

Author

Jon Schlinkert

- [GitHub Profile](#)
- [Twitter Profile](#)
- [LinkedIn Profile](#)

License

Copyright © 2017-present, [Jon Schlinkert](#). Released under the [MIT License](#).

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](#) 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](#) 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](#) to other libraries.

About

Contributing

Pull requests and stars are always welcome. For bugs and feature requests, [please create an issue](#). Please read the [contributing guide](#) 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:

```
npm install && npm test
```

Building docs

(This project's readme.md is generated by [verb](#), please don't edit the readme directly. Any changes to the readme must be made in the [verb.md](#) readme template.)

To generate the readme, run the following command:

```
npm install -g verbose/verb#dev verb-generate-readme && verb
```

against, we will always choose accuracy over performance. It's not helpful to anyone if our library is faster at returning the wrong answer.

```
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

Table of Contents

Click to expand

- [Install](#)
- [Usage](#)
- [API](#)
 - [picomatch](#)
 - [.test](#)
 - [.matchBase](#)
 - [.isMatch](#)
 - [.parse](#)
 - [.scan](#)
 - [.compileRe](#)
 - [.makeRe](#)
 - [.toRegex](#)
- [Options](#)
 - [Picomatch options](#)
 - [Scan Options](#)
 - [Options Examples](#)
- [Globbing features](#)
 - [Basic globbing](#)
 - [Advanced globbing](#)
 - [Braces](#)
 - [Matching special characters as literals](#)

- [Library Comparisons](#)
- [Benchmarks](#)
- [Philosophies](#)
- [About](#)
- [Author](#)
- [License](#)

(TOC generated by [verb](#) using [markdown-toc](#))

Install

Install with [npm](#):

```
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.

```
const pm = require('picomatch');
const isMatch = pm('*.*js');
```

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)

# .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
```

Library Comparisons

The following table shows which features are supported by [minimatch](#), [micromatch](#), [picomatch](#), [nanomatch](#), [extglob](#), [braces](#), and [expand-brackets](#).

Feature	minimatch	micromatch	picomatch	nanomatch	extglob
Wildcard matching (*?+)	✓	✓	✓	✓	-
Advancing globbing	✓	✓	✓	-	-
Brace <i>matching</i>	✓	✓	✓	-	-
Brace <i>expansion</i>	✓	✓	-	-	-
Extglobs	partial	✓	✓	-	✓
Posix brackets	-	✓	✓	-	-
Regular expression - syntax	-	✓	✓	✓	✓
File system operations	-	-	-	-	-

```
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](#)

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

```
const picomatch = require('picomatch');
// picomatch(glob[, options]);

const isMatch = picomatch('*.!(*a)');
```

```
console.log(isMatch('a.a')); // => false
console.log(isMatch('a.b')); // => true
```

.test

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

```
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

Match the basename of a filepath.

Params

- `input {String}`: String to test.

Braces

Picomatch does not do brace expansion. For [brace expansion](#) and advanced matching with braces, use [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:

```
console.log(pm.makeRe('foo/bar \\(1\\ \\)'));
```

```
console.log(pm.makeRe('foo/bar \\(1\\ \\)'));
```

The following named POSIX bracket expressions are supported:

- `[:alnum:]` - Alphanumeric characters, equivalent to `[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](#) for more information.

- `glob {RegExp|String}`: Glob pattern or regex created by [.makeRe](#).

- returns **{Boolean}**

Example

```
const picomatch = require('picomatch');
// picomatch.matchBase(input, glob[,
//                          options]);
console.log(picomatch.matchBase('foo/
bar.js', '*.js')); // true
```

.isMatch

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](#).
- returns **{Boolean}**: Returns true if any patterns match `str`

Example

```
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

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

```
const picomatch = require('picomatch');
const result =
  picomatch.parse(pattern[,
    options]);
```

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

```
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

```
console.log(pm.makeRe('[[:word:]]+', {
  posix: true }));
// => /\^(?:[A-Za-z0-9_+\\"/?]$)/
```

Supported POSIX classes

Advanced globbing

- [extglobs](#)
- [POSIX brackets](#)
- [Braces](#)

Extglobs

Pattern	Description
@(pattern)	Match <i>only one</i> consecutive occurrence of pattern
*(pattern)	Match <i>zero or more</i> consecutive occurrences of pattern
+(pattern)	Match <i>one or more</i> consecutive occurrences of pattern
?(pattern)	Match <i>zero or one</i> consecutive occurrences of pattern
!(pattern)	Match <i>anything but</i> pattern

Examples

```
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
```

Example

```
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](#)

Compile a regular expression from the state object returned by the [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

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

```
const picomatch = require('picomatch');
const state = picomatch.parse('*.*.js');
// picomatch.compileRe(state[, options]);
```

Character

Description

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.

