

# 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/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](#) 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 VCS systems 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](#) 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](#) will introduce all of the available options the server can set up to access your Git repository and the pros and cons of each.