**Git ignore patterns**

.gitignore uses [globbing patterns](http://linux.die.net/man/7/glob) to match against file names. You can construct your patterns using various symbols:

| **Pattern** | **Example matches** | **Explanation\*** |
| --- | --- | --- |
| \*\*/logs | logs/debug.log logs/monday/foo.bar build/logs/debug.log | You can prepend a pattern with a double asterisk to match directories anywhere in the repository. |
| \*\*/logs/debug.log | logs/debug.log build/logs/debug.log *but not* logs/build/debug.log | You can also use a double asterisk to match files based on their name and the name of their parent directory. |
| \*.log | debug.log foo.log .log logs/debug.log | An asterisk is a wildcard that matches zero or more characters. |
| \*.log  !important.log | debug.log trace.log *but not* important.log logs/important.log | Prepending an exclamation mark to a pattern negates it. If a file matches a pattern, but *also* matches a negating pattern defined later in the file, it will not be ignored. |
| \*.log  !important/\*.log trace.\* | debug.log important/trace.log *but not* important/debug.log | Patterns defined after a negating pattern will re-ignore any previously negated files. |
| /debug.log | debug.log *but not* logs/debug.log | Prepending a slash matches files only in the repository root. |
| debug.log | debug.log logs/debug.log | By default, patterns match files in any directory |
| debug?.log | debug0.log debugg.log *but not* debug10.log | A question mark matches exactly one character. |
| debug[0-9].log | debug0.log debug1.log *but not* debug10.log | Square brackets can also be used to match a single character from a specified range. |
| debug[01].log | debug0.log debug1.log *but not*  debug2.log debug01.log | Square brackets match a single character form the specified set. |
| debug[!01].log | debug2.log *but not* debug0.log debug1.log debug01.log | An exclamation mark can be used to match any character except one from the specified set. |
| debug[a-z].log | debuga.log debugb.log *but not* debug1.log | Ranges can be numeric or alphabetic. |
| logs | logs logs/debug.log logs/latest/foo.bar build/logs build/logs/debug.log | If you don't append a slash, the pattern will match both files and the contents of directories with that name. In the example matches on the left, both directories and files named *logs* are ignored |
| logs/ | logs/debug.log logs/latest/foo.bar build/logs/foo.bar build/logs/latest/debug.log | Appending a slash indicates the pattern is a directory. The entire contents of any directory in the repository matching that name – including all of its files and subdirectories – will be ignored |
| logs/  !logs/important.log | logs/debug.log logs/important.log | Wait a minute! Shouldn't logs/important.log be negated in the example on the left  Nope! Due to a performance-related quirk in Git, you *can not* negate a file that is ignored due to a pattern matching a directory |
| logs/\*\*/debug.log | logs/debug.log logs/monday/debug.log logs/monday/pm/debug.log | A double asterisk matches zero or more directories. |
| logs/\*day/debug.log | logs/monday/debug.log logs/tuesday/debug.log *but not* logs/latest/debug.log | Wildcards can be used in directory names as well. |
| logs/debug.log | logs/debug.log *but not* debug.log build/logs/debug.log | Patterns specifying a file in a particular directory are relative to the repository root. (You can prepend a slash if you like, but it doesn't do anything special.) |

\*\* these explanations assume your .gitignore file is in the top level directory of your repository, as is the convention. If your repository has multiple .gitignore files, simply mentally replace "repository root" with "directory containing the .gitignore file" (and consider unifying them, for the sanity of your team).\*

In addition to these characters, you can use # to include comments in your .gitignore file:

# ignore all logs

\*.log

You can use \ to escape .gitignore pattern characters if you have files or directories containing them:

# ignore the file literally named foo[01].txt

foo\[01\].txt

**Shared .gitignore files in your repository**

Git ignore rules are usually defined in a .gitignore file at the root of your repository. However, you can choose to define multiple .gitignore files in different directories in your repository. Each pattern in a particular .gitignore file is tested relative to the directory containing that file. However the convention, and simplest approach, is to define a single .gitignore file in the root. As your .gitignore file is checked in, it is versioned like any other file in your repository and shared with your teammates when you push. Typically you should only include patterns in .gitignore that will benefit other users of the repository.

**Personal Git ignore rules**

You can also define personal ignore patterns for a particular repository in a special file at .git/info/exclude. These are not versioned, and not distributed with your repository, so it's an appropriate place to include patterns that will likely only benefit you. For example if you have a custom logging setup, or special development tools that produce files in your repository's working directory, you could consider adding them to .git/info/exclude to prevent them from being accidentally committed to your repository.

**Global Git ignore rules**

In addition, you can define global Git ignore patterns for all repositories on your local system by setting the Git core.excludesFile property. You'll have to create this file yourself. If you're unsure where to put your global .gitignore file, your home directory isn't a bad choice (and makes it easy to find later). Once you've created the file, you'll need to configure its location with git config:

$ touch ~/.gitignore

$ git config --global core.excludesFile ~/.gitignore

You should be careful what patterns you choose to globally ignore, as different file types are relevant for different projects. Special operating system files (e.g. .DS\_Store and thumbs.db) or temporary files created by some developer tools are typical candidates for ignoring globally.

**Ignoring a previously committed file**

If you want to ignore a file that you've committed in the past, you'll need to delete the file from your repository and then add a .gitignore rule for it. Using the --cached option with git rm means that the file will be deleted from your repository, but will remain in your working directory as an ignored file.

$ echo debug.log >> .gitignore

$ git rm --cached debug.log

rm 'debug.log'

$ git commit -m "Start ignoring debug.log"

You can omit the --cached option if you want to delete the file from both the repository and your local file system.

**Committing an ignored file**

It is possible to force an ignored file to be committed to the repository using the -f (or --force) option with git add:

$ cat .gitignore

\*.log

$ git add -f debug.log

$ git commit -m "Force adding debug.log"

You might consider doing this if you have a general pattern (like \*.log) defined, but you want to commit a specific file. However a better solution is to define an exception to the general rule:

$ echo !debug.log >> .gitignore

$ cat .gitignore

\*.log

!debug.log

$ git add debug.log

$ git commit -m "Adding debug.log"

This approach is more obvious, and less confusing, for your teammates.

**Stashing an ignored file**

[git stash](https://www.atlassian.com/git/tutorials/git-stash) is a powerful Git feature for temporarily shelving and reverting local changes, allowing you to re-apply them later on. As you'd expect, by default git stash ignores ignored files and only stashes changes to files that are tracked by Git. However, you can invoke [git stash with the --all option](https://www.atlassian.com/git/tutorials/git-stash/#stashing-untracked-or-ignored) to stash changes to ignored and untracked files as well.

**Debugging .gitignore files**

If you have complicated .gitignore patterns, or patterns spread over multiple .gitignore files, it can be difficult to track down why a particular file is being ignored. You can use the git check-ignore command with the -v (or --verbose) option to determine which pattern is causing a particular file to be ignored:

$ git check-ignore -v debug.log

.gitignore:3:\*.log debug.log

The output shows:

<file containing the pattern> : <line number of the pattern> : <pattern> <file name>

You can pass multiple file names to git check-ignore if you like, and the names themselves don't even have to correspond to files that exist in your repository.