# yargs-parser

ci [NPM version](https://www.npmjs.com/package/yargs-parser) [Conventional Commits](https://conventionalcommits.org) nycrc config on GitHub

The mighty option parser used by [yargs](https://github.com/yargs/yargs).

visit the [yargs website](http://yargs.js.org/) for more examples, and thorough usage instructions.

## Example

npm i yargs-parser --save

const argv = require('yargs-parser')(process.argv.slice(2))

console.log(argv)

$ node example.js --foo=33 --bar hello

{ \_: [], foo: 33, bar: 'hello' }

*or parse a string!*

const argv = require('yargs-parser')('--foo=99 --bar=33')

console.log(argv)

{ \_: [], foo: 99, bar: 33 }

Convert an array of mixed types before passing to yargs-parser:

const parse = require('yargs-parser')

parse(['-f', 11, '--zoom', 55].join(' ')) // <-- array to string

parse(['-f', 11, '--zoom', 55].map(String)) // <-- array of strings

## Deno Example

As of v19 yargs-parser supports [Deno](https://github.com/denoland/deno):

import parser from "https://deno.land/x/yargs\_parser/deno.ts";

const argv = parser('--foo=99 --bar=9987930', {

string: ['bar']

})

console.log(argv)

## ESM Example

As of v19 yargs-parser supports ESM (*both in Node.js and in the browser*):

**Node.js:**

import parser from 'yargs-parser'

const argv = parser('--foo=99 --bar=9987930', {

string: ['bar']

})

console.log(argv)

**Browsers:**

<!doctype html>

<body>

<script type="module">

import parser from "https://unpkg.com/yargs-parser@19.0.0/browser.js";

const argv = parser('--foo=99 --bar=9987930', {

string: ['bar']

})

console.log(argv)

</script>

</body>

## API

### parser(args, opts={})

Parses command line arguments returning a simple mapping of keys and values.

**expects:**

* args: a string or array of strings representing the options to parse.
* opts: provide a set of hints indicating how args should be parsed:
  + opts.alias: an object representing the set of aliases for a key: {alias: {foo: ['f']}}.
  + opts.array: indicate that keys should be parsed as an array: {array: ['foo', 'bar']}.

Indicate that keys should be parsed as an array and coerced to booleans / numbers:

`{array: [{ key: 'foo', boolean: true }, {key: 'bar', number: true}]}`.

* + opts.boolean: arguments should be parsed as booleans: {boolean: ['x', 'y']}.
  + opts.coerce: provide a custom synchronous function that returns a coerced value from the argument provided (or throws an error). For arrays the function is called only once for the entire array:

`{coerce: {foo: function (arg) {return modifiedArg}}}`.

* + opts.config: indicate a key that represents a path to a configuration file (this file will be loaded and parsed).
  + opts.configObjects: configuration objects to parse, their properties will be set as arguments:

`{configObjects: [{'x': 5, 'y': 33}, {'z': 44}]}`.

* + opts.configuration: provide configuration options to the yargs-parser (see: [configuration](#configuration)).
  + opts.count: indicate a key that should be used as a counter, e.g., -vvv = {v: 3}.
  + opts.default: provide default values for keys: {default: {x: 33, y: 'hello world!'}}.
  + opts.envPrefix: environment variables (process.env) with the prefix provided should be parsed.
  + opts.narg: specify that a key requires n arguments: {narg: {x: 2}}.
  + opts.normalize: path.normalize() will be applied to values set to this key.
  + opts.number: keys should be treated as numbers.
  + opts.string: keys should be treated as strings (even if they resemble a number -x 33).

**returns:**

* obj: an object representing the parsed value of args
  + key/value: key value pairs for each argument and their aliases.
  + \_: an array representing the positional arguments.
  + [optional] --: an array with arguments after the end-of-options flag --.

### require('yargs-parser').detailed(args, opts={})

Parses a command line string, returning detailed information required by the yargs engine.

**expects:**

* args: a string or array of strings representing options to parse.
* opts: provide a set of hints indicating how args, inputs are identical to require('yargs-parser')(args, opts={}).

**returns:**

* argv: an object representing the parsed value of args
  + key/value: key value pairs for each argument and their aliases.
  + \_: an array representing the positional arguments.
  + [optional] --: an array with arguments after the end-of-options flag --.
* error: populated with an error object if an exception occurred during parsing.
* aliases: the inferred list of aliases built by combining lists in opts.alias.
* newAliases: any new aliases added via camel-case expansion:
  + boolean: { fooBar: true }
* defaulted: any new argument created by opts.default, no aliases included.
  + boolean: { foo: true }
* configuration: given by default settings and opts.configuration.

### Configuration

The yargs-parser applies several automated transformations on the keys provided in args. These features can be turned on and off using the configuration field of opts.

var parsed = parser(['--no-dice'], {

configuration: {

'boolean-negation': false

}

})

### short option groups

* default: true.
* key: short-option-groups.

Should a group of short-options be treated as boolean flags?

$ node example.js -abc

{ \_: [], a: true, b: true, c: true }

*if disabled:*

$ node example.js -abc

{ \_: [], abc: true }

### camel-case expansion

* default: true.
* key: camel-case-expansion.

Should hyphenated arguments be expanded into camel-case aliases?

$ node example.js --foo-bar

{ \_: [], 'foo-bar': true, fooBar: true }

*if disabled:*

$ node example.js --foo-bar

{ \_: [], 'foo-bar': true }

### dot-notation

* default: true
* key: dot-notation

Should keys that contain . be treated as objects?

