# braces [Donate](https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=W8YFZ425KND68) [NPM version](https://www.npmjs.com/package/braces) [NPM monthly downloads](https://npmjs.org/package/braces) [NPM total downloads](https://npmjs.org/package/braces) [Linux Build Status](https://travis-ci.org/micromatch/braces)

Bash-like brace expansion, implemented in JavaScript. Safer than other brace expansion libs, with complete support for the Bash 4.3 braces specification, without sacrificing speed.

Please consider following this project's author, [Jon Schlinkert](https://github.com/jonschlinkert), and consider starring the project to show your :heart: and support.

## Install

Install with [npm](https://www.npmjs.com/):

$ npm install --save braces

## v3.0.0 Released!!

See the [changelog](http://changelog.md) for details.

## Why use braces?

Brace patterns make globs more powerful by adding the ability to match specific ranges and sequences of characters.

* **Accurate** - complete support for the [Bash 4.3 Brace Expansion](http://www.gnu.org/software/bash/) specification (passes all of the Bash braces tests)
* [**fast and performant**](#benchmarks) - Starts fast, runs fast and [scales well](#performance) as patterns increase in complexity.
* **Organized code base** - The parser and compiler are easy to maintain and update when edge cases crop up.
* **Well-tested** - Thousands of test assertions, and passes all of the Bash, minimatch, and [brace-expansion](https://github.com/juliangruber/brace-expansion) unit tests (as of the date this was written).
* **Safer** - You shouldn't have to worry about users defining aggressive or malicious brace patterns that can break your application. Braces takes measures to prevent malicious regex that can be used for DDoS attacks (see [catastrophic backtracking](https://www.regular-expressions.info/catastrophic.html)).
* [Supports lists](#lists) - (aka "sets") a/{b,c}/d => ['a/b/d', 'a/c/d']
* [Supports sequences](#sequences) - (aka "ranges") {01..03} => ['01', '02', '03']
* [Supports steps](#steps) - (aka "increments") {2..10..2} => ['2', '4', '6', '8', '10']
* [Supports escaping](#escaping) - To prevent evaluation of special characters.

## Usage

The main export is a function that takes one or more brace patterns and options.

const braces = require('braces');

// braces(patterns[, options]);

console.log(braces(['{01..05}', '{a..e}']));

//=> ['(0[1-5])', '([a-e])']

console.log(braces(['{01..05}', '{a..e}'], { expand: true }));

//=> ['01', '02', '03', '04', '05', 'a', 'b', 'c', 'd', 'e']

### Brace Expansion vs. Compilation

By default, brace patterns are compiled into strings that are optimized for creating regular expressions and matching.

**Compiled**

console.log(braces('a/{x,y,z}/b'));

//=> ['a/(x|y|z)/b']

console.log(braces(['a/{01..20}/b', 'a/{1..5}/b']));

//=> [ 'a/(0[1-9]|1[0-9]|20)/b', 'a/([1-5])/b' ]

**Expanded**

Enable brace expansion by setting the expand option to true, or by using [braces.expand()](#expand) (returns an array similar to what you'd expect from Bash, or echo {1..5}, or [minimatch](https://github.com/isaacs/minimatch)):

console.log(braces('a/{x,y,z}/b', { expand: true }));

//=> ['a/x/b', 'a/y/b', 'a/z/b']

console.log(braces.expand('{01..10}'));

//=> ['01','02','03','04','05','06','07','08','09','10']

### Lists

Expand lists (like Bash "sets"):

console.log(braces('a/{foo,bar,baz}/\*.js'));

//=> ['a/(foo|bar|baz)/\*.js']

console.log(braces.expand('a/{foo,bar,baz}/\*.js'));

//=> ['a/foo/\*.js', 'a/bar/\*.js', 'a/baz/\*.js']

### Sequences

Expand ranges of characters (like Bash "sequences"):

console.log(braces.expand('{1..3}')); // ['1', '2', '3']

console.log(braces.expand('a/{1..3}/b')); // ['a/1/b', 'a/2/b', 'a/3/b']

console.log(braces('{a..c}', { expand: true })); // ['a', 'b', 'c']

console.log(braces('foo/{a..c}', { expand: true })); // ['foo/a', 'foo/b', 'foo/c']

// supports zero-padded ranges

console.log(braces('a/{01..03}/b')); //=> ['a/(0[1-3])/b']

console.log(braces('a/{001..300}/b')); //=> ['a/(0{2}[1-9]|0[1-9][0-9]|[12][0-9]{2}|300)/b']

See [fill-range](https://github.com/jonschlinkert/fill-range) for all available range-expansion options.

