

5 Ignoring Files in Git Written by Bhagat Singh & Chris Belanger

You’ve spent a fair bit of time learning how to get Git to track files in your repository, and how to deal with the ins and outs of Git’s near-constant surveillance of your activities. So it might come as a wonder that you’d ever want Git to actively *ignore* things in your repository. Why wouldn’t you want Git to track everything in your project? Well, there are quite a few situations in which you might not want Git to track *everything*. A good example would be any files that contain API keys, tokens, passwords or other secrets that you definitely need for testing, but you don’t want them sitting in a repository — especially a public repository — for all to see. Depending on your development platform, you may have lots of build artifacts or generated content sitting around inside your project directory, such as linker files, metadata, the resulting executable and other similar things. These files are regenerated each time you build your project, so you definitely don’t want Git to track these files. And then there are those persnickety things that some OSes add into your directories without asking, such as **.DS_Store** files on macOS.

Introducing .gitignore

Git’s answer to this is the **.gitignore** file, which is a set of rules held in a file that tell Git to not track files or sets of files. That seems like a very simple solution, and it is. But the real power of **.gitignore** is in its ability to pattern-match a wide range of files so that you don’t have to spell out every single file you want Git to ignore, and you can even instruct Git to ignore the same types of files across multiple projects. Taking that a step further, you can have a global **.gitignore** that applies to all of your repositories, and then put project-specific **.gitignore** files within directories or subdirectories under the projects that need a particularly pedantic level of control. In this chapter, you’ll learn how to configure your own **.gitignore**, how to use some prefabricated **.gitignore** files from places like GitHub, and how to set up a global **.gitignore** to apply to all of your projects.

Getting started

Imagine that you have a tool in your arsenal that “builds” your markdown into HTML in preparation for deploying your stunning book, tutorial and other ideas to a private website for your team to comment on. In this case, the HTML files would be the generated content that you *don’t* want to track in the repository. You’d like to render them locally as part of your build process so you could preview them, but you’d never edit the HTML directly: It’s always rendered using the tool. Create a new directory in the root folder of your project to hold these generated files, using the following command:

```
mkdir sitehtml
```

Now, create an empty HTML file in there (keep that imagination going, friend), with the following command:

```
touch sitehtml/all-todos.html
```

```
Run git status to see that Git recognizes the new content:
~/GitApprentice/ideas $ git status
On branch main
Your branch is ahead of 'origin/main' by 7 commits.
(use "git push" to publish your local commits)
```

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

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

So Git, once again, sees what you’re doing. But here’s how to tell Git to turn a blind eye. Create a new file named **.gitignore** in the root folder of your project:

```
touch .gitignore
```

And add the following line to your newly created **.gitignore** using a text editor:

```
*.html
```

Save and exit. What you’ve done is to tell Git, “For this project, ignore all files that match this pattern.” In this case, you’ve asked it to ignore all files that have an **.html** extension. Now, see what `git status` tells you:

```
~/GitApprentice/ideas $ git status
On branch main
Your branch is ahead of 'origin/main' by 7 commits.
(use "git push" to publish your local commits)
```

```
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)
```

Git sees that you’ve added **.gitignore**, but it no longer views that HTML file as “untracked,” even through it’s buried down in a subdirectory. Now, what if you were fine with ignoring HTML files in subdirectories, but you wanted all HTML files in the top-level directory of your project to be tracked? You *could* theoretically re-create the same **.gitignore** files in each of your subdirectories and remove this top-level **.gitignore**, but that would be amazingly tedious and would not scale well. Instead, you can use some clever pattern-matching in your top-level **.gitignore** to only ignore subdirectories. Edit the single line in your **.gitignore** as follows:

```
*/*.html
```

Save and exit. This new pattern tells Git, “Ignore all HTML files that *aren’t* in the top-level directory.” To see that this is true, create a new HTML file in the top-level directory of your project:

```
touch index.html
```

Run `git status` to see if Git does, in fact, recognize the HTML files in the top-level directory, while still ignoring the ones underneath:

```
~/GitApprentice/ideas $ git status
On branch main
Your branch is ahead of 'origin/main' by 7 commits.
(use "git push" to publish your local commits)
```

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

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

Git sees the top-level HTML file as untracked, but it’s still ignoring the other HTML file down in the **sitehtml** directory, just as you’d planned.

Nesting .gitignore files

You can easily nest **.gitignore** files in your project. Imagine that you have a subdirectory with HTML files that are referenced from your **index.html**. These aren’t generated by your imaginary build process but, rather, maintained by hand, and you want to make sure Git is able to track these. Create a new directory and name it **htmlrefs**:

```
mkdir htmlrefs
```

Now, create an HTML file in that subdirectory:

```
touch htmlrefs/utils.html
```

And create a **.gitignore** file in that directory as well:

```
touch htmlrefs/.gitignore
```

Open `htmlrefs/.gitignore` and add the following line to it:

```
!/*.html
```

Save and exit. The exclamation mark (!) negates the pattern in this case, and the slash (/) means “start this rule from this directory.” So this rule says, “Despite any higher-level rules, don’t ignore any HTML files, starting in this directory or lower.”

```
Execute git status to see if this is true:
~/GitApprentice/ideas $ git status
On branch main
Your branch is ahead of 'origin/main' by 7 commits.
(use "git push" to publish your local commits)
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
.gitignore
htmlrefs/
index.html
```

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

Git now sees the contents of your **htmlrefs** directory as untracked, just as you wanted. Now that you’re happy with the current arrangement of your **.gitignore** files, you can stage and commit those changes. Stage all changes with the following command:

```
git add .
```

And commit those changes as well:

```
git commit -m "Adding .gitignore files and HTML"
```

Setting up **.gitignore** files on a project-by-project basis will only get you so far, though. There are things — like the aforementioned **.DS_Store** files that macOS *so* helpfully adds to your directories — that you want to ignore all of the time. Git has the concept of a **global .gitignore** that you can use for cases like this.

