Bleach Documentation

Release 2.1.2 20171207

Will Kahn-Greene

Jan 08, 2018

Contents

Bleach is an allowed-list-based HTML sanitizing library that escapes or strips markup and attributes.

Bleach can also linkify text safely, applying filters that Django's `urlize` filter cannot, and optionally setting `rel` attributes, even on links already in the text.

Bleach is intended for sanitizing text from *untrusted* sources. If you find yourself jumping through hoops to allow your site administrators to do lots of things, you're probably outside the use cases. Either trust those users, or don't.

Because it relies on html5lib, Bleach is as good as modern browsers at dealing with weird, quirky HTML fragments. And *any* of Bleach's methods will fix unbalanced or mis-nested tags.

The version on GitHub is the most up-to-date and contains the latest bug fixes. You can find full documentation on ReadTheDocs.

Code https://github.com/mozilla/bleach

Documentation https://bleach.readthedocs.io/

Issue tracker https://github.com/mozilla/bleach/issues

IRC `#bleach` on irc.mozilla.org

License Apache License v2; see LICENSE file

CHAPTER 1

Reporting Bugs

For regular bugs, please report them in our issue tracker.

If you believe that you've found a security vulnerability, please file a secure bug report in our bug tracker or send an email to *security AT mozilla DOT org*.

For more information on security-related bug disclosure and the PGP key to use for sending encrypted mail or to verify responses received from that address, please read our wiki page at https://www.mozilla.org/en-US/security/#For_Developers.

CHAPTER 2

Installing Bleach

Bleach is available on PyPI, so you can install it with `pip`:

```
$ pip install bleach
```

Or with `easy_install`:

```
$ easy_install bleach
```

CHAPTER 3

Upgrading Bleach

Warning: Before doing any upgrades, read through Bleach Changes for backwards incompatible changes, newer versions, etc.

CHAPTER 4

Basic use

The simplest way to use Bleach is:

```
>>> import bleach

>>> bleach.clean('an <script>evil()</script> example')
u'an &lt;script&gt;evil()&lt;/script&gt; example'

>>> bleach.linkify('an http://example.com url')
u'an <a href="http://example.com" rel="nofollow">http://example.com</a> url
```

CHAPTER 5

Security

Bleach is a security-related library.

We have a responsible security vulnerability reporting process. Please use that if you're reporting a security issue.

Security issues are fixed in private. After we land such a fix, we'll do a release.

For every release, we mark security issues we've fixed in the `CHANGES` in the **Security issues** section. We include relevant CVE links.

CHAPTER 6

Code of conduct

This project and repository is governed by Mozilla's code of conduct and etiquette guidelines. For more details please see the Mozilla Community Participation Guidelines and Developer Etiquette Guidelines.

CHAPTER 7

Contents

Sanitizing text fragments

bleach.clean() is Bleach's HTML sanitization method.

Given a fragment of HTML, Bleach will parse it according to the HTML5 parsing algorithm and sanitize any disallowed tags or attributes. This algorithm also takes care of things like unclosed and (some) misnested tags.

You may pass in a `string` or a `unicode` object, but Bleach will always return `unicode`.

Note: *bleach.clean()* is for sanitizing HTML **fragments** and not entire HTML documents.

Warning: *bleach.clean()* is for sanitising HTML fragments to use in an HTML context–not for HTML attributes, CSS, JSON, xhtml, SVG, or other contexts.

For example, this is a safe use of `clean` output in an HTML context:

```
<p>
  {{ bleach.clean(user_bio) }}
</p>
```

This is a **not safe** use of `clean` output in an HTML attribute:

```
<body data-bio="{{ bleach.clean(user_bio} }}">
```

If you need to use the output of `bleach.clean()` in an HTML attribute, you need to pass it through your template library's escape function. For example, Jinja2's `escape` or `django.utils.html.escape` or something like that.

If you need to use the output of `bleach.clean()` in any other context, you need to pass it through an appropriate sanitizer/escaper for that context.

bleach.**clean**(*text, tags=[u'a', u'abbr', u'acronym', u'b', u'blockquote', u'code', u'em', u'i', u'li', u'ol', u'strong', u'ul'], attributes={u'a': [u'href', u'title'], u'acronym': [u'title'], u'abbr': [u'title']}, styles=[], protocols=[u'http', u'https', u'mailto'], strip=False, strip_comments=True*)

Clean an HTML fragment of malicious content and return it

This function is a security-focused function whose sole purpose is to remove malicious content from a string such that it can be displayed as content in a web page.

This function is not designed to use to transform content to be used in non-web-page contexts.

Example:

```
import bleach

better_text = bleach.clean(yucky_text)
```

Note: If you're cleaning a lot of text and passing the same argument values or you want more configurability, consider using a *bleach.sanitizer.Cleaner* instance.

Parameters

- **text** (*str*) – the text to clean
- **tags** (*list*) – allowed list of tags; defaults to bleach.sanitizer.ALLOWED_TAGS
- **attributes** (*dict*) – allowed attributes; can be a callable, list or dict; defaults to bleach.sanitizer.ALLOWED_ATTRIBUTES
- **styles** (*list*) – allowed list of css styles; defaults to bleach.sanitizer.ALLOWED_STYLES
- **protocols** (*list*) – allowed list of protocols for links; defaults to bleach.sanitizer.ALLOWED_PROTOCOLS
- **strip** (*bool*) – whether or not to strip disallowed elements
- **strip_comments** (*bool*) – whether or not to strip HTML comments

Returns cleaned text as unicode

Allowed tags (tags)

The tags kwarg specifies the allowed set of HTML tags. It should be a list, tuple, or other iterable. Any HTML tags not in this list will be escaped or stripped from the text.