### Steppped ranges

Steps, or increments, may be used with ranges:

console.log(braces.expand('{2..10..2}'));

//=> ['2', '4', '6', '8', '10']

console.log(braces('{2..10..2}'));

//=> ['(2|4|6|8|10)']

When the [.optimize](#optimize) method is used, or [options.optimize](#optionsoptimize) is set to true, sequences are passed to [to-regex-range](https://github.com/jonschlinkert/to-regex-range) for expansion.

### Nesting

Brace patterns may be nested. The results of each expanded string are not sorted, and left to right order is preserved.

**"Expanded" braces**

console.log(braces.expand('a{b,c,/{x,y}}/e'));

//=> ['ab/e', 'ac/e', 'a/x/e', 'a/y/e']

console.log(braces.expand('a/{x,{1..5},y}/c'));

//=> ['a/x/c', 'a/1/c', 'a/2/c', 'a/3/c', 'a/4/c', 'a/5/c', 'a/y/c']

**"Optimized" braces**

console.log(braces('a{b,c,/{x,y}}/e'));

//=> ['a(b|c|/(x|y))/e']

console.log(braces('a/{x,{1..5},y}/c'));

//=> ['a/(x|([1-5])|y)/c']

### Escaping

**Escaping braces**

A brace pattern will not be expanded or evaluted if *either the opening or closing brace is escaped*:

console.log(braces.expand('a\\{d,c,b}e'));

//=> ['a{d,c,b}e']

console.log(braces.expand('a{d,c,b\\}e'));

//=> ['a{d,c,b}e']

**Escaping commas**

Commas inside braces may also be escaped:

console.log(braces.expand('a{b\\,c}d'));

//=> ['a{b,c}d']

console.log(braces.expand('a{d\\,c,b}e'));

//=> ['ad,ce', 'abe']

**Single items**

Following bash conventions, a brace pattern is also not expanded when it contains a single character:

console.log(braces.expand('a{b}c'));

//=> ['a{b}c']

## Options

### options.maxLength

**Type**: Number

**Default**: 10,000

**Description**: Limit the length of the input string. Useful when the input string is generated or your application allows users to pass a string, et cetera.

console.log(braces('a/{b,c}/d', { maxLength: 3 })); //=> throws an error

### options.expand

**Type**: Boolean

**Default**: undefined

**Description**: Generate an "expanded" brace pattern (alternatively you can use the braces.expand() method, which does the same thing).

console.log(braces('a/{b,c}/d', { expand: true }));

//=> [ 'a/b/d', 'a/c/d' ]

### options.nodupes

**Type**: Boolean

**Default**: undefined

**Description**: Remove duplicates from the returned array.

### options.rangeLimit

**Type**: Number

**Default**: 1000

**Description**: To prevent malicious patterns from being passed by users, an error is thrown when braces.expand() is used or options.expand is true and the generated range will exceed the rangeLimit.

You can customize options.rangeLimit or set it to Inifinity to disable this altogether.

**Examples**

// pattern exceeds the "rangeLimit", so it's optimized automatically

console.log(braces.expand('{1..1000}'));

//=> ['([1-9]|[1-9][0-9]{1,2}|1000)']

// pattern does not exceed "rangeLimit", so it's NOT optimized

console.log(braces.expand('{1..100}'));

//=> ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41', '42', '43', '44', '45', '46', '47', '48', '49', '50', '51', '52', '53', '54', '55', '56', '57', '58', '59', '60', '61', '62', '63', '64', '65', '66', '67', '68', '69', '70', '71', '72', '73', '74', '75', '76', '77', '78', '79', '80', '81', '82', '83', '84', '85', '86', '87', '88', '89', '90', '91', '92', '93', '94', '95', '96', '97', '98', '99', '100']

