Posty Documentation

Release 2.0.3

Nick Pegg

Jan 26, 2018

Contents:

A simple static site generator tool. It reads in a series of posts and pages containing YAML metadata and Markdown text, and renders them as HTML.

CHAPTER 1

Quickstart

Important: If you are a Posty 1.x user, you can skip this and instead take a look at the *Posty 1.x* import docs.

To get started, create yourself an empty directory, cd into it, and then run `posty init` to create a site skeleton.

```
mkdir my_site
cd my_site
posty init
```

Posty will create the skeleton and give you some basic information:

```
Posty initialized!

Directories:
- posts -> Put all of your blog posts here
- pages -> Put all of your static pages here
- templates -> jinja2 HTML templates go here
- media -> Static files go here; JS, CSS, images, etc.

There is also a config file at config.yml that you should adjust to your
liking, like setting the site title and such.
```

Below are some introductions to some of these things. Once you have everything sorted and some pages and posts written, you just run the `posty build` command. This will render your site and place all of the resulting files into the `build/` directory. The contents of that directory can then be uploaded to your location of choice for hosting.

1.1 Config

You will want to adjust the config file `config.yml` to your liking, especially setting the `title`, `author`, and `base_url` for the site. The `base_url` is especially important and should be the URL at which your site will eventually be hosted, for example: `https://yourwebsite.com/blog/`.

See *Config File* for more information about the config file options.

1.2 Content

Content is stored in two directories, `posts` and `pages`. Each file in these directories are YAML metadata and a Markdown body, separated by a `---`. A couple of examples have been provided for you by the `posty init` command, but you should check out *Post and Page Schema* for information about what these types of files should look like.

The CLI has some commands available to easily create new pages and posts. Here are some examples:

```
$ posty new page --name "About me"
$ cat pages/about-me.yaml
parent: None
title: About me
---
This is your new page. Write what you want here!
```

```
$ posty new post --name "A nifty blog post"
$ cat posts/2018-01-20_a-nifty-blog-post.yaml
date: 2018-01-20
tags:
- tag1
- tag2
title: A nifty blog post
---
This the the summary/first paragraph of your new post
---
This is the rest of the post.

Write what you want here
```

1.3 Templates

The `templates/` directory is where you put all of your HTML templates. These will be rendered by Jinja. See *Templating* for more information about what these templates should look like.

1.4 Static files

The `media/` directory is where you should put all of your static files, such as JavaScript, CSS, images, etc. When you run `posty build`, it simply copies this directory over to your `build/` directory.

To help with accessing these files, there is a Jinja filter available in templates called `media_url`. It will translate any path relative to the `media` directory into a full absolute URL. It's used like so:

```
<link rel="stylesheet" href="{{ 'css/index.css' | media_url }}" />
```

CHAPTER 2

CLI Reference

See *Quickstart* for examples of how to use the CLI

2.1 Full CLI Reference

2.1.1 posty

```
posty [OPTIONS] COMMAND [ARGS]...
```

build

Build a Posty site as rendered HTML

```
posty build [OPTIONS]
```

Options

-o, --output <output>
Output directory

-c, --config <config>
Path to your config file

import

Import a site from another static site generator

```
posty import [OPTIONS] COMMAND [ARGS]...
```

posty1

Import a Posty 1.x site from PATH

```
posty import posty1 [OPTIONS] PATH
```

Arguments

PATH
Required argument

init

Initialize a Posty site

```
posty init [OPTIONS]
```

new

Create a new post or page

```
posty new [OPTIONS] COMMAND [ARGS]...
```

page

Create a new page from the template

```
posty new page [OPTIONS]
```

Options

--name <name>
Name of the new page

post

Create a new page from the template

```
posty new post [OPTIONS]
```

Options

--name <name>
Name of the new post

CHAPTER 3

Config File

Posty uses a simple YAML file for its config. Don't worry, Posty will validate your config and let you know if anything's missing or wrong.

