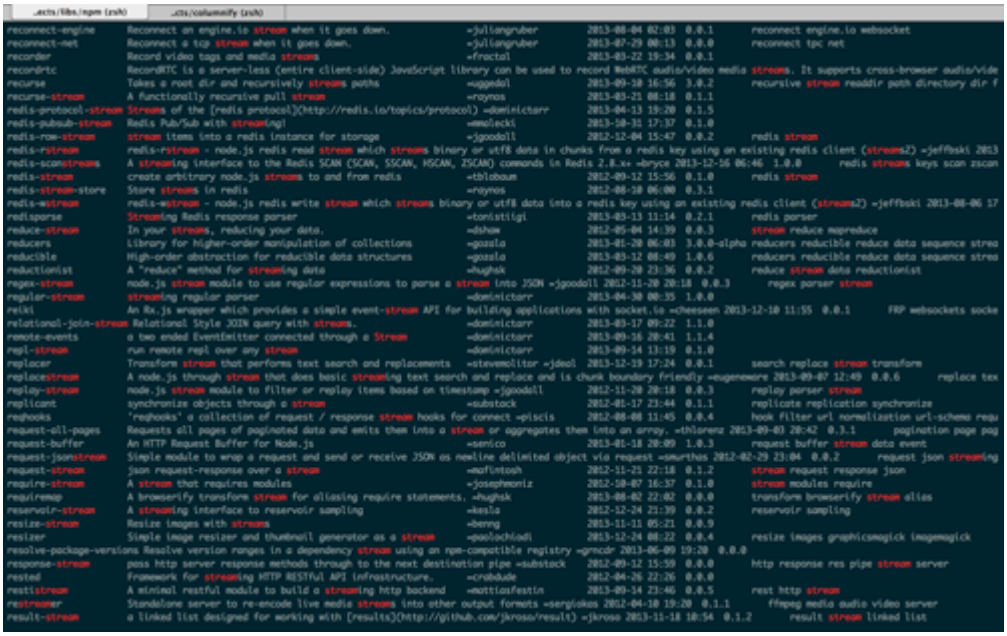
columnify

Create text-based columns suitable for console output from objects or arrays of objects.

Columns are automatically resized to fit the content of the largest cell. Each cell will be padded with spaces to fill the available space and ensure column contents are left-aligned.

Designed to handle sensible wrapping in npm search results.

`npm search` before & after integrating columnify:



Installation

```
$ npm install columnify
```

Usage

```
var columnify = require('columnify')
var columns = columnify(data, options)
console.log(columns)
```

Examples

Columnify Objects

Objects are converted to a list of key/value pairs:

```
var data = {
  "commander@0.6.1": 1,
  "minimatch@0.2.14": 3,
  "mkdirp@0.3.5": 2,
  "sigmund@1.0.0": 3
}

console.log(columnify(data))
```

Output:

```
KEY               VALUE
commander@0.6.1   1
minimatch@0.2.14  3
mkdirp@0.3.5      2
sigmund@1.0.0     3
```

Custom Column Names

```
var data = {
  "commander@0.6.1": 1,
  "minimatch@0.2.14": 3,
  "mkdirp@0.3.5": 2,
  "sigmund@1.0.0": 3
}

console.log(columnify(data, {columns: ['MODULE', 'COUNT']}))
```

Output:

```
MODULE            COUNT
commander@0.6.1   1
minimatch@0.2.14  3
mkdirp@0.3.5      2
sigmund@1.0.0     3
```

Columnify Arrays of Objects

Column headings are extracted from the keys in supplied objects.

```
var columnify = require('columnify')

var columns = columnify([{
  name: 'mod1',
  version: '0.0.1'
}, {
  name: 'module2',
  version: '0.2.0'
}])
```

```
console.log(columns)
```

Output:

```
NAME    VERSION
mod1    0.0.1
module2 0.2.0
```

Filtering & Ordering Columns

By default, all properties are converted into columns, whether or not they exist on every object or not.

