

Author

Jon Schlinkert

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

Picomatch

License

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

About

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

ANSWER.

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

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.

Please read the [contributing guide](#) for advice on opening issues, pull requests, and coding standards.

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 verb#dev verb-verbose

See the [library comparison](#) to other libraries.

Well tested - Thousands of unit tests
characters with \ or quotes.

Accurate matching - Using wildcards (* and ?), globs, braces, and POSIX brackets, and support for escaping special characters, for nested directives, advanced globbing with extglob, (*)

Performance - Use the returned matcher function to speed up repeat matching (like when watching files)

Fast - Loads in about 2ms (that's several times faster than a single frame of a HD movie at 60fps)

LARGEWAVELG	No dependencies
MINIMAL	Tiny API surface. Main export is a function that takes a global pattern and returns a matcher function.

Why picomatch?

No dependencies and just support for standard and extended Bash glob features, including braces, `extglob`, `POSIX` brackets, and regular expressions.

Blazing fast and accurate glob matcher written in JavaScript.

```
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 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](#)

```
const pm = require('picomatch');
```

The main export is a function that takes a glob pattern and an options object and returns a function for matching strings.

Usage

```
npm install --save picomatch
```

Install with `npm`:

Install

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

```
# .makeRe - basic braces
# .makeRe with leading star
# .makeRe with globs
# .makeRe globs
# .makeRe globsstar
# .makeRe globsstar
# .makeRe star; dot=true
# .makeRe star
# .makeRe comparison of picomatch and minimatch.
```

Performance comparison of picomatch and minimatch.

Benchmarks

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	extgl
Wildcard matching (*?+)	✓	✓	✓	✓	-
Advancing globbing	✓	✓	✓	-	-
Brace matching	✓	✓	✓	-	-
Brace expansion	✓	✓	-	-	-
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)');
```

Matching special characters as literals

Picomatch does not do brace expansion. For brace expansion and advanced matching with braces, use `micromatch` instead. Picomatch has very basic support for braces.

Braces

If you wish to match the following special characters in a regular expression, 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.

platforms.

Example

: ((, (\

[matchbase](#)

```
options] :  
    console.log(picomatch.test('foo/bar', {  
        '^': ([^/*?)(\/([^/*?)($/)):  
        // { isMatch: true, match: [ 'foo/' ,  
        'foo', 'bar' ], output: 'foo/  
        bar' }  
    })
```

Params	Test input with the given regex. This is used by the main <code>icomatch()</code> function to test the input string.
<code>input {String}</code>	String to test.
<code>regex {RegExp}</code>	Regular expression to match.
<code>example</code>	Example of how the function works.

test

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

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\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]);
```

```
Supports POSIX classes  
// + (pattern) matches ONE or more of  
// "pattern"  
console.log(pm.isMatch('azzz',  
    'a*(z)'), // true  
    console.log(pm.isMatch('az',  
        'a*(z)'), // true  
    console.log(pm.isMatch('a',  
        'a*(z)'), // true  
    console.log(pm.isMatch('azzz',  
        'a*(z)'), // true  
    console.log(pm.isMatch('azzz',  
        'a*'), // true  
    console.log(pm.isMatch('azzz',  
        'a* (z)'), // false  
    console.log(pm.isMatch('foo.bar',  
        '!(bar)'), // true  
    console.log(pm.isMatch('foo.bar',  
        '!(!(bar)'), // true  
    // supports multiple extglob  
    // supports nested extglob  
    console.log(pm.isMatch('foo',  
        '(foo)').!(!(bar))), // true  
    // supports  
    // POSIX brackets  
    // setting the Posix option to true.  
    // Enable POSIX bracket support  
    // => /\?:(?=.)[A-Za-Z0-9_-]+/\?)/$)/  
    console.log(pm.makeText(`[[${word}]]+`), {  
        posix: true  
    })  
}
```

