micromatch / **micromatch**

Watch 17 | Star 620 | Fork 61

Highly optimized wildcard and glob matching library. Faster, drop-in replacement to minimatch and multimatch. Used by babel core, yarn, jest, browser-sync, documentation.js, stylelint, nyc, and many others! https://github.com/micromatch

glob glob-pattern glob-matching matcher bash javascript multimatch minimatch regex regular-expression extglob globbing wildmat node-glob globby snapdragon node micromatch wildcard extended-glob

509 commits | **9** branches | **50** releases | **14** contributors | MIT

Branch: **master** | New pull request | Find file

Clone or download

jonschlinkert 3.1.10

Latest commit `10aacd7` Mar 23, 2018

.github	fixing typo in PR template	Jul 3, 2017
benchmark	update benchmark code	Sep 11, 2017
examples	fixes #119	Feb 20, 2018
lib	fixes #119	Feb 20, 2018
test	update tests	Mar 22, 2018
.editorconfig	run update, lint, upgrade deps	Sep 8, 2017
.eslintrc.json	run update, lint tests, upgrade deps	Feb 18, 2018
.gitattributes	run update	Oct 20, 2016
.gitignore	run update, lint tests, upgrade deps	Feb 18, 2018
.travis.yml	run update, lint tests, upgrade deps	Feb 18, 2018
.verb.md	update benchmark code	Sep 11, 2017
CHANGELOG.md	add link to release notes	Jun 5, 2017
LICENSE	run update, lint tests, upgrade deps	Feb 18, 2018
README.md	run verb to generate readme documentation	Feb 18, 2018

appveyor.yml	silence appveyor	Feb 20, 2018
gulpfile.js	fix url	May 29, 2017
index.js	fix typeof check on `.braces` method	Feb 24, 2018
package.json	3.1.10	Mar 22, 2018

README.md

micromatch npm v3.1.10 downloads 26M/month downloads 294M Travis passing AppVeyor passing

> Glob matching for javascript/node.js. A drop-in replacement and faster alternative to minimatch and multimatch.

Please consider following this project's author, Jon Schlinkert, and consider starring the project to show your❤ and support.

Table of Contents

▶ **Details**

Install

Install with npm:

```
$ npm install --save micromatch
```

Quickstart

```
var mm = require('micromatch');
mm(list, patterns[, options]);
```

The main export takes a list of strings and one or more glob patterns:

```
console.log(mm(['foo', 'bar', 'qux'], ['f*', 'b*']));
//=> ['foo', 'bar']
```

Use .isMatch() to get true/false:

```
console.log(mm.isMatch('foo', 'f*'));
//=> true
```

Switching from minimatch and multimatch is easy!

Why use micromatch?

> micromatch is a drop-in replacement for minimatch and multimatch

- Supports all of the same matching features as minimatch and multimatch
- Micromatch uses snapdragon for parsing and compiling globs, which provides granular control over the entire conversion process in a way that is easy to understand, reason about, and maintain.
- More consistently accurate matching than minimatch, with more than 36,000 test assertions to prove it.
- More complete support for the Bash 4.3 specification than minimatch and multimatch. In fact, micromatch passes *all of the spec tests* from bash, including some that bash still fails.
- Faster matching, from a combination of optimized glob patterns, faster algorithms, and regex caching.
- Micromatch is safer, and is not subject to DoS with brace patterns, like minimatch and multimatch.
- More reliable windows support than minimatch and multimatch.

Matching features

- Support for multiple glob patterns (no need for wrappers like multimatch)
- Wildcards (`**`, `*.js`)
- Negation (`'!a/*.js'`, `'*!(b).js']`)
- extglobs (`+(x|y)`, `!(a|b)`)
- POSIX character classes (`[[:alpha:][:digit:]]`)
- brace expansion (`foo/{1..5}.md`, `bar/{a,b,c}.js`)
- regex character classes (`foo-[1-5].js`)
- regex logical "or" (`foo/(abc|xyz).js`)

You can mix and match these features to create whatever patterns you need!