To explicitly specify which columns to include, and in which order, supply a "columns" or "include" array ("include" is just an alias).

```
var data = [{
  name: 'module1',
  description: 'some description',
  version: '0.0.1',
}, {
  name: 'module2',
  description: 'another description',
  version: '0.2.0',
}]

var columns = columnify(data, {
  columns: ['name', 'version']
})

console.log(columns)
```

Output:

```
NAME    VERSION
module1 0.0.1
module2 0.2.0
```

Global and Per Column Options

You can set a number of options at a global level (ie. for all columns) or on a per column basis.

Set options on a per column basis by using the `config` option to specify individual columns:

```
var columns = columnify(data, {
  optionName: optionValue,
  config: {
    columnName: {optionName: optionValue},
    columnName: {optionName: optionValue},
```

```
  }
})
```

Maximum and Minimum Column Widths

As with all options, you can define the `maxWidth` and `minWidth` globally, or for specified columns. By default, wrapping will happen at word boundaries. Empty cells or those which do not fill the `minWidth` will be padded with spaces.

```
var columns = columnify([{
  name: 'mod1',
  description: 'some description which happens to be far larger than the max',
  version: '0.0.1',
}, {
  name: 'module-two',
  description: 'another description larger than the max',
  version: '0.2.0',
}], {
  minWidth: 20,
  config: {
    description: {maxWidth: 30}
  }
})

console.log(columns)
```

Output:

```
NAME                 DESCRIPTION                    VERSION
mod1                 some description which happens 0.0.1
                     to be far larger than the max
module-two           another description larger     0.2.0
                     than the max
```

Maximum Line Width

You can set a hard maximum line width using the `maxLineWidth` option. Beyond this value data is unceremoniously truncated with no truncation marker.

This can either be a number or 'auto' to set the value to the width of stdout.

Setting this value to 'auto' prevent TTY-imposed line-wrapping when lines exceed the screen width.

Truncating Column Cells Instead of Wrapping

You can disable wrapping and instead truncate content at the maximum column width by using the `truncate` option. Truncation respects word boundaries. A truncation marker, `…`, will appear next to the last word in any truncated line.

```
var columns = columnify(data, {
  truncate: true,
  config: {
```

```
    description: {
      maxWidth: 20
    }
  }
})

console.log(columns)
```

Output:

```
NAME       DESCRIPTION          VERSION
mod1       some description…    0.0.1
module-two another description… 0.2.0
```

Align Right/Center

You can set the alignment of the column data by using the `align` option.

```
var data = {
  "mocha@1.18.2": 1,
  "commander@2.0.0": 1,
  "debug@0.8.1": 1
}

columnify(data, {config: {value: {align: 'right'}}})
```

Output:

```
KEY                 VALUE
mocha@1.18.2            1
commander@2.0.0         1
debug@0.8.1             1
```

`align: 'center'` works in a similar way.

Padding Character

Set a character to fill whitespace within columns with the `paddingChr` option.

```
var data = {
  "shortKey": "veryVeryVeryVeryVeryLongVal",
  "veryVeryVeryVeryVeryLongKey": "shortVal"
}

columnify(data, { paddingChr: '.'})
```

Output:

```
KEY..................... VALUE.....................
shortKey................ veryVeryVeryVeryVeryLongVal
```

```
veryVeryVeryVeryVeryLongKey shortVal...................
```

Preserve Existing Newlines

By default, `columnify` sanitises text by replacing any occurance of 1 or more whitespace characters with a single space.

`columnify` can be configured to respect existing new line characters using the `preserveNewLines` option. Note this will still collapse all other whitespace.

```
var data = [{
  name: "glob@3.2.9",
  paths: [
    "node_modules/tap/node_modules/glob",
    "node_modules/tape/node_modules/glob"
  ].join('\n')
}, {
  name: "nopt@2.2.1",
  paths: [
    "node_modules/tap/node_modules/nopt"
  ]
}, {
  name: "runforcover@0.0.2",
  paths: "node_modules/tap/node_modules/runforcover"
}]

console.log(columnify(data, {preserveNewLines: true}))
```

Output:

```
NAME              PATHS
glob@3.2.9        node_modules/tap/node_modules/glob
                  node_modules/tape/node_modules/glob
nopt@2.2.1        node_modules/tap/node_modules/nopt
runforcover@0.0.2 node_modules/tap/node_modules/runforcover
```