```
Example
and output to be used as a regex source string.
re.findall (objetc): returns an object with useful properties
const picomatch = require ('picomatch');
const result = picomatch.parse(pattern [, options]);
picomatch (result [, options]) :
Params
Scan a glob pattern to separate the pattern into segments.
input {String}: Glob pattern to scan.
options {Object}: Returns an object with properties
returns {Object}: Returns an object with properties
```

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.

makeRE	
• retURNS {RegExP}	property on the returned regex. Useful for implementors and debugging.
• retURNState {Boolean} : Adds the state to a state	properly from a parsed glob pattern. Thus, <code>foo/*/bar</code> is equivalent to <code>foo/**/bar</code> .
• optIons {Object}	Create a regular expression from a parsed glob pattern.
• 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.	returns from mutating the result.
• retURNState {Boolean} : Implementors may use this argument to return the state from the parsed glob with the returned regular expression.	regular expression.
• REGEXP : Returns a regex created from the given regular expression.	regular expression.
• Example	Example
<pre>const picomatch = require('picomatch'); const state = picomatch.parse('*.*'); const picomatch = picomatch.compile(state[0], state[1]);</pre>	<pre>// picomatch.compile(state[0], state[1]); // picomatch.compile(state[0], state[1]);</pre>

Description	Character
Picomatch's matching features and expected results in unit tests are based on Bash's unit tests and the Bash 4.3 specification, which the following exceptions:	• Bash will match <code>foo/bar/baz</code> with <code>*</code> . Picomatch only matches nested directories with <code>*</code> .
Picomatch's matching features and expected results in unit tests are based on Bash's unit tests and the Bash 4.3 specification, which the following exceptions:	• Bash 4.3 says that <code>!(foo)*</code> should match <code>foo</code> and <code>foobar</code> , since the trailing <code>*</code> backtracks to match the preceding pattern.
This is very memory-inefficient, and IMO, also incorrect. Picomatch would return <code>false</code> for both <code>foo</code> and <code>foobar</code> .	

Matching behavior vs. Bash

C, and nothing else.

match the characters `a`, `b` or

example, `[abc]` would

inside the brackets. For

Matches any characters

separators or leading dots.

time. Does not match path

excluding path separators one

Matches any character

`foo/*/bar`.

`***/bar` is equivalent to

a single star. Thus, `foo/`

path segment are regarded as

consecutive stars in a glob bar, and more than two

`[abc]`

?

makeRE	
• retURNS {RegExP}	property on the returned regex. Useful for implementors and debugging.
• retURNState {Boolean} : Adds the state to a state	properly from a parsed glob pattern. Thus, <code>foo/*/bar</code> is equivalent to <code>foo/**/bar</code> .
• optIons {Object}	Create a regular expression from a parsed glob pattern.
• 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.	regular expression.
• REGEXP : Returns a regex created from the given regular expression.	regular expression.
• Example	Example
<pre>const picomatch = picomatch.compile(state[0], state[1]); const state = picomatch.parse('*.*'); const picomatch = require('picomatch');</pre>	<pre>// picomatch.compile(state[0], state[1]); // picomatch.compile(state[0], state[1]);</pre>

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</i> match 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
<code>basename</code>	boolean	false	If set, then patterns matched against the basename of the path contains slashes. For example, a <code>?b</code> would match the path <code>/xyz</code> without slashes while <code>123/acb</code> , but not <code>xyz/acb/123</code> .
<code>bash</code>	boolean	false	Follow bash matching rules more strictly - <code>xyz/acb/123</code> .
<code>capture</code>	boolean	undefined	Return regex matches supporting methods. Allows glob to match any part of the given string(s).
<code>contains</code>	boolean	undefined	Allows glob to match any part of the given string(s). Current working directory. Used by <code>process.cwd()</code> method.
<code>cmd</code>	string	working directory. Used by <code>process.cwd()</code> method.	<code>picomatch.split()</code>

```

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 when an error is thrown.
dot	boolean	false	Enable dotfile matching.
expandRange	function	undefined	Custom function for expanding ranges in brace 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 are wrapped in parentheses. Throws an error if matches are found for ranges based on the base name of the same name. To speed up parsing, full parsing is skipped for a handful of common glob patterns. It's possible to change this behavior by setting this option to false. Regex flags to be applied to generated regexes. If this option is defined, the noMatch option will be overridden.
failglob	boolean	false	Based on the behavior of the same name.
fastpaths	boolean	true	To speed up parsing, full parsing is skipped for a handful of common glob patterns. It's possible to change this behavior by setting this option to false. Regex flags to be applied to generated regexes. If this option is defined, the noMatch option will be overridden.
flags	string	undefined	String containing regex flags to be applied to generated regexes.

Option	Description	Type	Default value	format
options.expandRange	Custom function for expanding ranges in brace patterns. The <code>full-range</code> library is ideal for this purpose, or you can use custom code to do whatever you need.	Function	undefined	function undefined
options.expandRange	Custom function for expanding ranges in brace patterns. The <code>full-range</code> library is ideal for this purpose, or you can use custom code to do whatever you need.	Function	undefined	function undefined
ignore	One or more glob patterns for excluding paths, etc.	Array string	false	ignore
keepQuotes	Retain quotes in the strings from the results matched from the regular expressions in the generated regex, since quoted strings may also be used as alternatives to backslashes.	Boolean	false	keepQuotes
literalBrackets	When true, braces escaped so that only the global pattern will be treated as literal brackets.	Boolean	false	literalBrackets
matchBase	Alias for basename	Boolean	false	matchBase
maxLength	Limit the max length of the input string if the input string is thrown if the input string is longer than the max length. An empty input string. An empty input string.	Boolean	65536	maxLength
noRace	Disable brace matching so that {a, b} and {1..3} would be treated as literal characters.	Boolean	false	noRace
noBracket	Disable matching with regular expressions.	Boolean	false	noBracket
noCase	Boolean	false	false	noCase

```

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
nodupes	boolean	true	Make matching insensitive. Equivalent to the regex <code>i</code> flag, but this option will be overridden by the <code>flags</code> option.
noext	boolean	false	Deprecated, use <code>nounique</code> instead. This option will be removed in a future major version. By default duplicates are removed. Disables uniqueification by this option to false.
noextglob	boolean	false	Alias for <code>noext</code> .
noglobstar	boolean	false	Disable support for matching nested directories with <code>(...)</code> .
nonegate	boolean	false	Disable support for negating with <code>!</code> .
noquantifiers	boolean	false	Disable support for quantifiers (like <code>a{1,2}</code>) and brace patterns expanded.
onIgnore	function	undefined	Function to be called for ignored items.
onMatch	function	undefined	Function to be called for matched items.
onResult	function	undefined	Function to be called for all items, regardless of whether they were matched or ignored.

