AUGUST 23, 2022 / #GIT

.gitignore File – How to Ignore Files and Folders in Git

Dionysia Lemonaki

Git is a popular version control system. It is how developers can collaborate and work together on projects.

Git allows you to track the changes you make to your project over

time. On top of that, it lets you revert to a previous version if you want to undo a change.

The way Git works is that you stage files in a project with the `git add` command and then commit them with the `git commit` command.

When working on a project as part of a team, there will be times when you don't want to share some files or parts of the project with others.

In other words, you don't want to include or *commit* those specific files to the main version of the project. This is why you may not want to use the period `.` with the `git add` command as this stages every single file in the current Git directory.

When you use the `git commit` command, every single file gets committed – this also includes files that do not need to be or shouldn't be.

You may instead want Git to ignore specific files, but there is no `git ignore` command for that purpose.

So, how do you tell Git to ignore and not track specific files? With a `.gitignore` file.

In this article, you will learn what a `.gitignore` file is, how to create one, and how to use it to ignore files and folders. You will also see how you can ignore a previously committed file.

Here is what we will cover:

1. What is a `.gitignore` file?
    1. How to create a `.gitignore` file

2. What to Include in a `.gitignore` file?

2. How to ignore a file and folder in Git
    1. How to ignore a previously committed file

What Is a `.gitignore` File? What Is a `.gitignore` File Used For?

Each of the files in any current working Git repository is either:

- **tracked** – these are all the files or directories Git knows about. These are the files and directories newly staged (added with `git add`) and committed (committed with `git commit`) to the main repo.
- **untracked** – these are any new files or directories created in the working directory but that have not yet been staged (or added using the `git add` command).
- **ignored** – these are all the files or directories that Git knows to completely exclude, ignore, and not be aware of in the Git repository. Essentially, this is a way to tell Git which untracked files should remain untracked and never get committed.

All ignored files get stored in a `.gitignore` file.

A `.gitignore` file is a plain text file that contains a list of all the specified files and folders from the project that Git should ignore and not track.

Inside `.gitignore`, you can tell Git to ignore only a single file or a single folder by mentioning the name or pattern of that specific file

or folder. You can also tell Git to ignore multiple files or folders using the same method.

How to Create a `.gitignore` File

Typically, a `.gitignore` file gets placed in the root directory of the repository. The root directory is also known as the parent and the current working directory. The root folder contains all the files and other folders that make up the project.

That said, you can place it in any folder in the repository. You can even have multiple `.gitignore` files, for that matter.

To create a `.gitignore` file on a Unix-based system such as macOS or Linux using the command line, open the terminal application (such as Terminal.app on macOS). Then, navigate to the root folder that contains the project using the `cd` command and enter the following command to create a `.gitignore` file for your directory:

```
touch .gitignore
```

Files with a dot (`.`) preceding their name are hidden by default.

Hidden files are not visible when using the `ls` command alone. To view all files – including hidden ones – from the command line, use the `-a` flag with the `ls` command like so:

```
ls -a
```

What to Include in a `.gitignore` File

The types of files you should consider adding to a `.gitignore` file are any files that do not need to get committed.

You may not want to commit them for security reasons or because they are local to you and therefore unnecessary for other developers working on the same project as you.

Some of these may include:

- Operating System files. Each Operating System (such as macOS, Windows, and Linux) generates system-specific hidden files that other developers don't need to use since their system also generates them. For example, on macOS, Finder generates a `.DS_Store` file that includes user preferences for the appearance and display of folders, such as the size and position of icons.
- Configuration files generated by applications such as code editors and IDEs (IDE stands for Integrated Development Environment). These files are custom to you, your configurations, and your preferences settings.
- Files that get automatically generated from the programming language or framework you are using in your project and compiled code-specific files, such as `.o` files.
- Folders generated by package managers, such as npm's `node_modules` folder. This is a folder used for saving and tracking the dependencies for each package you install locally.
- Files that contain sensitive data and personal information. Some examples of such files are files with your credentials (username and password) and files with environment variables like `.env` files (`.env` files contain API keys that

need to remain secure and private).

- Runtime `files, such as .log` files. They provide information on the Operating System's usage activities and errors, as well as a history of events that have taken place within the OS.

How to Ignore a File and Folder in Git

If you want to ignore only one specific file, you need to provide the full path to the file from the root of the project.

For example, if you want to ignore a `text.txt` file located in the root directory, you would do the following:

```
/text.txt
```

And if you wanted to ignore a `text.txt` file located in a `test` directory at the root directory, you would do the following:

```
/test/text.txt
```

You could also write the above like so:

```
test/text.txt
```

If you want to ignore all files with a specific name, you need to write the literal name of the file.

For example, if you wanted to ignore any `text.txt` files, you would add the following to `.gitignore`:

```
text.txt
```

In this case, you don't need to provide the full path to a specific file. This pattern will ignore all files with that particular name that are located anywhere in the project.

