Sphinx and Read the Docs Tips

Release 0.3.1

Thai Pangsakulyanont

March 11, 2015

Contents

1 Inferring Release Number from Git Tags 1

2 Creating Custom Link Roles 2

3 Better Fonts in PDF Output 3

4 Contributing 4

I am using Sphinx, in conjunction with the awesome Read the Docs documentation building and hosting service, to document my open source senior project Bemuse.

After fiddling for a couple of days, I have few tips and techniques to share.

Note: This is not a pro tip! I only use this name because the subdomain "tips" is already used. I'm also not a Pythonian; I just found Sphinx to be the right (and superior) tool for the right job. Therefore, my code may not be idiomatic – I mean, Pythonic.

1 Inferring Release Number from Git Tags

When starting a Sphinx documentation using `sphinx-quickstart`, it asks you for version number of the docs. This already makes me question about the maintenance burden.

The version number is already stored in the project as Git tags and `package.json` (but npm has a built-in command to increment the version number and also create a Git tag). Having to specify the version number in Sphinx documentation doesn't seem right to me.

After finding out that the configuration file is just a Python script, I modified it to infer the version number from Git tags:

```
import re
# The full version, including alpha/beta/rc tags.
release = re.sub('^v', '', os.popen('git describe').read().strip())
# The short X.Y version.
version = release
```

This code uses `git describe` to generate a version number based on current Git commit. If we are on a tag, such as `v0.10.1`, git describe returns that tag name:

```
$ git describe
v0.10.1
```

However, if we are not on the tag, `git describe` will find the closest tag to this commit, and append the number of commits since that tag, along with the commit ID:

```
$ git describe
v0.10.1-53-gd8be3ff
```

So there you have it. When building, the version number will describe exactly what commit the documentation is being built for.

Date 2015-03-11

Author Thai Pangsakulyanont

2 Creating Custom Link Roles

I think it'd be cool – especially in the developer's documentation – to link to the corresponding source code on GitHub.

In my project's docs, inside the architecture section, for example, describes the directory structure in the project. Each item links to corresponding GitHub source code:

Directory Structure

assets Image assets for use in the game. These assets can be referred from webpack code by `require('assets/...')`.

bin Useful scripts for routine work. Examples include setting up Git commit hooks and releasing a new version.

config Configuration code for webpack and other things.

And the corresponding source code:

```
**Directory Structure**

:tree:`assets`
  Image assets for use in the game.
  These assets can be referred from webpack code by ``require('assets/...')``.
:tree:`bin`
  Useful scripts for routine work.
  Examples include setting up Git commit hooks and releasing a new version.
```

```
:tree:`config`
  Configuration code for webpack and other things.
```

As you can see, a custom role, `:tree:`, is created to link to source code. To do that, you have to write a Sphinx extension.

Fortunately, this is easy. There are tutorials on defining custom roles in Sphinx and a more in-depth guide about creating a text role from Docutils.

First, you have to add a custom search path to `conf.py`, so that your extension may be loaded:

```
sys.path.insert(0, os.path.abspath('.') + '/_extensions')
```

Next, create the extension file. I call it `_extensions/bemuse.py`. It looks like this:

```
from docutils import nodes

def setup(app):
    app.add_role('github', autolink('https://github.com/%s'))
    app.add_role('module', autolink('https://github.com/bemusic/bemuse/tree/master/src/%s'))
    app.add_role('tree', autolink('https://github.com/bemusic/bemuse/tree/master/%s'))

def autolink(pattern):
    def role(name, rawtext, text, lineno, inliner, options={}, content=[]):
        url = pattern % (text,)
        node = nodes.reference(rawtext, text, refuri=url, **options)
        return [node], []
    return role
```

The `setup` function is called by Sphinx for an extension to add custom roles.

Since each role does similar thing: turn texts into links using some predefined pattern, I created a function, called `autolink`, that returns a function that turns text into link using some specified pattern.

Finally, add that extension to the configuration file:

```
# Add any Sphinx extension module names here, as strings. They can be
# extensions coming with Sphinx (named 'sphinx.ext.*') or your custom
# ones.
extensions = ['bemuse']
```

Date 2015-03-11

Author Thai Pangsakulyanont

3 Better Fonts in PDF Output

One thing I like a lot about Sphinx is that it can generate PDF files for your documentation, complete with table of contents, chapters and section numbers.

Read the Docs also automatically generate PDF documentations for you.

But there is one thing: the styling. While the HTML documentation looks nice and modern, thanks to Read the Docs' theme, the PDF documentation looks ugly and dull.

I think the major problem is the **typeface**. On HTML documentation, we have *Lato* and *Roboto Slab* font, which gives it a modern feel. For code, various modern monospace fonts are used (*Consolas*, *Andale Mono*, and so on.)

On PDF documentation, however, the default typefaces are *Helvetica*, *Times*, and *Courier*.

Are you serious?! Helvetica and Times makes your documentation look ancient and uninteresting. *Courier* is fat and thin and is kind of hard-to-read. This needs to change.

Sphinx uses pdfLaTeX to generate PDF documentation, and Read the Docs uses TeX Live in its build infrastructure. With TeX Live comes several nice fonts, which you can safely use in Read the Docs environment.

To use them, simply include the package, according to each font package's documentation, in the preamble option. For example, here is my configuration:

```
latex_elements = {
# The paper size ('letterpaper' or 'a4paper').
    'papersize': 'letterpaper',

# The font size ('10pt', '11pt' or '12pt').
    'pointsize': '11pt',

# Additional stuff for the LaTeX preamble.
    'preamble': r'''
        \usepackage{charter}
        \usepackage[defaultsans]{lato}
        \usepackage{inconsolata}
    ''',
}
```

In this configuration, **Charter** is used as serif font, **Lato** as sans-serif font, and **Inconsolata** as monospace font. Even though the colors and layout don't change, changing the typeface can give your PDF documentation a radically different feel. such wow. many modern.

Date 2015-03-11

Author Thai Pangsakulyanont

4 Contributing

The content lives on GitHub at dtinth/rtfd-protips.

If you have any tips to share, or any fixes or improvements, feel free to send me pull requests.