Switching to micromatch

There is one notable difference between micromatch and minimatch in regards to how backslashes are handled. See the notes about backslashes for more information.

From minimatch

Use mm.isMatch() instead of `minimatch()`:

```
mm.isMatch('foo', 'b*');
//=> false
```

Use mm.match() instead of `minimatch.match()`:

```
mm.match(['foo', 'bar'], 'b*');
//=> 'bar'
```

From multimatch

Same signature:

```
mm(['foo', 'bar', 'baz'], ['f*', '*z']);
//=> ['foo', 'baz']
```

API

micromatch

The main function takes a list of strings and one or more glob patterns to use for matching.

Params

- `list` **{Array}**: A list of strings to match
- `patterns` **{String|Array}**: One or more glob patterns to use for matching.
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Array}**: Returns an array of matches

Example

```
var mm = require('micromatch');
mm(list, patterns[, options]);

console.log(mm(['a.js', 'a.txt'], ['*.js']));
//=> [ 'a.js' ]
```

.match

Similar to the main function, but `pattern` must be a string.

Params

- `list` **{Array}**: Array of strings to match
- `pattern` **{String}**: Glob pattern to use for matching.
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Array}**: Returns an array of matches

Example

```js
var mm = require('micromatch');
mm.match(list, pattern[, options]);

console.log(mm.match(['a.a', 'a.aa', 'a.b', 'a.c'], '*.a'));
//=> ['a.a', 'a.aa']
```

.isMatch

Returns true if the specified `string` matches the given glob `pattern`.

Params

- `string` **{String}**: String to match
- `pattern` **{String}**: Glob pattern to use for matching.
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Boolean}**: Returns true if the string matches the glob pattern.

Example

```js
var mm = require('micromatch');
mm.isMatch(string, pattern[, options]);

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

.some

Returns true if some of the strings in the given `list` match any of the given glob `patterns`.

Params

- `list` **{String|Array}**: The string or array of strings to test. Returns as soon as the first match is found.
- `patterns` **{String|Array}**: One or more glob patterns to use for matching.
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Boolean}**: Returns true if any patterns match `str`

Example

```js
var mm = require('micromatch');
mm.some(list, patterns[, options]);

console.log(mm.some(['foo.js', 'bar.js'], ['*.js', '!foo.js']));
// true
console.log(mm.some(['foo.js'], ['*.js', '!foo.js']));
// false
```

.every

Returns true if every string in the given `list` matches any of the given glob `patterns`.

Params

- `list` **{String|Array}**: The string or array of strings to test.

- `patterns` **{String|Array}**: One or more glob patterns to use for matching.
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Boolean}**: Returns true if any patterns match `str`

Example

```
var mm = require('micromatch');
mm.every(list, patterns[, options]);

console.log(mm.every('foo.js', ['foo.js']));
// true
console.log(mm.every(['foo.js', 'bar.js'], ['*.js']));
// true
console.log(mm.every(['foo.js', 'bar.js'], ['*.js', '!foo.js']));
// false
console.log(mm.every(['foo.js'], ['*.js', '!foo.js']));
// false
```

.any

Returns true if **any** of the given glob `patterns` match the specified `string`.

Params

- `str` **{String|Array}**: The string to test.
- `patterns` **{String|Array}**: One or more glob patterns to use for matching.
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Boolean}**: Returns true if any patterns match `str`

Example

```
var mm = require('micromatch');
mm.any(string, patterns[, options]);

console.log(mm.any('a.a', ['b.*', '*.a']));
//=> true
console.log(mm.any('a.a', 'b.*'));
//=> false
```

.all

Returns true if **all** of the given `patterns` match the specified string.

Params

- `str` **{String|Array}**: The string to test.
- `patterns` **{String|Array}**: One or more glob patterns to use for matching.
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Boolean}**: Returns true if any patterns match `str`

Example

