**Duplicating a repository**

To maintain a mirror of a repository without forking it, you can run a special clone command, then mirror-push to the new repository.

**Platform navigation**

* [Mac](https://docs.github.com/en/repositories/creating-and-managing-repositories/duplicating-a-repository?platform=mac)
* [**Windows**](https://docs.github.com/en/repositories/creating-and-managing-repositories/duplicating-a-repository?platform=windows)
* [Linux](https://docs.github.com/en/repositories/creating-and-managing-repositories/duplicating-a-repository?platform=linux)

**Note:** If you have a project hosted on another version control system, you can automatically import your project to GitHub using the GitHub Importer tool. For more information, see "[About GitHub Importer](https://docs.github.com/en/migrations/importing-source-code/using-github-importer/about-github-importer)."

Before you can push the original repository to your new copy, or *mirror*, of the repository, you must [create the new repository](https://docs.github.com/en/repositories/creating-and-managing-repositories/creating-a-new-repository) on GitHub.com. In these examples, exampleuser/new-repository or exampleuser/mirrored are the mirrors.

[**Mirroring a repository**](https://docs.github.com/en/repositories/creating-and-managing-repositories/duplicating-a-repository#mirroring-a-repository)

1. Open Git Bash.
2. Create a bare clone of the repository.
3. git clone --bare https://github.com/EXAMPLE-USER/OLD-REPOSITORY.git
4. Mirror-push to the new repository.
5. cd OLD-REPOSITORY.git
6. git push --mirror https://github.com/EXAMPLE-USER/NEW-REPOSITORY.git
7. Remove the temporary local repository you created earlier.
8. cd ..
9. rm -rf OLD-REPOSITORY.git

[**Mirroring a repository that contains Git Large File Storage objects**](https://docs.github.com/en/repositories/creating-and-managing-repositories/duplicating-a-repository#mirroring-a-repository-that-contains-git-large-file-storage-objects)

1. Open Git Bash.
2. Create a bare clone of the repository. Replace the example username with the name of the person or organization who owns the repository, and replace the example repository name with the name of the repository you'd like to duplicate.
3. git clone --bare https://github.com/EXAMPLE-USER/OLD-REPOSITORY.git
4. Navigate to the repository you just cloned.
5. cd OLD-REPOSITORY.git
6. Pull in the repository's Git Large File Storage objects.
7. git lfs fetch --all
8. Mirror-push to the new repository.
9. git push --mirror https://github.com/EXAMPLE-USER/NEW-REPOSITORY.git
10. Push the repository's Git Large File Storage objects to your mirror.
11. git lfs push --all https://github.com/EXAMPLE-USER/NEW-REPOSITORY.git
12. Remove the temporary local repository you created earlier.
13. cd ..
14. rm -rf OLD-REPOSITORY.git

[**Mirroring a repository in another location**](https://docs.github.com/en/repositories/creating-and-managing-repositories/duplicating-a-repository#mirroring-a-repository-in-another-location)

If you want to mirror a repository in another location, including getting updates from the original, you can clone a mirror and periodically push the changes.

1. Open Git Bash.
2. Create a bare mirrored clone of the repository.
3. git clone --mirror https://github.com/EXAMPLE-USER/REPOSITORY-TO-MIRROR.git
4. Set the push location to your mirror.
5. cd REPOSITORY-TO-MIRROR
6. git remote set-url --push origin https://github.com/EXAMPLE-USER/MIRRORED

As with a bare clone, a mirrored clone includes all remote branches and tags, but all local references will be overwritten each time you fetch, so it will always be the same as the original repository. Setting the URL for pushes simplifies pushing to your mirror.

1. To update your mirror, fetch updates and push.
2. git fetch -p origin

git push --mirror

2.1 Git Basics - Getting a Git Repository

If you can read only one chapter to get going with Git, this is it. This chapter covers every basic command you need to do the vast majority of the things you’ll eventually spend your time doing with Git. By the end of the chapter, you should be able to configure and initialize a repository, begin and stop tracking files, and stage and commit changes. We’ll also show you how to set up Git to ignore certain files and file patterns, how to undo mistakes quickly and easily, how to browse the history of your project and view changes between commits, and how to push and pull from remote repositories.

