# Picomatch

   

**Blazing fast and accurate glob matcher written in JavaScript.**

*No dependencies and full support for standard and extended Bash glob features, including braces, extglobs, POSIX brackets, and regular expressions.*

## Why picomatch? \* \*\*Lightweight\*\* - No dependencies \* \*\*Minimal\*\* - Tiny API surface. Main export is a function that takes a glob pattern and returns a matcher function. \* \*\*Fast\*\* - Loads in about 2ms (that's several times faster than a [single frame of a HD movie](http://www.endmemo.com/sconvert/framespersecondframespermillisecond.php) at 60fps) \* \*\*Performant\*\* - Use the returned matcher function to speed up repeat matching (like when watching files) \* \*\*Accurate matching\*\* - Using wildcards (`\*` and `?`), globstars (`\*\*`) for nested directories, [advanced globbing](#advanced-globbing) with extglobs, braces, and POSIX brackets, and support for escaping special characters with `\` or quotes. \* \*\*Well tested\*\* - Thousands of unit tests See the [library comparison](#library-comparisons) to other libraries.

## Table of Contents Click to expand - [Install](#install) - [Usage](#usage) - [API](#api) \* [picomatch](#picomatch) \* [.test](#test) \* [.matchBase](#matchbase) \* [.isMatch](#ismatch) \* [.parse](#parse) \* [.scan](#scan) \* [.compileRe](#compilere) \* [.makeRe](#makere) \* [.toRegex](#toregex) - [Options](#options) \* [Picomatch options](#picomatch-options) \* [Scan Options](#scan-options) \* [Options Examples](#options-examples) - [Globbing features](#globbing-features) \* [Basic globbing](#basic-globbing) \* [Advanced globbing](#advanced-globbing) \* [Braces](#braces) \* [Matching special characters as literals](#matching-special-characters-as-literals) - [Library Comparisons](#library-comparisons) - [Benchmarks](#benchmarks) - [Philosophies](#philosophies) - [About](#about) \* [Author](#author) \* [License](#license) \_(TOC generated by [verb](https://github.com/verbose/verb) using [markdown-toc](https://github.com/jonschlinkert/markdown-toc))\_

## Install Install with [npm](https://www.npmjs.com/): ```sh npm install --save picomatch ```

## Usage The main export is a function that takes a glob pattern and an options object and returns a function for matching strings. ```js const pm = require('picomatch'); const isMatch = pm('\*.js'); console.log(isMatch('abcd')); //=> false console.log(isMatch('a.js')); //=> true console.log(isMatch('a.md')); //=> false console.log(isMatch('a/b.js')); //=> false ```