```js
var mm = require('micromatch');
mm.all(string, patterns[, options]);

console.log(mm.all('foo.js', ['foo.js']));
// true

console.log(mm.all('foo.js', ['*.js', '!foo.js']));
// false

console.log(mm.all('foo.js', ['*.js', 'foo.js']));
// true

console.log(mm.all('foo.js', ['*.js', 'f*', '*o*', '*o.js']));
// true
```

.not

Returns a list of strings that ***do not match any*** of the given `patterns`.

Params

- `list` **{Array}**: Array of strings to match.
- `patterns` **{String|Array}**: One or more glob pattern to use for matching.
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Array}**: Returns an array of strings that **do not match** the given patterns.

Example

```js
var mm = require('micromatch');
mm.not(list, patterns[, options]);

console.log(mm.not(['a.a', 'b.b', 'c.c'], '*.a'));
//=> ['b.b', 'c.c']
```

.contains

Returns true if the given `string` contains the given pattern. Similar to .isMatch but the pattern can match any part of the string.

Params

- `str` **{String}**: The string to match.
- `patterns` **{String|Array}**: Glob pattern to use for matching.
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Boolean}**: Returns true if the patter matches any part of `str`.

Example

```js
var mm = require('micromatch');
mm.contains(string, pattern[, options]);

console.log(mm.contains('aa/bb/cc', '*b'));
//=> true
console.log(mm.contains('aa/bb/cc', '*d'));
//=> false
```

.matchKeys

Filter the keys of the given object with the given `glob` pattern and `options`. Does not attempt to match nested keys. If you need this feature, use glob-object instead.

Params

- `object` **{Object}**: The object with keys to filter.
- `patterns` **{String|Array}**: One or more glob patterns to use for matching.

- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Object}**: Returns an object with only keys that match the given patterns.

Example

```
var mm = require('micromatch');
mm.matchKeys(object, patterns[, options]);

var obj = { aa: 'a', ab: 'b', ac: 'c' };
console.log(mm.matchKeys(obj, '*b'));
//=> { ab: 'b' }
```

.matcher

Returns a memoized matcher function from the given glob `pattern` and `options`. The returned function takes a string to match as its only argument and returns true if the string is a match.

Params

- `pattern` **{String}**: Glob pattern
- `options` **{Object}**: See available options for changing how matches are performed.
- `returns` **{Function}**: Returns a matcher function.

Example

```
var mm = require('micromatch');
mm.matcher(pattern[, options]);

var isMatch = mm.matcher('*.!(*a)');
console.log(isMatch('a.a'));
//=> false
console.log(isMatch('a.b'));
//=> true
```

.capture

Returns an array of matches captured by `pattern` in `string, or` null` if the pattern did not match.

Params

- `pattern` **{String}**: Glob pattern to use for matching.
- `string` **{String}**: String to match
- `options` **{Object}**: See available options for changing how matches are performed
- `returns` **{Boolean}**: Returns an array of captures if the string matches the glob pattern, otherwise `null`.

Example

```
var mm = require('micromatch');
mm.capture(pattern, string[, options]);

console.log(mm.capture('test/*.js', 'test/foo.js'));
//=> ['foo']
console.log(mm.capture('test/*.js', 'foo/bar.css'));
//=> null
```

.makeRe

Create a regular expression from the given glob `pattern`.

Params

- `pattern` **{String}**: A glob pattern to convert to regex.
- `options` **{Object}**: See available options for changing how matches are performed.
- `returns` **{RegExp}**: Returns a regex created from the given pattern.

Example

