RequirementsLib Documentation

Release 3.0.0

Dan Ryan <dan@danryan.co>

June 18, 2023

Contents:

CHAPTER 1

RequirementsLib: Requirement Management Library for Pip and Pipenv

RequirementsLib: Requirement Management Library for Pip and Pipenv

1.1 Installation

Install from PyPI:

```
$ pipenv install requirementslib
```

Install from Github:

```
$ pipenv install -e git+https://github.com/sarugaku/requirementslib.git
↪#egg=requirementslib
```

1.2 Summary

RequirementsLib provides a simple layer for building and interacting with requirements in both the Pipfile format and the requirements.txt format. This library was originally built for converting between these formats in Pipenv.

1.3 Usage

1.3.1 Importing a lockfile into your *setup.py* file

You can use RequirementsLib to import your lockfile into your setup file for including your **install_requires** dependencies:

```
from requirementslib import Lockfile
lockfile = Lockfile.create('/path/to/project/dir')
install_requires = lockfile.as_requirements(dev=False)
```

1.3.2 Interacting with a *Pipfile* directly

You can also interact directly with a Pipfile:

```
>>> from requirementslib import Pipfile
>>> pf = Pipfile.load('/home/hawk/git/pypa-pipenv')
>>> pf.sections
[Section(name='packages', requirements=[]), Section(name='dev-packages',␣
↪requirements=[Requirement(name='pipenv', vcs=None, req=FileRequirement(setup_
↪path=None, path='.', editable=True, uri='file:///home/hawk/git/pypa-pipenv', link=
↪<Link file:///home/hawk/git/pypa-pipenv>, name='pipenv', req=<Requirement: "-e␣
↪file:///home/hawk/git/pypa-pipenv">), markers='', specifiers=None, index=None,␣
↪editable=True, hashes=[], extras=None),...]
```

And you can even write it back out into Pipfile's native format:

```
>>> print(pf.dump(to_dict=False))
[packages]

[dev-packages]
pipenv = {path = ".", editable = true}
flake8 = ">=3.3.0,<4"
pytest = "*"
mock = "*"

[scripts]
tests = "bash ./run-tests.sh"

[pipenv]
allow_prereleases = true
```

1.3.3 Create a requirement object from *requirements.txt* format

```
>>> from requirementslib import Requirement
>>> r = Requirement.from_line('-e git+https://github.com/pypa/pipenv.git@master
↪#egg=pipenv')
>>> print(r)
Requirement(name='pipenv', vcs='git', req=VCSRequirement(editable=True, uri=
↪'git+https://github.com/pypa/pipenv.git', path=None, vcs='git', ref='master',␣
↪subdirectory=None, name='pipenv', link=<Link git+https://github.com/pypa/pipenv.
↪git@master#egg=pipenv>, req=<Requirement: "-e git+https://github.com/pypa/pipenv.
↪git@master#egg=pipenv">), markers=None, specifiers=None, index=None, editable=True,␣
↪hashes=[], extras=[])
```

(continues on next page)

(continued from previous page)

```
>>> r.as_pipfile()
{'pipenv': {'editable': True, 'ref': 'master', 'git': 'https://github.com/pypa/pipenv.
↪git'}}
```

Or move from *Pipfile* format to *requirements.txt*:

```
>>> r = Requirement.from_pipfile(name='pythonfinder', indexes=[], pipfile={'path': '..
↪/pythonfinder', 'editable': True})
>>> r.as_line()
'-e ../pythonfinder'
```

1.3.4 Resolving Editable Package Dependencies

Requirementslib also can resolve the dependencies of editable packages by calling the `run_requires` method. This method returns a detailed dictionary containing metadata parsed from the package built in a transient folder (unless it is already on the system or the call is run in a virtualenv).

The output of `run_requires` is very detailed and in most cases will be sufficient:

```
>>> from pprint import pprint
>>> from requirementslib.models.requirements import Requirement
>>> r = Requirement.from_line("-e git+git@github.com:sarugaku/vistir.git
↪#egg=vistir[spinner]")
>>> setup_info_dict = r.run_requires()
>>> from pprint import pprint
>>> pprint(setup_info_dict)
{'base_dir': '/tmp/requirementslib-t_ftl6no-src/src/vistir',
'build_backend': 'setuptools.build_meta',
'build_requires': ['setuptools>=36.2.2', 'wheel>=0.28.0'],
'extra_kwargs': {'build_dir': '/tmp/requirementslib-t_ftl6no-src/src',
                 'download_dir': '/home/hawk/.cache/pipenv/pkgs',
                 'src_dir': '/tmp/requirementslib-t_ftl6no-src/src',
                 'wheel_download_dir': '/home/hawk/.cache/pipenv/wheels'},
'extras': {'spinner': [Requirement.parse('cursor'),
                       Requirement.parse('yaspin')],
           'tests': [Requirement.parse('pytest'),
                     Requirement.parse('pytest-xdist'),
                     Requirement.parse('pytest-cov'),
                     Requirement.parse('pytest-timeout'),
                     Requirement.parse('hypothesis-fspaths'),
                     Requirement.parse('hypothesis')]},
'ireq': <InstallRequirement object: vistir[spinner] from git+ssh://git@github.com/
↪sarugaku/vistir.git#egg=vistir editable=True>,
'name': 'vistir',
'pyproject': PosixPath('/tmp/requirementslib-t_ftl6no-src/src/vistir/pyproject.toml'),
'python_requires': '>=2.6,!=3.0,!=3.1,!=3.2,!=3.3',
'requires': {'backports.functools_lru_cache;python_version<="3.4"': Requirement.parse(
↪'backports.functools_lru_cache; python_version <= "3.4"'),
             'backports.shutil_get_terminal_size;python_version<"3.3"': Requirement.
↪parse('backports.shutil_get_terminal_size; python_version < "3.3"'),
             'backports.weakref;python_version<"3.3"': Requirement.parse('backports.
↪weakref; python_version < "3.3"'),
             'colorama': Requirement.parse('colorama'),
             'pathlib2;python_version<"3.5"': Requirement.parse('pathlib2; python_
↪version < "3.5"'),
```