For example:

```
>>> import bleach

>>> bleach.clean(
...     u'<b><i>an example</i></b>',
...     tags=['b'],
... )
u'<b>&lt;i&gt;an example&lt;/i&gt;</b>'
```

The default value is a relatively conservative list found in bleach.sanitizer.ALLOWED_TAGS.

bleach.sanitizer.**ALLOWED_TAGS = [u'a', u'abbr', u'acronym', u'b', u'blockquote', u'code', u'em', u'i', u'li', u'ol', u'str**

List of allowed tags

Allowed Attributes (`attributes`)

The `attributes` kwarg lets you specify which attributes are allowed. The value can be a list, a callable or a map of tag name to list or callable.

The default value is also a conservative dict found in `bleach.sanitizer.ALLOWED_ATTRIBUTES`.

bleach.sanitizer.**ALLOWED_ATTRIBUTES = {u'a': [u'href', u'title'], u'acronym': [u'title'], u'abbr': [u'title']}**

Map of allowed attributes by tag

Changed in version 2.0: Prior to 2.0, the `attributes` kwarg value could only be a list or a map.

As a list

The `attributes` value can be a list which specifies the list of attributes allowed for any tag.

For example:

```
>>> import bleach

>>> bleach.clean(
...     u'<p class="foo" style="color: red; font-weight: bold;">blah blah blah</p>',
...     tags=['p'],
...     attributes=['style'],
...     styles=['color'],
... )
u'<p style="color: red;">blah blah blah</p>'
```

As a dict

The `attributes` value can be a dict which maps tags to what attributes they can have.

You can also specify `*`, which will match any tag.

For example, this allows "href" and "rel" for "a" tags, "alt" for the "img" tag and "class" for any tag (including "a" and "img"):

```
>>> import bleach

>>> attrs = {
...     '*': ['class'],
...     'a': ['href', 'rel'],
...     'img': ['alt'],
... }

>>> bleach.clean(
...    u'<img alt="an example" width=500>',
...    tags=['img'],
...    attributes=attrs
... )
u'<img alt="an example">'
```

Using functions

You can also use callables that take the tag, attribute name and attribute value and returns `True` to keep the attribute or `False` to drop it.

You can pass a callable as the attributes argument value and it'll run for every tag/attr.

For example:

```
>>> import bleach

>>> def allow_h(tag, name, value):
...     return name[0] == 'h'

>>> bleach.clean(
...    u'<a href="http://example.com" title="link">link</a>',
...    tags=['a'],
...    attributes=allow_h,
... )
u'<a href="http://example.com">link</a>'
```

You can also pass a callable as a value in an attributes dict and it'll run for attributes for specified tags:

```
>>> from urlparse import urlparse
>>> import bleach

>>> def allow_src(tag, name, value):
...     if name in ('alt', 'height', 'width'):
...         return True
...     if name == 'src':
...         p = urlparse(value)
...         return (not p.netloc) or p.netloc == 'mydomain.com'
...     return False

>>> bleach.clean(
...    u'<img src="http://example.com" alt="an example">',
...    tags=['img'],
...    attributes={
...        'img': allow_src
...    }
... )
u'<img alt="an example">'
```

Changed in version 2.0: In previous versions of Bleach, the callable took an attribute name and a attribute value. Now it takes a tag, an attribute name and an attribute value.

Allowed styles (`styles`)

If you allow the `style` attribute, you will also need to specify the allowed styles users are allowed to set, for example `color` and `background-color`.

The default value is an empty list. In other words, the `style` attribute will be allowed but no style declaration names will be allowed.

For example, to allow users to set the color and font-weight of text:

```
>>> import bleach
>>> tags = ['p', 'em', 'strong']
>>> attrs = {
...     '*': ['style']
... }
>>> styles = ['color', 'font-weight']

>>> bleach.clean(
...     u'<p style="font-weight: heavy;">my html</p>',
...     tags=tags,
...     attributes=attrs,
...     styles=styles
... )
u'<p style="font-weight: heavy;">my html</p>'
```

Default styles are stored in `bleach.sanitizer.ALLOWED_STYLES`.

`bleach.sanitizer.`**`ALLOWED_STYLES`** **= []**

List of allowed styles

Allowed protocols (`protocols`)

If you allow tags that have attributes containing a URI value (like the `href` attribute of an anchor tag, you may want to adapt the accepted protocols.

For example, this sets allowed protocols to http, https and smb:

```
>>> import bleach

>>> bleach.clean(
...     '<a href="smb://more_text">allowed protocol</a>',
...     protocols=['http', 'https', 'smb']
... )
u'<a href="smb://more_text">allowed protocol</a>'
```

This adds smb to the Bleach-specified set of allowed protocols:

```
>>> import bleach

>>> bleach.clean(
...     '<a href="smb://more_text">allowed protocol</a>',
...     protocols=bleach.ALLOWED_PROTOCOLS + ['smb']
... )
u'<a href="smb://more_text">allowed protocol</a>'
```

Default protocols are in `bleach.sanitizer.ALLOWED_PROTOCOLS`.

`bleach.sanitizer.`**`ALLOWED_PROTOCOLS`** **= [u'http', u'https', u'mailto']**

List of allowed protocols

Stripping markup (`strip`)

By default, Bleach *escapes* tags that aren't specified in the allowed tags list and invalid markup. For example:

```
>>> import bleach

>>> bleach.clean('<span>is not allowed</span>')
```

```
u'&lt;span&gt;is not allowed&lt;/span&gt;'

>>> bleach.clean('<b><span>is not allowed</span></b>', tags=['b'])
u'<b>&lt;span&gt;is not allowed&lt;/span&gt;</b>'
```

If you would rather Bleach stripped this markup entirely, you can pass `strip=True`:

```
>>> import bleach

>>> bleach.clean('<span>is not allowed</span>', strip=True)
u'is not allowed'

>>> bleach.clean('<b><span>is not allowed</span></b>', tags=['b'], strip=True)
u'<b>is not allowed</b>'
```

Stripping comments (`strip_comments`)