**Getting a Git Repository**

You typically obtain a Git repository in one of two ways:

1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
2. You can **clone** an existing Git repository from elsewhere.

In either case, you end up with a Git repository on your local machine, ready for work.

**Initializing a Repository in an Existing Directory**

If you have a project directory that is currently not under version control and you want to start controlling it with Git, you first need to go to that project’s directory. If you’ve never done this, it looks a little different depending on which system you’re running:

for Linux:

$ cd /home/user/my\_project

for macOS:

$ cd /Users/user/my\_project

for Windows:

$ cd C:/Users/user/my\_project

and type:

$ git init

This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet. See [Git Internals](https://git-scm.com/book/en/v2/ch00/ch10-git-internals) for more information about exactly what files are contained in the .git directory you just created.

If you want to start version-controlling existing files (as opposed to an empty directory), you should probably begin tracking those files and do an initial commit. You can accomplish that with a few git add commands that specify the files you want to track, followed by a git commit:

$ git add \*.c

$ git add LICENSE

$ git commit -m 'Initial project version'

We’ll go over what these commands do in just a minute. At this point, you have a Git repository with tracked files and an initial commit.

**Cloning an Existing Repository**

If you want to get a copy of an existing Git repository — for example, a project you’d like to contribute to — the command you need is git clone. If you’re familiar with other VCSs such as Subversion, you’ll notice that the command is "clone" and not "checkout". This is an important distinction — instead of getting just a working copy, Git receives a full copy of nearly all data that the server has. Every version of every file for the history of the project is pulled down by default when you run git clone. In fact, if your server disk gets corrupted, you can often use nearly any of the clones on any client to set the server back to the state it was in when it was cloned (you may lose some server-side hooks and such, but all the versioned data would be there — see [Getting Git on a Server](https://git-scm.com/book/en/v2/ch00/_getting_git_on_a_server) for more details).

You clone a repository with git clone <url>. For example, if you want to clone the Git linkable library called libgit2, you can do so like this:

$ git clone https://github.com/libgit2/libgit2

That creates a directory named libgit2, initializes a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version. If you go into the new libgit2 directory that was just created, you’ll see the project files in there, ready to be worked on or used.

If you want to clone the repository into a directory named something other than libgit2, you can specify the new directory name as an additional argument:

$ git clone https://github.com/libgit2/libgit2 mylibgit

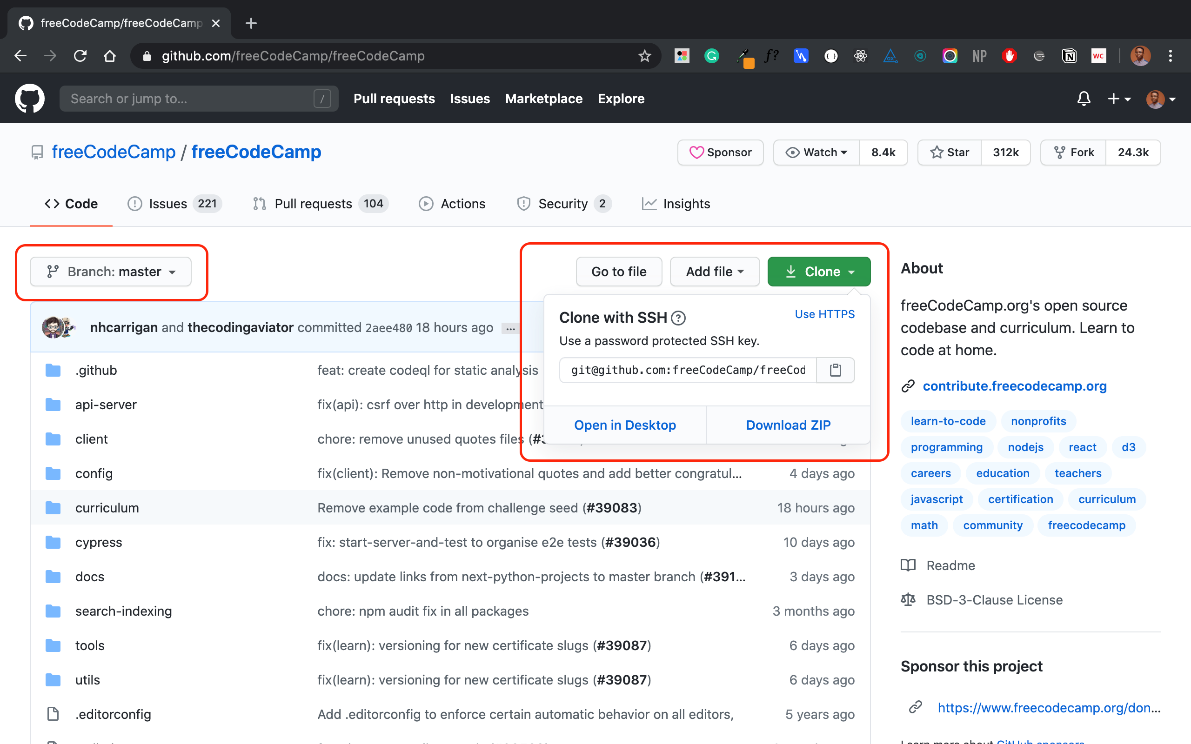
That command does the same thing as the previous one, but the target directory is called mylibgit.

