Sphinx API Documentation: Step 0 towards Reusable Code

Yubo “Paul” Yang, THW-IL, 2017/10/18

CODING HORROR
programming and human factors

Google Custom Search

23 Jan 2007

If It Isn't Documented, It Doesn't Exist

Nicholas Zakas enumerates the number one reason why good JavaScript libraries fail:

> **Lack of documentation.** No matter how wonderful your library is and how intelligent its design, if you're the only one who understands it, it doesn't do any good. Documentation means not just autogenerated API references, but also *annotated* examples and in-depth tutorials. You need all three to make sure your library can be easily adopted.

Act I: It is all so **simple**

Sphinx API Documentation Requires docstring only

"A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition." – PEP 257, Python.org

```
6  def isosurf(ax,vol,level_frac=0.25):
7      """draw iso surface of volumetric data on matplotlib axis at given level
8
9      Example usage:
10       from mpl_toolkits.mplot3d import Axes3D # enable 3D projection
11       vol = np.random.randn(10,10,10)
12       fig = plt.figure()
13       ax  = fig.add_subplot(1,1,1,projection='3d')
14       isosurf(ax,vol)
15       plt.show()
16
17     Args:
18       ax (plt.Axes3D): ax = fig.add_subplot(1,1,1,projection="3d")
19       vol (np.array): 3D volumetric data having shape (nx,ny,nz)
20       level_frac (float): 0.0->1.0, isosurface value as a fraction between min and max
21     Returns:
22       Poly3DCollection: mesh
23     Effect:
24       draw on ax """
25     from skimage import measure
26     from mpl_toolkits.mplot3d.art3d import Poly3DCollection
27     nx,ny,nz = vol.shape
28     lmin,lmax = vol.min(),vol.max()
```

Sphinx

qharv.inspect.volumetric module

qharv.inspect.volumetric.**axes_func_on_grid3d**(*axes*, *func*, *grid_shape*) [source]

put a function define in axes units on a 3D grid :param axes: dtype=float, shape=(3,3); 3D lattice vectors in row major (i.e. a1 = axes[0]) :type axes: np.array :param func: 3D function defined on the unit cube :type func: RegularGridInterpolator :param grid_shape: dtype=int, shape=(3,); shape of real space grid :type grid_shape: np.array

Returns: dtype=float, shape=grid_shape; volumetric data
Return type: grid (np.array)

qharv.inspect.volumetric.**isosurf**(*ax*, *vol*, *level_frac=0.25*) [source]

draw iso surface of volumetric data on matplotlib axis at given level

Example usage:

from mpl_toolkits.mplot3d import Axes3D # enable 3D projection vol = np.random.randn(10,10,10) fig = plt.figure() ax = fig.add_subplot(1,1,1,projection='3d') isosurf(ax,vol) plt.show()

Parameters:
- **ax** (*plt.Axes3D*) – ax = fig.add_subplot(1,1,1,projection="3d")
- **vol** (*np.array*) – 3D volumetric data having shape (nx,ny,nz)
- **level_frac** (*float*) – 0.0->1.0, isosurface value as a fraction between min and max

Returns: mesh
Return type: Poly3DCollection

Effect:

draw on ax

Sphinx API Documentation Setup Takes **ONE** Command

sphinx-apidoc -A "Paul" -F -o docs src/

- sphinx-apidoc is a command available in the sphinx package, which can be installed without admin privileges using `pip install –user sphinx`.
- -A specifies the author name
- -F triggers a full setup
- -o specifies the documentation directory
- src/ is the folder that contains your Python package

Gotcha:
sphinx-apidoc cannot build documentation if it cannot **import your module**!

