.gitignore

[(https://github.com/learn-co-curriculum/git-github-gitignore)](https://github.com/learn-co-curriculum/git-github-gitignore) [(https://github.com/learn-co-curriculum/git-github-gitignore/issues/new)](https://github.com/learn-co-curriculum/git-github-gitignore/issues/new)

Learning Goals

- Describe use cases for a `.gitignore` file.
- Use a `.gitignore` file to identify files that should not be tracked by Git.

Introduction

In an earlier lesson, we mentioned that sometimes we don't want Git to track changes for all of the files in our repository. Git allows us to tell it to ignore specific files or directories in our project by using a `.gitignore` file. In this lesson, we'll discuss the types of things that might go into a `.gitignore` file as well as how to create one.

Why `.gitignore` ?

If version control systems like Git are designed to track all of your changes, why would we want to tell it to ignore some files? There are three main reasons:

1. Private files
2. Irrelevant files
3. Files that are too large for GitHub

Private Files

(https://commons.wikimedia.org/wiki/File:Breezeicons-actions-22-view-certificate.svg)

In some cases, you will need credentials in order for your code to access some service. This includes free services! For example, if you are using a rate-limited API, the API needs to know who is making the request in order to determine whether you have exceeded your limit of requests.

These credentials might be in the form of *keys* or *secrets*, often stored in JSON or YML file formats.

You want to avoid pushing these files to public GitHub repositories, because people have set up bots to crawl GitHub in order to steal these valuable credentials!

Using `.gitignore` you can make sure that your credentials are not pushed.

Irrelevant Files

(https://commons.wikimedia.org/wiki/File:Breezeicons-actions-22-noisereduction.svg)

Some of the files generated in the process of coding are not actually relevant for future users of the repository.

These include:

- Log files;
- OS-specific files (e.g. `.DS_Store` , which is only applicable to Mac computers); and
- Configuration files for code editors (e.g. `.vscode` , which is used by VS Code).

If you are working with collaborators, not only are these files *not useful* to them, they can actually cause problems in merging together the work of different people on the team! (These are known as *merge conflicts*, which we'll return to when we learn about branching.)

Files That Are Too Large for GitHub

(https://commons.wikimedia.org/wiki/File:Breezeicons-actions-22-project-development-close.svg)

GitHub **limits the size** **(https://docs.github.com/en/repositories/working-with-files/managing-large-files/about-large-files-on-github)** of files that can be pushed to its repositories. If you try to push a file that exceeds the 100 MB threshold, you will get an error message and the push will fail. An example might be large data files.

`.gitignore` Syntax

Now that we understand some reasons that we might want to ignore certain files, how does that actually work?

To start with, you put a hidden text file called `.gitignore` at the root of the repository. It is hidden because it starts with `.`, which means that in order to view it you will need to type `ls -a` rather than just `ls` in the terminal. We demonstrate the `.gitignore` in this very repository below:

```
$ ls
    CONTRIBUTING.md  LICENSE.md  README.md
```

```
$ ls -a

.               ..              .git            .gitignore      CONTRIBUTING.md LICENSE.md      README.md
```

If you take a look at the `.gitignore` in this repo, you'll see that it contains a list of things to ignore, as well as whitespace and comments (which start with `#`). The items in the list are separated by newlines. You'll also see examples of the types of files discussed above: irrelevant files (e.g., `.DS_Store`, log files) and files that are too large to be pushed to GitHub (e.g., database files).

If we need to add additional files — for example, if we have a YML file containing our credentials for an API — we could add it like this:

```
# credential files
secrets.yml
```

Alternatively, if we want to make sure that our `secrets` file is ignored whether it's stored as YML or JSON, we could use **glob syntax (https://en.wikipedia.org/wiki/Glob_(programming))**, which allows us to use the `*` wildcard character:

```
# credential files
secrets.*
```

This will ignore any file with the name `secrets`, regardless of its format.

Another option is to use the `*` to tell Git to ignore files of a certain type. For example, if we add the following to our `.gitignore`, all CSV files will be ignored:

```
# ignore all CSV files
*.csv
```

Or, if we only wanted to ignore CSV files that are in the `data/` folder, we could do that like this:

```
# ignore CSV files in the data/ folder
data/*.csv
```

For more examples with comments, see the **Git documentation (https://git-scm.com/docs/gitignore)**.

