Note: You are not using the most up to date version of the library. 3.6.3 is the newest version.

Getting started

Installing Pelican

You're ready? Let's go! You can install Pelican via several different methods. The simplest is via pip:

```
$ pip install pelican
```

If you don't have `pip` installed, an alternative method is `easy_install`:

```
$ easy_install pelican
```

While the above is the simplest method, the recommended approach is to create a virtual environment for Pelican via virtualenv and virtualenvwrapper before installing Pelican. Assuming you've followed the virtualenvwrapper installation and shell configuration steps, you can then open a new terminal session and create a new virtual environment for Pelican:

```
$ mkvirtualenv pelican
```

Once the virtual environment has been created and activated, Pelican can be be installed via `pip` or `easy_install` as noted above. Alternatively, if you have the project source, you can install Pelican using the distutils method:

```
$ cd path-to-Pelican-source
$ python setup.py install
```

If you have Git installed and prefer to install the latest bleeding-edge version of Pelican rather than a stable release, use the following command:

```
$ pip install -e git://github.com/getpelican/pelican#e
```

If you plan on using Markdown as a markup format, you'll need to install the Markdown library as well:

```
$ pip install Markdown
```

If you want to use AsciiDoc you need to install it from source or use your operating system's package manager.

Upgrading

If you installed a stable Pelican release via `pip` or `easy_install` and wish to upgrade to the latest stable release, you can do so by adding `--upgrade` to the relevant command. For pip, that would be:

```
$ pip install --upgrade pelican
```

If you installed Pelican via distutils or the bleeding-edge method, simply perform the same step to install the most recent version.

Dependencies

At this time, Pelican is dependent on the following Python packages:

- feedgenerator, to generate the Atom feeds
- jinja2, for templating support
- docutils, for supporting reStructuredText as an input format

If you're not using Python 2.7, you will also need the `argparse` package.

Optionally:

- pygments, for syntax highlighting
- Markdown, for supporting Markdown as an input format
- Typogrify, for typographical enhancements

Kickstart a blog

Following is a brief tutorial for those who want to get started right away. We're going to assume that virtualenv and virtualenvwrapper are installed and configured; if you've installed Pelican outside of a virtual environment, you can skip to the `pelican-quickstart` command. Let's first create a new virtual environment and install Pelican into it:

```
$ mkvirtualenv pelican
$ pip install pelican Markdown
```

Next we'll create a directory to house our site content and configuration files, which can be located any place you prefer, and associate this new project with the currently-active virtual environment:

```
$ mkdir ~/code/yoursitename
$ cd ~/code/yoursitename
$ setvirtualenvproject
```

Now we can run the `pelican-quickstart` command, which will ask some questions about your site:

```
$ pelican-quickstart
```

Once you finish answering all the questions, you can begin adding content to the *content* folder that has been created for you. (See *Writing articles using Pelican* section below for more information about how to format your content.) Once you have some content to generate, you can convert it to HTML via the following command:

```
$ make html
```

If you'd prefer to have Pelican automatically regenerate your site every time a change is detected (handy when testing locally), use the following command instead:

```
$ make regenerate
```

To serve the site so it can be previewed in your browser at http://localhost:8000:

```
$ make serve
```

Normally you would need to run `make regenerate` and `make serve` in two separate terminal sessions, but you can run both at once via:

```
$ make devserver
```

The above command will simultaneously run Pelican in regeneration mode as well as serve the output at http://localhost:8000. Once you are done testing your changes, you should stop the development server via:

```
$ ./develop_server.sh stop
```

When you're ready to publish your site, you can upload it via the method(s) you chose during the `pelican-quickstart` questionnaire. For this example, we'll use rsync over ssh:

```
$ make rsync_upload
```

That's it! Your site should now be live.

Writing articles using Pelican

File metadata

Pelican tries to be smart enough to get the information it needs from the file system (for instance, about the category of your articles), but some information you need to provide in the form of metadata inside your files.

You can provide this metadata in reStructuredText text files via the following syntax (give your file the `.rst` extension):

```
My super title
```

```
##############

:date: 2010-10-03 10:20
:tags: thats, awesome
:category: yeah
:slug: my-super-post
:author: Alexis Metaireau
:summary: Short version for index and feeds
```

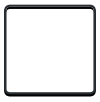
Pelican implements an extension to reStructuredText to enable support for the `abbr` HTML tag. To use it, write something like this in your post:

```
This will be turned into :abbr:`HTML (HyperText Markup
```

You can also use Markdown syntax (with a file ending in `.md`, `.markdown`, or `.mkd`). Markdown generation will not work until you explicitly install the `Markdown` package, which can be done via `pip install Markdown`. Metadata syntax for Markdown posts should follow this pattern:

```
Title: My super title
Date: 2010-12-03 10:20
Tags: thats, awesome
Category: yeah
Slug: my-super-post
Author: Alexis Metaireau
Summary: Short version for index and feeds

This is the content of my super blog post.
```



Note that, aside from the title, none of this metadata is mandatory: if the date is not specified, Pelican can rely on the file's "mtime" timestamp through the `DEFAULT_DATE` setting, and the category can be determined by the directory in which the file resides. For example, a file located at `python/foobar/myfoobar.rst` will have a category of `foobar`. If you would like to organize your files in other ways where the name of the subfolder would not be a good category name, you can set the setting `USE_FOLDER_AS_CATEGORY` to `False`. If there is no summary metadata for a given post, the `SUMMARY_MAX_LENGTH` setting can be used to specify how many words from the beginning of an article are used as the summary.