## API ### [picomatch](lib/picomatch.js#L32) Creates a matcher function from one or more glob patterns. The returned function takes a string to match as its first argument, and returns true if the string is a match. The returned matcher function also takes a boolean as the second argument that, when true, returns an object with additional information. \*\*Params\*\* \* `globs` \*\*{String|Array}\*\*: One or more glob patterns. \* `options` \*\*{Object=}\*\* \* `returns` \*\*{Function=}\*\*: Returns a matcher function. \*\*Example\*\* ```js const picomatch = require('picomatch'); // picomatch(glob[, options]); const isMatch = picomatch('\*.!(\*a)'); console.log(isMatch('a.a')); //=> false console.log(isMatch('a.b')); //=> true ``` ### [.test](lib/picomatch.js#L117) Test `input` with the given `regex`. This is used by the main `picomatch()` function to test the input string. \*\*Params\*\* \* `input` \*\*{String}\*\*: String to test. \* `regex` \*\*{RegExp}\*\* \* `returns` \*\*{Object}\*\*: Returns an object with matching info. \*\*Example\*\* ```js const picomatch = require('picomatch'); // picomatch.test(input, regex[, options]); console.log(picomatch.test('foo/bar', /^(?:([^/]\*?)\/([^/]\*?))$/)); // { isMatch: true, match: [ 'foo/', 'foo', 'bar' ], output: 'foo/bar' } ``` ### [.matchBase](lib/picomatch.js#L161) Match the basename of a filepath. \*\*Params\*\* \* `input` \*\*{String}\*\*: String to test. \* `glob` \*\*{RegExp|String}\*\*: Glob pattern or regex created by [.makeRe](#makeRe). \* `returns` \*\*{Boolean}\*\* \*\*Example\*\* ```js const picomatch = require('picomatch'); // picomatch.matchBase(input, glob[, options]); console.log(picomatch.matchBase('foo/bar.js', '\*.js'); // true ``` ### [.isMatch](lib/picomatch.js#L183) Returns true if \*\*any\*\* of the given glob `patterns` match the specified `string`. \*\*Params\*\* \* \*\*{String|Array}\*\*: str The string to test. \* \*\*{String|Array}\*\*: patterns One or more glob patterns to use for matching. \* \*\*{Object}\*\*: See available [options](#options). \* `returns` \*\*{Boolean}\*\*: Returns true if any patterns match `str` \*\*Example\*\* ```js const picomatch = require('picomatch'); // picomatch.isMatch(string, patterns[, options]); console.log(picomatch.isMatch('a.a', ['b.\*', '\*.a'])); //=> true console.log(picomatch.isMatch('a.a', 'b.\*')); //=> false ``` ### [.parse](lib/picomatch.js#L199) Parse a glob pattern to create the source string for a regular expression. \*\*Params\*\* \* `pattern` \*\*{String}\*\* \* `options` \*\*{Object}\*\* \* `returns` \*\*{Object}\*\*: Returns an object with useful properties and output to be used as a regex source string. \*\*Example\*\* ```js const picomatch = require('picomatch'); const result = picomatch.parse(pattern[, options]); ``` ### [.scan](lib/picomatch.js#L231) Scan a glob pattern to separate the pattern into segments. \*\*Params\*\* \* `input` \*\*{String}\*\*: Glob pattern to scan. \* `options` \*\*{Object}\*\* \* `returns` \*\*{Object}\*\*: Returns an object with \*\*Example\*\* ```js const picomatch = require('picomatch'); // picomatch.scan(input[, options]); const result = picomatch.scan('!./foo/\*.js'); console.log(result); { prefix: '!./', input: '!./foo/\*.js', start: 3, base: 'foo', glob: '\*.js', isBrace: false, isBracket: false, isGlob: true, isExtglob: false, isGlobstar: false, negated: true } ``` ### [.compileRe](lib/picomatch.js#L245) Compile a regular expression from the `state` object returned by the [parse()](#parse) method. \*\*Params\*\* \* `state` \*\*{Object}\*\* \* `options` \*\*{Object}\*\* \* `returnOutput` \*\*{Boolean}\*\*: Intended for implementors, this argument allows you to return the raw output from the parser. \* `returnState` \*\*{Boolean}\*\*: Adds the state to a `state` property on the returned regex. Useful for implementors and debugging. \* `returns` \*\*{RegExp}\*\* ### [.makeRe](lib/picomatch.js#L286) Create a regular expression from a parsed glob pattern. \*\*Params\*\* \* `state` \*\*{String}\*\*: The object returned from the `.parse` method. \* `options` \*\*{Object}\*\* \* `returnOutput` \*\*{Boolean}\*\*: Implementors may use this argument to return the compiled output, instead of a regular expression. This is not exposed on the options to prevent end-users from mutating the result. \* `returnState` \*\*{Boolean}\*\*: Implementors may use this argument to return the state from the parsed glob with the returned regular expression. \* `returns` \*\*{RegExp}\*\*: Returns a regex created from the given pattern. \*\*Example\*\* ```js const picomatch = require('picomatch'); const state = picomatch.parse('\*.js'); // picomatch.compileRe(state[, options]); console.log(picomatch.compileRe(state)); //=> /^(?:(?!\.)(?=.)[^/]\*?\.js)$/ ``` ### [.toRegex](lib/picomatch.js#L321) Create a regular expression from the given regex source string. \*\*Params\*\* \* `source` \*\*{String}\*\*: Regular expression source string. \* `options` \*\*{Object}\*\* \* `returns` \*\*{RegExp}\*\* \*\*Example\*\* ```js const picomatch = require('picomatch'); // picomatch.toRegex(source[, options]); const { output } = picomatch.parse('\*.js'); console.log(picomatch.toRegex(output)); //=> /^(?:(?!\.)(?=.)[^/]\*?\.js)$/ ```

