Reqompyler Documentation

Release 0.1.0

Mar 25, 2020

CONTENTS:

1 Reqompyler **1**
- 1.1 Table of Contents . . . 1
- 1.2 Description . . . 1
- 1.3 Installation . . . 2
- 1.4 Usage . . . 2
- 1.5 Credits . . . 2

2 Installation **3**
- 2.1 Stable release . . . 3
- 2.2 From sources . . . 3

3 Usage **5**
- 3.1 reqompyler . . . 5

4 Contributing **7**
- 4.1 Types of Contributions . . . 7
- 4.2 Get Started! . . . 8
- 4.3 Pull Request Guidelines . . . 9
- 4.4 Tips . . . 9
- 4.5 Deploying . . . 9

5 API Reference **11**
- 5.1 reqompyler . . . 11

6 Dependencies Graph **13**

7 Credits **15**
- 7.1 Development Lead . . . 15
- 7.2 Contributors . . . 15

8 Indices and tables **17**

Python Module Index **19**

Index **21**

CHAPTER

ONE

REQOMPYLER

1.1 Table of Contents

- *Table of Contents*
- *Description*
- *Installation*
- *Usage*
- *Credits*

1.2 Description

Use pip-tools to compile a requirements.in folder into proper pinned dependencies.

1.3 Installation

```
pip install reqompyler
```

1.4 Usage

```
$ reqompyler --help

Usage: reqompyler [OPTIONS]

  Console script for reqompyler.

Options:
  -i, --in PATH      Path to requirements.in folder  [default:
                     ./requirements.in]
  -o, --output PATH  Path to pinned requirements folder  [default:
                     ./requirements]
  --tld PATH         Ideally the top-level-directory of the package (ie, were
                     there is a setup.py). If not None will be used to export
                     a copy of your dev.txt as requirements.txt  [default: .]
  --ignore TEXT      Requirements.in file (without extension) to ignore,
                     if they are required by another they will be used but no␣
↪dedicated
                     .txt file will be generated.  [default: linting]
  --help             Show this message and exit.
```

1.5 Credits

This package was created with Cookiecutter and the zurutech/cookie-monster project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install Reqompyler, run this command in your terminal:

```
$ pip install reqompyler
```

This is the preferred method to install Reqompyler, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The source for Reqompyler can be downloaded from the GitHub repo.

You can either clone the public repository:

```
$ git clone https://github.com/zurutech/reqompyler.git
```

Once you have a copy of the source, you can install it with:

```
$ cd reqompyler
$ pip install .
```

CHAPTER

THREE

USAGE

To use `reqompyler` as a CLI tool:

```
$ reqompyler --help

Usage: reqompyler [OPTIONS]

  Console script for reqompyler.

Options:
  -i, --in PATH      Path to requirements.in folder  [default:
                     ./requirements.in]
  -o, --output PATH  Path to pinned requirements folder  [default:
                     ./requirements]
  --tld PATH         Ideally the top-level-directory of the package (ie, were
                     there is a setup.py). If not None will be used to export
                     a copy of your dev.txt as requirements.txt  [default: .]
  --ignore TEXT      Requirements.in file (without extension) to ignore,
                     if they are required by another they will be used but no dedicated
                     .txt file will be generated.  [default: linting]
  --help             Show this message and exit.
```

3.1 reqompyler

Console script for reqompyler.

```
reqompyler [OPTIONS]
```

Options

-i, --in <req_in>
Path to requirements.in folder [default: ./requirements.in]

-o, --output <req_pinned>
Path to pinned requirements folder [default: ./requirements]

--tld <tld>
Ideally the top-level-directory of the package (ie, were there is a setup.py). If not None will be used to export a copy of your dev.txt as requirements.txt [default: .]

--ignore <ignore>
Requirements.in file (without extension) to ignore,if they are required by another they will be used but no dedicate.txt file will be generated. [default: linting]

CHAPTER

FOUR

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/zurutech/reqompyler/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitLab issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitLab issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

Reqompyler could always use more documentation, whether as part of the official Reqompyler docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/zurutech/reqompyler/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up `reqompyler` for local development.

1. Clone `reqompyler` from the GitHub repository:

```
$ git clone https://github.com/zurutech/reqompyler.git
```

2. Install your local copy into a virtualenv. Assuming you have `virtualenvwrapper` installed, this is how you set up your fork for local development:

```
$ mkvirtualenv reqompyler
$ cd reqompyler/
$ python setup.py develop
```

3. To get the dev toolchain just pip install the provided requirements into your virtualenv.

```
$ pip install -r requirements/dev.txt
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8, pylint, black and the tests, including testing other Python versions with tox:

   - Automatically run the pipeline with `tox`.

```
   $ tox

**Note:** ``tox`` can be run parallely with ``tox -p auto -o``
```

   - Manually run them:

```
$ pytest -x -s -vvv --doctest-modules reqompyler tests --cov=reqompyler
$ black reqompyler tests
$ isort reqompyler tests
$ flake8 reqompyler tests
$ pylint reqompyler tests
```

6. Commit your changes and push your branch to GitLab:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitLab website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.md.
3. The pull request should work for the specified Python Versions.
4. If you have made change to the CI Pipeline test them locally

4.4 Tips

To run a subset of tests:

```
$ pytest -x -s -vvv --doctest-modules WHAT_MODULE/TEST_SUBSET_TO_TEST --cov=ashpy
```

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.md). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

FIVE

API REFERENCE

reqompyler	Top-level package for Reqompyler.

5.1 reqompyler

Top-level package for Reqompyler.

Functions

reqcompyle	Compile requirements files using pip-tools.

5.1.1 reqompyler.reqcompyle

reqompyler.**reqcompyle**(*in_folder*, *out_folder*, *tld*, *ignore*)

Compile requirements files using pip-tools.

Parameters

- **in_folder** (pathlib.Path) – Path to the folder with your requirements.in files.
- **out_folder** (pathlib.Path) – Path to the folder were the pinned requirements file will be saved.
- **tld** (pathlib.Path) – Top level directory of your package, if passed, copies pinned the `dev.txt` as `requirements.txt` to this location.
- **ignore** (list of [pathlib.Path]) – Array of requirements files (without extension) to ignore.

Return type None

CHAPTER

SIX

DEPENDENCIES GRAPH

CHAPTER

SEVEN

CREDITS

7.1 Development Lead

- AI Lab ml@zuru.tech

7.2 Contributors

None yet. Why not be the first?

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

reqompyler, 11

INDEX

Symbols

--ignore <ignore>
    reqompyler command line option, 5
--in <req_in>
    reqompyler command line option, 5
--output <req_pinned>
    reqompyler command line option, 5
--tld <tld>
    reqompyler command line option, 5
-i
    reqompyler command line option, 5
-o
    reqompyler command line option, 5

R

reqcompyle() (*in module reqompyler*), 11
reqompyler (*module*), 11
reqompyler command line option
    --ignore <ignore>, 5
    --in <req_in>, 5
    --output <req_pinned>, 5
    --tld <tld>, 5
    -i, 5
    -o, 5