# Arg

arg is an unopinionated, no-frills CLI argument parser.

## Installation

npm install arg

## Usage

arg() takes either 1 or 2 arguments:

1. Command line specification object (see below)
2. Parse options (*Optional*, defaults to {permissive: false, argv: process.argv.slice(2), stopAtPositional: false})

It returns an object with any values present on the command-line (missing options are thus missing from the resulting object). Arg performs no validation/requirement checking - we leave that up to the application.

All parameters that aren't consumed by options (commonly referred to as "extra" parameters) are added to result.\_, which is *always* an array (even if no extra parameters are passed, in which case an empty array is returned).

const arg = require('arg');

// `options` is an optional parameter

const args = arg(

spec,

(options = { permissive: false, argv: process.argv.slice(2) })

);

For example:

$ node ./hello.js --verbose -vvv --port=1234 -n 'My name' foo bar --tag qux --tag=qix -- --foobar

// hello.js

const arg = require('arg');

const args = arg({

// Types

'--help': Boolean,

'--version': Boolean,

'--verbose': arg.COUNT, // Counts the number of times --verbose is passed

'--port': Number, // --port <number> or --port=<number>

'--name': String, // --name <string> or --name=<string>

'--tag': [String], // --tag <string> or --tag=<string>

// Aliases

'-v': '--verbose',

'-n': '--name', // -n <string>; result is stored in --name

'--label': '--name' // --label <string> or --label=<string>;

// result is stored in --name

});

console.log(args);

/\*

{

\_: ["foo", "bar", "--foobar"],

'--port': 1234,

'--verbose': 4,

'--name': "My name",

'--tag': ["qux", "qix"]

}

\*/

The values for each key=>value pair is either a type (function or [function]) or a string (indicating an alias).

* In the case of a function, the string value of the argument's value is passed to it, and the return value is used as the ultimate value.
* In the case of an array, the only element *must* be a type function. Array types indicate that the argument may be passed multiple times, and as such the resulting value in the returned object is an array with all of the values that were passed using the specified flag.
* In the case of a string, an alias is established. If a flag is passed that matches the *key*, then the *value* is substituted in its place.

Type functions are passed three arguments:

1. The parameter value (always a string)
2. The parameter name (e.g. --label)
3. The previous value for the destination (useful for reduce-like operations or for supporting -v multiple times, etc.)

This means the built-in String, Number, and Boolean type constructors "just work" as type functions.

Note that Boolean and [Boolean] have special treatment - an option argument is *not* consumed or passed, but instead true is returned. These options are called "flags".

For custom handlers that wish to behave as flags, you may pass the function through arg.flag():

const arg = require('arg');

const argv = [

'--foo',

'bar',

'-ff',

'baz',

'--foo',

'--foo',

'qux',

'-fff',

'qix'

];

function myHandler(value, argName, previousValue) {

/\* `value` is always `true` \*/

return 'na ' + (previousValue || 'batman!');

}

const args = arg(

{

'--foo': arg.flag(myHandler),

'-f': '--foo'

},

{

argv

}

);

console.log(args);

/\*

{

\_: ['bar', 'baz', 'qux', 'qix'],

'--foo': 'na na na na na na na na batman!'

}

\*/

As well, arg supplies a helper argument handler called arg.COUNT, which equivalent to a [Boolean] argument's .length property - effectively counting the number of times the boolean flag, denoted by the key, is passed on the command line.. For example, this is how you could implement ssh's multiple levels of verbosity (-vvvv being the most verbose).

const arg = require('arg');

const argv = ['-AAAA', '-BBBB'];

const args = arg(

{

'-A': arg.COUNT,

'-B': [Boolean]

},

{

argv

}

);

console.log(args);

/\*

{

\_: [],

'-A': 4,

'-B': [true, true, true, true]

}

\*/

### Options

If a second parameter is specified and is an object, it specifies parsing options to modify the behavior of arg().

#### argv

If you have already sliced or generated a number of raw arguments to be parsed (as opposed to letting arg slice them from process.argv) you may specify them in the argv option.

For example:

const args = arg(

{

'--foo': String

},

{

argv: ['hello', '--foo', 'world']

}

);

results in:

const args = {

\_: ['hello'],

'--foo': 'world'

};

#### permissive

When permissive set to true, arg will push any unknown arguments onto the "extra" argument array (result.\_) instead of throwing an error about an unknown flag.

For example:

const arg = require('arg');

const argv = [

'--foo',

'hello',

'--qux',

'qix',

'--bar',

'12345',

'hello again'

];

const args = arg(

{

'--foo': String,

'--bar': Number

},

{

argv,

permissive: true

}

);

results in:

const args = {

\_: ['--qux', 'qix', 'hello again'],

'--foo': 'hello',

'--bar': 12345

};

#### stopAtPositional

When stopAtPositional is set to true, arg will halt parsing at the first positional argument.

For example:

const arg = require('arg');

const argv = ['--foo', 'hello', '--bar'];

const args = arg(

{

'--foo': Boolean,

'--bar': Boolean

},

{

argv,

stopAtPositional: true

}

);

results in:

const args = {

\_: ['hello', '--bar'],

'--foo': true

};

### Errors

Some errors that arg throws provide a .code property in order to aid in recovering from user error, or to differentiate between user error and developer error (bug).

##### ARG\_UNKNOWN\_OPTION

If an unknown option (not defined in the spec object) is passed, an error with code ARG\_UNKNOWN\_OPTION will be thrown:

// cli.js

try {

require('arg')({ '--hi': String });

} catch (err) {

if (err.code === 'ARG\_UNKNOWN\_OPTION') {

console.log(err.message);

} else {

throw err;

}

}

node cli.js --extraneous true

Unknown or unexpected option: --extraneous

# FAQ

A few questions and answers that have been asked before:

### How do I require an argument with arg?

Do the assertion yourself, such as:

const args = arg({ '--name': String });

if (!args['--name']) throw new Error('missing required argument: --name');

# License

Released under the [MIT License](http://license.md).