## Options ### Picomatch options The following options may be used with the main `picomatch()` function or any of the methods on the picomatch API. | \*\*Option\*\* | \*\*Type\*\* | \*\*Default value\*\* | \*\*Description\*\* | | --- | --- | --- | --- | | `basename` | `boolean` | `false` | If set, then patterns without slashes will be matched against the basename of the path if it contains slashes. For example, `a?b` would match the path `/xyz/123/acb`, but not `/xyz/acb/123`. | | `bash` | `boolean` | `false` | Follow bash matching rules more strictly - disallows backslashes as escape characters, and treats single stars as globstars (`\*\*`). | | `capture` | `boolean` | `undefined` | Return regex matches in supporting methods. | | `contains` | `boolean` | `undefined` | Allows glob to match any part of the given string(s). | | `cwd` | `string` | `process.cwd()` | Current working directory. Used by `picomatch.split()` | | `debug` | `boolean` | `undefined` | Debug regular expressions when an error is thrown. | | `dot` | `boolean` | `false` | Enable dotfile matching. By default, dotfiles are ignored unless a `.` is explicitly defined in the pattern, or `options.dot` is true | | `expandRange` | `function` | `undefined` | Custom function for expanding ranges in brace patterns, such as `{a..z}`. The function receives the range values as two arguments, and it must return a string to be used in the generated regex. It's recommended that returned strings be wrapped in parentheses. | | `failglob` | `boolean` | `false` | Throws an error if no matches are found. Based on the bash option of the same name. | | `fastpaths` | `boolean` | `true` | To speed up processing, full parsing is skipped for a handful common glob patterns. Disable this behavior by setting this option to `false`. | | `flags` | `string` | `undefined` | Regex flags to use in the generated regex. If defined, the `nocase` option will be overridden. | | [format](#optionsformat) | `function` | `undefined` | Custom function for formatting the returned string. This is useful for removing leading slashes, converting Windows paths to Posix paths, etc. | | `ignore` | `array\|string` | `undefined` | One or more glob patterns for excluding strings that should not be matched from the result. | | `keepQuotes` | `boolean` | `false` | Retain quotes in the generated regex, since quotes may also be used as an alternative to backslashes. | | `literalBrackets` | `boolean` | `undefined` | When `true`, brackets in the glob pattern will be escaped so that only literal brackets will be matched. | | `matchBase` | `boolean` | `false` | Alias for `basename` | | `maxLength` | `boolean` | `65536` | Limit the max length of the input string. An error is thrown if the input string is longer than this value. | | `nobrace` | `boolean` | `false` | Disable brace matching, so that `{a,b}` and `{1..3}` would be treated as literal characters. | | `nobracket` | `boolean` | `undefined` | Disable matching with regex brackets. | | `nocase` | `boolean` | `false` | Make matching case-insensitive. Equivalent to the regex `i` flag. Note that this option is overridden by the `flags` option. | | `nodupes` | `boolean` | `true` | Deprecated, use `nounique` instead. This option will be removed in a future major release. By default duplicates are removed. Disable uniquification by setting this option to false. | | `noext` | `boolean` | `false` | Alias for `noextglob` | | `noextglob` | `boolean` | `false` | Disable support for matching with extglobs (like `+(a\|b)`) | | `noglobstar` | `boolean` | `false` | Disable support for matching nested directories with globstars (`\*\*`) | | `nonegate` | `boolean` | `false` | Disable support for negating with leading `!` | | `noquantifiers` | `boolean` | `false` | Disable support for regex quantifiers (like `a{1,2}`) and treat them as brace patterns to be expanded. | | [onIgnore](#optionsonIgnore) | `function` | `undefined` | Function to be called on ignored items. | | [onMatch](#optionsonMatch) | `function` | `undefined` | Function to be called on matched items. | | [onResult](#optionsonResult) | `function` | `undefined` | Function to be called on all items, regardless of whether or not they are matched or ignored. | | `posix` | `boolean` | `false` | Support POSIX character classes ("posix brackets"). | | `posixSlashes` | `boolean` | `undefined` | Convert all slashes in file paths to forward slashes. This does not convert slashes in the glob pattern itself | | `prepend` | `boolean` | `undefined` | String to prepend to the generated regex used for matching. | | `regex` | `boolean` | `false` | Use regular expression rules for `+` (instead of matching literal `+`), and for stars that follow closing parentheses or brackets (as in `)\*` and `]\*`). | | `strictBrackets` | `boolean` | `undefined` | Throw an error if brackets, braces, or parens are imbalanced. | | `strictSlashes` | `boolean` | `undefined` | When true, picomatch won't match trailing slashes with single stars. | | `unescape` | `boolean` | `undefined` | Remove backslashes preceding escaped characters in the glob pattern. By default, backslashes are retained. | | `unixify` | `boolean` | `undefined` | Alias for `posixSlashes`, for backwards compatibility. | picomatch has automatic detection for regex positive and negative lookbehinds. If the pattern contains a negative lookbehind, you must be using Node.js >= 8.10 or else picomatch will throw an error. ### Scan Options In addition to the main [picomatch options](#picomatch-options), the following options may also be used with the [.scan](#scan) method. | \*\*Option\*\* | \*\*Type\*\* | \*\*Default value\*\* | \*\*Description\*\* | | --- | --- | --- | --- | | `tokens` | `boolean` | `false` | When `true`, the returned object will include an array of tokens (objects), representing each path "segment" in the scanned glob pattern | | `parts` | `boolean` | `false` | When `true`, the returned object will include an array of strings representing each path "segment" in the scanned glob pattern. This is automatically enabled when `options.tokens` is true | \*\*Example\*\* ```js const picomatch = require('picomatch'); const result = picomatch.scan('!./foo/\*.js', { tokens: true }); console.log(result); // { // prefix: '!./', // input: '!./foo/\*.js', // start: 3, // base: 'foo', // glob: '\*.js', // isBrace: false, // isBracket: false, // isGlob: true, // isExtglob: false, // isGlobstar: false, // negated: true, // maxDepth: 2, // tokens: [ // { value: '!./', depth: 0, isGlob: false, negated: true, isPrefix: true }, // { value: 'foo', depth: 1, isGlob: false }, // { value: '\*.js', depth: 1, isGlob: true } // ], // slashes: [ 2, 6 ], // parts: [ 'foo', '\*.js' ] // } ```