By default, Bleach will strip out HTML comments. To disable this behavior, set `strip_comments=False`:

```
>>> import bleach

>>> html = 'my<!-- commented --> html'

>>> bleach.clean(html)
u'my html'

>>> bleach.clean(html, strip_comments=False)
u'my<!-- commented --> html'
```

Using `bleach.sanitizer.Cleaner`

If you're cleaning a lot of text or you need better control of things, you should create a *`bleach.sanitizer.Cleaner`* instance.

class `bleach.sanitizer.`**`Cleaner`**(*tags=[u'a', u'abbr', u'acronym', u'b', u'blockquote', u'code', u'em', u'i', u'li', u'ol', u'strong', u'ul'], attributes={u'a': [u'href', u'title'], u'acronym': [u'title'], u'abbr': [u'title']}, styles=[], protocols=[u'http', u'https', u'mailto'], strip=False, strip_comments=True, filters=None*)

Cleaner for cleaning HTML fragments of malicious content

This cleaner is a security-focused function whose sole purpose is to remove malicious content from a string such that it can be displayed as content in a web page.

This cleaner is not designed to use to transform content to be used in non-web-page contexts.

To use:

```
from bleach.sanitizer import Cleaner

cleaner = Cleaner()

for text in all_the_yucky_things:
    sanitized = cleaner.clean(text)
```

Initializes a Cleaner

Parameters

- **tags** (*list*) – allowed list of tags; defaults to `bleach.sanitizer.ALLOWED_TAGS`
- **attributes** (*dict*) – allowed attributes; can be a callable, list or dict; defaults to `bleach.sanitizer.ALLOWED_ATTRIBUTES`
- **styles** (*list*) – allowed list of css styles; defaults to `bleach.sanitizer.ALLOWED_STYLES`
- **protocols** (*list*) – allowed list of protocols for links; defaults to `bleach.sanitizer.ALLOWED_PROTOCOLS`
- **strip** (*bool*) – whether or not to strip disallowed elements
- **strip_comments** (*bool*) – whether or not to strip HTML comments
- **filters** (*list*) – list of html5lib Filter classes to pass streamed content through

  See also:

  http://html5lib.readthedocs.io/en/latest/movingparts.html#filters

  Warning: Using filters changes the output of `bleach.Cleaner.clean`. Make sure the way the filters change the output are secure.

clean (*text*)
Cleans text and returns sanitized result as unicode

Parameters **text** (*str*) – text to be cleaned

Returns sanitized text as unicode

Raises **TypeError** – if `text` is not a text type

New in version 2.0.

html5lib Filters (`filters`)

Bleach sanitizing is implemented as an html5lib filter. The consequence of this is that we can pass the streamed content through additional specified filters after the `bleach.sanitizer.BleachSanitizingFilter` filter has run.

This lets you add data, drop data and change data as it is being serialized back to a unicode.

Documentation on html5lib Filters is here: http://html5lib.readthedocs.io/en/latest/movingparts.html#filters

Trivial Filter example:

```
>>> from bleach.sanitizer import Cleaner
>>> from html5lib.filters.base import Filter

>>> class MooFilter(Filter):
...     def __iter__(self):
...         for token in Filter.__iter__(self):
...             if token['type'] in ['StartTag', 'EmptyTag'] and token['data']:
...                 for attr, value in token['data'].items():
...                     token['data'][attr] = 'moo'
...             yield token
...
>>> ATTRS = {
...     'img': ['rel', 'src']
... }
...
>>> TAGS = ['img']
>>> cleaner = Cleaner(tags=TAGS, attributes=ATTRS, filters=[MooFilter])
>>> dirty = 'this is cute! <img src="http://example.com/puppy.jpg" rel="nofollow">'
>>> cleaner.clean(dirty)
u'this is cute! <img rel="moo" src="moo">'
```

Warning: Filters change the output of cleaning. Make sure that whatever changes the filter is applying maintain the safety guarantees of the output.

New in version 2.0.

Using `bleach.sanitizer.BleachSanitizerFilter`

`bleach.clean` creates a `bleach.sanitizer.Cleaner` which creates a `bleach.sanitizer.BleachSanitizerFilter` which does the sanitizing work.

`BleachSanitizerFilter` is an html5lib filter and can be used anywhere you can use an html5lib filter.

class `bleach.sanitizer.`**`BleachSanitizerFilter`**(*source, attributes={u'a': [u'href', u'title'], u'acronym': [u'title'], u'abbr': [u'title']}, strip_disallowed_elements=False, strip_html_comments=True, **kwargs*)

html5lib Filter that sanitizes text

This filter can be used anywhere html5lib filters can be used.

Creates a BleachSanitizerFilter instance

Parameters

- **`source`** (*`Treewalker`*) – stream
- **`tags`** (*`list`*) – allowed list of tags; defaults to `bleach.sanitizer.ALLOWED_TAGS`
- **`attributes`** (*`dict`*) – allowed attributes; can be a callable, list or dict; defaults to `bleach.sanitizer.ALLOWED_ATTRIBUTES`
- **`styles`** (*`list`*) – allowed list of css styles; defaults to `bleach.sanitizer.ALLOWED_STYLES`
- **`protocols`** (*`list`*) – allowed list of protocols for links; defaults to `bleach.sanitizer.ALLOWED_PROTOCOLS`
- **`strip_disallowed_elements`** (*`bool`*) – whether or not to strip disallowed elements
- **`strip_html_comments`** (*`bool`*) – whether or not to strip HTML comments

New in version 2.0.

Linkifying text fragments

`bleach.linkify()` searches text for links, URLs, and email addresses and lets you control how and when those links are rendered.

It works by building a document tree, so it's guaranteed never to do weird things to URLs in attribute values, can modify the value of attributes on `<a>` tags and can even do things like skip `<pre>` sections.

Note: You may pass a `string` or `unicode` object, but Bleach will always return `unicode`.

