nexb-skeleton

AboutCode.org authors and contributors

Feb 19, 2024

CONTENTS:

CHAPTER

ONE

USAGE

1.1 A brand new project

```
git init my-new-repo
cd my-new-repo
git pull git@github.com:nexB/skeleton

# Create the new repo on GitHub, then update your remote
git remote set-url origin git@github.com:nexB/your-new-repo.git
```

From here, you can make the appropriate changes to the files for your specific project.

1.2 Update an existing project

```
cd my-existing-project
git remote add skeleton git@github.com:nexB/skeleton
git fetch skeleton
git merge skeleton/main --allow-unrelated-histories
```

This is also the workflow to use when updating the skeleton files in any given repository.

1.3 Customizing

You typically want to perform these customizations:

- remove or update the src/README.rst and tests/README.rst files
- set project info and dependencies in setup.cfg
- check the configure and configure.bat defaults

1.4 Initializing a project

All projects using the skeleton will be expected to pull all of it dependencies from thirdparty.aboutcode.org/pypi or the local thirdparty directory, using requirements.txt and/or requirements-dev.txt to determine what version of a package to collect. By default, PyPI will not be used to find and collect packages from.

In the case where we are starting a new project where we do not have requirements.txt and requirements-dev.txt and whose dependencies are not yet on thirdparty.aboutcode.org/pypi, we run the following command after adding and customizing the skeleton files to your project:

```
./configure
```

This will initialize the virtual environment for the project, pull in the dependencies from PyPI and add them to the virtual environment.

1.5 Generating requirements.txt and requirements-dev.txt

After the project has been initialized, we can generate the requirements.txt and requirements-dev.txt files.

Ensure the virtual environment is enabled.

```
source venv/bin/activate
```

To generate requirements.txt:

```
python etc/scripts/gen_requirements.py -s venv/lib/python<version>/site-packages/
```

Replace <version> with the version number of the Python being used, for example: `venv/lib/python3.6/site-packages/`

To generate requirements-dev.txt after requirements.txt has been generated:

```
./configure --dev
python etc/scripts/gen_requirements_dev.py -s venv/lib/python<version>/site-packages/
```

Note: on Windows, the `site-packages` directory is located at `venv\Lib\site-packages\`

```
python .\\etc\\scripts\\gen_requirements.py -s .\\venv\\Lib\\site-packages\\
.\configure --dev
python .\\etc\\scripts\\gen_requirements_dev.py -s .\\venv\\Lib\\site-packages\\
```

1.6 Collecting and generating ABOUT files for dependencies

Ensure that the dependencies used by `etc/scripts/fetch_thirdparty.py` are installed:

```
pip install -r etc/scripts/requirements.txt
```

Once we have requirements.txt and requirements-dev.txt, we can fetch the project dependencies as wheels and generate ABOUT files for them:

```
python etc/scripts/fetch_thirdparty.py -r requirements.txt -r requirements-dev.txt
```

There may be issues with the generated ABOUT files, which will have to be corrected. You can check to see if your corrections are valid by running:

```
python etc/scripts/check_thirdparty.py -d thirdparty
```

Once the wheels are collected and the ABOUT files are generated and correct, upload them to third-party.aboutcode.org/pypi by placing the wheels and ABOUT files from the thirdparty directory to the pypi directory at https://github.com/nexB/thirdparty-packages

1.7 Usage after project initialization

Once the `requirements.txt` and `requirements-dev.txt` have been generated and the project dependencies and their ABOUT files have been uploaded to thirdparty.aboutcode.org/pypi, you can configure the project as needed, typically when you update dependencies or use a new checkout.

If the virtual env for the project becomes polluted, or you would like to remove it, use the `--clean` option:

```
./configure --clean
```

Then you can run `./configure` again to set up the project virtual environment.

To set up the project for development use:

```
./configure --dev
```

To update the project dependencies (adding, removing, updating packages, etc.), update the dependencies in `setup.cfg`, then run:

```
./configure --clean # Remove existing virtual environment
source venv/bin/activate # Ensure virtual environment is activated
python etc/scripts/gen_requirements.py -s venv/lib/python<version>/site-packages/ #␣
↪Regenerate requirements.txt
python etc/scripts/gen_requirements_dev.py -s venv/lib/python<version>/site-packages/ #␣
↪Regenerate requirements-dev.txt
pip install -r etc/scripts/requirements.txt # Install dependencies needed by etc/scripts/
↪bootstrap.py
python etc/scripts/fetch_thirdparty.py -r requirements.txt -r requirements-dev.txt #␣
↪Collect dependency wheels and their ABOUT files
```

Ensure that the generated ABOUT files are valid, then take the dependency wheels and ABOUT files and upload them to thirdparty.aboutcode.org/pypi.