### options.transform

**Type**: Function

**Default**: undefined

**Description**: Customize range expansion.

**Example: Transforming non-numeric values**

const alpha = braces.expand('x/{a..e}/y', {

transform(value, index) {

// When non-numeric values are passed, "value" is a character code.

return 'foo/' + String.fromCharCode(value) + '-' + index;

},

});

console.log(alpha);

//=> [ 'x/foo/a-0/y', 'x/foo/b-1/y', 'x/foo/c-2/y', 'x/foo/d-3/y', 'x/foo/e-4/y' ]

**Example: Transforming numeric values**

const numeric = braces.expand('{1..5}', {

transform(value) {

// when numeric values are passed, "value" is a number

return 'foo/' + value \* 2;

},

});

console.log(numeric);

//=> [ 'foo/2', 'foo/4', 'foo/6', 'foo/8', 'foo/10' ]

### options.quantifiers

**Type**: Boolean

**Default**: undefined

**Description**: In regular expressions, quanitifiers can be used to specify how many times a token can be repeated. For example, a{1,3} will match the letter a one to three times.

Unfortunately, regex quantifiers happen to share the same syntax as [Bash lists](#lists)

The quantifiers option tells braces to detect when [regex quantifiers](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp#quantifiers) are defined in the given pattern, and not to try to expand them as lists.

**Examples**

const braces = require('braces');

console.log(braces('a/b{1,3}/{x,y,z}'));

//=> [ 'a/b(1|3)/(x|y|z)' ]

console.log(braces('a/b{1,3}/{x,y,z}', { quantifiers: true }));

//=> [ 'a/b{1,3}/(x|y|z)' ]

console.log(braces('a/b{1,3}/{x,y,z}', { quantifiers: true, expand: true }));

//=> [ 'a/b{1,3}/x', 'a/b{1,3}/y', 'a/b{1,3}/z' ]

### options.keepEscaping

**Type**: Boolean

**Default**: undefined

**Description**: Do not strip backslashes that were used for escaping from the result.

## What is "brace expansion"?

Brace expansion is a type of parameter expansion that was made popular by unix shells for generating lists of strings, as well as regex-like matching when used alongside wildcards (globs).

In addition to "expansion", braces are also used for matching. In other words:

* [brace expansion](#brace-expansion) is for generating new lists
* [brace matching](#brace-matching) is for filtering existing lists

**More about brace expansion** (click to expand) There are two main types of brace expansion:

1. **lists**: which are defined using comma-separated values inside curly braces: {a,b,c}
2. **sequences**: which are defined using a starting value and an ending value, separated by two dots: a{1..3}b. Optionally, a third argument may be passed to define a "step" or increment to use: a{1..100..10}b. These are also sometimes referred to as "ranges".

Here are some example brace patterns to illustrate how they work:

**Sets**

{a,b,c} => a b c

{a,b,c}{1,2} => a1 a2 b1 b2 c1 c2

**Sequences**

{1..9} => 1 2 3 4 5 6 7 8 9

{4..-4} => 4 3 2 1 0 -1 -2 -3 -4

{1..20..3} => 1 4 7 10 13 16 19

{a..j} => a b c d e f g h i j

{j..a} => j i h g f e d c b a

{a..z..3} => a d g j m p s v y

**Combination**

Sets and sequences can be mixed together or used along with any other strings.

{a,b,c}{1..3} => a1 a2 a3 b1 b2 b3 c1 c2 c3

foo/{a,b,c}/bar => foo/a/bar foo/b/bar foo/c/bar

The fact that braces can be "expanded" from relatively simple patterns makes them ideal for quickly generating test fixtures, file paths, and similar use cases.

## Brace matching

In addition to *expansion*, brace patterns are also useful for performing regular-expression-like matching.

For example, the pattern foo/{1..3}/bar would match any of following strings:

foo/1/bar

foo/2/bar

foo/3/bar

But not:

baz/1/qux

baz/2/qux

baz/3/qux

Braces can also be combined with [glob patterns](https://github.com/jonschlinkert/micromatch) to perform more advanced wildcard matching. 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

## Brace matching pitfalls

Although brace patterns offer a user-friendly way of matching ranges or sets of strings, there are also some major disadvantages and potential risks you should be aware of.