Git has a number of different transfer protocols you can use. The previous example uses the https:// protocol, but you may also see git:// or user@server:path/to/repo.git, which uses the SSH transfer protocol. [Getting Git on a Server](https://git-scm.com/book/en/v2/ch00/_getting_git_on_a_server) will introduce all of the available options the server can set up to access your Git repository and the pros and cons of each.

## Introduction to Git Clone

Git allows you to manage and version your project(s) in a "repository". This repository is stored on a web-based hosting service for version control, like GitHub.

You can then clone this repository to your local machine and have all the files and branches locally (I'll explain more about branches soon).



For example, you can clone freeCodeCamp's repository with SSH like so:

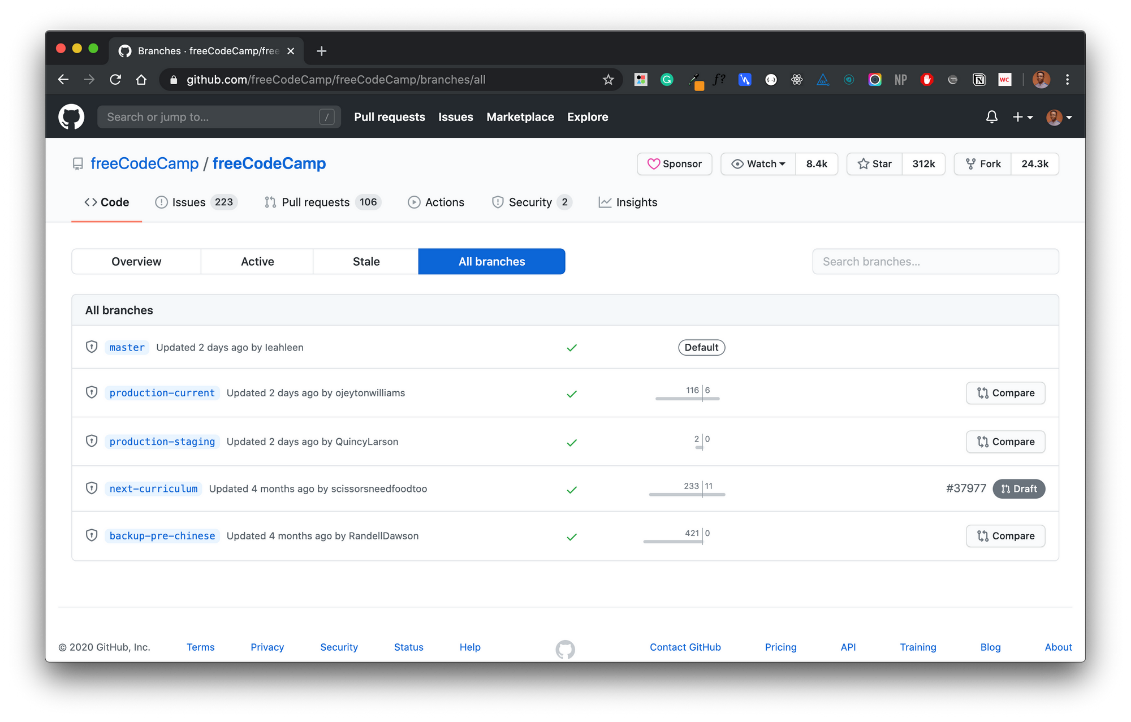
git clone git@github.com:freeCodeCamp/freeCodeCamp.git

## Introduction to Git Branches

When working on a project, you will likely have different features. And multiple contributors will be working on this project and its features.