You can also extract any metadata from the filename through a regular expression to be set in the `FILENAME_METADATA` setting. All named groups that are matched will be set in the metadata object.

The default value for the `FILENAME_METADATA` setting will only extract the date from the filename. For example, if you would like to extract both the date and the slug, you could set something like: `'(?P<date>\d{4}-\d{2}-\d{2})_(?P<slug>.*)'`

Please note that the metadata available inside your files takes precedence over the metadata extracted from the filename.

Generate your blog

The `make` shortcut commands mentioned in the *Kickstart a blog* section are mostly wrappers around the `pelican` command that generates the HTML from the content. The `pelican` command can also be run directly:

```
$ pelican /path/to/your/content/ [-s path/to/your/set
```

The above command will generate your weblog and save it in the `output/` folder, using the default theme to produce a simple site. The default theme is simple HTML without styling and is provided so folks may use it as a basis for creating their own themes.

Pelican has other command-line switches available. Have a look at the help to see all the options you can use:

```
$ pelican --help
```

Auto-reload

It's possible to tell Pelican to watch for your modifications, instead of manually re-running it every time you want to see your changes. To enable this, run the `pelican` command with the `-r` or `--autoreload` option.

Pages

If you create a folder named `pages` inside the content folder, all the files in it will be used to generate static pages.

Then, use the `DISPLAY_PAGES_ON_MENU` setting to add all those

pages to the primary navigation menu.

If you want to exclude any pages from being linked to or listed in the menu then add a `status: hidden` attribute to its metadata. This is useful for things like making error pages that fit the generated theme of your site.

Linking to internal content

From Pelican 3.1 onwards, it is now possible to specify intra-site links to files in the *source content* hierarchy instead of files in the *generated* hierarchy. This makes it easier to link from the current post to other posts and images that may be sitting alongside the current post (instead of having to determine where those resources will be placed after site generation).

To link to internal content, use the following syntax: `|filename|path/to/file`.

For example, you may want to add links between “article1” and “article2” given the structure:

```
website/
├── content
│   ├── article1.rst
│   └── cat/
│       └── article2.md
└── pelican.conf.py
```

In this example, `article1.rst` could look like:

```
Title: The first article
Date: 2012-12-01

See below intra-site link examples in reStructuredTex

`a link relative to content root <|filename|/cat/artic
`a link relative to current file <|filename|cat/artic
```

and `article2.md`:

```
Title: The second article
Date: 2012-12-01

See below intra-site link examples in Markdown format
```

```
[a link relative to content root](|filename|/article1
[a link relative to current file](|filename|../articl
```

Note: You can use the same syntax to link to internal pages or even static content (like images) which would be available in a directory listed in `settings["STATIC_PATHS"]`.

Importing an existing blog

It is possible to import your blog from Dotclear, WordPress, and RSS feeds using a simple script. See *Import from other blog software*.

Translations

It is possible to translate articles. To do so, you need to add a `lang` meta attribute to your articles/pages and set a `DEFAULT_LANG` setting (which is English [en] by default). With those settings in place, only articles with the default language will be listed, and each article will be accompanied by a list of available translations for that article.

Pelican uses the article's URL "slug" to determine if two or more articles are translations of one another. The slug can be set manually in the file's metadata; if not set explicitly, Pelican will auto-generate the slug from the title of the article.

Here is an example of two articles, one in English and the other in French.

The English article:

```
Foobar is not dead
##################

:slug: foobar-is-not-dead
:lang: en

That's true, foobar is still alive!
```

And the French version:

```
Foobar n'est pas mort !
```

```
######################

:slug: foobar-is-not-dead
:lang: fr

Oui oui, foobar est toujours vivant !
```

Post content quality notwithstanding, you can see that only item in common between the two articles is the slug, which is functioning here as an identifier. If you'd rather not explicitly define the slug this way, you must then instead ensure that the translated article titles are identical, since the slug will be auto-generated from the article title.

Syntax highlighting

Pelican is able to provide colorized syntax highlighting for your code blocks. To do so, you have to use the following conventions inside your content files.

For reStructuredText, use the code-block directive:

```
.. code-block:: identifier

   <indented code block goes here>
```

For Markdown, include the language identifier just above the code block, indenting both the identifier and code:

```
A block of text.

    :::identifier
    <code goes here>
```

The specified identifier (e.g. `python`, `ruby`) should be one that appears on the list of available lexers.

Publishing drafts

If you want to publish an article as a draft (for friends to review before publishing, for example), you can add a `status: draft` attribute to its metadata. That article will then be output to the `drafts` folder and not listed on the index page nor on any category page.

Viewing the generated files

The files generated by Pelican are static files, so you don't actually need anything special to see what's happening with the generated files.

You can either use your browser to open the files on your disk:

```
firefox output/index.html
```

Or run a simple web server using Python:

```
cd output && python -m SimpleHTTPServer
```