```
var mm = require('micromatch');
mm.makeRe(pattern[, options]);

console.log(mm.makeRe('*.js'));
//=> /^(?:(\.[\\\/])?(?!\.)(?=.)[^\/]*?\.js)$/
```

.braces

Expand the given brace `pattern`.

Params

- `pattern` **{String}**: String with brace pattern to expand.
- `options` **{Object}**: Any options to change how expansion is performed. See thebraces library for all available options.
- `returns` **{Array}**

Example

```
var mm = require('micromatch');
console.log(mm.braces('foo/{a,b}/bar'));
//=> ['foo/(a|b)/bar']

console.log(mm.braces('foo/{a,b}/bar', {expand: true}));
//=> ['foo/(a|b)/bar']
```

.create

Parses the given glob `pattern` and returns an array of abstract syntax trees (ASTs), with the compiled `output` and optional source `map` on each AST.

Params

- `pattern` **{String}**: Glob pattern to parse and compile.
- `options` **{Object}**: Any options to change how parsing and compiling is performed.
- `returns` **{Object}**: Returns an object with the parsed AST, compiled string and optional source map.

Example

```
var mm = require('micromatch');
mm.create(pattern[, options]);

console.log(mm.create('abc/*.js'));
// [{ options: { source: 'string', sourcemap: true },
//   state: {},
//   compilers:
//    { ... },
//   output: '(\\.[\\\\\\/])?abc\\/(?!\\.)(?=.)[^\\/]*?\\.js',
//   ast:
//    { type: 'root',
//      errors: [],
//      nodes:
//       [ ... ],
//      dot: false,
//      input: 'abc/*.js' },
//   parsingErrors: [],
//   map:
//    { version: 3,
//      sources: [ 'string' ],
//      names: [],
//      mappings: 'AAAA,GAAG,EAAC,kBAAC,EAAC,EAAE',
//      sourcesContent: [ 'abc/*.js' ] },
//   position: { line: 1, column: 28 },
//   content: {},
//   files: {},
//   idx: 6 }]
```

.parse

Parse the given `str` with the given `options`.

Params

- `str` **{String}**
- `options` **{Object}**
- `returns` **{Object}**: Returns an AST

Example

```
var mm = require('micromatch');
mm.parse(pattern[, options]);

var ast = mm.parse('a/{b,c}/d');
console.log(ast);
// { type: 'root',
//   errors: [],
//   input: 'a/{b,c}/d',
//   nodes:
//    [ { type: 'bos', val: '' },
//      { type: 'text', val: 'a/' },
//      { type: 'brace',
//        nodes:
//         [ { type: 'brace.open', val: '{' },
//           { type: 'text', val: 'b,c' },
//           { type: 'brace.close', val: '}' } ] },
//      { type: 'text', val: '/d' },
//      { type: 'eos', val: '' } ] }
```

.compile

Compile the given `ast` or string with the given `options`.

Params

- `ast` **{Object|String}**
- `options` **{Object}**
- `returns` **{Object}**: Returns an object that has an `output` property with the compiled string.

Example

```
var mm = require('micromatch');
mm.compile(ast[, options]);

var ast = mm.parse('a/{b,c}/d');
console.log(mm.compile(ast));
// { options: { source: 'string' },
//   state: {},
//   compilers:
//    { eos: [Function],
//      noop: [Function],
//      bos: [Function],
//      brace: [Function],
//      'brace.open': [Function],
//      text: [Function],
//      'brace.close': [Function] },
//   output: [ 'a/(b|c)/d' ],
//   ast:
//    { ... },
//   parsingErrors: [] }
```

.clearCache

Clear the regex cache.

Example

```
mm.clearCache();
```

Options

- basename
- bash
- cache
- dot
- failglob
- ignore
- matchBase
- nobrace
- nocase
- nodupes
- noext
- noglobstar
- nonull
- nullglob
- snapdragon
- sourcemap
- unescape
- unixify

options.basename

Allow glob patterns without slashes to match a file path based on its basename. Same behavior as minimatch option `matchBase`.

Type: `Boolean`

Default: `false`

Example