?

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 `*` backtracks to match the preceding pattern. This is very memory-inefficient, and IMHO, also incorrect. Picomatch would return `false` for both `foo` and `foobar`.

Globbing features

- [Basic globbing](#) (Wildcard matching)
- [Advanced globbing](#) (extglobs, posix brackets, brace matching)

Basic globbing

Character	Description
*	Matches any character zero or more times, excluding path separators. Does <i>not match</i> path separators or hidden files or directories (“dotfiles”), unless explicitly enabled by setting the <code>dot</code> option to <code>true</code> .
**	Matches any character zero or more times, including path separators. Note that <code>**</code> will only match path separators (<code>/</code> , and <code>\\</code> on Windows) when they are the only characters in a path segment. Thus, <code>foo**/bar</code> is equivalent to <code>foo*/bar</code> , and <code>foo/a**b/bar</code> is equivalent to <code>foo/a*b/</code>

```
console.log(picomatch.compileRe(state));  
//=> /^((?:?!\\.)?(?=.)[^/]*?\\.js)$/
```

.toRegex

Create a regular expression from the given regex source string.

Params

- `source {String}`: Regular expression source string.
- `options {Object}`
- returns `{RegExp}`

Example

```
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 instead of the whole path. For example, <code>a?b</code> would match the path <code>/xyz/123/abc</code> , but not <code>xyz/abc/123</code> .
----------	---------	-------	---

bash	boolean	false	Follow bash matching rules more strictly - disallows backslashes to escape characters, and treats single stars as globs (e.g. <code>**</code>).
------	---------	-------	--

capture	boolean	undefined	Return regex matches supporting methods.
contains	boolean	undefined	Allows glob to match any part of the given string(s).

cwd	string	<code>process.cwd()</code>	Current working directory. Used by <code>picomatch.split</code>
-----	--------	----------------------------	---

```
const isMatch = picomatch('*', { onMatch });
isMatch('foo');
isMatch('bar');
isMatch('baz');
```

options.ignore

```
const onIgnore = ({ glob, regex, input, output }) => {
  console.log({ glob, regex, input, output });
};
```

```
const isMatch = picomatch('*', { onIgnore, ignore: '*' });
isMatch('foo');
isMatch('bar');
isMatch('baz');
```

options.onResult

```
const onResult = ({ glob, regex, input, output }) => {
  console.log({ glob, regex, input, output });
};
```

```
const isMatch = picomatch('*', { onResult, ignore: '*' });
isMatch('foo');
isMatch('bar');
isMatch('baz');
```

```

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

```

// strip leading './' from strings
const format = str => str.replace(/^\.
  \/, '');
const isMatch = picomatch('foo/*.js',
  { format });
console.log(isMatch('./foo/bar.js')); //
  => true

```

options.onMatch

```

const onMatch = ({ glob, regex, input,
  output }) => {
  console.log({ glob, regex, input,
    output });
};

```

Option	Type	Default value	Description
debug	boolean	undefined	Debug regular expressions where an error is thrown.
dot	boolean	false	Enable dotfile matching. By default, dotfiles are ignored unless explicitly defined in a pattern, or options.dot is true.
expandRange	function	undefined	Custom function for expanding range patterns, e.g. {a..z}. The function receives the range as two arguments and must return a string 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 behavior of the same name in Bash.
fastpaths	boolean	true	To speed up processing, full parsing is skipped for a handful of common glob patterns. If this behavior bothers you, use this option to force full parsing.
flags	string	undefined	Regex flags to be generated. If defined, the noFlags option will be overridden.

