

---

## unfluff

An automatic web page content extractor for Node.js!

build unknown

Automatically grab the main text out of a webpage like this:

```
1 extractor = require('unfluff');
2 data = extractor(my_html_data);
3 console.log(data.text);
```

In other words, it turns pretty webpages into boring plain text/json data:



This might be useful for: - Writing your own Instapaper clone - Easily building ML data sets from web pages - Reading your favorite articles from the console?

Please don't use this for: - Stealing other peoples' web pages - Making crappy spam sites with stolen content from other sites - Being a jerk

## Credits / Thanks

This library is largely based on python-goose by Xavier Grangier which is in turn based on goose by Gravity Labs. However, it's not an exact port so it may behave differently on some pages and the

---

feature set is a little bit different. If you are looking for a python or Scala/Java/JVM solution, check out those libraries!

## Install

To install the command-line `unfluff` utility:

```
1 npm install -g unfluff
```

To install the `unfluff` module for use in your Node.js project:

```
1 npm install --save unfluff
```

## Usage

You can use `unfluff` from node or right on the command line!

### Extracted data elements

This is what `unfluff` will try to grab from a web page: - `title` - The document's title (from the `<title>` tag) - `softTitle` - A version of `title` with less truncation - `date` - The document's publication date - `copyright` - The document's copyright line, if present - `author` - The document's author - `publisher` - The document's publisher (website name) - `text` - The main text of the document with all the junk thrown away - `image` - The main image for the document (what's used by facebook, etc.) - `videos` - An array of videos that were embedded in the article. Each video has `src`, `width` and `height`. - `tags` - Any tags or keywords that could be found by checking `<rel>` tags or by looking at `href` urls. - `canonicalLink` - The canonical url of the document, if given. - `lang` - The language of the document, either detected or supplied by you. - `description` - The description of the document, from `<meta>` tags - `favicon` - The url of the document's favicon. - `links` - An array of links embedded within the article text. (text and `href` for each)

This is returned as a simple json object.

### Command line interface

You can pass a webpage to `unfluff` and it will try to parse out the interesting bits.

You can either pass in a file name:

```
1 unfluff my_file.html
```

---

Or you can pipe it in:

```
1 curl -s "http://somesite.com/page" | unfluff
```

You can easily chain this together with other unix commands to do cool stuff. For example, you can download a web page, parse it and then use jq to print it just the body text.

```
1 curl -s "https://www.polygon.com/2014/6/26/5842180/shovel-knight-review-pc-3ds-wii-u" | unfluff | jq -r .text
```

And here's how to find the top 10 most common words in an article:

```
1 curl -s "https://www.polygon.com/2014/6/26/5842180/shovel-knight-review-pc-3ds-wii-u" | unfluff | tr -c '[:alnum:]' '[\n*]' | sort | uniq -c | sort -nr | head -10
```

## Module Interface

**extractor(html, language)** html: The html you want to parse

language (optional): The document's two-letter language code. This will be auto-detected as best as possible, but there might be cases where you want to override it.

The extraction algorithm depends heavily on the language, so it probably won't work if you have the language set incorrectly.

```
1 extractor = require('unfluff');  
2  
3 data = extractor(my_html_data);
```

Or supply the language code yourself:

```
1 extractor = require('unfluff');  
2  
3 data = extractor(my_html_data, 'en');
```

data will then be a json object that looks like this:

```
1 {  
2   "title": "Shovel Knight review",  
3   "softTitle": "Shovel Knight review: rewrite history",  
4   "date": "2014-06-26T13:00:03Z",  
5   "copyright": "2016 Vox Media Inc Designed in house",  
6   "author": [  
7     "Griffin McElroy"  
8   ],  
9   "publisher": "Polygon",
```

---

```
10  "text": "Shovel Knight is inspired by the past in all the right ways
11      - but it's far from stuck in it. [... snip ...]",
12  "image": "http://cdn2.vox-cdn.com/uploads/chorus_image/image
13      /34834129/jellyfish_hero.0_cinema_1280.0.png",
14  "tags": [],
15  "videos": [],
16  "canonicalLink": "http://www.polygon.com/2014/6/26/5842180/shovel-
17      knight-review-pc-3ds-wii-u",
18  "lang": "en",
19  "description": "Shovel Knight is inspired by the past in all the
20      right ways - but it's far from stuck in it.",
21  "favicon": "http://cdn1.vox-cdn.com/community_logos/42931/favicon.ico",
22  "links": [
23      { "text": "Six Thirty", "href": "http://www.sixthirty.co/" }
24  ]
25  }
```

**extractor.lazy(html, language)** Lazy version of `extractor(html, language)`.

The text extraction algorithm can be somewhat slow on large documents. If you only need access to elements like `title` or `image`, you can use the lazy extractor to get them more quickly without running the full processing pipeline.

This returns an object just like the regular extractor except all fields are replaced by functions and evaluation is only done when you call those functions.

```
1  extractor = require('unfluff');
2
3  data = extractor.lazy(my_html_data, 'en');
4
5  // Access whichever data elements you need directly.
6  console.log(data.title());
7  console.log(data.softTitle());
8  console.log(data.date());
9  console.log(data.copyright());
10 console.log(data.author());
11 console.log(data.publisher());
12 console.log(data.text());
13 console.log(data.image());
14 console.log(data.tags());
15 console.log(data.videos());
16 console.log(data.canonicalLink());
17 console.log(data.lang());
18 console.log(data.description());
19 console.log(data.favicon());
```

Some of these data elements require calculating intermediate representations of the html document.

---

Everything is cached so looking up multiple data elements and looking them up multiple times should be as fast as possible.

## Demo

The easiest way to try out `unfluff` is to just install it:

```
1 $ npm install -g unfluff
2 $ curl -s "http://www.cnn.com/2014/07/07/world/americas/mexico-
    earthquake/index.html" | unfluff
```

But if you can't be bothered, you can check out `fetch text`. It's a site by Andy Jiang that uses `unfluff`. You send an email with a url and it emails back with the cleaned content of that url. It should give you a good idea of how `unfluff` handles different urls.

## What is broken

- Parsing web pages in languages other than English is poorly tested and probably is buggy right now.
- This definitely won't work yet for languages like Chinese / Arabic / Korean / etc that need smarter word tokenization.
- This has only been tested on a limited set of web pages. There are probably lots of lurking bugs with web pages that haven't been tested yet.