Branches allow you to create a "playground" with the same files in the master branch. You can use this branch to build independent features, test new features, make breaking changes, create fixes, write docs or try out ideas without breaking or affecting the production code. When you're done, you merge the branch into the production master branch.

Branching is a core concept in Git which is also used in GitHub to manage workflows of different versions of one project. The master branch is always the default branch in a repository that is most often considered "production and deployable code". New branches like passwordless-auth or refactor-signup-ux can be created from the master branch.



All branches in freeCodeCamp's repository

## How to Clone Git Branches

While you can clone repositories with the git clone command, keep in mind that this clones the branch and the remote HEAD. This is usually master by default and includes all other branches in the repository.

So when you clone a repository, you clone the master and all other branches. This means you will have to checkout another branch yourself.

Let's say your task on a project is to work on a feature to add passwordless authentication to a user dashboard. And this feature is in the passwordless-auth branch.

You really don't need the master branch since your "feature branch" will be merged into master afterward. How then do you clone this passwordless-auth branch without fetching all other branches with "a bunch of files you don't need"?

I created this sample repository to explain this. This repository holds a simple blog built with Nextjs and has four dummy branches:

* master
* dev
* staging
* passwordless-auth

In Nextjs, any file inside the folder pages/api is mapped to the /api/\* path and will be treated as an API endpoint instead of a page. In our repository, I have created different dummy APIs [in this directory](https://github.com/BolajiAyodeji/nextjs-blog/tree/master/pages/api) to make each branch different.

The master branch holds the file **pages/api/hello.js** while passwordless-auth holds the file **pages/api/auth.js**. Each file just returns a dummy text response. See master's hello API response [here](https://nextjs-blog.bolajiayodeji.vercel.app/api/hello) (with a special message for you ?).

Let's clone the repository:

git clone git@github.com:BolajiAyodeji/nextjs-blog.git

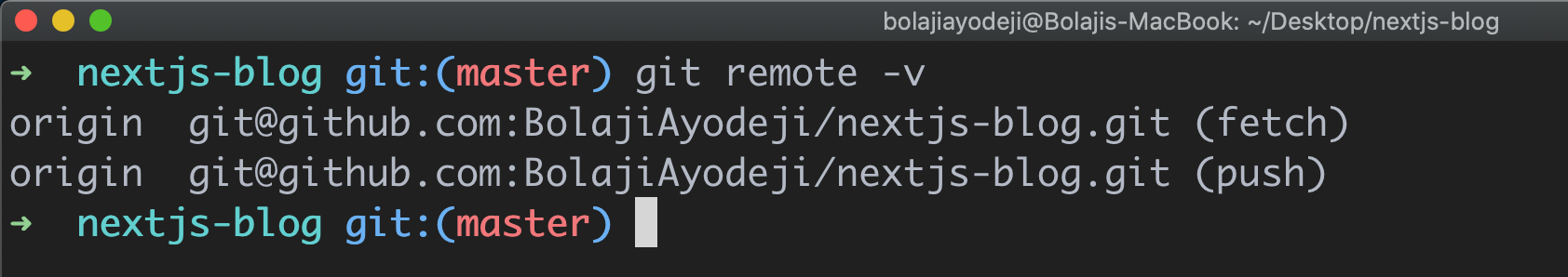
This gives us access to all branches in this repository and you can easily toggle between each to see each version and its files.