CHAPTER

TWO

CONTRIBUTING TO THE DOCUMENTATION

2.1 Setup Local Build

To get started, create or identify a working directory on your local machine.

Open that directory and execute the following command in a terminal session:

```
git clone https://github.com/nexB/skeleton.git
```

That will create an `/skeleton` directory in your working directory. Now you can install the dependencies in a virtualenv:

```
cd skeleton
./configure --docs
```

Note: In case of windows, run `configure --docs` instead of this.

Now, this will install the following prerequisites:

- Sphinx
- sphinx_rtd_theme (the format theme used by ReadTheDocs)
- docs8 (style linter)

These requirements are already present in setup.cfg and *./configure –docs* installs them.

Now you can build the HTML documents locally:

```
source venv/bin/activate
cd docs
make html
```

Assuming that your Sphinx installation was successful, Sphinx should build a local instance of the documentation .html files:

```
open build/html/index.html
```

Note: In case this command did not work, for example on Ubuntu 18.04 you may get a message like “Couldn’t get a file descriptor referring to the console”, try:

```
see build/html/index.html
```

You now have a local build of the AboutCode documents.

2.2 Share Document Improvements

Ensure that you have the latest files:

```
git pull
git status
```

Before commiting changes run Continious Integration Scripts locally to run tests. Refer *Continuous Integration* for instructions on the same.

Follow standard git procedures to upload your new and modified files. The following commands are examples:

```
git status
git add source/index.rst
git add source/how-to-scan.rst
git status
git commit -m "New how-to document that explains how to scan"
git status
git push
git status
```

The Scancode-Toolkit webhook with ReadTheDocs should rebuild the documentation after your Pull Request is Merged.

Refer the Pro Git Book available online for Git tutorials covering more complex topics on Branching, Merging, Rebasing etc.

2.3 Continuous Integration

The documentations are checked on every new commit through Travis-CI, so that common errors are avoided and documentation standards are enforced. Travis-CI presently checks for these 3 aspects of the documentation :

1. Successful Builds (By using `sphinx-build`)
2. No Broken Links (By Using `link-check`)
3. Linting Errors (By Using `Doc8`)

So run these scripts at your local system before creating a Pull Request:

```
cd docs
./scripts/sphinx_build_link_check.sh
./scripts/doc8_style_check.sh
```

If you don't have permission to run the scripts, run:

```
chmod u+x ./scripts/doc8_style_check.sh
```

2.4 Style Checks Using Doc8

2.4.1 How To Run Style Tests

In the project root, run the following commands:

```
$ cd docs
$ ./scripts/doc8_style_check.sh
```

A sample output is:

```
Scanning...
Validating...
docs/source/misc/licence_policy_plugin.rst:37: D002 Trailing whitespace
docs/source/misc/faq.rst:45: D003 Tabulation used for indentation
docs/source/misc/faq.rst:9: D001 Line too long
docs/source/misc/support.rst:6: D005 No newline at end of file
========
Total files scanned = 34
Total files ignored = 0
Total accumulated errors = 326
Detailed error counts:
    - CheckCarriageReturn = 0
    - CheckIndentationNoTab = 75
    - CheckMaxLineLength = 190
    - CheckNewlineEndOfFile = 13
    - CheckTrailingWhitespace = 47
    - CheckValidity = 1
```

Now fix the errors and run again till there isn't any style error in the documentation.

2.4.2 What is Checked?

PyCQA is an Organization for code quality tools (and plugins) for the Python programming language. Doc8 is a sub-project of the same Organization. Refer this README for more details.

What is checked:

- invalid rst format - D000
- lines should not be longer than 100 characters - D001
  - RST exception: line with no whitespace except in the beginning
  - RST exception: lines with http or https URLs
  - RST exception: literal blocks
  - RST exception: rst target directives
- no trailing whitespace - D002
- no tabulation for indentation - D003
- no carriage returns (use UNIX newlines) - D004
- no newline at end of file - D005

2.5 Interspinx

ScanCode toolkit documentation uses Intersphinx to link to other Sphinx Documentations, to maintain links to other Aboutcode Projects.