`bleach.`**`linkify`**(*text, callbacks=[<function nofollow>], skip_tags=None, parse_email=False*)

Convert URL-like strings in an HTML fragment to links

This function converts strings that look like URLs, domain names and email addresses in text that may be an HTML fragment to links, while preserving:

1. links already in the string
2. urls found in attributes
3. email addresses

linkify does a best-effort approach and tries to recover from bad situations due to crazy text.

Note: If you're linking a lot of text and passing the same argument values or you want more configurability, consider using a *`bleach.linkifier.Linker`* instance.

Note: If you have text that you want to clean and then linkify, consider using the *`bleach.linkifier.LinkifyFilter`* as a filter in the clean pass. That way you're not parsing the HTML twice.

Parameters

- **`text`** (*`str`*) – the text to linkify
- **`callbacks`** (*`list`*) – list of callbacks to run when adjusting tag attributes; defaults to `bleach.linkifier.DEFAULT_CALLBACKS`
- **`skip_tags`** (*`list`*) – list of tags that you don't want to linkify the contents of; for example, you could set this to `['pre']` to skip linkifying contents of `pre` tags
- **`parse_email`** (*`bool`*) – whether or not to linkify email addresses

Returns linkified text as unicode

Callbacks for adjusting attributes (`callbacks`)

The second argument to `linkify()` is a list or other iterable of callback functions. These callbacks can modify links that exist and links that are being created, or remove them completely.

Each callback will get the following arguments:

```
def my_callback(attrs, new=False):
```

The `attrs` argument is a dict of attributes of the `<a>` tag. Keys of the `attrs` dict are namespaced attr names. For example `(None, 'href')`. The `attrs` dict also contains a `_text` key, which is the innerText of the `<a>` tag.

The `new` argument is a boolean indicating if the link is new (e.g. an email address or URL found in the text) or already existed (e.g. an `<a>` tag found in the text).

The callback must return a dict of attributes (including `_text`) or `None`. The new dict of attributes will be passed to the next callback in the list.

If any callback returns `None`, new links will not be created and existing links will be removed leaving the innerText left in its place.

The default callback adds `rel="nofollow"`. See `bleach.callbacks` for some included callback functions.

This defaults to `bleach.linkify.DEFAULT_CALLBACKS`.

`bleach.linkifier.`**`DEFAULT_CALLBACKS` = [<function nofollow>]**
List of default callbacks

Changed in version 2.0: In previous versions of Bleach, the attribute names were not namespaced.

Setting Attributes

For example, you could add a `title` attribute to all links:

```
>>> from bleach.linkifier import Linker

>>> def set_title(attrs, new=False):
...     attrs[(None, u'title')] = u'link in user text'
...     return attrs
...
>>> linker = Linker(callbacks=[set_title])
>>> linker.linkify('abc http://example.com def')
u'abc <a href="http://example.com" title="link in user text">http://example.com</a>␣
↪def'
```

This would set the value of the `rel` attribute, stomping on a previous value if there was one.

Here's another example that makes external links open in a new tab and look like an external link:

```
>>> from urlparse import urlparse
>>> from bleach.linkifier import Linker

>>> def set_target(attrs, new=False):
...     p = urlparse(attrs[(None, u'href')])
...     if p.netloc not in ['my-domain.com', 'other-domain.com']:
...         attrs[(None, u'target')] = u'_blank'
...         attrs[(None, u'class')] = u'external'
...     else:
...         attrs.pop((None, u'target'), None)
...     return attrs
...
>>> linker = Linker(callbacks=[set_target])
>>> linker.linkify('abc http://example.com def')
u'abc <a class="external" href="http://example.com" target="_blank">http://example.com
↪</a> def'
```

Removing Attributes

You can easily remove attributes you don't want to allow, even on existing links (`<a>` tags) in the text. (See also *clean()* for sanitizing attributes.)

```
>>> from bleach.linkifier import Linker

>>> def allowed_attrs(attrs, new=False):
...     """Only allow href, target, rel and title."""
...     allowed = [
...         (None, u'href'),
...         (None, u'target'),
...         (None, u'rel'),
...         (None, u'title'),
...         u'_text',
...     ]
...     return dict((k, v) for k, v in attrs.items() if k in allowed)
...
>>> linker = Linker(callbacks=[allowed_attrs])
>>> linker.linkify('<a style="font-weight: super bold;" href="http://example.com">link
↪</a>')
u'<a href="http://example.com">link</a>'
```

Or you could remove a specific attribute, if it exists:

```
>>> from bleach.linkifier import Linker

>>> def remove_title(attrs, new=False):
...     attrs.pop((None, u'title'), None)
...     return attrs
...
>>> linker = Linker(callbacks=[remove_title])
>>> linker.linkify('<a href="http://example.com">link</a>')
u'<a href="http://example.com">link</a>'

>>> linker.linkify('<a title="bad title" href="http://example.com">link</a>')
u'<a href="http://example.com">link</a>'
```

Altering Attributes

You can alter and overwrite attributes, including the link text, via the `_text` key, to, for example, pass outgoing links through a warning page, or limit the length of text inside an `<a>` tag.

Example of shortening link text:

```
>>> from bleach.linkifier import Linker

>>> def shorten_url(attrs, new=False):
...     """Shorten overly-long URLs in the text."""
...     # Only adjust newly-created links
...     if not new:
...         return attrs
...     # _text will be the same as the URL for new links
...     text = attrs[u'_text']
...     if len(text) > 25:
...         attrs[u'_text'] = text[0:22] + u'...'
...     return attrs
...
>>> linker = Linker(callbacks=[shorten_url])
>>> linker.linkify('http://example.com/longlonglonglonglongurl')
u'<a href="http://example.com/longlonglonglonglongurl">http://example.com/lon...</a>'
```