Useful Defaults

The examples above were specific to a particular project, and you will typically need to write your own `.gitignore` lines for these specialized use cases.

Outside of these narrow use cases, there are resources available for developing `.gitignore` files that are appropriate for your type of project in general. For example, GitHub maintains templates for a wide range of languages and tools (e.g., **Python (https://github.com/github/gitignore/blob/main/Python.gitignore)**, **JavaScript (https://github.com/github/gitignore/blob/main/Node.gitignore)**).

There is also a tool called **gitignore.io (https://gitignore.io)** that will help you identify useful `.gitignore` lines for your operating system or code editor. If you take a look at the top line of the `.gitignore` file in this repo, you'll see that's where it came from! It's usually a good idea to make use of these resources, and then add anything else you need for your particular use case.

Ignoring a Tracked File

If you previously used `git add` on a file then ran `git commit`, that means that the file is already tracked, regardless of what the `.gitignore` says, and you will need to tell Git to stop tracking it.

Irrelevant Files

If the file is just irrelevant for your collaborators, you can use the `rm` command to tell Git to stop tracking it. For example, if you wanted to tell Git to stop tracking `log.txt`, you could use this command:

```
$ git rm --cached log.txt
```

Note that this is a command you run in the terminal, *not* a line to be added to the `.gitignore` file.

For the changes to be reflected on GitHub, you will also need to run `git commit` and `git push`. (You also probably want to add that file name to the `.gitignore` at the same time!)

> A note about the `git rm` command: if you run `git rm <filename>` the file will be deleted from the directory, and that file deletion will be staged for the next commit. Adding `--cached` to `git rm` means that Git will no longer *track* the file, but it will not actually be deleted from the directory. In either case, the deleted or unstaged file will still be visible in your Git history.

Private or Too Large Files

If the file is private/secret or too large for GitHub, you will need to rewrite your Git history. This can be fairly complicated (see this **blog post (https://medium.com/analytics-vidhya/tutorial-removing-large-files-from-git-78dbf4cf83a?sk=c3763d466c7f2528008c3777192dfb95)** for more details) so it's always better to add things to `.gitignore` sooner rather than later!

Telling Git Not to Ignore

Many developer setups will include a global `.gitignore` file. This file is not located within a specific repository, and can be especially useful for OS-related settings (e.g. ignoring `.DS_Store` on a Mac).

Sometimes you might want to override these global settings for the current repository, particularly if your global `.gitignore` is relatively broad. To do that, you use the same glob syntax, placing a `!` at the beginning.

So, for example, if you want Git *not* to ignore `log.txt` in a particular repository, you could add this to the `.gitignore` :

```
# do not ignore log.txt
!log.txt
```

Example

Let's go back to our todo's project that we created earlier and get it set up with a `.gitignore` file. Start by navigating back into that directory, then create a new file, `secrets.yml` , and add the following to it:

```
secret_password: 12345678
```

Run `git status` ; you should see the following:

```
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    secrets.yml

nothing added to commit but untracked files present (use "git add" to track)
```

Next, create the `.gitignore` file and run `git status` again. You should now see both files listed as untracked:

```
$ git status
On branch main
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    secrets.yml

nothing added to commit but untracked files present (use "git add" to track)
```

Finally, add the following to the `.gitignore` file and save it:

```
# credential files
secrets.yml
```

Now, if you run `git status` one more time, you'll see that Git is "ignoring" the `secrets.yml` file, just as we want:

```
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

We can now continue to make changes and commit them, without worrying that our secret password will accidentally get posted to GitHub.

Check for Understanding

Before moving on to the next lesson, check for your understanding of this material by describing in your own words what a `.gitignore` file enables us to do and why it is important.

Summary

Sometimes you want Git to ignore certain files. The common reasons for this are that the files are private, irrelevant to collaborators, or too big for GitHub. To tell Git to ignore the files, you use a hidden file called `.gitignore` at the root of the repository. This file uses glob syntax to specify which files should be ignored, including `*` to indicate a wildcard and `!` to indicate that a file should *not* be ignored.

We recommend that you use available tools such as **gitignore.io** (https://gitignore.io) rather than writing your own `.gitignore` lines most of the time, and that you always make sure to add private or too-large files to the `.gitignore` as soon as possible so you don't have to rewrite your Git history.