$ node example.js --foo.bar

{ \_: [], foo: { bar: true } }

*if disabled:*

$ node example.js --foo.bar

{ \_: [], "foo.bar": true }

### parse numbers

* default: true
* key: parse-numbers

Should keys that look like numbers be treated as such?

$ node example.js --foo=99.3

{ \_: [], foo: 99.3 }

*if disabled:*

$ node example.js --foo=99.3

{ \_: [], foo: "99.3" }

### parse positional numbers

* default: true
* key: parse-positional-numbers

Should positional keys that look like numbers be treated as such.

$ node example.js 99.3

{ \_: [99.3] }

*if disabled:*

$ node example.js 99.3

{ \_: ['99.3'] }

### boolean negation

* default: true
* key: boolean-negation

Should variables prefixed with --no be treated as negations?

$ node example.js --no-foo

{ \_: [], foo: false }

*if disabled:*

$ node example.js --no-foo

{ \_: [], "no-foo": true }

### combine arrays

* default: false
* key: combine-arrays

Should arrays be combined when provided by both command line arguments and a configuration file.

### duplicate arguments array

* default: true
* key: duplicate-arguments-array

Should arguments be coerced into an array when duplicated:

$ node example.js -x 1 -x 2

{ \_: [], x: [1, 2] }

*if disabled:*

$ node example.js -x 1 -x 2

{ \_: [], x: 2 }

### flatten duplicate arrays

* default: true
* key: flatten-duplicate-arrays

Should array arguments be coerced into a single array when duplicated:

$ node example.js -x 1 2 -x 3 4

{ \_: [], x: [1, 2, 3, 4] }

*if disabled:*

$ node example.js -x 1 2 -x 3 4

{ \_: [], x: [[1, 2], [3, 4]] }

### greedy arrays

* default: true
* key: greedy-arrays

Should arrays consume more than one positional argument following their flag.

$ node example --arr 1 2

{ \_: [], arr: [1, 2] }

*if disabled:*

$ node example --arr 1 2

{ \_: [2], arr: [1] }

**Note: in v18.0.0 we are considering defaulting greedy arrays to false.**

### nargs eats options

* default: false
* key: nargs-eats-options

Should nargs consume dash options as well as positional arguments.

### negation prefix

* default: no-
* key: negation-prefix

The prefix to use for negated boolean variables.

$ node example.js --no-foo

{ \_: [], foo: false }

*if set to quux:*

$ node example.js --quuxfoo

{ \_: [], foo: false }

### populate --

* default: false.
* key: populate--

Should unparsed flags be stored in -- or \_.

*If disabled:*

$ node example.js a -b -- x y

{ \_: [ 'a', 'x', 'y' ], b: true }

*If enabled:*

$ node example.js a -b -- x y

{ \_: [ 'a' ], '--': [ 'x', 'y' ], b: true }

### set placeholder key

* default: false.
* key: set-placeholder-key.

Should a placeholder be added for keys not set via the corresponding CLI argument?

*If disabled:*

$ node example.js -a 1 -c 2

{ \_: [], a: 1, c: 2 }

*If enabled:*

$ node example.js -a 1 -c 2

{ \_: [], a: 1, b: undefined, c: 2 }

### halt at non-option

* default: false.
* key: halt-at-non-option.

Should parsing stop at the first positional argument? This is similar to how e.g. ssh parses its command line.

*If disabled:*

$ node example.js -a run b -x y

{ \_: [ 'b' ], a: 'run', x: 'y' }

*If enabled:*

$ node example.js -a run b -x y

{ \_: [ 'b', '-x', 'y' ], a: 'run' }

### strip aliased

* default: false
* key: strip-aliased

Should aliases be removed before returning results?

*If disabled:*

$ node example.js --test-field 1

{ \_: [], 'test-field': 1, testField: 1, 'test-alias': 1, testAlias: 1 }

*If enabled:*

$ node example.js --test-field 1

{ \_: [], 'test-field': 1, testField: 1 }

### strip dashed

* default: false
* key: strip-dashed

Should dashed keys be removed before returning results? This option has no effect if camel-case-expansion is disabled.

*If disabled:*

$ node example.js --test-field 1

{ \_: [], 'test-field': 1, testField: 1 }

*If enabled:*

$ node example.js --test-field 1

{ \_: [], testField: 1 }

### unknown options as args

* default: false
* key: unknown-options-as-args

Should unknown options be treated like regular arguments? An unknown option is one that is not configured in opts.

*If disabled*

$ node example.js --unknown-option --known-option 2 --string-option --unknown-option2

{ \_: [], unknownOption: true, knownOption: 2, stringOption: '', unknownOption2: true }

*If enabled*

$ node example.js --unknown-option --known-option 2 --string-option --unknown-option2

{ \_: ['--unknown-option'], knownOption: 2, stringOption: '--unknown-option2' }

## Supported Node.js Versions

Libraries in this ecosystem make a best effort to track [Node.js' release schedule](https://nodejs.org/en/about/releases/). Here's [a post on why we think this is important](https://medium.com/the-node-js-collection/maintainers-should-consider-following-node-js-release-schedule-ab08ed4de71a).

## Special Thanks

The yargs project evolves from optimist and minimist. It owes its existence to a lot of James Halliday's hard work. Thanks [substack](https://github.com/substack) **beep** **boop** \o/

## License

ISC