git branch -a

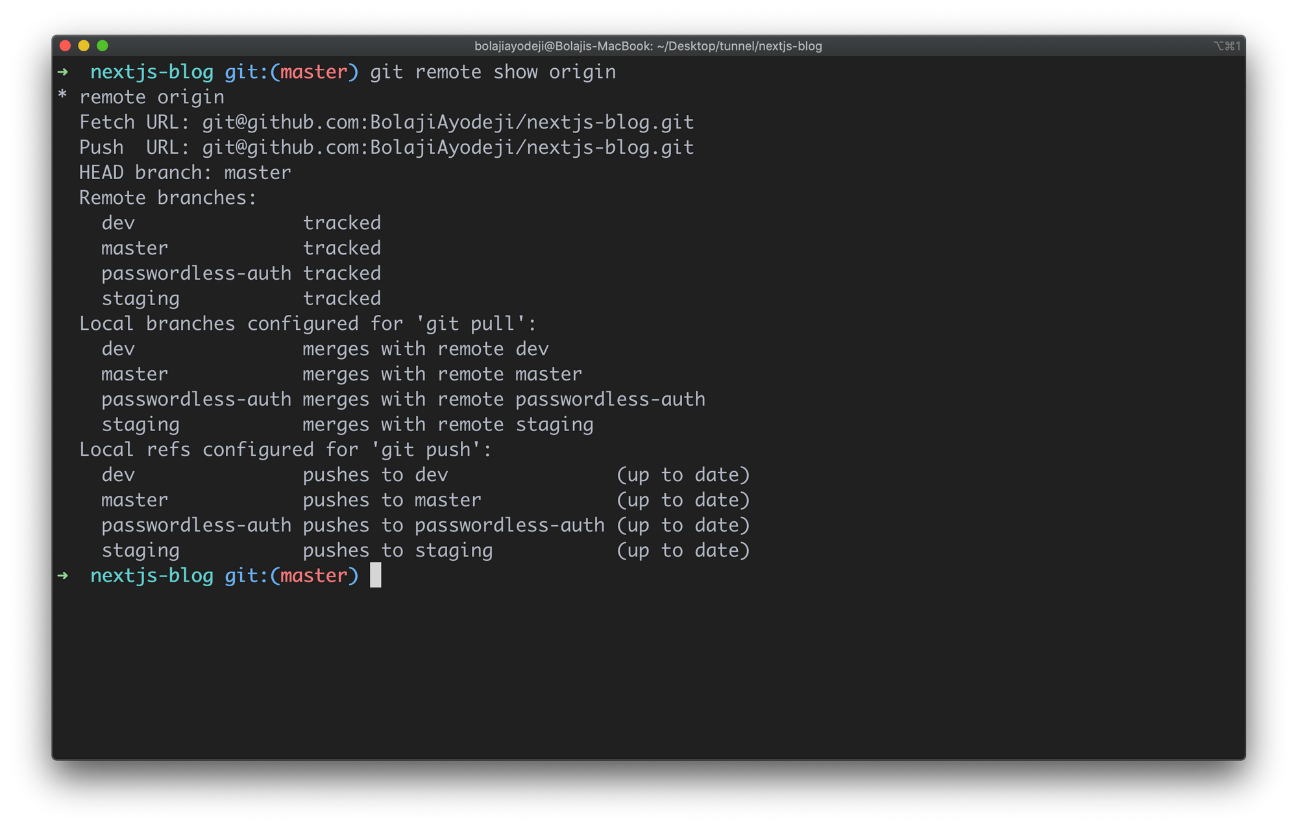


Wondering where the **remotes/origin/..** branches came from?  
  
When you clone a repository, you pull data from a repository on the internet or an internal server known as the **remote**. The word origin is an alias created by your Git to replace the remote URL (you can change or specify another alias if you want).

These **remotes/origin/..**branches point you back to the origin repository you cloned from the internet so you can still perform pull/push from the origin.



So when you clone master onto your machine, remotes/origin/master is the original master branch on the internet, and master is on your local machine. So you will pull/push from and to the remotes/origin/master.  
  
In summary **Remote** is the URL that points you to the repository on the internet while **Origin** is an alias for this remote URL.



## How to Clone a Specific Branch

Now let's clone a specific branch from our demo repository. There are two ways to clone a specific branch. You can either:

* Clone the repository, fetch all branches, and checkout to a specific branch immediately.
* Clone the repository and fetch only a single branch.