3.1 Example Config

This is an example config file, which is what you get in the skeleton site when you run `posty init`.

```
author: you!
title: My website
description: Thoughts and stuff

# URL of where this site will be hosted, must end with a /
base_url: http://example.org/

num_top_tags: 5
num_posts_per_page: 5

# Set rss or atom to False if you do not want to generate those feeds
feeds:
  rss: True
  atom: True

# Backward compatibility tunables
compat:
  redirect_posty1_urls: False
```

3.2 Config Variables

These are all of the config variables that Posty will recognize. It will ignore any others that you set, so if you wanted to pass any extra config to your templates for example, you can do so!

These config variables are all accessible in the templates via `{{ site.config }}`.

- `author` [required] - Your name. This gets used in the copyright string if you choose to add that to your templates.
- `title` [required] - The title for your site
- `description` - An optional description of your site
- `base_url` - The location at which your Posty-powered website will be hosted. So if it'll be hosted at https://example.org/blog/, then that should be the value set here. This value **must** have a trailing slash.
- `num_top_tags` - The number of tags to include in the 'top tags' list
- `num_posts_per_page` - When generating HTML files containing posts from the entire list of posts, Posty will break them up into files containing this number of posts
- `feeds.rss` - Set to `true` to generate an RSS feed XML file
- `feeds.atom` - Set to `true` to generate an Atom feed XML file

3.2.1 Compatibility Config

- `compat.redirect_posty1_urls` - If set to `true`, Posty will generate HTML files which redirect from an old Posty 1.x post URL to Posty 2.x post URLs. Use this when you are converting a Posty 1.x site to 2.x.

CHAPTER 4

Post and Page Schema

A site is made up of two main components: *Page* and *Post*. Each type of file consists of two parts: a YAML header and the actual content. See below for the format of each.

Like the *config file*, you can set extra keys in the metadata which will simply be passed along, enabling you to use them in your templates.

4.1 Page

Pages are the simplest object. These are located in the `pages/` directory in your site root. They are simply some YAML metadata, followed by three dashes, followed my the Markdown content of the page itself.

When a page is rendered into HTML, its URL will be `:title_slug/index.html` relative to your site's *base_url*.

4.1.1 Example

```
title: About Me
---
This is a page. Write what you want here!
```

4.1.2 Required Metadata Fields

- `title` - The title of the page

4.2 Post

Posts are a little bit more complex. They have three sections separated by `---`: the YAML metadata, a blurb (or summary), and the body.

The 'blurb' is the start of your post, usually a single paragraph. It gets stored on to the Post object as the `blurb` field and is most useful when you have a list of posts and only want to show the blurb, but then have readers click a link to view the entire post.

The next section in the file, the `body`, is a separate piece from the blurb and Posty behind the scenes will combine the two when rendering a full-post HTML file.

If you do not have a `---` separating a blurb from a body, then it will just be considered to be just the body as a whole. The `blurb` field on the Post object will just be set to be the `body` so that the field is still populated with something.

All posts will be rendered into HTML files with URLs that look like `:year/:zero_padded_month/:slug/index.html` relative to the *base_url*.

Additionally a series of HTML files will be rendered with a certain number of posts per file (controlled by *num_posts_per_page*) in reverse-chronological order. The first page will be rendered as `index.html` and following pages will be rendered as `page/:page_num/index.html`.

4.2.1 Tags

If a post has a list of *tags* associated with it, these tags will be collected and rendered into lists of posts similar to the main list. They follow the same URL pattern as the main list but are prefixed with `tags/:tag_name/`, e.g.: `tags/foo/page/2/`.

4.2.2 Example

```
title: New Post
date: 1970-01-01
tags:
    - tag1
    - tag2
---
This the the summary/first paragraph of your new post
---
This is the rest of the post.

Write what you want here!
```

4.2.3 Blurb-less Example

```
title: New Post
date: 1970-01-01
tags:
    - tag1
    - tag2
---
This the the summary/first paragraph of your new post

This is the rest of the post.

Write what you want here!
```

4.2.4 Required Metadata Fields

- `title` - The title of the post
- `date` - The date of the post, formatted as `YYYY-MM-DD`

4.2.5 Optional Metadata Fields

- `tags` - A list of *tags* to be associated with the post.

