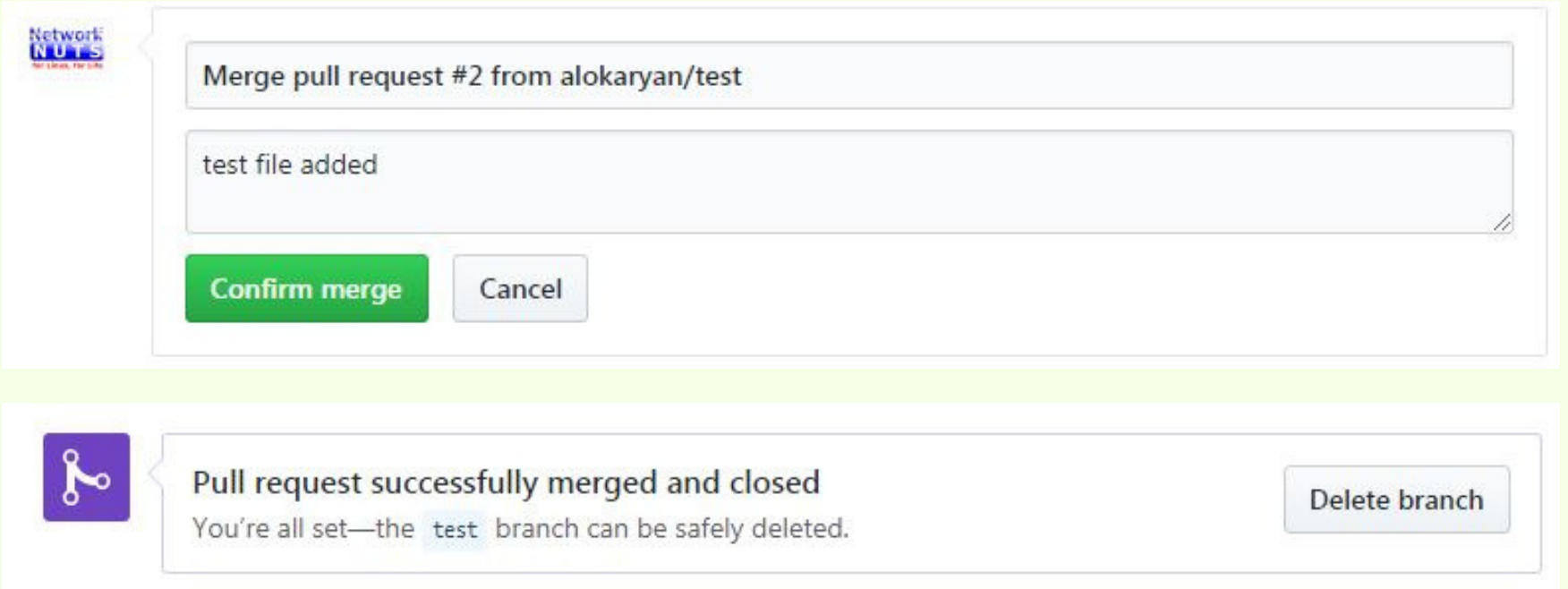
Now create a pull request to our master branch, allowing us to merge the changes from our new branch. Select the **Compare & pull request** button to create and configure our pull request



The base branch is the branch you want to merge to, and the compare branch is the branch you want to merge. We will then press the **Create pull request** button to create our pull request. This lands us on the pull request page, where we can see a few more things regarding our project. Of most importance is the output in the **Merge pull request section**

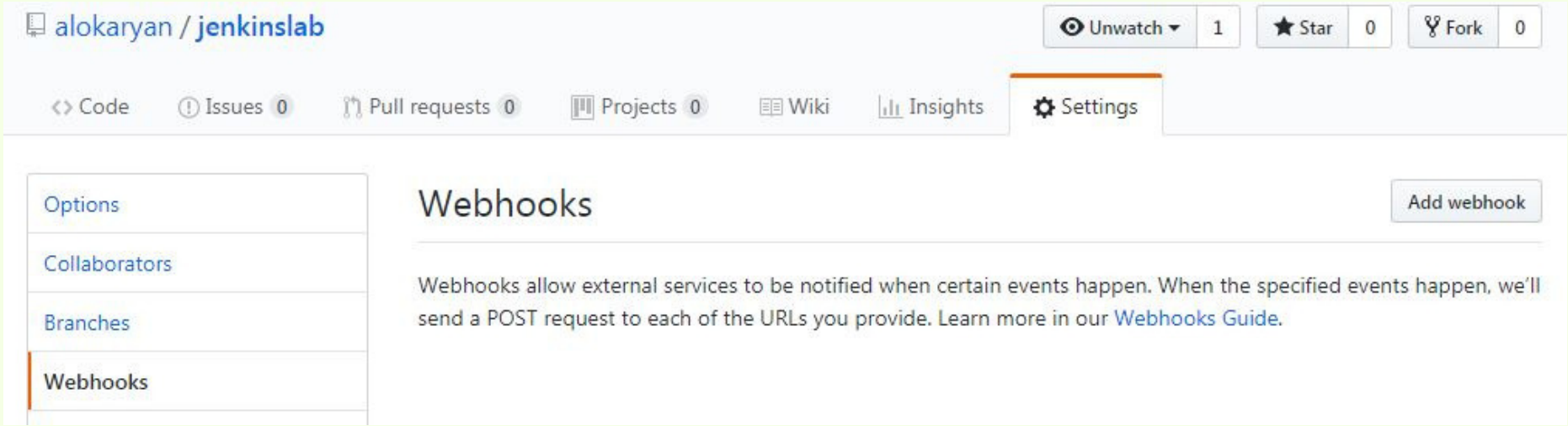


GitHub informs us that CI has not been set up. Before a Pull Request is considered mergeable, GitHub will ensure all integrated checks have certified the PR. This includes what we mentioned earlier, that is testing, linting, and so on



GitHub Repository and Integrating Jenkins

Go to the repository settings and add the Jenkins GitHub plugin integration. This is under **Settings -> Webhooks**.



For completing this integration, the jenkins server must be publicly accessible. Like a EC2 instance.