To ignore an entire directory with all its contents, you need to include the name of the directory with the slash `/` at the end:

```
test/
```

This command will ignore any directory (including other files and other sub-directories inside the directory) named `test` located anywhere in your project.

Something to note is that if you write the name of a file alone or the name of the directory alone without the slash `/`, then this pattern will match both any files or directories with that name:

```
# matches any files and directories with the name test
test
`
```

What if you want to ignore any files or directories that *start* with a

specific word?

Say that you want to ignore all files and directories that have a name starting with `img`. To do this, you would need to specify the name you want to ignore followed by the `*` wildcard selector like so:

```
img*
```

This command will ignore all files and directories that have a name starting with `img`.

But what if you want to ignore any files or directories that *end* with a specific word?

If you wanted to ignore all files that end with a specific file extension, you would need to use the `*` wildcard selector followed by the file extension you want to ignore.

For example, if you wanted to ignore all markdown files that end with a `.md` file extension, you would add the following to your `.gitignore` file:

```
*.md
```

This pattern will match any file ending with the `.md` extension located anywhere in the project.

Earlier, you saw how to ignore all files ending with a specific suffix. What happens when you want to make an exception, and there is one file with that suffix that you *don't* want to ignore?

Say you added the following to your `.gitignore` file:

```
.md
```

This pattern ignores all files ending in `.md`, but you *don't* want Git to ignore a `README.md` file.

To do this, you would need to use the negating pattern with an exclamation mark, `!`, to negate a file that would otherwise be ignored:

```
# ignores all .md files
.md

# does not ignore the README.md file
!README.md
```

With both of those patterns in the `.gitignore` file, all files ending in `.md` get ignored except for the `README.md` file.

Something to keep in mind is that this pattern will not work if you ignore an entire directory.

Say that you ignore all `test` directories:

```
test/
```

Say that inside a `test` folder, you have a file, `example.md`, that you *don't* want to ignore.

You *cannot* negate a file inside an ignored directory like so:

```
# ignore all directories with the name test
test/

# trying to negate a file inside an ignored directory won't work
!test/example.md
```

How to Ignore a Previously Committed File

It's a best practice to create a `.gitignore` file with all the files and the different file patterns you want to ignore when you create a new repository – before committing it.

Git can only ignore untracked files that haven't yet been committed to the repository.

What happens when you have already committed a file in the past and wish you hadn't?

Say you accidentally committed a `.env` file that stores environment variables.

You first need to update the `.gitignore` file to include the `.env` file:

```
# add .env file to .gitignore
echo ".env" >> .gitignore
```

Now, you will need to tell Git not to track this file by removing it from the index:

```
git rm --cached .env
```

The `git rm` command, along with the `--cached` option, deletes the file from the repository but does not delete the actual file. This means the file remains on your local system and in your working directory as an ignored file.

A `git status` will show that the file is no longer in the repository, and entering the `ls` command will show that the file exists on your local file system.

If you want to delete the file from the repository and your local system, omit the `--cached` option.

Next, add the `.gitignore` to the staging area using the `git add` command:

```
git add .gitignore
```

Finally, commit the `.gitignore` file using the `git commit` command:

```
git commit -m "update ignored files"
```

Conclusion

And there you have it – you now know the basics of ignoring files and folders in Git.

Hopefully, you found this article helpful.

To learn more about Git, check out the following free resources:

- Git and GitHub for Beginners - Crash Course
- Git for Professionals Tutorial - Tools & Concepts for Mastering Version Control with Git
- Advanced Git Tutorial - Interactive Rebase, Cherry-Picking, Reflog, Submodules and more

Thank you for reading and happy coding :)

Dionysia Lemonaki

Read more posts.

---

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) charity organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can make a tax-deductible donation here.

Trending Books and Handbooks

- Learn CSS Transform
- What is Programming?
- HTTP Networking in JS
- How to Write Clean Code
- Learn Swift
- Learn CSS Grid
- Learn JS Modules
- Front-End JS Development
- Command Line for Beginners
- OSS Security Best Practices
- Build a Static Blog
- Python Code Examples
- Write React Unit Tests
- Learn PHP
- Learn Golang
- Learn Solidity
- Learn Apache Kafka
- Learn to Build REST APIs
- Intro to Operating Systems
- Distributed Systems Patterns
- Build an AI Chatbot
- Open Source for Devs
- Learn Algorithms in JS
- Learn Java
- Learn Node.js
- Learn Express.js
- REST API Best Practices
- Intermediate TS and React
- Learn to Build GraphQL APIs
- Software Architecture Patterns

Mobile App

Download on the App Store

GET IT ON Google Play

Our Charity

About Alumni Network Open Source Shop Support Sponsors Academic Honesty