Example of switching all links to go through a bouncer first:

```
>>> from six.moves.urllib.parse import quote, urlparse
>>> from bleach.linkifier import Linker
>>> def outgoing_bouncer(attrs, new=False):
...     """Send outgoing links through a bouncer."""
...     href_key = (None, u'href')
...     p = urlparse(attrs.get(href_key, None))
...     if p.netloc not in ['example.com', 'www.example.com', '']:
...         bouncer = 'http://bn.ce/?destination=%s'
...         attrs[href_key] = bouncer % quote(attrs[href_key])
...     return attrs
...
>>> linker = Linker(callbacks=[outgoing_bouncer])
>>> linker.linkify('http://example.com')
u'<a href="http://example.com">http://example.com</a>'

>>> linker.linkify('http://foo.com')
u'<a href="http://bn.ce/?destination=http%3A//foo.com">http://foo.com</a>'
```

Preventing Links

A slightly more complex example is inspired by Crate, where strings like `models.py` are often found, and linkified. `.py` is the ccTLD for Paraguay, so `example.py` may be a legitimate URL, but in the case of a site dedicated to Python packages, odds are it is not. In this case, Crate could write the following callback:

```
>>> from bleach.linkifier import Linker

>>> def dont_linkify_python(attrs, new=False):
...     # This is an existing link, so leave it be
...     if not new:
...         return attrs
...     # If the TLD is '.py', make sure it starts with http: or https:.
...     # Use _text because that's the original text
...     link_text = attrs[u'_text']
...     if link_text.endswith('.py') and not link_text.startswith(('http:', 'https:
↪')):
...         # This looks like a Python file, not a URL. Don't make a link.
...         return None
...     # Everything checks out, keep going to the next callback.
...     return attrs
...
>>> linker = Linker(callbacks=[dont_linkify_python])
>>> linker.linkify('abc http://example.com def')
u'abc <a href="http://example.com">http://example.com</a> def'

>>> linker.linkify('abc models.py def')
u'abc models.py def'
```

Removing Links

If you want to remove certain links, even if they are written in the text with `<a>` tags, have the callback return `None`.

For example, this removes any `mailto:` links:

```
>>> from bleach.linkifier import Linker

>>> def remove_mailto(attrs, new=False):
```

```
...     if attrs[(None, u'href')].startswith(u'mailto:'):
...         return None
...     return attrs
...
>>> linker = Linker(callbacks=[remove_mailto])
>>> linker.linkify('<a href="mailto:janet@example.com">mail janet!</a>')
u'mail janet!'
```

Skipping links in specified tag blocks (`skip_tags`)

`<pre>` tags are often special, literal sections. If you don't want to create any new links within a `<pre>` section, pass `skip_tags=['pre']`.

This works for `code`, `div` and any other blocks you want to skip over.

Changed in version 2.0: This used to be `skip_pre`, but this makes it more general.

Linkifying email addresses (`parse_email`)

By default, *`bleach.linkify()`* does not create `mailto:` links for email addresses, but if you pass `parse_email=True`, it will. `mailto:` links will go through exactly the same set of callbacks as all other links, whether they are newly created or already in the text, so be careful when writing callbacks that may need to behave differently if the protocol is `mailto:`.

Using `bleach.linkifier.Linker`

If you're linking a lot of text and passing the same argument values or you want more configurability, consider using a *`bleach.linkifier.Linker`* instance.

```
>>> from bleach.linkifier import Linker

>>> linker = Linker(skip_tags=['pre'])
>>> linker.linkify('a b c http://example.com d e f')
u'a b c <a href="http://example.com" rel="nofollow">http://example.com</a> d e f'
```

class `bleach.linkifier.`**`Linker`**(*callbacks=[<function nofollow>], skip_tags=None, parse_email=False, url_re=<_sre.SRE_Pattern object at 0x2d8e180>, email_re=<_sre.SRE_Pattern object at 0x2d5c910>*)

Convert URL-like strings in an HTML fragment to links

This function converts strings that look like URLs, domain names and email addresses in text that may be an HTML fragment to links, while preserving:

1. links already in the string
2. urls found in attributes
3. email addresses

linkify does a best-effort approach and tries to recover from bad situations due to crazy text.

Creates a Linker instance

Parameters

- **`callbacks`** (*`list`*) – list of callbacks to run when adjusting tag attributes; defaults to `bleach.linkifier.DEFAULT_CALLBACKS`

- **skip_tags** (*list*) – list of tags that you don't want to linkify the contents of; for example, you could set this to `['pre']` to skip linkifying contents of `pre` tags
- **parse_email** (*bool*) – whether or not to linkify email addresses
- **url_re** (*re*) – url matching regex
- **email_re** (*re*) – email matching regex

Returns linkified text as unicode

linkify (*text*)
Linkify specified text

Parameters **text** (*str*) – the text to add links to

Returns linkified text as unicode

Raises **TypeError** – if `text` is not a text type

New in version 2.0.

Using `bleach.linkifier.LinkifyFilter`

`bleach.linkify` works by paring an HTML fragment and then running it through the `bleach.linkifier.LinkifyFilter` when walking the tree and serializing it back into text.

You can use this filter wherever you can use an html5lib Filter. For example, you could use it with `bleach.Cleaner` to clean and linkify in one step.

For example, using all the defaults:

```
>>> from functools import partial

>>> from bleach import Cleaner
>>> from bleach.linkifier import LinkifyFilter

>>> cleaner = Cleaner(tags=['pre'])
>>> cleaner.clean('<pre>http://example.com</pre>')
u'<pre>http://example.com</pre>'

>>> cleaner = Cleaner(tags=['pre'], filters=[LinkifyFilter])
>>> cleaner.clean('<pre>http://example.com</pre>')
u'<pre><a href="http://example.com">http://example.com</a></pre>'
```

And passing parameters to `LinkifyFilter`:

```
>>> from functools import partial

>>> from bleach.sanitizer import Cleaner
>>> from bleach.linkifier import LinkifyFilter

>>> cleaner = Cleaner(
...     tags=['pre'],
...     filters=[partial(LinkifyFilter, skip_tags=['pre'])]
... )
...
>>> cleaner.clean('<pre>http://example.com</pre>')
u'<pre>http://example.com</pre>'
```

class `bleach.linkifier.`**`LinkifyFilter`**(*source*, *callbacks=None*, *skip_tags=None*, *parse_email=False*, *url_re=<_sre.SRE_Pattern object at 0x2d8e180>*, *email_re=<_sre.SRE_Pattern object at 0x2d5c910>*)

html5lib filter that linkifies text

This will do the following:

- convert email addresses into links
- convert urls into links
- edit existing links by running them through callbacks–the default is to add a `rel="nofollow"`

This filter can be used anywhere html5lib filters can be used.