### tldr

**"brace bombs"**

* brace expansion can eat up a huge amount of processing resources
* as brace patterns increase *linearly in size*, the system resources required to expand the pattern increase exponentially
* users can accidentally (or intentially) exhaust your system's resources resulting in the equivalent of a DoS attack (bonus: no programming knowledge is required!)

For a more detailed explanation with examples, see the [geometric complexity](#geometric-complexity) section.

### The solution

Jump to the [performance section](#performance) to see how Braces solves this problem in comparison to other libraries.

### Geometric complexity

At minimum, brace patterns with sets limited to two elements have quadradic or O(n^2) complexity. But the complexity of the algorithm increases exponentially as the number of sets, *and elements per set*, increases, which is O(n^c).

For example, the following sets demonstrate quadratic (O(n^2)) complexity:

{1,2}{3,4} => (2X2) => 13 14 23 24

{1,2}{3,4}{5,6} => (2X2X2) => 135 136 145 146 235 236 245 246

But add an element to a set, and we get a n-fold Cartesian product with O(n^c) complexity:

{1,2,3}{4,5,6}{7,8,9} => (3X3X3) => 147 148 149 157 158 159 167 168 169 247 248

249 257 258 259 267 268 269 347 348 349 357

358 359 367 368 369

Now, imagine how this complexity grows given that each element is a n-tuple:

{1..100}{1..100} => (100X100) => 10,000 elements (38.4 kB)

{1..100}{1..100}{1..100} => (100X100X100) => 1,000,000 elements (5.76 MB)

Although these examples are clearly contrived, they demonstrate how brace patterns can quickly grow out of control.

**More information**

Interested in learning more about brace expansion?

* [linuxjournal/bash-brace-expansion](http://www.linuxjournal.com/content/bash-brace-expansion)
* [rosettacode/Brace\_expansion](https://rosettacode.org/wiki/Brace_expansion)
* [cartesian product](https://en.wikipedia.org/wiki/Cartesian_product)

## Performance

Braces is not only screaming fast, it's also more accurate the other brace expansion libraries.

### Better algorithms

Fortunately there is a solution to the ["brace bomb" problem](#brace-matching-pitfalls): *don't expand brace patterns into an array when they're used for matching*.

Instead, convert the pattern into an optimized regular expression. This is easier said than done, and braces is the only library that does this currently.

**The proof is in the numbers**

Minimatch gets exponentially slower as patterns increase in complexity, braces does not. The following results were generated using braces() and minimatch.braceExpand(), respectively.

| **Pattern** | **braces** | **[minimatch][]** |
| --- | --- | --- |
| {1..9007199254740991}[^1] | 298 B (5ms 459μs) | N/A (freezes) |
| {1..1000000000000000} | 41 B (1ms 15μs) | N/A (freezes) |
| {1..100000000000000} | 40 B (890μs) | N/A (freezes) |
| {1..10000000000000} | 39 B (2ms 49μs) | N/A (freezes) |
| {1..1000000000000} | 38 B (608μs) | N/A (freezes) |
| {1..100000000000} | 37 B (397μs) | N/A (freezes) |
| {1..10000000000} | 35 B (983μs) | N/A (freezes) |
| {1..1000000000} | 34 B (798μs) | N/A (freezes) |
| {1..100000000} | 33 B (733μs) | N/A (freezes) |
| {1..10000000} | 32 B (5ms 632μs) | 78.89 MB (16s 388ms 569μs) |
| {1..1000000} | 31 B (1ms 381μs) | 6.89 MB (1s 496ms 887μs) |
| {1..100000} | 30 B (950μs) | 588.89 kB (146ms 921μs) |
| {1..10000} | 29 B (1ms 114μs) | 48.89 kB (14ms 187μs) |
| {1..1000} | 28 B (760μs) | 3.89 kB (1ms 453μs) |
| {1..100} | 22 B (345μs) | 291 B (196μs) |
| {1..10} | 10 B (533μs) | 20 B (37μs) |
| {1..3} | 7 B (190μs) | 5 B (27μs) |

### Faster algorithms

When you need expansion, braces is still much faster.