```
mm(['a/b.js', 'a/c.md'], '*.js');
//=> []

mm(['a/b.js', 'a/c.md'], '*.js', {matchBase: true});
//=> ['a/b.js']
```

options.bash

Enabled by default, this option enforces bash-like behavior with stars immediately following a bracket expression. Bash bracket expressions are similar to regex character classes, but unlike regex, a star following a bracket expression **does not repeat the bracketed characters**. Instead, the star is treated the same as an other star.

Type: `Boolean`

Default: `true`

Example

```
var files = ['abc', 'ajz'];
console.log(mm(files, '[a-c]*'));
//=> ['abc', 'ajz']

console.log(mm(files, '[a-c]*', {bash: false}));
```

options.cache

Disable regex and function memoization.

Type: `Boolean`

Default: `undefined`

options.dot

Match dotfiles. Same behavior as minimatch option `dot`.

Type: `Boolean`

Default: `false`

options.failglob

Similar to the `--failglob` behavior in Bash, throws an error when no matches are found.

Type: `Boolean`

Default: `undefined`

options.ignore

String or array of glob patterns to match files to ignore.

Type: `String|Array`

Default: `undefined`

options.matchBase

Alias for options.basename.

options.nobrace

Disable expansion of brace patterns. Same behavior asminimatch option `nobrace`.

Type: `Boolean`

Default: `undefined`

See braces for more information about extended brace expansion.

options.nocase

Use a case-insensitive regex for matching files. Same behavior asminimatch.

Type: `Boolean`

Default: `undefined`

options.nodupes

Remove duplicate elements from the result array.

Type: `Boolean`

Default: `undefined`

Example

Example of using the `unescape` and `nodupes` options together:

```
mm.match(['a/b/c', 'a/b/c'], 'a/b/c');
//=> ['a/b/c', 'a/b/c']

mm.match(['a/b/c', 'a/b/c'], 'a/b/c', {nodupes: true});
//=> ['abc']
```

options.noext

Disable extglob support, so that extglobs are regarded as literal characters.

Type: `Boolean`

Default: `undefined`

Examples

```
mm(['a/z', 'a/b', 'a/!(z)'], 'a/!(z)');
//=> ['a/b', 'a/!(z)']

mm(['a/z', 'a/b', 'a/!(z)'], 'a/!(z)', {noext: true});
//=> ['a/!(z)'] (matches only as literal characters)
```

options.nonegate

Disallow negation (`!`) patterns, and treat leading `!` as a literal character to match.

Type: `Boolean`

Default: `undefined`

options.noglobstar

Disable matching with globstars (`**`).

Type: `Boolean`

Default: `undefined`

```
mm(['a/b', 'a/b/c', 'a/b/c/d'], 'a/**');
//=> ['a/b', 'a/b/c', 'a/b/c/d']

mm(['a/b', 'a/b/c', 'a/b/c/d'], 'a/**', {noglobstar: true});
//=> ['a/b']
```

options.nonull

Alias for options.nullglob.

options.nullglob

If `true` , when no matches are found the actual (arrayified) glob pattern is returned instead of an empty array. Same behavior as minimatch option `nonull` .

Type: `Boolean`

Default: `undefined`

options.snapdragon

Pass your own instance of snapdragon, to customize parsers or compilers.

Type: `Object`

Default: `undefined`

options.sourcemap

Generate a source map by enabling the `sourcemap` option with the `.parse` , `.compile` , or `.create` methods.

(Note that sourcemaps are currently not enabled for brace patterns)

Examples