### Options Examples #### options.expandRange \*\*Type\*\*: `function` \*\*Default\*\*: `undefined` Custom function for expanding ranges in brace patterns. The [fill-range](https://github.com/jonschlinkert/fill-range) library is ideal for this purpose, or you can use custom code to do whatever you need. \*\*Example\*\* The following example shows how to create a glob that matches a folder ```js const fill = require('fill-range'); const regex = pm.makeRe('foo/{01..25}/bar', { expandRange(a, b) { return `(${fill(a, b, { toRegex: true })})`; } }); console.log(regex); //=> /^(?:foo\/((?:0[1-9]|1[0-9]|2[0-5]))\/bar)$/ console.log(regex.test('foo/00/bar')) // false console.log(regex.test('foo/01/bar')) // true console.log(regex.test('foo/10/bar')) // true console.log(regex.test('foo/22/bar')) // true console.log(regex.test('foo/25/bar')) // true console.log(regex.test('foo/26/bar')) // false ``` #### options.format \*\*Type\*\*: `function` \*\*Default\*\*: `undefined` Custom function for formatting strings before they're matched. \*\*Example\*\* ```js // strip leading './' from strings const format = str => str.replace(/^\.\//, ''); const isMatch = picomatch('foo/\*.js', { format }); console.log(isMatch('./foo/bar.js')); //=> true ``` #### options.onMatch ```js const onMatch = ({ glob, regex, input, output }) => { console.log({ glob, regex, input, output }); }; const isMatch = picomatch('\*', { onMatch }); isMatch('foo'); isMatch('bar'); isMatch('baz'); ``` #### options.onIgnore ```js const onIgnore = ({ glob, regex, input, output }) => { console.log({ glob, regex, input, output }); }; const isMatch = picomatch('\*', { onIgnore, ignore: 'f\*' }); isMatch('foo'); isMatch('bar'); isMatch('baz'); ``` #### options.onResult ```js const onResult = ({ glob, regex, input, output }) => { console.log({ glob, regex, input, output }); }; const isMatch = picomatch('\*', { onResult, ignore: 'f\*' }); isMatch('foo'); isMatch('bar'); isMatch('baz'); ```