Ref: jlk's blog post on raxcloud

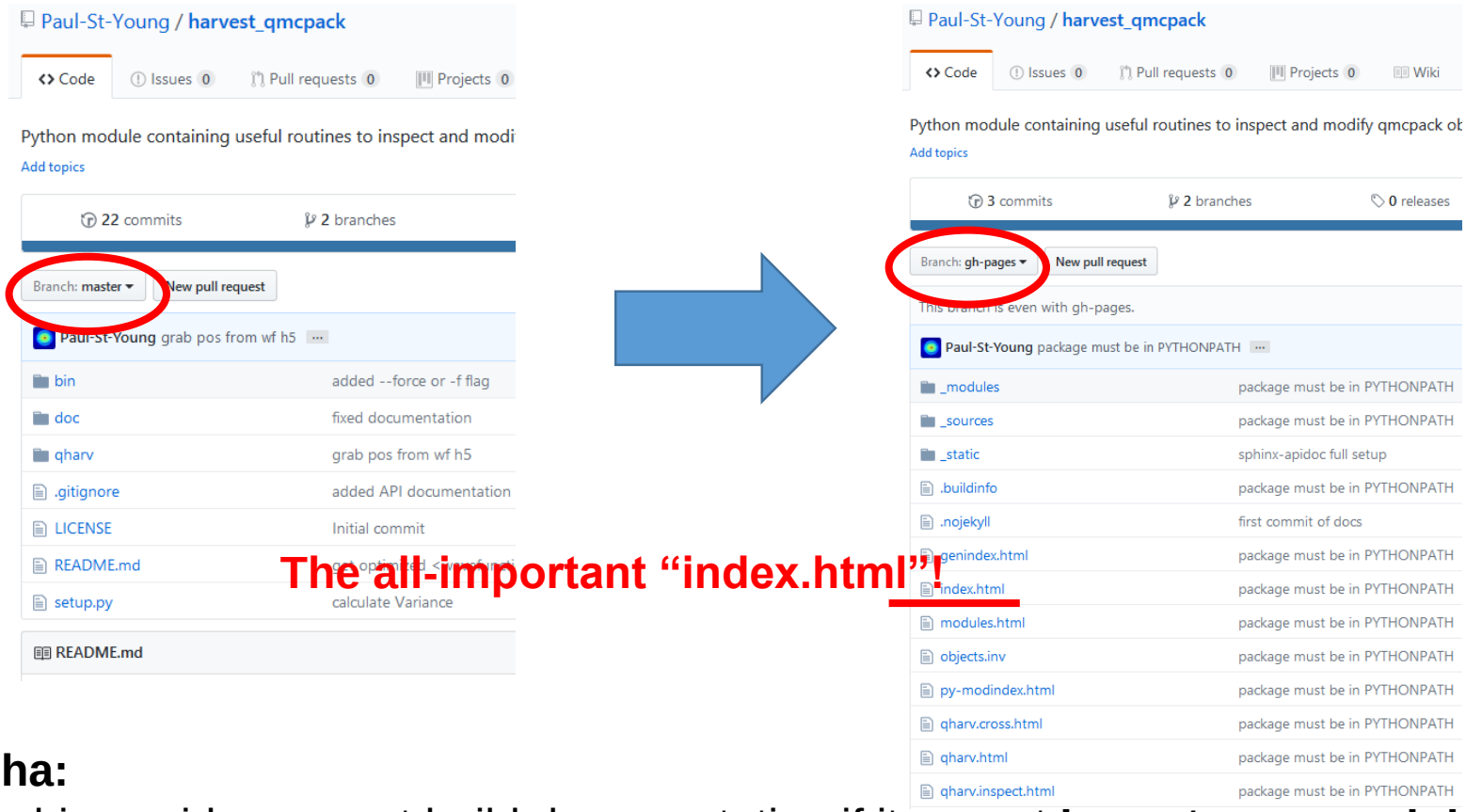
Sphinx API Documentation Generation also takes **ONE** Command

```
make html
```

- pdf version can be generated using `make latexpdf`. You will need the latexmk package, which should be easily obtainable via `apt-get/dnf install latexmk`.
- Read the Makefile to see what other options are available.

