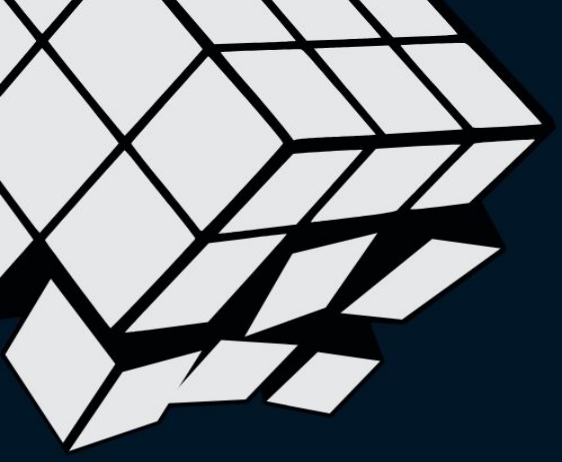


INTRODUCTION TO GIT & GITHUB

Table of Contents

Creating a new branch

Let's start by creating a new branch called `newFeature`. In order to create the branch, you could use the following command:

```
git branch newFeature
```

Now, in order to switch to that new branch, you would need to run the following command:

```
git checkout newFeature
```

Note: You can use the `git checkout` command to switch between different branches.

The above two commands could be combined into 1, so that you don't have to create the branch first and then switch to the new branch. You could use this command instead, which would do both:

```
git checkout -b newFeature
```

Once you run this command, you will see the following output:

```
Switched to a new branch 'newFeature'
```

In order to check what branch you are currently on, you can use the following command:

```
git branch
```

Output:

```
  main
* newFeature
```

We can tell that we have 2 branches: the `main` one and the `newFeature` one that we just created. The star before the `newFeature` branch name indicates that we are currently on the `newFeature` branch.

If you were to use the `git checkout` command to switch to the `main` branch:

```
git checkout main
```

And then run `git branch` again. You will see the following output indicating that you are now on the `main` branch:

```
* main
  newFeature
```

Making changes to the new branch

Now let's go ahead and make a change on the new feature branch. First switch to the branch with the `git checkout` command:

```
git checkout newFeature
```

Note: we only need to add the `-b` argument when creating new branches

Check that you've actually switched to the correct branch:

```
git branch
```

Output:

```
  main
* newFeature
```

Now let's create a new file with some demo content. You can do that with the following command:

```
echo "<h1>My First Feature Branch</h1>" > feature1.html
```

The above will echo out the `<h1>My First Feature Branch</h1>` string and store it in a new file called `feature1.html`.

After that, stage the file and commit the change:

```
git add feature1.html
git commit -m "Add feature1.html"
```

The new `feature1.html` file will only be present on the `newFeature` branch. If you were to switch to the `main` branch and run the `ls` command or check the `git log`, you will be able to see that the file is not there.

You can check that by using the `git log` command:

```
git log
```

With that, we've used quite a bit of the commands that we've covered in the previous chapters!

Compare branches

You can also compare two branches with the following commands.

- Shows the commits on `branchA` that are not on `branchB`:

```
git log BranchA..BranchB
```

- Shows the difference of what is in `branchA` but not in `branchB`:

```
git diff BranchB...BranchA
```

Renaming a branch

In case that you've created a branch with the wrong name or if you think that the name could be improved as it is not descriptive enough, you can rename a branch by running the following command:

```
git branch -m wrong-branch-name correct-branch-name
```

If you want to rename your **current branch**, you could just run the following:

```
git branch -m my-branch-name
```

After that, if you run `git branch` again you will be able to see the correct branch name.

Deleting a branch

If you wanted to completely delete a specific branch you could run the following command:

```
git branch -d name_of_the_branch
```

This would only delete the branch from your local repository, in case that you've already pushed the branch to GitHub, you can use the following command to delete the remote branch:

```
git push origin --delete name_of_the_branch
```

If you wanted to synchronize your local branches with the remote branches you could run the following command:

```
git fetch
```

Conclusion

With that, our `newFeature` branch is now ahead of the main branch with 1 commit. So in order to get that new changes over to the main branch, we need to merge the `newFeature` branch into our `main` branch.