Looking at the global .gitignore

Execute the following command to find out if you already have a global **.gitignore**:

```
git config --global core.excludesfile
```

If that command returns nothing, then you don’t have one set up just yet. No worries; it’s easy to create one. Create a file in a convenient location — in this case, your home directory — and name it something obvious:

```
touch ~/.gitignore_global
```

And now you can use the `git config` command to tell Git that it should look at this file from now on as your global **.gitignore**:

```
git config --global core.excludesfile ~/.gitignore_global
```







So now if I ask Git where my global **.gitignore** lives, it tells me the following:

```
~/GitApprentice/ideas $ git config --global core.excludesfile
/Users/chrisbelanger/.gitignore_global
```

But now that you have a global **.gitignore**... what should you put in it?

Finding sample .gitignore files

This is one of those situations wherein you don’t have to reinvent the wheel. Hundreds of thousands of developers have come before you, and they’ve already figured out what the best configuration is for your particular situation. One of the better collections of prefabricated **.gitignore** files is hosted by GitHub — no surprise there, I’m sure. GitHub has files for most OSes, programming languages and code editors. Head over to <https://github.com/github/gitignore> and have a look through the packages it offers. Sample files that are appropriate for your OS can be found in the **Global** subfolder of the repository. Go into the **Global** subfolder (or simply navigate to <https://github.com/github/gitignore/tree/master/Global>) and find the one for your local system.

 VisualStudioCode.gitignore	Modified VS Code .gitignore	2 years ago
 WebMethods.gitignore	Capitalise initial letter in template filenames for consistency/sorting	4 years ago
 Windows.gitignore	Add a new .msix extension	10 months ago
 Xcode.gitignore	Revert "Update Xcode.gitignore"	3 months ago
 XilinxISE.gitignore	Update to include iMPACT and Core Generator files	3 years ago
 macOS.gitignore	macOS low cap m	2 months ago

There’s a **Windows.gitignore**, a **macOS.gitignore**, a **Linux.gitignore** and many more, all waiting for you to add them to your own **.gitignore**. And that brings you to the challenge for this chapter!

Challenge

Challenge: Populate your global .gitignore

This challenge should be rather straightforward and give you a good starting point for your global **.gitignore**. Your goal is to find the correct **.gitignore** for your own OS, get that file from the GitHub repository, and add the contents of that file to your global **.gitignore**.

. Navigate to <https://github.com/github/gitignore/tree/master/Global>.

. Find the correct **.gitignore** for your own OS.

. Take the contents of that OS-specific **.gitignore**, and add it to your own global **.gitignore**.

If you get stuck, or want to check your solution, you can always find the answer to this challenge under the **challenge** folder for this chapter.

Key points

.gitignore lets you configure Git so that it ignores specific files or files that match a certain pattern.

*.html in your **.gitignore** matches on all files with an **.html** extension, in any directory or subdirectory of your project.

/.html matches all files with an **.html** extension, but only in subdirectories of your project.

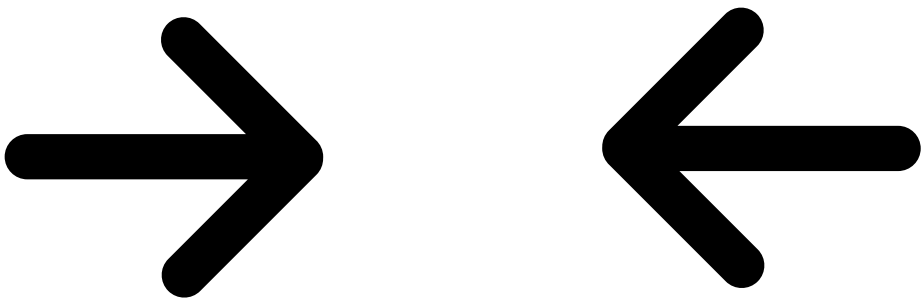
! negates a matching rule.

You can have multiple **.gitignore** files inside various directories of your project to override higher-level matches in your project.

You can find where your global **.gitignore** lives with the command `git config --global core.excludesfile`. GitHub hosts some excellent started **.gitignore** files at <https://github.com/github/gitignore>.

Where to go from here?

As you work on more and more complex projects, especially across multiple code-based and coding languages, you'll find that the power of the global **.gitignore**, coupled with the project-specific (and even folder-specific) **.gitignore** files, will be an indispensable part of your Git workflow. The next chapter will take you through a short diversion into the various workings of `git log`. Yes, you've already used this command, but this command has some clever options that will help you view the history of your project in an efficient and highly readable manner.



6. Git Log & History

Have a technical question? Want to report a bug? You can ask questions and report bugs to the book authors in our official book forum [here](#).

© 2022 Razeware LLC

4. The Staging Area