Option	Type	Default value	Description
format	function undefined		Custom function for formatting the return string. This is useful 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 results.
keepQuotes	boolean false		Retain quotes in the generated regex, since quotes may also be used as an alternative to escaping.
literalBrackets	boolean undefined		When true, bracket the glob pattern will be escaped so that only literal brackets will be matched.
matchBase	boolean false		Alias for basenameLimit.
maxLength	boolean 65536		Limit the max length of the input string. An error is thrown if the input string is longer than the specified value.
nobrace	boolean false		Disable brace matching so that {a,b} and {1..3} would be treated as literal characters.
nobreaket	boolean undefined		Disable matching with regex brackets.
nocase	boolean false		

14

Custom function for formatting the return string. This is useful removing leading slashes, converting Windows paths to POSIX paths, etc.

One or more glob patterns for excluding strings that should not be matched from the results.

Retain quotes in the generated regex, since quotes may also be used as an alternative to escaping.

When true, bracket the glob pattern will be escaped so that only literal brackets will be matched.

Alias for basenameLimit.

Limit the max length of the input string. An error is thrown if the input string is longer than the specified value.

Disable brace matching so that {a,b} and {1..3} would be treated as literal characters.

Disable matching with regex brackets.

Options Examples

options.expandRange

Type: function

Default: undefined

Custom function for expanding ranges in brace patterns. The [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

```
const fill = require('fill-range');
const regex = pm.makeRe('foo/{01..25}/bar', {
  expandRange(a, b) {
    return `_${fill(a, b, { toRegex: true })}`;
  }
});
console.log(regex);
//=> /^(?:foo\/(?:[1-9]|1[0-9]|2[0-5])\/bar)$/
```

```
console.log(regex.test('foo/00/bar')) // false
console.log(regex.test('foo/01/bar')) // true
```

19

```

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' ]
// }

```

Option	Type	Default value	Description
			Make matching insensitive. Equivalent to the regex <code>i</code> flag, but that this option is overridden by the <code>flags</code> option.
			Deprecated, use <code>unique</code> instead. This option will be removed in a future major version. By default duplicates are removed. Disable uniquification by setting this option to <code>false</code> .
<code>nodupes</code>	boolean	<code>true</code>	Alias for <code>noext</code> .
<code>noext</code>	boolean	<code>false</code>	Disable support for matching with <code>extglob</code> (like <code>+(a b)</code>).
<code>noextglob</code>	boolean	<code>false</code>	Disable support for matching nested directories with <code>(**)</code> .
<code>noglobstar</code>	boolean	<code>false</code>	Disable support for negating with <code>!</code> .
<code>nonegate</code>	boolean	<code>false</code>	Disable support for quantifiers (like <code>{1,2}</code>) and treat <code>*</code> as brace pattern unless expanded.
<code>noquantifiers</code>	boolean	<code>false</code>	Function to be called for ignored items.
<code>onIgnore</code>	function	undefined	Function to be called for matched items.
<code>onMatch</code>	function	undefined	Function to be called for all items, regardless of whether or not they are matched.
<code>onResult</code>	function	undefined	

Option	Type	Default value	Description
posix	boolean	false	Support POSIX character classes ("posix" brackets"). Convert all slashes in paths to forward slashes. This does not convert slashes in the glob pattern itself
posixSlashes	boolean	undefined	String to prepend to generated regex used matching.
prepend	boolean	undefined	Use regular expression rules for + (instead of matching literal +), a for stars that follow closing parentheses or brackets (as in) * and [*).
regex	boolean	false	Throw an error if brackets, braces, or parens are imbalanced. When true, picomatch won't match trailing slashes with single slashes. Remove backslashes preceding escaped characters in the glob pattern. By default, backslashes are retained. Alias for posixSlashes, for backwards compatibility
strictBrackets	boolean	undefined	
strictSlashes	boolean	undefined	
unescape	boolean	undefined	
unify	boolean	undefined	

16

Option	Type	Default value	Description
negative	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
parts	boolean	false	When true, the returned object will include an array of tokens (objects), representing each path "segment" in the scanned glob pattern
tokens	boolean	false	When true, the returned object will include an array of tokens (objects), representing each path "segment" in the scanned glob pattern
Example			

17

Option	Type	Default value	Description
negative	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
parts	boolean	false	When true, the returned object will include an array of tokens (objects), representing each path "segment" in the scanned glob pattern
tokens	boolean	false	When true, the returned object will include an array of tokens (objects), representing each path "segment" in the scanned glob pattern
Example			

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.

In addition to the main [picomatch.options](#), the following options may also be used with the [.scan](#) method.

Scan Options