(continues on next page)

(continued from previous page)

```
            'requests': Requirement.parse('requests'),
            'six': Requirement.parse('six'),
            'spinner': [Requirement.parse('cursor'),
                        Requirement.parse('yaspin')]},
'setup_cfg': PosixPath('/tmp/requirementslib-t_ftl6no-src/src/vistir/setup.cfg'),
'setup_py': PosixPath('/tmp/requirementslib-t_ftl6no-src/src/vistir/setup.py')}
```

As a side-effect of calls to `run_requires`, new metadata is made available on the requirement itself via the property `requirement.req.dependencies`:

```
>>> pprint(r.req.dependencies)
({'backports.functools_lru_cache;python_version<="3.4"': Requirement.parse('backports.
↪functools_lru_cache; python_version <= "3.4"'),
'backports.shutil_get_terminal_size;python_version<"3.3"': Requirement.parse(
↪'backports.shutil_get_terminal_size; python_version < "3.3"'),
'backports.weakref;python_version<"3.3"': Requirement.parse('backports.weakref;␣
↪python_version < "3.3"'),
'colorama': Requirement.parse('colorama'),
'pathlib2;python_version<"3.5"': Requirement.parse('pathlib2; python_version < "3.5"
↪'),
'requests': Requirement.parse('requests'),
'six': Requirement.parse('six'),
'spinner': [Requirement.parse('cursor'), Requirement.parse('yaspin')]},
[],
['setuptools>=36.2.2', 'wheel>=0.28.0'])
```

1.4 Integrations

- Pip
- Pipenv
- Pipfile

1.5 Contributing

1. Fork the repository and clone the fork to your local machine: `git clone git@github.com:yourusername/requirementslib.git`
2. Move into the repository directory and update the submodules: `git submodule update --init --recursive`
3. Install the package locally in a virtualenv using pipenv: `pipenv install --dev`
    a. You can also install the package into a virtualenv by running `pip install -e .[dev,tests,typing]` to ensure all the development and test dependencies are installed
4. Before making any changes to the code, make sure to file an issue. The best way to ensure a smooth collaboration is to communicate *before* investing significant time and energy into any changes! Make sure to consider not just your own use case but others who might be using the library
5. Create a new branch. For bugs, you can simply branch to `bugfix/<issuenumber>`. Features can be branched to `feature/<issuenumber>`. This convention is to streamline the branching process and to

encourage good practices around filing issues and associating pull requests with specific issues. If you find yourself addressing many issues in one pull request, that should give you pause

6. Make your desired changes. Don't forget to add additional tests to account for your new code – continuous integration **will** fail without it
7. Test your changes by running `pipenv run pytest -ra tests` or simply `pytest -ra tests` if you are inside an activated virtual environment
8. Create a corresponding `.rst` file in the `news` directory with a one sentence description of your change, e.g. `Resolved an issue which sometimes prevented requirements from being converted from Pipfile entries to pip lines correctly`
9. Commit your changes. The first line of your commit should be a summary of your changes, no longer than 72 characters, followed by a blank line, followed by a bulleted description of your changes. Don't forget to add separate lines with the phrase `- Fixes #<issuenumber>` for each issue you are addressing in your pull request
10. Before submitting your pull request, make sure to `git remote add upstream git@github.com:sarugaku/requirementslib.git` and then `git fetch upstream && git pull upstream master` to ensure your code is in sync with the latest version of the master branch,
11. Create a pull request describing your fix, referencing the issues in question. If your commit message from step 8 was detailed, you should be able to copy and paste it

CHAPTER 2

requirementslib package

2.1 Submodules

2.1.1 requirementslib.models package

2.1.1.1 Submodules

requirementslib.models.cache module

requirementslib.models.dependencies module

requirementslib.models.lockfile module

requirementslib.models.markers module

requirementslib.models.metadata module

requirementslib.models.pipfile module

requirementslib.models.project module

requirementslib.models.requirements module

requirementslib.models.resolvers module

requirementslib.models.setup_info module

2.1.2 requirementslib.utils module

CHAPTER 3

Indices and tables

- genindex
- modindex
- search