```js
var mm = require('micromatch');
var pattern = '*(*(of*(a)x)z)';

var res = mm.create('abc/*.js', {sourcemap: true});
console.log(res.map);
// { version: 3,
//   sources: [ 'string' ],
//   names: [],
//   mappings: 'AAAA,GAAG,EAAC,iBAAC,EAAC,EAAE',
//   sourcesContent: [ 'abc/*.js' ] }

var ast = mm.parse('abc/**/*.js');
var res = mm.compile(ast, {sourcemap: true});
console.log(res.map);
// { version: 3,
//   sources: [ 'string' ],
//   names: [],
//   mappings: 'AAAA,GAAG,EAAC,2BAAE,EAAC,iBAAC,EAAC,EAAE',
//   sourcesContent: [ 'abc/**/*.js' ] }

var ast = mm.parse(pattern);
var res = mm.compile(ast, {sourcemap: true});
console.log(res.map);
// { version: 3,
//   sources: [ 'string' ],
//   names: [],
//   mappings: 'AAAA,CAAE,CAAE,EAAE,CAAE,CAAC,EAAC,CAAC,EAAC,CAAC,EAAC',
//   sourcesContent: [ '*(*(of*(a)x)z)' ] }
```

options.unescape

Remove backslashes from returned matches.

Type: `Boolean`

Default: `undefined`

Example

In this example we want to match a literal `*`:

```js
mm.match(['abc', 'a\\*c'], 'a\\*c');
//=> ['a\\*c']

mm.match(['abc', 'a\\*c'], 'a\\*c', {unescape: true});
//=> ['a*c']
```

options.unixify

Convert path separators on returned files to posix/unix-style forward slashes.

Type: `Boolean`

Default: `true` on windows, `false` everywhere else

Example

```js
mm.match(['a\\b\\c'], 'a/**');
//=> ['a/b/c']

mm.match(['a\\b\\c'], {unixify: false});
//=> ['a\\b\\c']
```

Extended globbing

Micromatch also supports extended globbing features.

extglobs

Extended globbing, as described by the bash man page:

pattern	regex equivalent	description
`?(pattern)`	`(pattern)?`	Matches zero or one occurrence of the given patterns
`*(pattern)`	`(pattern)*`	Matches zero or more occurrences of the given patterns
`+(pattern)`	`(pattern)+`	Matches one or more occurrences of the given patterns
`@(pattern)`	`(pattern)` *	Matches one of the given patterns
`!(pattern)`	N/A (equivalent regex is much more complicated)	Matches anything except one of the given patterns

* Note that `@` isn't a RegEx character.

Powered by extglob. Visit that library for the full range of options or to report extglob related issues.

braces

Brace patterns can be used to match specific ranges or sets of characters. For example, the pattern `*/{1..3}/*` would match any of following strings:

```
foo/1/bar
foo/2/bar
foo/3/bar
baz/1/qux
baz/2/qux
baz/3/qux
```

Visit braces to see the full range of features and options related to brace expansion, or to create brace matching or expansion related issues.

regex character classes

Given the list: `['a.js', 'b.js', 'c.js', 'd.js', 'E.js']`:

- `[ac].js`: matches both `a` and `c`, returning `['a.js', 'c.js']`
- `[b-d].js`: matches from `b` to `d`, returning `['b.js', 'c.js', 'd.js']`
- `[b-d].js`: matches from `b` to `d`, returning `['b.js', 'c.js', 'd.js']`
- `a/[A-Z].js`: matches and uppercase letter, returning `['a/E.md']`

Learn about regex character classes.

regex groups

Given `['a.js', 'b.js', 'c.js', 'd.js', 'E.js']`:

- `(a|c).js`: would match either `a` or `c`, returning `['a.js', 'c.js']`
- `(b|d).js`: would match either `b` or `d`, returning `['b.js', 'd.js']`
- `(b|[A-Z]).js`: would match either `b` or an uppercase letter, returning `['b.js', 'E.js']`

As with regex, parens can be nested, so patterns like `((a|b)|c)/b` will work. Although brace expansion might be friendlier to use, depending on preference.

POSIX bracket expressions

POSIX brackets are intended to be more user-friendly than regex character classes. This of course is in the eye of the beholder.

Example