In the next chapter, you will learn how to merge your changes from one branch to another!

One thing that you might want to keep in mind is that in the past when creating a new GitHub repository the default branch name was called `master`. However, new repositories created on GitHub use `main` instead of `master` as the default branch name. This is part of GitHub's effort to

remove unnecessary references to slavery and replace them with more inclusive terms.

Forking in Git

When contributing to an open-source project, you will not be able to make the changes directly to the project. Only the repository maintainers have that privilege.

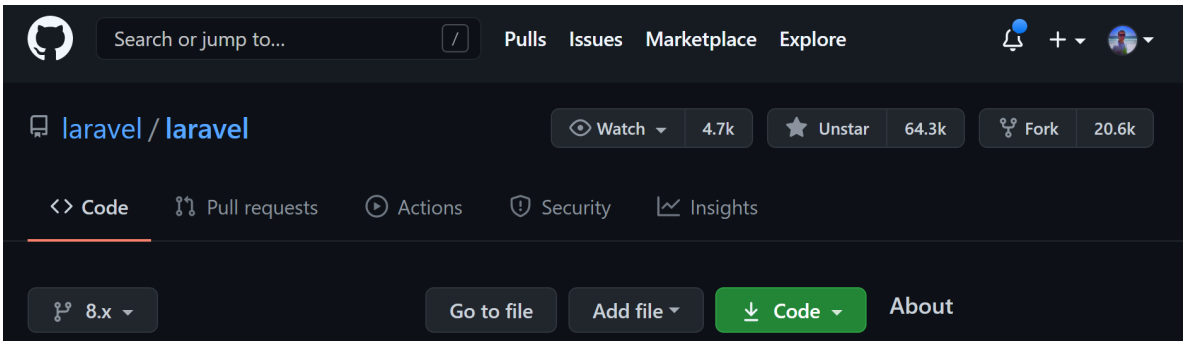
What you need to do instead is to fork the specific repository, make the changes to the forked project and then submit a pull request to the original project. You will learn more about pull requests in the next chapters.

If you clone a repository that you don't have the access to and then try to push the changes directly to that repository, you would get the following error:

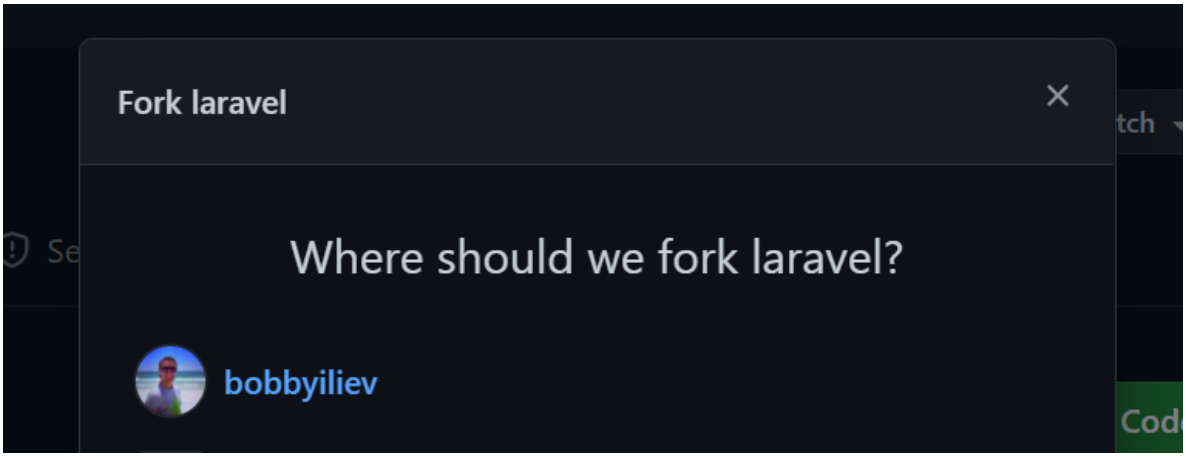
```
ERROR: Permission to laravel/laravel.git denied to bobbyiliev.
Fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

This is where Forks come into play!

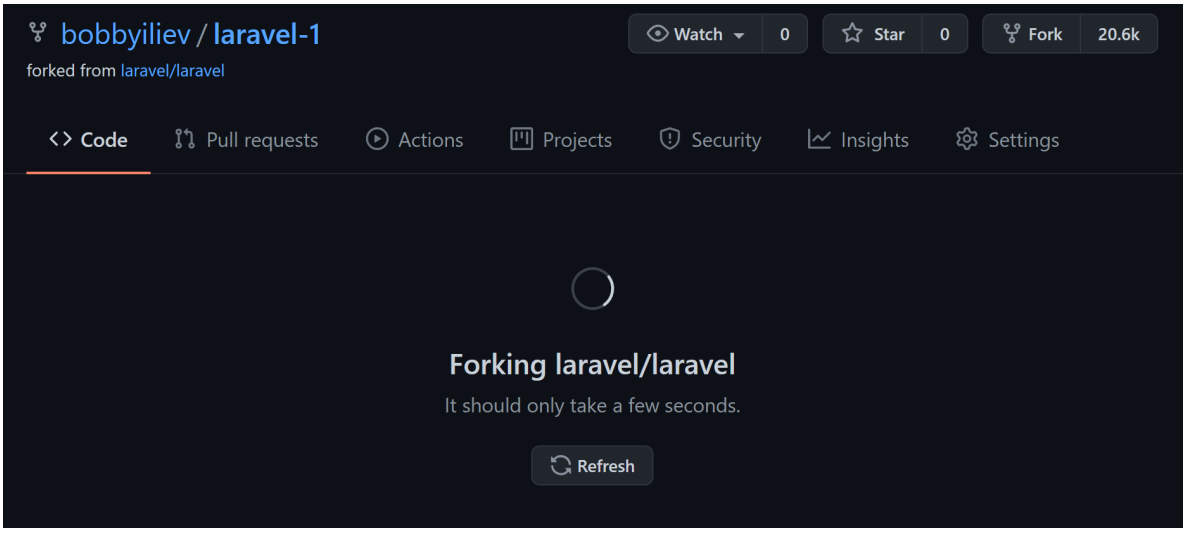
In order to fork a repository, you need to visit the repository via your browser and click on the Fork button on the top right:



Then choose the account that you want to fork the repository to:



Then it might take a few seconds for the repository to be forked under your account:



With that, you would have an exact copy of the repository in question under your account.

The benefit here is that you can now clone the forked repository under your account, make the changes to that repository as normal, and then once you are ready, you can submit a pull request to the original repository contributing your changes.

As we've now mentioned submitting pull requests a few times already, let's go ahead and learn more about pull requests in the next chapter!

Git Workflow

Now that you know the basic commands, let's put it all together and go through a basic Git workflow.

Usually, the workflow looks something like this:

- First, you clone an existing project with the `git clone` command, or if you are starting a new project, you initialize it with the `git init` command.

- After that, before starting with your code changes, it's best to create a new Git branch where you would work on. You can do that with the `git checkout -b YOUR_BRANCH_NAME` command.

- Once you have your branch ready, you would start making the changes to your code.

- Then, once you are ready with the changes, you need to stage them with the `git add` command.

- Then, to commit/save the changes to your local Git repository, you need to run the `git commit` command and provide a descriptive commit message.

- To push your local changes to your remote GitHub project, you would use the `git push origin YOUR_BRANCH_NAME` command