### Option One

git clone --branch <branchname> <remote-repo-url>

or

git clone -b <branchname> <remote-repo-url>

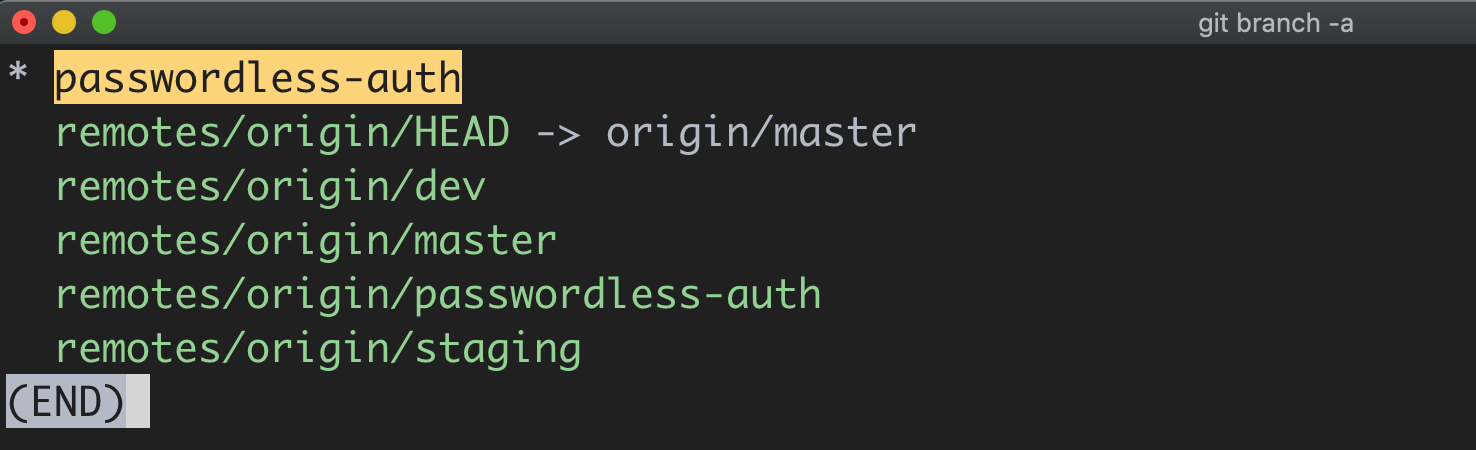
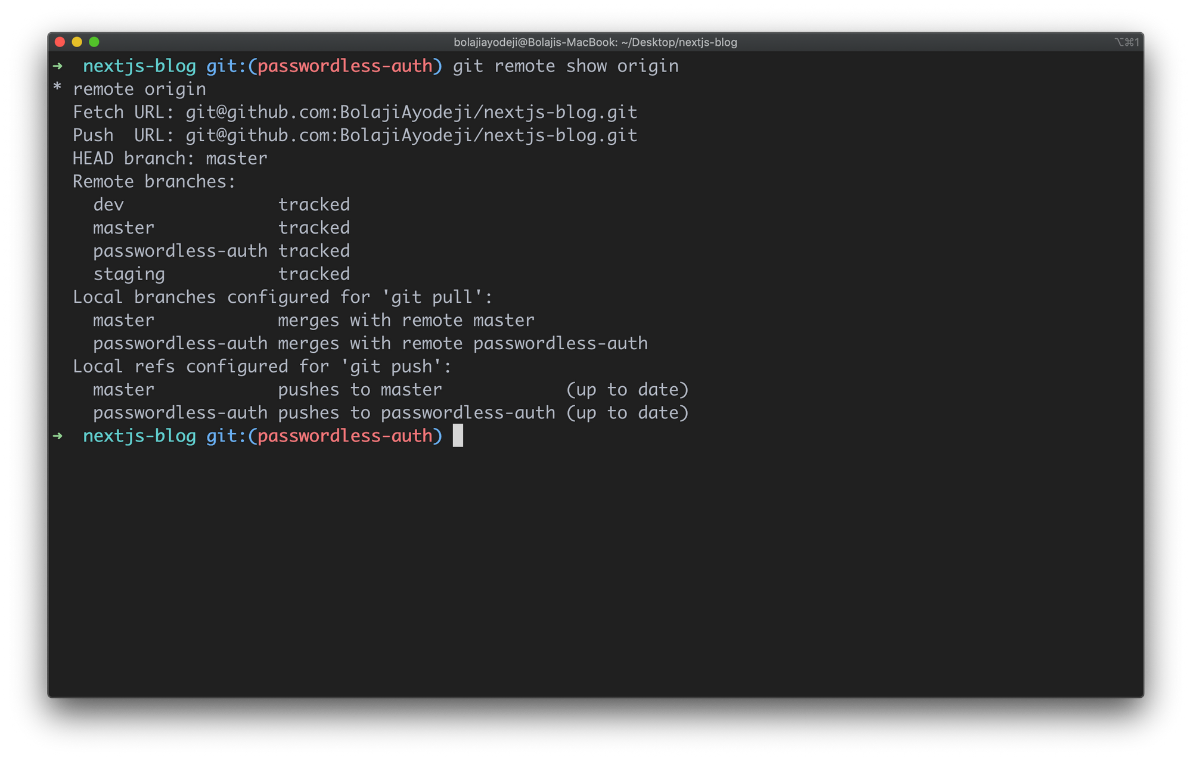
Here **-b** is just an alias for **--branch**