Creates a LinkifyFilter instance

Parameters

- **`source`** (*`TreeWalker`*) – stream
- **`callbacks`** (*`list`*) – list of callbacks to run when adjusting tag attributes; defaults to `bleach.linkifier.DEFAULT_CALLBACKS`
- **`skip_tags`** (*`list`*) – list of tags that you don't want to linkify the contents of; for example, you could set this to `['pre']` to skip linkifying contents of `pre` tags
- **`parse_email`** (*`bool`*) – whether or not to linkify email addresses
- **`url_re`** (*`re`*) – url matching regex
- **`email_re`** (*`re`*) – email matching regex

New in version 2.0.

Goals of Bleach

This document lists the goals and non-goals of Bleach. My hope is that by focusing on these goals and explicitly listing the non-goals, the project will evolve in a stronger direction.

Contents

Goals

Always take a allowed-list-based approach

Bleach should always take a allowed-list-based approach to markup filtering. Specifying disallowed lists is error-prone and not future proof.

For example, you should have to opt-in to allowing the `onclick` attribute, not opt-out of all the other `on*` attributes. Future versions of HTML may add new event handlers, like `ontouch`, that old disallow would not prevent.

Main goal is to sanitize input of malicious content

The primary goal of Bleach is to sanitize user input that is allowed to contain *some* HTML as markup and is to be included in the content of a larger page in an HTML context.

Examples of such content might include:

- User comments on a blog.
- "Bio" sections of a user profile.
- Descriptions of a product or application.

These examples, and others, are traditionally prone to security issues like XSS or other script injection, or annoying issues like unclosed tags and invalid markup. Bleach will take a proactive, allowed-list-only approach to allowing HTML content, and will use the HTML5 parsing algorithm to handle invalid markup.

See the *chapter on clean()* for more info.

Safely create links

The secondary goal of Bleach is to provide a mechanism for finding or altering links (`<a>` tags with `href` attributes, or things that look like URLs or email addresses) in text.

While Bleach itself will always operate on a allowed-list-based security model, the *linkify() method* is flexible enough to allow the creation, alteration, and removal of links based on an extremely wide range of use cases.

Non-Goals

Bleach is designed to work with fragments of HTML by untrusted users. Some non-goal use cases include:

Sanitize complete HTML documents

Bleach's `clean` is not for sanitizing entire HTML documents. Once you're creating whole documents, you have to allow so many tags that a disallow-list approach (e.g. forbidding `<script>` or `<object>`) may be more appropriate.

Sanitize for use in HTML attributes, CSS, JSON, xhtml, SVG, or other contexts

Bleach's `clean` is used for sanitizing content to be used in an HTML context–not for HTML attributes, CSS, JSON, xhtml, SVG, or other contexts.

For example, this is a safe use of `clean` output in an HTML context:

```
<p>
  {{ bleach.clean(user_bio) }}
</p>
```

This is a **not safe** use of `clean` output in an HTML attribute:

```
<body data-bio="{{ bleach.clean(user_bio} }}">
```

If you need to use the output of `bleach.clean()` in an HTML attribute, you need to pass it through your template library's escape function. For example, Jinja2's `escape` or `django.utils.html.escape` or something like that.

If you need to use the output of `bleach.clean()` in any other context, you need to pass it through an appropriate sanitizer/escaper for that context.

Remove all HTML or transforming content for some non-web-page purpose

There are much faster tools available if you want to remove or escape all HTML from a document.

Clean up after trusted users

Bleach is powerful but it is not fast. If you trust your users, trust them and don't rely on Bleach to clean up their mess.

Make malicious content look pretty or sane

Malicious content is designed to be malicious. Making it safe is a design goal of Bleach. Making it pretty or sane-looking is not.

If you want your malicious content to look pretty, you should pass it through Bleach to make it safe and then do your own transform afterwards.

Allow arbitrary styling

There are a number of interesting CSS properties that can do dangerous things, like Opera's `-o-link`. Painful as it is, if you want your users to be able to change nearly anything in a `style` attribute, you should have to opt into this.

Usage with Javascript frameworks and template languages

A number of Javascript frameworks and template languages allow XSS via Javascript Gadgets. While Bleach usually produces output safe for these contexts, it is not tested against them nor guaranteed to produce safe output. Check that bleach properly strips or escapes language-specific syntax like `data-bind` attributes for Knockout.js or `ng-*` attributes from Angular templates before using bleach-sanitized output with your framework or template language.

Bleach vs html5lib

Bleach is built upon html5lib, and html5lib has a built-in sanitizer filter, so why use Bleach?

- Bleach's API is simpler.
- Bleach's sanitizer allows a map to be provided for `ALLOWED_ATTRIBUTES`, giving you a lot more control over sanitizing attributes: you can sanitize attributes for specific tags, you can sanitize based on value, etc.
- Bleach's sanitizer always alphabetizes attributes, but uses an alphabetizer that works with namespaces — the html5lib one is broken in that regard.
- Bleach's sanitizer always quotes attribute values because that's the safe thing to do. The html5lib one makes that configurable. In this case, Bleach doesn't make something configurable that isn't safe.
- Bleach's sanitizer has a very restricted set of `ALLOWED_PROTOCOLS` by default. html5lib has a much more expansive one that Bleach's authors claim is less safe.
- `html5lib.filters.sanitizer.Filter`'s `sanitize_css` is broken and doesn't work.