Publish Documentation on GitHub Pages

Goal Make gh-pages branch to hold the output of `make html`.



The all-important "index.html"!

Gotcha:

1. sphinx-apidoc cannot build documentation if it cannot **import your module**!
2. There are more than one way to **create an empty gh-pages branch** in your repository (see Ref [1] vs. Ref [3]). No need to be stubborn.

Ref [1] Luca Sbardella blog post on 2010/02/09
Ref [2] Ryan Dale's sphinxdoc-test GitHub repository
Ref [3] jlk's blog post on raxcloud

Conclusions: Sphinx is as easy as 1,2,3!

1. Write docstring for your functions

2. One command to setup: sphinx-apidoc –A “Paul” –o docs src/

3. One command to run: make html

Ready to publish!

Act II: *What* just happened?

`sphinx-apidoc` converts .py to .rst

sphinx-apidoc -A "Paul" -F -o docs src/

- docs folder contains one .rst file for each .py file in your src/ (with one extra index.rst)
- -F generates conf.py and Makefile, you may wish to add a few extensions in conf.py

default

```
extensions = ['sphinx.ext.autodoc',
    'sphinx.ext.todo',
    'sphinx.ext.viewcode']
```

customized

```
extensions = ['sphinx.ext.autodoc',
    'sphinx.ext.napoleon',
    'sphinx.ext.coverage',
    'sphinx.ext.mathjax',
    'sphinx.ext.viewcode',
    'sphinx.ext.githubpages']
```

Goolge style docstring

pages

- For more custom setup, run `sphinx-quickstart` for *the quiz*.

For Our Beloved Python 2

- conf.py uses Python 3 style by default!

- To change style, edit conf.py:

1. change `html_theme = 'alabaster'` to `html_theme = 'classic'`

2. remove unnecessary cram:

```
html_sidebars = {
    '**': [
        #'about.html',
        'searchbox.html',
        'globaltoc.html',
        #'navigation.html',
        'relations.html',
        #'donate.html',
    ]
}
```

Image credit: Chris Okorie, http://chrisokorie.com/

`make html` converts .rst to .html

make html

- `make html` fills docs/_build/html folder with html files
- Try `firefox docs/_build/html/index.html`

```
[yyang173@localhost html]$ ls
genindex.html   _modules        objects.inv       search.html     _sources
index.html      nexus_obj.html  py-modindex.html  searchindex.js  _static
```

- The ‘githubpages’ extension will add an empty ‘**.nojekyll**’ file to docs/_build/html/
- `touch docs/_build/html/.nojekyll` should also work, but is not automatic

Act III: Beginning of the End

API Documentation is Step 0 towards Reusable Code

CODING HORROR
programming and human factors

Google Custom Search

23 Jan 2007

If It Isn't Documented, It Doesn't Exist

Nicholas Zakas enumerates the number one reason why good JavaScript libraries fail:

> **Lack of documentation.** No matter how wonderful your library is and how intelligent its design, if you're the only one who understands it, it doesn't do any good. Documentation means not just autogenerated API references, but also *annotated* examples and in-depth tutorials. You need all three to make sure your library can be easily adopted.

What can Withstand the Revenges of Time?

- API Documentation
- Tests
- Examples

scikit-learn

scikit-learn is a Python module for machine le

The project was started in 2007 by David Cou
have contributed. See the AUTHORS.rst file fo

It is currently maintained by a team of volunte

Website: http://scikit-learn.org

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...
— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...
— Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ...
— Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization...
— Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.
— Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.
— Examples

Optional Python Module

Python Module = folder having __init__.py

Module:

```
[yyang173@localhost qharv]$ ls
cross    __init__.py  inspect  reel  seed
[yyang173@localhost qharv]$ ls inspect/
crystal.py    __init__.py  jastrow.py  test  volumetric.py
```