```
mm.isMatch('a1', '[[:alpha:][:digit:]]');
//=> true

mm.isMatch('a1', '[[:alpha:][:alpha:]]');
//=> false
```

See expand-brackets for more information about bracket expressions.

Notes

Bash 4.3 parity

Whenever possible matching behavior is based on behavior Bash 4.3, which is mostly consistent with minimatch.

However, it's suprising how many edge cases and rabbit holes there are with glob matching, and since there is no real glob specification, and micromatch is more accurate than both Bash and minimatch, there are cases where best-guesses were made for behavior. In a few cases where Bash had no answers, we used wildmatch (used by git) as a fallback.

Backslashes

There is an important, notable difference between minimatch and micromatch*in regards to how backslashes are handled*in glob patterns.

- Micromatch exclusively and explicitly reserves backslashes for escaping characters in a glob pattern, even on windows. This is consistent with bash behavior.
- Minimatch converts all backslashes to forward slashes, which means you can't use backslashes to escape any characters in your glob patterns.

We made this decision for micromatch for a couple of reasons:

- consistency with bash conventions.
- glob patterns are not filepaths. They are a type ofregular language that is converted to a JavaScript regular expression. Thus, when forward slashes are defined in a glob pattern, the resulting regular expression will match windows or POSIX path separators just fine.

A note about joining paths to globs

Note that when you pass something like `path.join('foo', '*')` to micromatch, you are creating a filepath and expecting it to still work as a glob pattern. This causes problems on windows, since the `path.sep` is `\\`.

In other words, since `\\` is reserved as an escape character in globs, on windows `path.join('foo', '*')` would result in `foo\\*`, which tells micromatch to match `*` as a literal character. This is the same behavior as bash.

Contributing

All contributions are welcome! Please read the contributing guide to get started.

Bug reports

Please create an issue if you encounter a bug or matching behavior that doesn't seem correct. If you find a matching-related issue, please:

- research existing issues first (open and closed)
- visit the GNU Bash documentation to see how Bash deals with the pattern
- visit the minimatch documentation to cross-check expected behavior in node.js
- if all else fails, since there is no real specification for globs we will probably need to discuss expected behavior and decide how to resolve it. which means any detail you can provide to help with this discussion would be greatly appreciated.

Platform issues

It's important to us that micromatch work consistently on all platforms. If you encounter any platform-specific matching or path related issues, please let us know (pull requests are also greatly appreciated).

Benchmarks

Running benchmarks

Install dev dependencies:

```
npm i -d && npm run benchmark
```

Latest results

As of February 18, 2018 (longer bars are better):

```
# braces-globstar-large-list (485691 bytes)
  micromatch ██████████████████████████████████████████████████ (517 ops/sec ±0.49%)
  minimatch  █ (18.92 ops/sec ±0.54%)
  multimatch █ (18.94 ops/sec ±0.62%)

  micromatch is faster by an avg. of 2,733%

# braces-multiple (3362 bytes)
  micromatch ██████████████████████████████████████████████████ (33,625 ops/sec ±0.45%)
  minimatch    (2.92 ops/sec ±3.26%)
  multimatch  (2.90 ops/sec ±2.76%)

  micromatch is faster by an avg. of 1,156,935%

# braces-range (727 bytes)
  micromatch █████████████████████████████████████████████████ (155,220 ops/sec ±0.56%)
  minimatch  ██████ (20,186 ops/sec ±1.27%)
  multimatch ██████ (19,809 ops/sec ±0.60%)

  micromatch is faster by an avg. of 776%

# braces-set (2858 bytes)
  micromatch █████████████████████████████████████████████████ (24,354 ops/sec ±0.92%)
  minimatch  █████ (2,566 ops/sec ±0.56%)
  multimatch ████ (2,431 ops/sec ±1.25%)