## Globbing features \* [Basic globbing](#basic-globbing) (Wildcard matching) \* [Advanced globbing](#advanced-globbing) (extglobs, posix brackets, brace matching) ### Basic globbing | \*\*Character\*\* | \*\*Description\*\* | | --- | --- | | `\*` | Matches any character zero or more times, excluding path separators. Does \_not match\_ path separators or hidden files or directories ("dotfiles"), unless explicitly enabled by setting the `dot` option to `true`. | | `\*\*` | Matches any character zero or more times, including path separators. Note that `\*\*` will only match path separators (`/`, and `\\` on Windows) when they are the only characters in a path segment. Thus, `foo\*\*/bar` is equivalent to `foo\*/bar`, and `foo/a\*\*b/bar` is equivalent to `foo/a\*b/bar`, and \_more than two\_ consecutive stars in a glob path segment are regarded as \_a single star\_. Thus, `foo/\*\*\*/bar` is equivalent to `foo/\*/bar`. | | `?` | Matches any character excluding path separators one time. Does \_not match\_ path separators or leading dots. | | `[abc]` | Matches any characters inside the brackets. For example, `[abc]` would match the characters `a`, `b` or `c`, and nothing else. | #### Matching behavior vs. Bash Picomatch's matching features and expected results in unit tests are based on Bash's unit tests and the Bash 4.3 specification, with the following exceptions: \* Bash will match `foo/bar/baz` with `\*`. Picomatch only matches nested directories with `\*\*`. \* Bash greedily matches with negated extglobs. For example, Bash 4.3 says that `!(foo)\*` should match `foo` and `foobar`, since the trailing `\*` bracktracks to match the preceding pattern. This is very memory-inefficient, and IMHO, also incorrect. Picomatch would return `false` for both `foo` and `foobar`.

