# Addign .gitignore file

Often, when working on a project that uses Git, you'll want to exclude specific files or directories from being pushed to the remote repository. This is where `.gitignore` file comes in handy.

The `.gitignore` file specifies what untracked files Git should ignore.

**What Files Should be Ignored?**

Ignored files are usually platform-specific files or automatically created files from the build systems. Some common examples include:

- Runtime files such as log, lock, cache, or temporary files.
- Files with sensitive information, such as passwords or API keys.
- Compiled code, such as `.class` or `.o`.
- Dependency directories, such as `/vendor` or `/node_modules` .
- Build directories, such as `/public`, `/out`, or `/dist`.
- System files like `.DS_Store` or `Thumbs.db`
- IDE or text editor configuration files.

**`.gitignore` Patterns**

`.gitignore` is a plain text file in which each line contains a pattern for files or directories to ignore.

It uses globbing patterns to match filenames with wildcard characters. If you have files or directories containing a wildcard pattern, you can use a single backslash (`\`) to escape the character.

**Comments**

Lines starting with a hash mark (`#`) are comments and are ignored. Empty lines can be used to improve the readability of the file and to group related lines of patterns.

**Slash**

The slash symbol (`/`) represents a directory separator. The slash at the beginning of a pattern is relative to the directory where the `.gitignore` resides.

If the pattern starts with a slash, it matches files and directories only in the repository root.

If the pattern doesn't start with a slash, it matches files and directories in any directory or subdirectory.

If the pattern ends with a slash, it matches only directories. When a directory is ignored, all of its files and subdirectories are also ignored.

### Literal File Names

The most straightforward pattern is a literal file name without any special characters.

| Pattern | Example matches |
|---|---|
| `/access.log` | `access.log` |
| `access.log` | `access.log`<br>`logs/access.log`<br>`var/logs/access.log` |
| `build/` | `build` |

### Wildcard Symbols

`*` - The asterisk symbol matches zero or more characters.

| Pattern | Example matches |
|---|---|
| `*.log` | `error.log`<br>`logs/debug.log`<br>`build/logs/error.log` |

`**` - Two adjacent asterisk symbols match any file or zero or more directories. When followed by a slash (`/`), it matches only directories.

| Pattern | Example matches |
|---|---|
| `logs/**` | Matches anything inside the `logs` directory. |

| Pattern | Example matches |
|---|---|
| `**/build` | `var/build`<br>`pub/build`<br>`build` |
| `foo/**/bar` | `foo/bar`<br>`foo/a/bar`<br>`foo/a/b/c/bar` |

`?` - The question mark matches any single character.

| Pattern | Example matches |
|---|---|
| `access?.log` | `access0.log`<br>`access1.log`<br>`accessA.log` |
| `foo??` | `fooab`<br>`foo23`<br>`foo0s` |

**Square brackets**

`[...]` - Matches any of the characters enclosed in the square brackets. When two characters are separated by a hyphen `-` it denotes a range of characters. The range includes all characters that are between those two characters. The ranges can be alphabetic or numeric.

If the first character following the `[` is an exclamation mark (`!`), then the pattern matches any character except those from the specified set.

| Pattern | Example matches |
|---|---|
| `*.[oa]` | `file.o`<br>`file.a` |
| `*.[!oa]` | `file.s`<br>`file.1`<br>`file.0` |
| `access.[0-2].log` | `access.0.log`<br>`access.1.log`<br>`access.2.log` |

| Pattern | Example matches |
|---|---|
| `file.[a-c].out` | `file.a.out`<br>`file.b.out`<br>`file.c.out` |
| `file.[a-cx-z].out` | `file.a.out`<br>`file.b.out`<br>`file.c.out`<br>`file.x.out`<br>`file.y.out`<br>`file.z.out` |
| `access.[!0-2].log` | `access.3.log`<br>`access.4.log`<br>`access.Q.log` |

### Negating Patterns

A pattern that starts with an exclamation mark (`!`) negates (re-include) any file that is ignored by the previous pattern. The exception to this rule is to re-include a file if its parent directory is excluded.

| Pattern | Example matches |
|---|---|
| `*.log`<br>`!error.log` | `error.log` or `logs/error.log` will not be ignored |

### `.gitignore` Example

Below is an example of what your `.gitignore` file could look like:

```
# Ignore the node_modules directory
node_modules/

# Ignore Logs
logs
*.log

# Ignore the build directory
/dist
```

```
# The file containing environment variables
.env

# Ignore IDE specific files
.idea/
.vscode/
*.sw*
```

### Local `.gitignore`

A local `.gitignore` file is usually placed in the repository's root directory. However, you can create multiple `.gitignore` files in different subdirectories in your repository. The patterns in the `.gitignore` files are matched relative to the directory where the file resides.

Patterns defined in the files that reside in lower-level directories (sub-directories) have precedence over those in higher-level directories.

Local `.gitignore` files are shared with other developers and should contain patterns that are useful for all other users of the repository.

### Personal Ignore Rules

Patterns that are specific to your local repository and should not be distributed to other repositories should be set in the `.git/info/exclude` file.

For example, you can use this file to ignore generated files from your personal project tools.

### Global `.gitignore`

Git also allows you to create a global `.gitignore` file, where you can define ignore rules for every Git repository on your local system.

The file can be named anything you like and stored in any location. The most common place to keep this file is the home directory. You'll have to manually create the file and configure Git to use it.

For example, to set `~/.gitignore_global` as the global Git ignore file, you would do the following:

1. Create the file:

   ```
   touch ~/.gitignore_global
   ```

2. Add the file to the Git configuration:

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

3. Open the file with your text editor and add your rules to it.

Global rules are particularly useful for ignoring particular files that you never want to commit, such as files with sensitive information or compiled executables.

**Ignoring a Previously Committed Files**

The files in your working copy can be either tracked or untracked.

To ignore a file that has been previously committed, you'll need to unstage and remove the file from the index, and then add a rule for the file in `.gitignore`:

```
git rm --cached filename
```

The `--cached` option tells git not to delete the file from the working tree but only to remove it from the index.

To recursively remove a directory, use the `-r` option:

```
git rm --cached filename
```

If you want to remove the file from both the index and local filesystem, omit the `--cached` option.

When recursively deleting files, use the `-n` option that will perform a "dry run" and show you what files will be deleted:

```
git rm -r -n directory
```

## Debugging `.gitignore` File

Sometimes it can be challenging to determine why a specific file is being ignored, especially when you're are using multiple `.gitignore` files or complex patterns. This is where the `git check-ignore` command with the `-v` option, which tells git to display details about the matching pattern, comes handy.

For example, to check why the `www/yarn.lock` file is ignored you would run:

```
git check-ignore -v www/yarn.lock
```

The output shows the path to the `gitignore` file, the number of the matching line, and the actual pattern.

```
www/.gitignore:31:/yarn.lock  www/yarn.lock
```

The command also accepts more than one filename as arguments, and the file doesn't have to exist in your working tree.

## Displaying All Ignored Files

The `git status` command with the `--ignored` option displays a list of all ignored files:

```
git status --ignored
```

## Conclusion

The `.gitignore` file allows you to exclude files from being checked into the repository. The file contains globbing patterns that describe which files and directories should be ignored.

gitignore.io is an online service that allows you to generate `.gitignore` files for your operating system, programming language, or IDE.

If you have any questions or feedback, feel free to leave a comment.