With this, you fetch all the branches in the repository, checkout to the one you specified, and the specific branch becomes the configured local branch for git push and git pull . But you still fetched all files from each branch. This might not be what you want right? ?

Let's test it:

git clone -b passwordless-auth git@github.com:BolajiAyodeji/nextjs-blog.git

This automatically configures passwordless-auth as the local branch but still tracks other branches.



### Option Two

git clone --branch <branchname> --single-branch <remote-repo-url>

or

git clone -b <branchname> --single-branch <remote-repo-url>

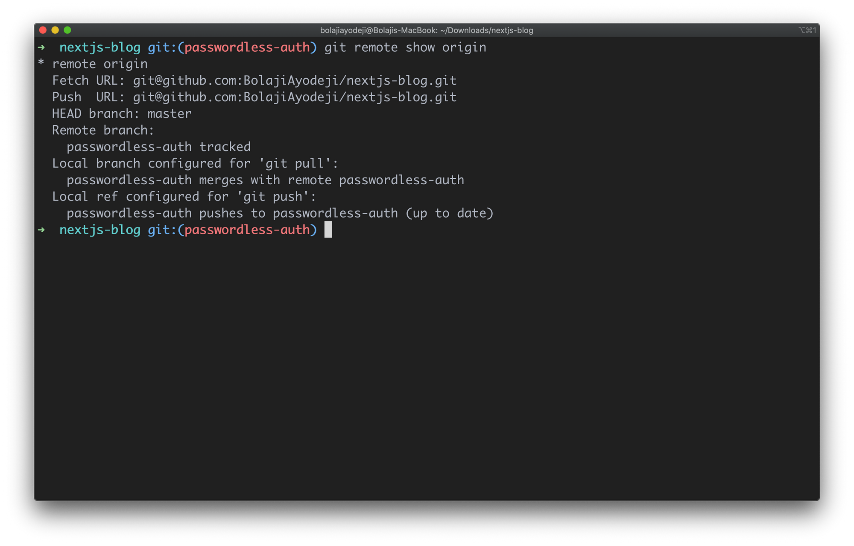
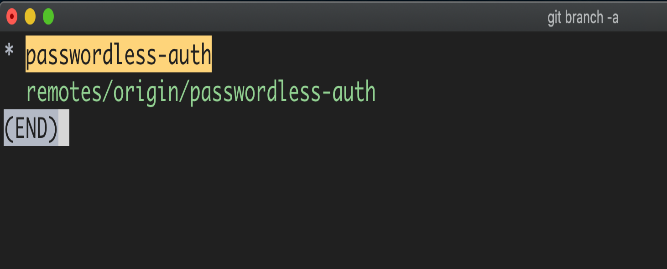
Here **-b** is just an alias for **--branch**

This performs the same action as option one, except that the --single-branch option was introduced in Git version 1.7.10 and later. It allows you to only fetch files from the specified branch without fetching other branches.

Let's test it:

git clone -b passwordless-auth --single-branch git@github.com:BolajiAyodeji/nextjs-blog.git

This automatically configures passwordless-auth as the local branch and only tracks this branch.



If you run cd pages/api you'll find the auth.js file in the passwordless-auth branch as expected from the previous setup.

## Conclusion

You might be running out of internet or storage space but you need to work on a task in a specific branch. Or you might want to clone a specific branch with limited files for various reasons. Fortunately, Git provides you the flexibility to do this. Flex your muscles and try it out, there's much more "Git" to learn.

a