Compare this with output without `preserveNewLines`:

```
console.log(columnify(data, {preserveNewLines: false}))
// or just
console.log(columnify(data))
```

```
NAME              PATHS
glob@3.2.9        node_modules/tap/node_modules/glob
node_modules/tape/node_modules/glob
nopt@2.2.1        node_modules/tap/node_modules/nopt
runforcover@0.0.2 node_modules/tap/node_modules/runforcover
```

Custom Truncation Marker

You can change the truncation marker to something other than the default `…` by using the `truncateMarker` option.

```
var columns = columnify(data, {
  truncate: true,
  truncateMarker: '>',
  widths: {
    description: {
      maxWidth: 20
    }
  }
})

console.log(columns)
```

Output:

```
NAME       DESCRIPTION          VERSION
mod1       some description>    0.0.1
module-two another description> 0.2.0
```

Custom Column Splitter

If your columns need some bling, you can split columns with custom characters by using the `columnSplitter` option.

```
var columns = columnify(data, {
  columnSplitter: ' | '
})

console.log(columns)
```

Output:

```
NAME       | DESCRIPTION                                                | VERSION
mod1       | some description which happens to be far larger than the max | 0.0.1
module-two | another description larger than the max                    | 0.2.0
```

Control Header Display

Control whether column headers are displayed by using the `showHeaders` option.

```
var columns = columnify(data, {
  showHeaders: false
})
```

This also works well for hiding a single column header, like an `id` column:

```
var columns = columnify(data, {
  config: {
    id: { showHeaders: false }
  }
})
```

Transforming Column Data and Headers

If you need to modify the presentation of column content or heading content there are two useful options for doing that: `dataTransform` and `headingTransform`. Both of these take a function and need to return a valid string.

```
var columns = columnify([{
    name: 'mod1',
    description: 'SOME DESCRIPTION TEXT.'
}, {
    name: 'module-two',
    description: 'SOME SLIGHTLY LONGER DESCRIPTION TEXT.'
}], {
    dataTransform: function(data) {
        return data.toLowerCase()
    },
    headingTransform: function(heading) {
        return heading.toLowerCase()
    },
    config: {
        name: {
            headingTransform: function(heading) {
              heading = "module " + heading
              return "*" +  heading.toUpperCase() + "*"
            }
        }
    }
})
```

Output:

```
*MODULE NAME* description
mod1          some description text.
module-two    some slightly longer description text.
```

Multibyte Character Support

`columnify` uses [mycoboco/wcwidth.js] to calculate length of multibyte characters:

```
var data = [{
  name: 'module-one',
  description: 'some description',
  version: '0.0.1',
}, {
```

```
  name: '这是一个很长的名字的模块',
  description: '这真的是一个描述的内容这个描述很长',
  version: "0.3.3"
}]

console.log(columnify(data))
```

Without multibyte handling:

i.e. before columnify added this feature

```
NAME         DESCRIPTION       VERSION
module-one   some description  0.0.1
这是一个很长的名字的模块  这真的是一个描述的内容这个描述很长  0.3.3
```

With multibyte handling:

```
NAME                    DESCRIPTION                       VERSION
module-one              some description                  0.0.1
这是一个很长的名字的模块  这真的是一个描述的内容这个描述很长  0.3.3
```

Contributions

```
project  : columnify
repo age : 8 years
active   : 47 days
commits  : 180
files    : 57
authors  :
  123  Tim Oxley           68.3%
   11  Nicholas Hoffman    6.1%
    8  Tim                 4.4%
    7  Arjun Mehta         3.9%
    6  Dany                3.3%
    5  Tim Kevin Oxley     2.8%
    5  Wei Gao             2.8%
    4  Matias Singers      2.2%
    3  Michael Kriese      1.7%
    2  sreekanth370        1.1%
    2  Dany Shaanan        1.1%
    1  Tim Malone          0.6%
    1  Seth Miller         0.6%
    1  andyfusniak         0.6%
    1  Isaac Z. Schlueter  0.6%
```

License

MIT