CHAPTER 5

Templating

Posty uses Jinja for its templates. Your templates should live in the `templates/` directory in your site root.

5.1 Template Files

page.html post_base.html post.html posts.html redirect.html

There are a few required templates to be able to generate a site. Each of these should live in the *templates/* directory in your site root.

5.1.1 page.html

This renders a single Page.

The following variables are available:

- *site*
- *page*

5.1.2 post.html

Renders a single Post.

These variables are available:

- *site*
- *post*

5.1.3 posts.html

Renders a list of Posts. This is used both for rendering N number of posts in a series as pages, as well as rendering the same thing for each tag.

These variables are available:

- *site*
- *posts*
- *next_page_url*
- *prev_page_url*

5.1.4 redirect.html

This is a special optional template that is used if Posty 1.x URL compatibility is turned on. The only variable passed in is a `url` which is the real URL of the post.

This is provided for you if you run `posty init` and it's unlikely that you'll need to modify it.

5.2 Template Variables

These are the variables and the fields available on each that you can use in your templates. Note that not all variables are available in all tempaltes, see above for which ones are.

5.2.1 site

This is a representation of the site as a whole. Note that if you're trying to access a list of posts, you should likely use the

It has the following fields available on it:

- `pages` - The list of all Pages on the site
- `posts` - The list of all Posts on the site, in reverse chronological order
- `tags` - The list of all tags found in Posts on the site
- `config` - The configuration loaded from the *Config File*. See *Config Variables* for details on what fields are available.
- `copyright` - The copyright string for the site, based on the year of your earliest post, the year of the latest post, and the `author` set in the *Config File*.

5.2.2 page

The representation of the *Page* being rendered.

It has these fields and functions available on it:

- `title` - The title of the page
- `body` - The body text of the page
- `parent` - The parent page to this page. Will be `None` if this is a top-level page.

- `slug` - The slugified title of the page, as used in the URL
- `url()` - Function which returns the absolute URL to this page

5.2.3 post

The represenatation of the *Post* being rendererd.

It has these fields and functions available on it:

- `title`
- `date`
- `tags` - The list of tags for this post
- `blurb` - The blurb (summary) of this post as defined in the YAML file. If no blurb was set, then this is identical to the body.
- `body` - The body of this post as defined in the YAML file. This will include the `blurb` if one was set in the YAML file.
- `slug` - The slugified title of this post, as used in the URL
- `url()` - Function which returns the absolute URL to this post

5.2.4 posts

A list of *post* objects.

5.2.5 next_page_url

If not null, provides the absolute URL to the next page of post items.

5.2.6 prev_page_url

If not null, provides the absolute URL to the previous page of post items.

5.3 Jinja Filters

These functions are available as Jinja filters in all templates.

5.3.1 markdown

This filter takes text and returns the Markdown-rendered version of it.

Usage: `{{ post.body | markdown }}`

5.3.2 media_url

This filter takes a URL relative to the `media/` directory and returns an abosulte URL to that thing.

For example, if your `base_url` in your config is https://example.org/site/:

```
{{ "css/index.css" | media_url }}
```

Returns `https://example.org/site/media/css/index.css`.

5.3.3 absolute_url

This filter works in a similar way to *media_url*, but instead returning the absolute URL for an arbitrary relative URL. It does this by concatenating the `base_url` from config with the given relative URL.

This is handy if you're directly linking to a page from some other page or a post.

Usage: `{{ "/some-page-name/" | absolute_url }}`

CHAPTER 6

Importing From Other Static Site Generators

6.1 Posty 1.x

The `posty import posty1` command lets you import from an old Posty 1.x site into a new Posty 2.x site. This command should be ran from a directory which will house your new Posty 2.x site.

It doesn't matter if this Posty 2.x site directory has been initialized or not, but you will need to make sure that a *valid config* exists in your site directory.