  micromatch is faster by an avg. of 975%

# globstar-large-list (485686 bytes)
  micromatch █████████████████████████████████████████████████ (504 ops/sec ±0.45%)
  minimatch  ███ (33.36 ops/sec ±1.08%)
  multimatch ███ (33.19 ops/sec ±1.35%)

  micromatch is faster by an avg. of 1,514%

# globstar-long-list (90647 bytes)
  micromatch ██████████████████████████████████████████████████ (2,694 ops/sec ±1.08%)
  minimatch  ████████████████ (870 ops/sec ±1.09%)
  multimatch ████████████████ (862 ops/sec ±0.84%)

  micromatch is faster by an avg. of 311%

# globstar-short-list (182 bytes)
  micromatch ██████████████████████████████████████████████████ (328,921 ops/sec ±1.06%)
  minimatch  █████████ (64,808 ops/sec ±1.42%)
  multimatch ████████ (57,991 ops/sec ±2.11%)

  micromatch is faster by an avg. of 536%

# no-glob (701 bytes)
  micromatch █████████████████████████████████████████████████ (415,935 ops/sec ±0.36%)
  minimatch  ███████████ (92,730 ops/sec ±1.44%)
  multimatch █████████ (81,958 ops/sec ±2.13%)

  micromatch is faster by an avg. of 476%

# star-basename-long (12339 bytes)
  micromatch █████████████████████████████████████████████████ (7,963 ops/sec ±0.36%)
  minimatch  ███████████████████████████████ (5,072 ops/sec ±0.83%)
  minimatch  ███████████████████████████████ (5,072 ops/sec ±0.83%)
  multimatch ███████████████████████████████ (5,028 ops/sec ±0.40%)

  micromatch is faster by an avg. of 158%

# star-basename-short (349 bytes)
  micromatch ██████████████████████████████████████████████████ (269,552 ops/sec ±0.70%)
  minimatch  ██████████████████████ (122,457 ops/sec ±1.39%)
  multimatch ████████████████████ (110,788 ops/sec ±1.99%)

  micromatch is faster by an avg. of 231%

# star-folder-long (19207 bytes)
  micromatch █████████████████████████████████████████████████ (3,806 ops/sec ±0.38%)
  minimatch  ████████████████████████████ (2,204 ops/sec ±0.32%)
  multimatch ██████████████████████████ (2,020 ops/sec ±1.07%)

  micromatch is faster by an avg. of 180%

# star-folder-short (551 bytes)
  micromatch ██████████████████████████████████████████████████ (249,077 ops/sec ±0.40%)
  minimatch  ███████████ (59,431 ops/sec ±1.67%)
  multimatch ███████████ (55,569 ops/sec ±1.43%)

  micromatch is faster by an avg. of 433%
```

About

▶ **Contributing**

▶ **Running Tests**

▶ **Building docs**

Related projects

You might also be interested in these projects:

- braces: Bash-like brace expansion, implemented in JavaScript. Safer than other brace expansion libs, with complete support… more | homepage
- expand-brackets: Expand POSIX bracket expressions (character classes) in glob patterns. | homepage
- extglob: Extended glob support for JavaScript. Adds (almost) the expressive power of regular expressions to glob… more | homepage
- fill-range: Fill in a range of numbers or letters, optionally passing an increment or `step` to… more | homepage
- nanomatch: Fast, minimal glob matcher for node.js. Similar to micromatch, minimatch and multimatch, but complete Bash… more | homepage

Contributors

Commits	Contributor
457	jonschlinkert
12	es128
8	doowb
3	paulmillr
2	TrySound
2	MartinKolarik
2	charlike-old
1	amilajack
1	mrmlnc
1	devongovett
1	DianeLooney
1	UltCombo
1	tomByrer
1	fidian

Author

Jon Schlinkert

- linkedin/in/jonschlinkert
- github/jonschlinkert
- twitter/jonschlinkert

License

This file was generated by verb-generate-readme, v0.6.0, on February 18, 2018.