Bleach development

Install for development

To install Bleach to make changes to it:

1. Clone the repo from GitHub:

```
$ git clone git://github.com/mozilla/bleach.git
```

2. Create a virtual environment using whatever method you want.
3. Install Bleach into the virtual environment such that you can see changes:

```
$ pip install -e .
```

Reporting Bugs

For regular bugs, please report them in our issue tracker.

If you believe that you've found a security vulnerability, please file a secure bug report in our bug tracker or send an email to *security AT mozilla DOT org*.

For more information on security-related bug disclosure and the PGP key to use for sending encrypted mail or to verify responses received from that address, please read our wiki page at https://www.mozilla.org/en-US/security/#For_Developers.

Code of conduct

This project and repository is governed by Mozilla's code of conduct and etiquette guidelines. For more details please see the Mozilla Community Participation Guidelines and Developer Etiquette Guidelines.

Docs

Docs are in `docs/`. We use Sphinx. Docs are pushed to ReadTheDocs via a GitHub webhook.

Testing

Run:

```
$ tox
```

That'll run Bleach tests in all the supported Python environments. Note that you need the necessary Python binaries for them all to be tested.

Tests are run in Travis CI via a GitHub webhook.

Release process

1. Checkout master tip.
2. Check to make sure `setup.py` and `requirements.txt` are correct and match requirements-wise.
3. Update version numbers in `bleach/__init__.py`.
   (a) Set `__version__` to something like `2.0`.
   (b) Set `__releasedate__` to something like `20120731`.
4. Update `CONTRIBUTORS`, `CHANGES` and `MANIFEST.in`.
5. Verify correctness.
   (a) Run tests with tox:

   ```
   $ tox
   ```

   (b) Build the docs:

   ```
   $ cd docs
   $ make html
   ```

   (c) Run the doctests:

   ```
   $ cd docs/
   $ make doctests
   ```

   (d) Verify everything works
6. Commit the changes.
7. Push the changes to GitHub. This will cause Travis to run the tests.
8. After Travis is happy, create a signed tag for the release:

   ```
   $ git tag -s v0.4
   ```

   Copy the details from `CHANGES` into the tag comment.
9. Push the new tag:

   ```
   $ git push --tags official master
   ```

   That will push the release to PyPI.
10. Blog posts, twitter, update topic in `#bleach`, etc.

Bleach Changes

Version 2.1.2 (December 7th, 2017)

Security fixes

None

Backwards incompatible changes

None

Features

None

Bug fixes

- Support html5lib-python 1.0.1. (#337)
- Add deprecation warning for supporting html5lib-python < 1.0.
- Switch to semver.

Version 2.1.1 (October 2nd, 2017)

Security fixes

None

Backwards incompatible changes

None

Features

None

Bug fixes

- Fix `setup.py` opening files when `LANG=`. (#324)

Version 2.1 (September 28th, 2017)

Security fixes

- Convert control characters (backspace particularly) to "?" preventing malicious copy-and-paste situations. (#298)

  See https://github.com/mozilla/bleach/issues/298 for more details.

  This affects all previous versions of Bleach. Check the comments on that issue for ways to alleviate the issue if you can't upgrade to Bleach 2.1.

Backwards incompatible changes

- Redid versioning. `bleach.VERSION` is no longer available. Use the string version at `bleach.__version__` and parse it with `pkg_resources.parse_version`. (#307)
- clean, linkify: linkify and clean should only accept text types; thank you, Janusz! (#292)
- clean, linkify: accept only unicode or utf-8-encoded str (#176)

Features

Bug fixes

- `bleach.clean()` no longer unescapes entities including ones that are missing a `;` at the end which can happen in urls and other places. (#143)
- linkify: fix http links inside of mailto links; thank you, sedrubal! (#300)
- clarify security policy in docs (#303)
- fix dependency specification for html5lib 1.0b8, 1.0b9, and 1.0b10; thank you, Zoltán! (#268)
- add Bleach vs. html5lib comparison to README; thank you, Stu Cox! (#278)
- fix KeyError exceptions on tags without href attr; thank you, Alex Defsen! (#273)
- add test website and scripts to test `bleach.clean()` output in browser; thank you, Greg Guthe!

Version 2.0 (March 8th, 2017)

Security fixes

- None

Backwards incompatible changes

- Removed support for Python 2.6. #206
- Removed support for Python 3.2. #224
- Bleach no longer supports html5lib < 0.99999999 (8 9s).

  This version is a rewrite to use the new sanitizing API since the old one was dropped in html5lib 0.99999999 (8 9s).

  If you're using 0.9999999 (7 9s) upgrade to 0.99999999 (8 9s) or higher.

  If you're using 1.0b8 (equivalent to 0.9999999 (7 9s)), upgrade to 1.0b9 (equivalent to 0.99999999 (8 9s)) or higher.
- `bleach.clean` and friends were rewritten

  `clean` was reimplemented as an html5lib filter and happens at a different step in the HTML parsing -> traversing -> serializing process. Because of that, there are some differences in clean's output as compared with previous versions.

  Amongst other things, this version will add end tags even if the tag in question is to be escaped.
- `bleach.clean` and friends attribute callables now take three arguments: tag, attribute name and attribute value. Previously they only took attribute name and attribute value.

  All attribute callables will need to be updated.
- `bleach.linkify` was rewritten

  `linkify` was reimplemented as an html5lib Filter. As such, it no longer accepts a `tokenizer` argument.

  The callback functions for adjusting link attributes now takes a namespaced attribute.

  Previously you'd do something like this:

  ```
  def check_protocol(attrs, is_new):
      if not attrs.get('href', '').startswith('http:', 'https:')):
          return None
      return attrs
  ```

Now it's more like this:

```
def check_protocol(attrs, is_new):
    if not attrs.get((None, u'href'), u'').startswith(('http:', 'https:')):
        #                ^^^^^^^^^^^^^^^
        return None
    return attrs
```

Further, you need to make sure you're always using unicode values. If you don't then html5lib will raise an assertion error that the value is not unicode.

