Git Checkout Remote Branch Tutorial

freecodecamp.org/news/git-checkout-remote-branch-tutorial

January 12, 2021

Git is a version control tool that allows you to maintain and view different versions of your application. When a new update breaks your app, Git lets you revert those changes to the previous version.

In addition to versioning, Git allows you to work in multiple environments at the same time. Multiple environments in this context means **branches**.

Why you need branches

When you're working with git, you'll have a master (also called main) environment (branch). This particular branch holds the source code that gets deployed when your app is ready for production.

When you want to update your app, you can also add more commits (changes) to this branch. For minor changes, this may not be a big deal, but for big changes, doing this is not ideal. And that's why other branches exist.

To create and use a new branch, you use the following command in your terminal in the project directory:

```
# create a new branch
git branch branch-name
# change environment to the new branch
git checkout branch-name
```

On this new branch, you can create the new changes. Then when you're done, you can merge them with the master branch.

Another benefit of branches is that they allow multiple developers to work on the same project simultaneously. If you have multiple developers working on the same master branch, it can be disastrous. You have too many changes between each developer's code, and this usually ends in merge conflicts.

With Git, you can jump on another branch (another environment) and make changes there, while work goes on in other branches.

What does Git Checkout Remote Branch mean?

When you begin a project with Git, you get two environments: the local master branch (which exists in your computer), and the remote master branch (which exists in a Git-supported platform like GitHub).

You can push changes from the local master branch to the remote master branch and also pull changes from the remote branch.

When you create a branch locally, it exists only locally until it is pushed to GitHub where it becomes the remote branch. This is shown in the following example:

```
# create a new branch
git branch new-branch
# change environment to the new branch
git checkout new-branch
# create a change
touch new-file.js
# commit the change
git add .
git commit -m "add new file"
# push to a new branch
git push --set-upstream origin new-branch
```

From the example above, `origin new-branch` becomes the remote branch. As you may have noticed, we created a new branch and committed a change on it before pushing to the new remote branch.

But what if the remote branch already existed, and we wanted to pull the branch and all of its changes to our local environment?

That's where we "Git Checkout Remote Branch".

How to Git Checkout Remote Branch

Let's say there's a remote branch created by another developer, and you want to pull that branch. Here's how you go about it:

1. Fetch all remote branches

```
git fetch origin
```

This fetches all the remote branches from the repository. `origin` is the remote name you're targetting. So if you had an `upstream` remote name, you can call `git fetch upstream`.

2. List the branches available for checkout

To see the branches available for checkout, run the following:

```
git branch -a
```

The output of this command is the list of branches available for checkout. For the remote branches, you'll find them prefixed with `remotes/origin`.

3. Pull changes from a remote branch

Note that you cannot make changes directly on a remote branch. Hence, you need a copy of that branch. Say you wanted to copy the remote branch `fix-failing-tests`, here's how you would do it:

```
git checkout -b fix-failing-tests origin/fix-failing-tests
```

What this does is:

- it creates a new branch called `fix-failing-tests`
- it `checkout`s that branch
- it pulls changes from `origin/fix-failing-tests` to that branch

And now you have a copy of that remote branch. Also, you can push commits to that remote branch. For example, you make push a new commit like so:

```
touch new-file.js
git add .
git commit -m "add new file"
git push
```

This will push the committed changes to `origin/fix-failing-tests`. If you noticed, we didn't have to specify where we were pushing the changes (like `git push origin fix-failing-tests`). That's because git automatically sets the local branch to track the remote branch.

Conclusion

Git branching makes it very easy to collaborate during application development.

With branches, different developers can easily work on different parts of the application simultaneously.

With checkout remote branch, collaboration even becomes more seamless as developers can also copy remote branches locally on their systems, make changes, and push to the remote branches.

Dillion Megida

Frontend Web Engineer and Technical Writer. I love teaching what I know. Javascript / ReactJS / NodeJS