6.1.1 CLI Example

```
# From within your Posty 2.x site directory (empty or not)
posty import posty1 /path/to/your/posty1_site
```

6.1.2 Import Process

Here's what happens when you run the `posty import posty1` command:

1. All of your media files are copied from `old_site/_media/` to `new_site/media/`.
2. All of your templates are copied from `old_site/_templates/` to `new_site/templates/`.
3. All of your pages are copied from `old_site/_pages/` to `new_site/pages/`.
4. All of your posts are copied and converted from `old_site/_posts/` to `new_site/posts/`.

In each post, the first paragraph of each body will be converted into a *blurb*, so it's a good idea to inspect each post to make sure it was converted to your liking.

Additionally it's a good idea to check that your templates will work with Posty 2.x. See *Templating* for more info on what variables are available.

CHAPTER 7

Development

All development is currently done against Python 3.5 (the latest available in Ubuntu 16.04), and all CI tests are ran against version 2.7, 3.5, and 3.6.

7.1 Bootstrapping

Bootstrapping your development environment is easy! Once you set up whatever virtualenv or container or VM that you want to do development in, there's one command to run:

```
make develop
```

This will install all of the pip requirements in `requirements.dev.txt` and install Posty into your environment in 'editable mode'. Editable mode means that Posty will be installed into your env as if it were any other package, but any changes you make to the source code will be instantly reflected in the CLI binary.

7.2 Testing

To run the test suite, simply run `make test`. This will run:

1. `pycodestyle` (formerly `pep8`)
2. `flake8`
3. `pytest`

CHAPTER 8

posty

8.1 posty package

8.1.1 Subpackages

posty.renderer package

Submodules

posty.renderer.atom module

class posty.renderer.atom.**AtomRenderer**(*site*, *output_path='build'*)

Bases: *posty.renderer.feed.FeedRenderer*

Renderer that outputs an Atom feed XML file

filename = 'atom.xml'

output()

Output the Atom feed file

url()

Return the URL to this feed file

posty.renderer.base module

class posty.renderer.base.**Renderer**(*site*, *output_path='build'*)

Bases: object

Base class that all renderers inherit off of. Each child class must implement `render_site()` with their own rendering logic.

ensure_output_path()
Ensure that the output directory `self.output_path` exists

render_site()

posty.renderer.feed module

class posty.renderer.feed.**FeedRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.base.Renderer*

Base class for all feed Renderers (RSS, Atom)

output()
This method must be implemented by child classes. It gets called during render_site to output the specific file, such as the RSS file or Atom file

render_posts()
Add each post to the feed

render_site()

url()
Return the URL to this feed file

posty.renderer.html module

class posty.renderer.html.**HtmlRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.base.Renderer*

Renderer that outputs HTML files

prepare_content()
Do a first-pass rendering of each post and page text, treating the text as Jinja2 templates. This lets us use basic jinja in the markdown to be able to use filter functions like this:

```
{{ "img/cool_pic.jpg" | media_url }}
```

render_page(*page*, *template_name='page.html'*)

Parameters **page** – a Page object

render_post(*post*, *template_name='post.html'*)

Parameters **post** – a Post object

render_posts(*posts*, *prefix=''*, *template_name='posts.html'*)
Render a list of posts as sets of pages where each page has `num_posts_per_page` posts. Each page of posts will be rendered to the path page/:page/index.html relative to the Renderer output_path

If `prefix` is given, add that will be put in between the output_path and page path. For example if the prefix is 'tags/foo/' then a page path would look like 'tags/foo/page/:page/index.html'

render_site()
Given a Site object, render all of its components

Parameters **site** – a loaded Site object

render_site_posts()
Renders all of the multi-post pages, N per page

render_site_tags(*template_name='posts.html'*)
Renders all of the per-tag multi-post pages, N per page

posty.renderer.json module

class posty.renderer.json.**JsonRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.base.Renderer*

