mincss Documentation

Release 0.1

Peter Bengtsson

Sep 27, 2017

Contents

`mincss` is a Python library that makes it possible to evaluate which CSS is actually being used. It does this by download the whole page(s) and finds all inline and linked CSS and analyses which selectors are still in use somewhere.

Optionally, you can use PhantomJS to download the HTML source from a URL which means it will at least load all the Javascript that gets executed onload.

Installation should be as simple as `pip install mincss`. The code is available on Github.

CHAPTER 1

Getting started

Suppose you have a page like this:

```
<!doctype html>
<html>
  <head>
    <style type="text/css">
    .foo, input:hover { color: black; }
    .bar { color: blue; }
    </style>
  </head>
  <body>
    <div id="content">
      <p class="foo">Foo!</p>
    </div>
  </body>
</html>
```

And, let's assume that this is available as `http://localhost/page.html`.

Now, let's use `mincss` as follows:

```
>>> from mincss.processor import Processor
>>> p = Processor()
>>> p.process('http://localhost/page.html')
>>> inline = p.inlines[0]
>>> inline.before
'\n    .foo, input:hover { color: black; }\n    .bar { color: blue; }\n    '
>>> inline.after
'\n    .foo { color: black; }\n    '
```

As you can see, it automatically discovered that the `input:hover` and the `.bar` selectors are not used in the HTML DOM tree.

If you have `phantomjs` installed and can do things like `$ phantomjs --help` on your command line you can run mincss like this:

```
>>> from mincss.processor import Processor
>>> p = Processor(phantomjs=True)
>>> p.process('http://localhost/page-with-javascript.html')
```

CHAPTER 2

Supported Features and Limitations

Things that work:

- Any selector that `lxml`'s `CSSSelector` can match such as `#foo > .bar input[type="submit"]`
- media queries are supported (this is treated as nested CSS basically)
- keyframes are left untouched
- You can manually over selectors that should be untouched for things you definitely know will be needed by Javascript code but isn't part of the initial HTML tree.
- You can analyze multiple URLs and find the common CSS amongst them. (This doesn't work for inline CSS)
- Comments are left untouched and minute whitespace details are preseved so the generated output looks similar to its input, but with the selectors not needed removed.
- A proxy server apps is available that can help you preview the result of just one URL.

Things that don't yet work:

- Javascript events that manipulate the DOM tree. You can use PhantomJS to do the downloading but it still won't get every possible piece of HTML generated based on complex Javascript.
- keyframes are always left untouched even if it's never referenced
- Broken HTML or broken/invalid CSS isn't supported and good results can not be guaranteed.

Things that don't work:

- link tags wrapped in IE-only style comments (e.g `<!--[if lte IE 7]>`) is not supported.

CHAPTER 3

API

This is work in progress and is likely to change in future version

- `process.Processor([debug=False, preserve_remote_urls=True])` Creates a processor instance that you can feed HTML and URLs.

  The arguments:

  - `debug=False` Currently does nothing particular.
  - `preserve_remote_urls=True` If you run a URL like `http://www.example.org` that references `http://cdn.cloudware.com/foo.css` which contains `url(/background.png)` then the CSS will be rewritten to become `url(http://cdn.cloudware.com/background.png)`
  - `phantomjs=None` If `True` will default to `phantomjs`, If a string it's assume it's the path to the executable `phantomjs` path.
  - `phantomjs_options={}` Additional options/switches to the `phantomjs` command. This has to be a dict. So, for example `{'script-encoding': 'latin1'}` becomes `--script-encoding=latin1`.
  - `optimize_lookup=True` If true, will make a set of all ids and class names in all processed documents and use these to avoid some expensive CSS query searches.

  Instances of this allows you to use the following methods:

  - `process(*urls)` Downloads the HTML from that URL(s) and expects it to be 200 return code. The content will be transformed to a unicode string in UTF-8.

    Once all URLs have been processed the CSS is analyzed.
  - `process_url(url)` Given a specific URL it will download it and parse the HTML. This method will download the HTML then called `process_html()`.
  - `process_html(html, url)` If you for some reason already have the HTML you can jump straight to this method. Note, you still need to provide the URL where you got the HTML from so it can use that to download any external CSS.

  The `Processor` instance will make two attributes available

  - `instance.inlines` A list of `InlineResult` instances (see below)
  - `instance.links` A list of `LinkResult` instances (see below)

- `InlineResult`

  This is where the results are stored for inline CSS. It holds the following attributes:

  - `line` Which line in the original HTML this starts on
  - `url` The URL this was found on
  - `before` The inline CSS before it was analyzed
  - `after` The new CSS with the selectors presumably not used removed

- `LinkResult`

  This is where the results are stored for all referenced links to CSS files. i.e. from things like `<link rel="stylesheet" href="foo.css">` It contains the following attributes:

  - `href` The `href` attribute on the link tag. e.g. `/static/main.css`
  - `before` The CSS before it was analyzed
  - `after` The new CSS with the selectors presumably not used removed

CHAPTER 4

Changelog

v0.8.1 (2013-04-05)

The file `download.js` was missing from the tarball.

v0.8.0 (2013-02-26)

Much faster! Unless you pass `Processor(optimize_lookup=False)` when creating the processor instance. See http://www.peterbe.com/plog/mincss-0.8

v0.7.0 (2013-02-13)

Fixed bug with make absolute url of url like *http://peterbe.com* + *./style.css*. Thanks @erfaan!

v0.6.1 (2013-02-12)

The proxy app would turn <*script src="foo"*></*script*> into <*script src="http://remote/foo"*/>. Same for iframe, textarea and divs.

v0.6.0 (2013-02-01)

New option, *phantomjs* that allows you to download the HTML using phantomjs instead of regular Python's urllib.

v0.5.0 (2013-01-24)

New option *preserve_remote_urls* to *Processor()* class. Useful when the hrefs in link tags are of different domain than the URL you're processing.

v0.1 (2013-01-14)

Initial release.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index