To link sections in the same documentation, standart reST labels are used. Refer Cross-Referencing for more information.

For example:

```
.. _my-reference-label:

Section to cross-reference
--------------------------

This is the text of the section.

It refers to the section itself, see :ref:`my-reference-label`.
```

Now, using Intersphinx, you can create these labels in one Sphinx Documentation and then referance these labels from another Sphinx Documentation, hosted in different locations.

You just have to add the following in the `conf.py` file for your Sphinx Documentation, where you want to add the links:

```
extensions = [
'sphinx.ext.intersphinx'
]

intersphinx_mapping = {'aboutcode': ('https://aboutcode.readthedocs.io/en/latest/',␣
↪None)}
```

To show all Intersphinx links and their targets of an Intersphinx mapping file, run:

```
python -msphinx.ext.intersphinx https://aboutcode.readthedocs.io/en/latest/objects.inv
```

Warning: `python -msphinx.ext.intersphinx https://aboutcode.readthedocs.io/objects.inv` will give error.

This enables you to create links to the `aboutcode` Documentation in your own Documentation, where you modified the configuration file. Links can be added like this:

```
For more details refer :ref:`aboutcode:doc_style_guide`.
```

You can also not use the `aboutcode` label assigned to all links from aboutcode.readthedocs.io, if you don't have a label having the same name in your Sphinx Documentation. Example:

```
For more details refer :ref:`doc_style_guide`.
```

If you have a label in your documentation which is also present in the documentation linked by Intersphinx, and you link to that label, it will create a link to the local label.

For more information, refer this tutorial named Using Intersphinx.

2.6 Style Conventions for the Documentaion

1. Headings

   (Refer) Normally, there are no heading levels assigned to certain characters as the structure is determined from the succession of headings. However, this convention is used in Python's Style Guide for documenting which you may follow:

   # with overline, for parts

   - with overline, for chapters

   =, for sections

   -, for subsections

   ^, for sub-subsections

   “, for paragraphs

2. Heading Underlines

   Do not use underlines that are longer/shorter than the title headline itself. As in:

   ```
   Correct :

   Extra Style Checks
   ------------------

   Incorrect :

   Extra Style Checks
   ------------------------
   ```

Note: Underlines shorter than the Title text generates Errors on sphinx-build.

3. Internal Links

   Using `:ref:` is advised over standard reStructuredText links to sections (like `` `Section title`_ ``) because it works across files, when section headings are changed, will raise warnings if incorrect, and works for all builders that support cross-references. However, external links are created by using the standard `` `Section title`_ `` method.

4. Eliminate Redundancy

   If a section/file has to be repeated somewhere else, do not write the exact same section/file twice. Use `.. include:  ../README.rst` instead. Here, `../` refers to the documentation root, so file location can be used accordingly. This enables us to link documents from other upstream folders.

5. Using `:ref:` only when necessary

   Use `:ref:` to create internal links only when needed, i.e. it is referenced somewhere. Do not create references for all the sections and then only reference some of them, because this created unnecessary references. This also generates ERROR in `restructuredtext-lint`.

6. Spelling

   You should check for spelling errors before you push changes. Aspell is a GNU project Command Line tool you can use for this purpose. Download and install Aspell, then execute `aspell check <file-name>` for all the files changed. Be careful about not changing commands or other stuff as

Aspell gives prompts for a lot of them. Also delete the temporary `.bak` files generated. Refer the manual for more information on how to use.

7. Notes and Warning Snippets

   Every `Note` and `Warning` sections are to be kept in `rst_snippets/note_snippets/` and `rst_snippets/warning_snippets/` and then included to eliminate redundancy, as these are frequently used in multiple files.

2.7 Converting from Markdown

If you want to convert a `.md` file to a `.rst` file, this tool does it pretty well. You'd still have to clean up and check for errors as this contains a lot of bugs. But this is definitely better than converting everything by yourself.

This will be helpful in converting GitHub wiki's (Markdown Files) to reStructuredtext files for Sphinx/ReadTheDocs hosting.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search