Use:

```
if __name__ == '__main__':
  import argparse
  from qharv.inspect import crystal

  parser = argparse.ArgumentParser()
  parser.add_argument('fname',type=str,help='xml input file name')
  parser.add_argument('--pset_name','-pset',type=str,default='ion0'
    ,help='name of static/source/classical particle set')
  parser.add_argument('--super','-s',action='store_true'
    ,help='draw 2x2x2 supercell')
  parser.add_argument('--text','-t',action='store_true'
    ,help='print axes,pos of the crystal structure instead of plotting')
  args = parser.parse_args()

  axes = crystal.lattice_vectors(args.fname)
  pos  = crystal.atomic_coords(args.fname,pset_name=args.pset_name)

  if args.text:
```

https://www.halloweenexpress.com/adult-corn-stalk-costume-p-25781.html

Gotha ImportError: No module named qharv.inspect

- Make sure your package is in your PYTHONPATH
- `export PYTHONPATH=~/harvest_qmcpack:$ PYTHONPATH`
- `pip install --user ~/harvest_qmcpack`

Remind yourself in README

Unit Tests: Executable Documentation

```
[yyang173@localhost qharv]$ ls
cross    __init__.py  inspect  reel  seed
[yyang173@localhost qharv]$ ls inspect/
crystal.py   __init__.py  jastrow.py  test  volumetric.py
```

```
[yyang173@localhost inspect]$ ls test/
test_crystal.py
```

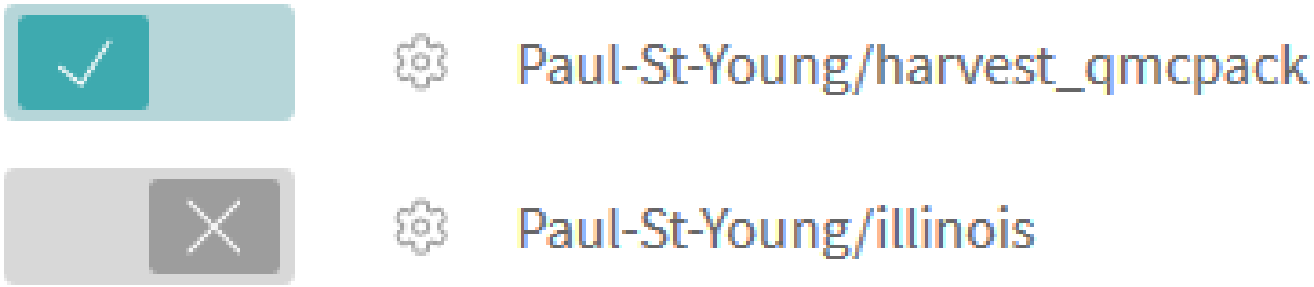
```
import urllib
import numpy as np

fname = 'TIP5P_PIMC.32.P5.0C.0.ptcl.xml'

def save_xml_example():
  if not os.path.isfile(fname):
    flink = 'https://sites.google.com/a/cmscc.org/qmcpac
.P5.0C.0.ptcl.xml'
    response = urllib.urlopen(flink)
    text = response.read()
    with open(fname,'w') as fp:
      fp.write(text)
    # end with
  # end if
# end def

def test_axes():
  save_xml_example()
  from qharv.inspect.crystal import lattice_vectors
  axes = lattice_vectors(fname)
  assert np.allclose(axes,18.330056710775050*np.eye(3))
# end def test_axes
```

```
[yyang173@localhost harvest_qmcpack]$ nosetests -v
test_crystal.test_axes ... ok

----------------------------------------------------------------------
Ran 1 test in 1.031s

OK
```

Continuous Integration: Test at Every Commit

Travis CI

(almost) as simple as flipping a switch



Continuous Integration: Add **One** File to Repo. (.travis.yml)

```
language: python
python:
  - "2.7"
install:
  - pip install -r requirements.txt
script:
  - nosetests -v
```