### Advanced globbing \* [extglobs](#extglobs) \* [POSIX brackets](#posix-brackets) \* [Braces](#brace-expansion) #### Extglobs | \*\*Pattern\*\* | \*\*Description\*\* | | --- | --- | | `@(pattern)` | Match \_only one\_ consecutive occurrence of `pattern` | | `\*(pattern)` | Match \_zero or more\_ consecutive occurrences of `pattern` | | `+(pattern)` | Match \_one or more\_ consecutive occurrences of `pattern` | | `?(pattern)` | Match \_zero or \*\*one\*\*\_ consecutive occurrences of `pattern` | | `!(pattern)` | Match \_anything but\_ `pattern` | \*\*Examples\*\* ```js const pm = require('picomatch'); // \*(pattern) matches ZERO or more of "pattern" console.log(pm.isMatch('a', 'a\*(z)')); // true console.log(pm.isMatch('az', 'a\*(z)')); // true console.log(pm.isMatch('azzz', 'a\*(z)')); // true // +(pattern) matches ONE or more of "pattern" console.log(pm.isMatch('a', 'a\*(z)')); // true console.log(pm.isMatch('az', 'a\*(z)')); // true console.log(pm.isMatch('azzz', 'a\*(z)')); // true // supports multiple extglobs console.log(pm.isMatch('foo.bar', '!(foo).!(bar)')); // false // supports nested extglobs console.log(pm.isMatch('foo.bar', '!(!(foo)).!(!(bar))')); // true ``` #### POSIX brackets POSIX classes are disabled by default. Enable this feature by setting the `posix` option to true. \*\*Enable POSIX bracket support\*\* ```js console.log(pm.makeRe('[[:word:]]+', { posix: true })); //=> /^(?:(?=.)[A-Za-z0-9\_]+\/?)$/ ``` \*\*Supported POSIX classes\*\* The following named POSIX bracket expressions are supported: \* `[:alnum:]` - Alphanumeric characters, equ `[a-zA-Z0-9]` \* `[:alpha:]` - Alphabetical characters, equivalent to `[a-zA-Z]`. \* `[:ascii:]` - ASCII characters, equivalent to `[\\x00-\\x7F]`. \* `[:blank:]` - Space and tab characters, equivalent to `[ \\t]`. \* `[:cntrl:]` - Control characters, equivalent to `[\\x00-\\x1F\\x7F]`. \* `[:digit:]` - Numerical digits, equivalent to `[0-9]`. \* `[:graph:]` - Graph characters, equivalent to `[\\x21-\\x7E]`. \* `[:lower:]` - Lowercase letters, equivalent to `[a-z]`. \* `[:print:]` - Print characters, equivalent to `[\\x20-\\x7E ]`. \* `[:punct:]` - Punctuation and symbols, equivalent to `[\\-!"#$%&\'()\\\*+,./:;<=>?@[\\]^\_`{|}~]`. \* `[:space:]` - Extended space characters, equivalent to `[ \\t\\r\\n\\v\\f]`. \* `[:upper:]` - Uppercase letters, equivalent to `[A-Z]`. \* `[:word:]` - Word characters (letters, numbers and underscores), equivalent to `[A-Za-z0-9\_]`. \* `[:xdigit:]` - Hexadecimal digits, equivalent to `[A-Fa-f0-9]`. See the [Bash Reference Manual](https://www.gnu.org/software/bash/manual/html\_node/Pattern-Matching.html) for more information. ### Braces Picomatch does not do brace expansion. For [brace expansion](https://www.gnu.org/software/bash/manual/html\_node/Brace-Expansion.html) and advanced matching with braces, use [micromatch](https://github.com/micromatch/micromatch) instead. Picomatch has very basic support for braces. ### Matching special characters as literals If you wish to match the following special characters in a filepath, and you want to use these characters in your glob pattern, they must be escaped with backslashes or quotes: \*\*Special Characters\*\* Some characters that are used for matching in regular expressions are also regarded as valid file path characters on some platforms. To match any of the following characters as literals: `$^\*+?()[] Examples: ```js console.log(pm.makeRe('foo/bar \\(1\\)')); console.log(pm.makeRe('foo/bar \\(1\\)')); ```