Renderer that outputs a JSON representation of the Site to `site.json` within the output directory

render_site()
Render the Site to `site.json`

posty.renderer.posty1_redirect module

class posty.renderer.posty1_redirect.**Posty1RedirectRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.base.Renderer*

Renderer which creates pages to redirect old Posty1 URLs to new Posty2 URLs

Old Posty1 post URLs are in the form of: /:year/:month/:old_slug.html

Posty2 URLs are in the form of: /:year/:month/:slug/index.html

render_site()

posty.renderer.rss module

class posty.renderer.rss.**RssRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.feed.FeedRenderer*

Renderer that outputs a RSS feed XML file

filename = 'rss.xml'

output()
Output the RSS feed file

url()
Return the URL to this feed file

posty.renderer.util module

posty.renderer.util.**absolute_url_func**(*site*)
Returns a markdown filter function that returns an absolute URL for the given relative URL, simply concatenating config['base_url'] with the URL.

posty.renderer.util.**markdown**(*text*)
Returns the rendered version of the given Markdown text

posty.renderer.util.**media_url_func**(*site*)
Returns a filter function that returns a full media URL for the given file, scoped to the given Site object.

For example, if the Site has its base_url set to '/foo/' then: img/my_picture.jpg -> /foo/media/img/my_picture.jpg

Module contents

class posty.renderer.**AtomRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.feed.FeedRenderer*

Renderer that outputs an Atom feed XML file

filename = 'atom.xml'

output()
Output the Atom feed file

url()
Return the URL to this feed file

class posty.renderer.**HtmlRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.base.Renderer*

Renderer that outputs HTML files

prepare_content()
Do a first-pass rendering of each post and page text, treating the text as Jinja2 templates. This lets us use basic jinja in the markdown to be able to use filter functions like this:

```
{{ "img/cool_pic.jpg" | media_url }}
```

render_page(*page*, *template_name='page.html'*)

Parameters **page** – a Page object

render_post(*post*, *template_name='post.html'*)

Parameters **post** – a Post object

render_posts(*posts*, *prefix=''*, *template_name='posts.html'*)
Render a list of posts as sets of pages where each page has `num_posts_per_page` posts. Each page of posts will be rendered to the path page/:page/index.html relative to the Renderer output_path

If `prefix` is given, add that will be put in between the output_path and page path. For example if the prefix is 'tags/foo/' then a page path would look like 'tags/foo/page/:page/index.html'

render_site()
Given a Site object, render all of its components

Parameters **site** – a loaded Site object

render_site_posts()
Renders all of the multi-post pages, N per page

render_site_tags(*template_name='posts.html'*)
Renders all of the per-tag multi-post pages, N per page

class posty.renderer.**JsonRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.base.Renderer*

Renderer that outputs a JSON representation of the Site to `site.json` within the output directory

render_site()
Render the Site to `site.json`

class posty.renderer.**RssRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.feed.FeedRenderer*

Renderer that outputs a RSS feed XML file

filename = 'rss.xml'

output()
Output the RSS feed file

url()
Return the URL to this feed file

class posty.renderer.**Posty1RedirectRenderer**(*site*, *output_path='build'*)
Bases: *posty.renderer.base.Renderer*

Renderer which creates pages to redirect old Posty1 URLs to new Posty2 URLs

Old Posty1 post URLs are in the form of: /:year/:month/:old_slug.html

Posty2 URLs are in the form of: /:year/:month/:slug/index.html

render_site()

8.1.2 Submodules

posty.cli module

posty.config module

class posty.config.**Config**(*path='config.yml'*)
Bases: _abcoll.MutableMapping

Config object that gets passed around to various other objects. Loads config from a given YAML file.