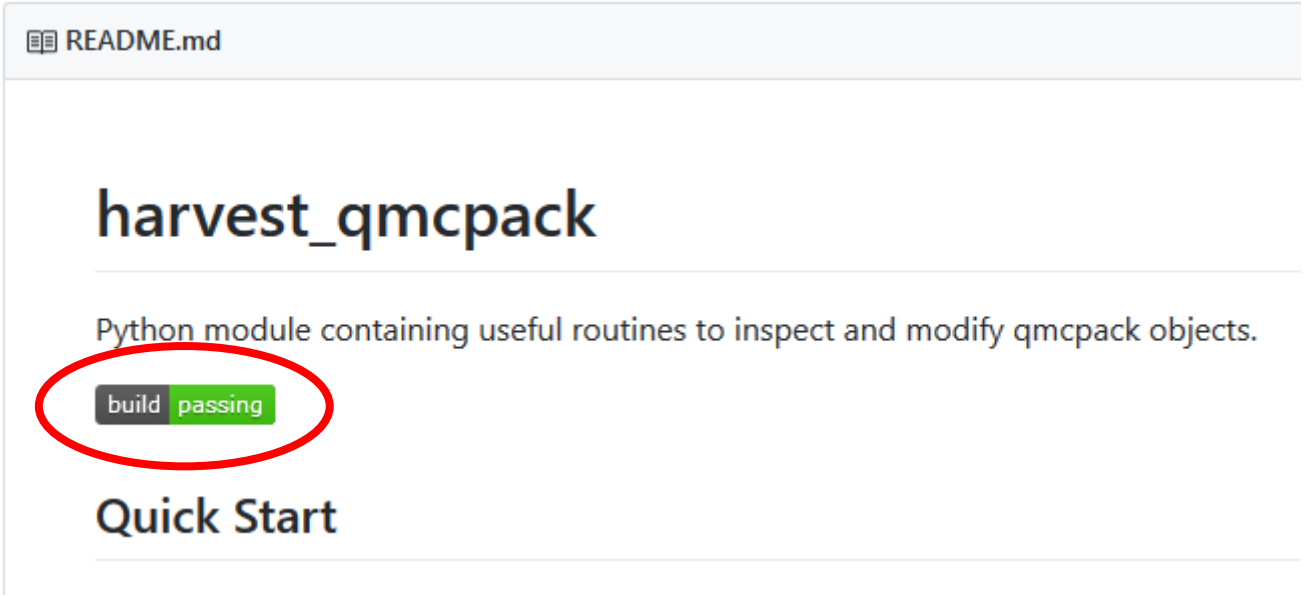
```
[yyang173@localhost harvest_qmcpack]$ cat requirements.txt
nose
lxml
h5py
numpy
```

Continuous Integration: Now Brag!

Add the following line to README.md:

```
![master build status]
(https://api.travis-ci.org/Paul-St-Young/harvest_qmcpack.svg?branch=master)
```

And you will receive a stamp of approval!



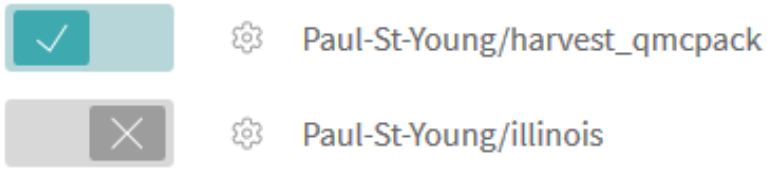
Conclusions:

- □ Create Python module with `touch src/__init__.py`
- □ Write your docstring **Goolge style**
- □ Generate API documentation:

```
sphinx-apidoc -A "Paul" -F -o docs src/
```

```
make html
```

- □ Copy html to  **pages**
- □ Add examples and unit tests
- □ Befriend Travis




```yaml
language: python
python:
  - "2.7"
install:
  - pip install -r requirements.txt
script:
  - nosetests -v
```