## Library Comparisons The following table shows which features are supported by [minimatch](https://github.com/isaacs/minimatch), [micromatch](https://github.com/micromatch/micromatch), [picomatch](https://github.com/micromatch/picomatch), [nanomatch](https://github.com/micromatch/nanomatch), [extglob](https://github.com/micromatch/extglob), [braces](https://github.com/micromatch/braces), and [expand-brackets](https://github.com/micromatch/expand-brackets). | \*\*Feature\*\* | `minimatch` | `micromatch` | `picomatch` | `nanomatch` | `extglob` | `braces` | `expand-brackets` | | --- | --- | --- | --- | --- | --- | --- | --- | | Wildcard matching (`\*?+`) | ✔ | ✔ | ✔ | ✔ | - | - | - | | Advancing globbing | ✔ | ✔ | ✔ | - | - | - | - | | Brace \_matching\_ | ✔ | ✔ | ✔ | - | - | ✔ | - | | Brace \_expansion\_ | ✔ | ✔ | - | - | - | ✔ | - | | Extglobs | partial | ✔ | ✔ | - | ✔ | - | - | | Posix brackets | - | ✔ | ✔ | - | - | - | ✔ | | Regular expression syntax | - | ✔ | ✔ | ✔ | ✔ | - | ✔ | | File system operations | - | - | - | - | - | - | - |

## Benchmarks Performance comparison of picomatch and minimatch. ``` # .makeRe star picomatch x 1,993,050 ops/sec ±0.51% (91 runs sampled) minimatch x 627,206 ops/sec ±1.96% (87 runs sampled)) # .makeRe star; dot=true picomatch x 1,436,640 ops/sec ±0.62% (91 runs sampled) minimatch x 525,876 ops/sec ±0.60% (88 runs sampled) # .makeRe globstar picomatch x 1,592,742 ops/sec ±0.42% (90 runs sampled) minimatch x 962,043 ops/sec ±1.76% (91 runs sampled)d) # .makeRe globstars picomatch x 1,615,199 ops/sec ±0.35% (94 runs sampled) minimatch x 477,179 ops/sec ±1.33% (91 runs sampled) # .makeRe with leading star picomatch x 1,220,856 ops/sec ±0.40% (92 runs sampled) minimatch x 453,564 ops/sec ±1.43% (94 runs sampled) # .makeRe - basic braces picomatch x 392,067 ops/sec ±0.70% (90 runs sampled) minimatch x 99,532 ops/sec ±2.03% (87 runs sampled)) ```

## Philosophies The goal of this library is to be blazing fast, without compromising on accuracy. \*\*Accuracy\*\* The number one of goal of this library is accuracy. However, it's not unusual for different glob implementations to have different rules for matching behavior, even with simple wildcard matching. It gets increasingly more complicated when combinations of different features are combined, like when extglobs are combined with globstars, braces, slashes, and so on: `!(\*\*/{a,b,\*/c})`. Thus, given that there is no canonical glob specification to use as a single source of truth when differences of opinion arise regarding behavior, sometimes we have to implement our best judgement and rely on feedback from users to make improvements. \*\*Performance\*\* Although this library performs well in benchmarks, and in most cases it's faster than other popular libraries we benchmarked against, we will always choose accuracy over performance. It's not helpful to anyone if our library is faster at returning the wrong answer.

## About **Contributing** Pull requests and stars are always welcome. For bugs and feature requests, [please create an issue](../../issues/new). Please read the [contributing guide](.github/contributing.md) for advice on opening issues, pull requests, and coding standards. **Running Tests** Running and reviewing unit tests is a great way to get familiarized with a library and its API. You can install dependencies and run tests with the following command: ```sh npm install && npm test ``` **Building docs** \_(This project's readme.md is generated by [verb](https://github.com/verbose/verb-generate-readme), please don't edit the readme directly. Any changes to the readme must be made in the [.verb.md](.verb.md) readme template.)\_ To generate the readme, run the following command: ```sh npm install -g verbose/verb#dev verb-generate-readme && verb ``` ### Author \*\*Jon Schlinkert\*\* \* [GitHub Profile](https://github.com/jonschlinkert) \* [Twitter Profile](https://twitter.com/jonschlinkert) \* [LinkedIn Profile](https://linkedin.com/in/jonschlinkert) ### License Copyright © 2017-present, [Jon Schlinkert](https://github.com/jonschlinkert). Released under the [MIT License](LICENSE).