Parameters **path** – Path to a YAML file to read in as config

clean_config()
Validate and clean the already-loaded config

load()
Load the YAML config from the given path, return the config object

posty.exceptions module

exception posty.exceptions.**InvalidConfig**(*config_obj*, *reason*)
Bases: *posty.exceptions.PostyError*

exception posty.exceptions.**InvalidObject**
Bases: *posty.exceptions.PostyError*

exception posty.exceptions.**MalformedInput**
Bases: *posty.exceptions.PostyError*

exception posty.exceptions.**PostyError**
Bases: exceptions.RuntimeError

exception posty.exceptions.**UnableToImport**
Bases: *posty.exceptions.PostyError*

posty.importers module

Functions to import from various other static site generators

class posty.importers.**Importer**(*site*, *src_path*)
Bases: *posty.model.ABC*

Base class for all importers

Parameters

- **site** – Site object for the destination
- **src_path** – Path to the thing to import

ensure_directories()

run()

class posty.importers.**Posty1Importer**(*site*, *src_path*)
Bases: *posty.importers.Importer*

Importer to pull from a Posty 1.x site

import_media()

import_pages()

import_posts()

import_templates()

run()

posty.model module

class posty.model.**ABC**
Bases: object

class posty.model.**Model**(*payload*, *config=None*)
Bases: *posty.model.ABC*, _abcoll.MutableMapping

Base class for objects representing things stored as YAML, such as a Post or a Page

Parameters

- **payload** – A dict representing the backing payload for this object
- **config** – A Config object

as_dict()
Return a true dict representation of this object, suitable for serialization into JSON or YAML

classmethod from_yaml(*file_contents*, *config=None*)
Load an object from its YAML file representation

path_on_disk()
Returns the relative path on disk to the object, for rendering purposes

url()
Returns the URL path to this resource

validate()
This should be implemented by the child class to verify that all fields that are expected exist on the payload, and set any that aren't

posty.page module

class posty.page.**Page**(*payload*, *config=None*)
Bases: *posty.model.Model*

Representation of a page

classmethod from_yaml(*file_contents*, *config=None*)
Return a Page from the given file_contents

path_on_disk()

to_yaml()
Returns a string of the YAML and text representation of this Post. This is the reverse of from_yaml

url()

validate()

posty.post module

class posty.post.**Post**(*payload*, *config=None*)
Bases: *posty.model.Model*

Representation of a post

classmethod from_yaml(*file_contents*, *config=None*)
Returns a Post from the given file_contents

path_on_disk()

to_yaml()
Returns the YAML and text representation of this Post. This is the reverse of `from_yaml()`

url()

validate()

posty.site module

class posty.site.**Site**(*site_path='.'*, *config_path=None*)
Bases: `object`

Representation of an entire site with posts and pages. This is the main class that conrols everything.

Parameters

- **site_path** – Path to the directory containing site content (pages, posts, templates)
- **config_path** – Path to the config file, defaults to `$SITE_PATH/config.yml`

config
Returns this site's config as read from the config file

copyright
Returns a string of the copyright info, based on the configured author and the years of the first and last post

init()
Initialize a new Posty site at the given path

load()
Load the site from files on disk into our internal representation

new_page(*name='New Page'*)
Create a new page in the site directory from the skeleton page

new_post(*name='New Post'*)
Create a new post in the site directory from the skeleton post

page(*slug*)
Returns a Page object by its slug

Parameters **slug** – slug of the page to find

Returns A page dict

Raises ***PostyError*** – if no page could be found

post(*slug*)
Returns a Post object by its slug

Parameters **slug** – slug of the post to find

Returns A post dict

Raises ***PostyError*** – if no post could be found

render(*output_path='build'*)
Render the site with the various renderers

- HTML
- JSON
- RSS (if `feeds.rss` is True in the config)
- Atom (if `feeds.atom` is True in the config)

posty.util module

Various utility functions

`posty.util.`**bucket**(*_list*, *size*)
Bucket the list `_list` into chunks of up to size `size`

Example: bucket([1,2,3,4,5], 2) -> [[1,2], [3,4], [5]]

`posty.util.`**slugify**(*text*)
Returns a slugified version of the given `text`

`posty.util.`**slugify_posty1**(*text*)
Returns a Posty 1.x compatible slugified version of `text`

8.1.3 Module contents

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

E

F

H

I

J

L

M

N

O

P

R

S

T

U

V