All linkify filters will need to be updated.

- `bleach.linkify` and friends had a `skip_pre` argument–that's been replaced with a more general `skip_tags` argument.

  Before, you might do:

  ```
  bleach.linkify(some_text, skip_pre=True)
  ```

  The equivalent with Bleach 2.0 is:

  ```
  bleach.linkify(some_text, skip_tags=['pre'])
  ```

  You can skip other tags, too, like `style` or `script` or other places where you don't want linkification happening.

  All uses of linkify that use `skip_pre` will need to be updated.

Changes

- Supports Python 3.6.
- Supports html5lib >= 0.99999999 (8 9s).
- There's a `bleach.sanitizer.Cleaner` class that you can instantiate with your favorite clean settings for easy reuse.
- There's a `bleach.linkifier.Linker` class that you can instantiate with your favorite linkify settings for easy reuse.
- There's a `bleach.linkifier.LinkifyFilter` which is an htm5lib filter that you can pass as a filter to `bleach.sanitizer.Cleaner` allowing you to clean and linkify in one pass.
- `bleach.clean` and friends can now take a callable as an attributes arg value.
- Tons of bug fixes.
- Cleaned up tests.
- Documentation fixes.

Version 1.5 (November 4th, 2016)

Security fixes

- None

Backwards incompatible changes

- clean: The list of `ALLOWED_PROTOCOLS` now defaults to http, https and mailto.

Previously it was a long list of protocols something like ed2k, ftp, http, https, irc, mailto, news, gopher, nntp, telnet, webcal, xmpp, callto, feed, urn, aim, rsync, tag, ssh, sftp, rtsp, afs, data. #149

Changes

- clean: Added `protocols` to arguments list to let you override the list of allowed protocols. Thank you, Andreas Malecki! #149
- linkify: Fix a bug involving periods at the end of an email address. Thank you, Lorenz Schori! #219
- linkify: Fix linkification of non-ascii ports. Thank you Alexandre, Macabies! #207
- linkify: Fix linkify inappropriately removing node tails when dropping nodes. #132
- Fixed a test that failed periodically. #161
- Switched from nose to py.test. #204
- Add test matrix for all supported Python and html5lib versions. #230
- Limit to html5lib `>=0.999,!=0.9999,!=0.99999,<0.99999999` because 0.9999 and 0.99999 are busted.
- Add support for `python setup.py test`. #97

Version 1.4.3 (May 23rd, 2016)

Security fixes

- None

Changes

- Limit to html5lib `>=0.999,<0.99999999` because of impending change to sanitizer api. #195

Version 1.4.2 (September 11, 2015)

Changes

- linkify: Fix hang in linkify with `parse_email=True`. #124
- linkify: Fix crash in linkify when removing a link that is a first-child. #136
- Updated TLDs.
- linkify: Don’t remove exterior brackets when linkifying. #146

Version 1.4.1 (December 15, 2014)

Changes

- Consistent order of attributes in output.
- Python 3.4 support.

Version 1.4 (January 12, 2014)

Changes

- linkify: Update linkify to use etree type Treewalker instead of simpletree.
- Updated html5lib to version `>=0.999`.
- Update all code to be compatible with Python 3 and 2 using six.
- Switch to Apache License.

Version 1.3

- Used by Python 3-only fork.

Version 1.2.2 (May 18, 2013)

- Pin html5lib to version 0.95 for now due to major API break.

Version 1.2.1 (February 19, 2013)

- `clean()` no longer considers `feed:` an acceptable protocol due to inconsistencies in browser behavior.

Version 1.2 (January 28, 2013)

- `linkify()` has changed considerably. Many keyword arguments have been replaced with a single callbacks list. Please see the documentation for more information.
- Bleach will no longer consider unacceptable protocols when linkifying.
- `linkify()` now takes a tokenizer argument that allows it to skip sanitization.
- `delinkify()` is gone.
- Removed exception handling from `_render`. `clean()` and `linkify()` may now throw.
- `linkify()` correctly ignores case for protocols and domain names.
- `linkify()` correctly handles markup within an <a> tag.

Version 1.1.5

Version 1.1.4

Version 1.1.3 (July 10, 2012)

- Fix parsing bare URLs when parse_email=True.

Version 1.1.2 (June 1, 2012)

- Fix hang in style attribute sanitizer. (#61)
- Allow `/` in style attribute values.

Version 1.1.1 (February 17, 2012)

- Fix tokenizer for html5lib 0.9.5.

Version 1.1.0 (October 24, 2011)

- `linkify()` now understands port numbers. (#38)
- Documented character encoding behavior. (#41)
- Add an optional target argument to `linkify()`.
- Add `delinkify()` method. (#45)
- Support subdomain whitelist for `delinkify()`. (#47, #48)

Version 1.0.4 (September 2, 2011)

- Switch to SemVer git tags.
- Make `linkify()` smarter about trailing punctuation. (#30)
- Pass `exc_info` to logger during rendering issues.
- Add wildcard key for attributes. (#19)
- Make `linkify()` use the `HTMLSanitizer` tokenizer. (#36)
- Fix URLs wrapped in parentheses. (#23)
- Make `linkify()` UTF-8 safe. (#33)

Version 1.0.3 (June 14, 2011)

- `linkify()` works with 3rd level domains. (#24)
- `clean()` supports vendor prefixes in style values. (#31, #32)
- Fix `linkify()` email escaping.

Version 1.0.2 (June 6, 2011)

- `linkify()` supports email addresses.
- `clean()` supports callables in attributes filter.

Version 1.0.1 (April 12, 2011)

- `linkify()` doesn't drop trailing slashes. (#21)
- `linkify()` won't linkify 'libgl.so.1'. (#22)

CHAPTER 8

Indices and tables

- genindex
- search

Index