*(the following results were generated using braces.expand() and minimatch.braceExpand(), respectively)*

| **Pattern** | **braces** | **[minimatch][]** |
| --- | --- | --- |
| {1..10000000} | 78.89 MB (2s 698ms 642μs) | 78.89 MB (18s 601ms 974μs) |
| {1..1000000} | 6.89 MB (458ms 576μs) | 6.89 MB (1s 491ms 621μs) |
| {1..100000} | 588.89 kB (20ms 728μs) | 588.89 kB (156ms 919μs) |
| {1..10000} | 48.89 kB (2ms 202μs) | 48.89 kB (13ms 641μs) |
| {1..1000} | 3.89 kB (1ms 796μs) | 3.89 kB (1ms 958μs) |
| {1..100} | 291 B (424μs) | 291 B (211μs) |
| {1..10} | 20 B (487μs) | 20 B (72μs) |
| {1..3} | 5 B (166μs) | 5 B (27μs) |

If you'd like to run these comparisons yourself, see [test/support/generate.js](http://test/support/generate.js).

## Benchmarks

### Running benchmarks

Install dev dependencies:

npm i -d && npm benchmark

### Latest results

Braces is more accurate, without sacrificing performance.

● expand - range (expanded)

braces x 53,167 ops/sec ±0.12% (102 runs sampled)

minimatch x 11,378 ops/sec ±0.10% (102 runs sampled)

● expand - range (optimized for regex)

braces x 373,442 ops/sec ±0.04% (100 runs sampled)

minimatch x 3,262 ops/sec ±0.18% (100 runs sampled)

● expand - nested ranges (expanded)

braces x 33,921 ops/sec ±0.09% (99 runs sampled)

minimatch x 10,855 ops/sec ±0.28% (100 runs sampled)

● expand - nested ranges (optimized for regex)

braces x 287,479 ops/sec ±0.52% (98 runs sampled)

minimatch x 3,219 ops/sec ±0.28% (101 runs sampled)

● expand - set (expanded)

braces x 238,243 ops/sec ±0.19% (97 runs sampled)

minimatch x 538,268 ops/sec ±0.31% (96 runs sampled)

● expand - set (optimized for regex)

braces x 321,844 ops/sec ±0.10% (97 runs sampled)

minimatch x 140,600 ops/sec ±0.15% (100 runs sampled)

● expand - nested sets (expanded)

braces x 165,371 ops/sec ±0.42% (96 runs sampled)

minimatch x 337,720 ops/sec ±0.28% (100 runs sampled)

● expand - nested sets (optimized for regex)

braces x 242,948 ops/sec ±0.12% (99 runs sampled)

minimatch x 87,403 ops/sec ±0.79% (96 runs sampled)

## About

**Contributing** Pull requests and stars are always welcome. For bugs and feature requests, [please create an issue](http://../../issues/new).

**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*](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*](http://.verb.md) *readme template.)*

To generate the readme, run the following command:

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

### Contributors

| **Commits** | **Contributor** |
| --- | --- |
| 197 | [jonschlinkert](https://github.com/jonschlinkert) |
| 4 | [doowb](https://github.com/doowb) |
| 1 | [es128](https://github.com/es128) |
| 1 | [eush77](https://github.com/eush77) |
| 1 | [hemanth](https://github.com/hemanth) |
| 1 | [wtgtybhertgeghgtwtg](https://github.com/wtgtybhertgeghgtwtg) |

### Author

**Jon Schlinkert**

* [GitHub Profile](https://github.com/jonschlinkert)
* [Twitter Profile](https://twitter.com/jonschlinkert)
* [LinkedIn Profile](https://linkedin.com/in/jonschlinkert)

### License

Copyright © 2019, [Jon Schlinkert](https://github.com/jonschlinkert). Released under the [MIT License](http://license).

*This file was generated by* [*verb-generate-readme*](https://github.com/verbose/verb-generate-readme)*, v0.8.0, on April 08, 2019.*