Setupext: Pip

Release 1.0.5

December 21, 2015

Contents

1 Wait... Why? What?? 3

2 Ok... Where? 5

2.1 Using the Setuptools Command 5

2.2 Using Requirement Files 6

2.3 Changelog 7

A setuptools extension that installs requirements based on pip requirements files.

CHAPTER 1

Wait... Why? What??

Installing runtime dependencies is already handled by setuptools right? It's as easy as running `python setup.py develop`?! Not quite...

setup.py is a great tool and the *develop* command is useful for setting up basic development and installing the dependencies identified by the `setup_requires` keyword. What it does not do is install any tools that are used for actually hacking on the code base. Many projects include a pip-formatted requirements file named *requirements.txt* for this very purposes. It contains the dependencies that you need to have installed to work on the project instead of use it. This extension aims to automate that pattern and make it easier to set up a development environment by providing a new setup command named *requirements*.

Having a separate *requirements.txt* is a good pattern but it is not without its flaws. Having dependencies identified in two places is an outright violation of the *Don't Repeat Yourself* principle. That is something else that we can solve pretty easily with a function that you can read a pip-formatted requirements file and generate a list that is usable as the setup `setup_requires`, `install_requires`, or `tests_require` keywords. This is where the `setupext.pip.read_requirements_from_file` function comes in. You can use this function to populate the `setup_requires`, `tests_require`, and `extras_require` keywords.

CHAPTER 2

Ok... Where?

Source	https://github.com/dave-shawley/setupext-pip
Status	https://travis-ci.org/dave-shawley/setupext-pip
Download	https://pypi.python.org/pypi/setupext-pip
Documentation	http://setupext-pip.readthedocs.org/en/latest/
Issues	https://github.com/dave-shawley/setupext-pip

2.1 Using the Setuptools Command

This library extends the setuptools utility by defining a single new command called **requirements**. This command installs your package's dependencies identified by the `setup_requires` keyword passed to `setuptools.setup()` without installing the project in development mode.

```
$ virtualenv -q env
$ env/bin/python setup.py requirements
```

The *requirements* command supports a number of command line options that are passed through to the underlying `pip install` execution.

`--index-url` `URL`
Use *URL* as the Python Package Index instead of the default (https://pypi.python.org/simple).

`--find-links` `URL`
Fetch additional packages by extracting the links from *URL*. If *URL* refers to a directory (via a `file://` URL), then the contents of the directory are used.

`--no-use-wheel`
Do not find or prefer wheel archives when searching indexes and find-links locations.

`--pre`
Include pre-release and development versions. By default, only stable versions are installed.

`--install-test-requirements`
Install dependencies listed in the `tests_require` keyword passed to `setuptools.setup()`.

`--install-extra-requirements` `EXTRA`
Install the `extras_require` dependencies associated with *EXTRA*.

Since the *requirements* command uses **pip** to perform the installation, you can use any of the pip configuration files.

Warning: Import Usage Note
Now there is a very important problem with taking this approach – **setupext has to be installed BEFORE setup.py runs**. This means that you cannot depend on `setup_requires` to install the extension. This is very unfortunate and made me consider not releasing this helper at all.

2.2 Using Requirement Files

`setuptools.setup` takes three keyword parameters that control which requirements are installed and when. Here's what the setuptools documentation has to say about these parameters.

`setup_requires`

> A string or list of strings specifying what other distributions need to be present in order for the setup script to run. `setuptools` will attempt to obtain these (even going so far as to download them using `EasyInstall`) before processing the rest of the setup script or commands. This argument is needed if you are using distutils extensions as part of your build process; for example, extensions that process `setup()` arguments and turn them into EGG-INFO metadata files.
>
> (Note: projects listed in `setup_requires` will NOT be automatically installed on the system where the setup script is being run. They are simply downloaded to the setup directory if they're not locally available already. If you want them to be installed, as well as being available when the setup script is run, you should add them to `install_requires` and `setup_requires`.)

`tests_require`

> If your project's tests need one or more additional packages besides those needed to install it, you can use this option to specify them. It should be a string or list of strings specifying what other distributions need to be present for the package's tests to run. When you run the test command, `setuptools` will attempt to obtain these (even going so far as to download them using EasyInstall). Note that these required projects will not be installed on the system where the tests are run, but only downloaded to the project's setup directory if they're not already installed locally.

`extras_require`

> A dictionary mapping names of "extras" (optional features of your project) to strings or lists of strings specifying what other distributions must be installed to support those features. See the section below on Declaring Dependencies for details and examples of the format of this argument.

You can use `read_requirements_from_file()` to read each set of dependencies from a separate file. This practice makes it possible to organize simple or complex dependencies in a DRY manner.

```
import setuptools
from setupext import pip

setuptools.setup(
    name='my-package',
    # ...
    setup_requires=pip.read_requirements_from_file('requirements.txt'),
    tests_require=pip.read_requirements_from_file('dev-requirements.txt'),
)
```

Now back to the ominous warning above. If you can't guarantee that the extension is installed into the environment that your package is installed into, then you can roll your own implementation of reading the requirements file. I've used the following snippet to work through this problem with, admittedly, simple requirements files.

```
try:
    from setupext.pip import read_requirements_from_file
except ImportError:
    def read_requirements_from_file(req_name):
        with open(req_name, 'r') as req_file:
            return [
                line[0:line.find('#')] if '#' in line else line.strip()
                for line in req_file
            ]
```

It's no where near perfect, but it does work in most cases.

`setupext.pip.`**`read_requirements_from_file`**(*file_name*)

Read requirements from a pip-formatted requirements file.

Parameters **file_name** (*str*) – the name of the file to read from

Returns a `list` of requirements as strings

This function reads the specified file, processes it as the **pip** utility would, and returns the list of dependency specifiers. Each line of the requirements file is parsed using the functionality provided by pkg_resources so even the hairiest dependency specifiers will be parsed correctly. However, all command line parameter overrides (e.g., `--index-file=...`) are ignored.

2.3 Changelog

- 1.0.5 (15-Aug-2014)
  - Add the *–install-test-requirements* command line option.
  - Add the *–install-extra-requirements* command line option.
- 1.0.4 (02-Aug-2014)
  - Add `from future import absolute_import` to make the extension safe on older Python versions.
  - Make it safe to use without a `setup_requires` keyword.
- 1.0.3 (30-Jul-2014)
  - Fix `setup.py test`. It was hanging forever unless you passed it additional parameters.
- 1.0.2 (30-Jul-2014)
  - Added `read_requirements_from_file` function.
- 1.0.1 (27-Jul-2014)
  - Worked around a problem with py.test output capturing and Python 3.x. It looks like they are patching `sys.stdout` and friends with something that does not have a `errors` attribute so *distutils/log.py* is blowing up on line 30: `if stream.errors == 'strict'`.

    For some reason, switching to `--capture=sys` seems to work correctly. I have a feeling that this will be fixed when py.test 2.6 shows up.
- 1.0.0 (27-Jul-2014)
  - Initial revision that supported the `requirements` command.

Symbols

–find-links URL
    command line option, 5
–index-url URL
    command line option, 5
–install-extra-requirements EXTRA
    command line option, 5
–install-test-requirements
    command line option, 5
–no-use-wheel
    command line option, 5
–pre
    command line option, 5

C

command line option
    –find-links URL, 5
    –index-url URL, 5
    –install-extra-requirements EXTRA, 5
    –install-test-requirements, 5
    –no-use-wheel, 5
    –pre, 5

R

read_requirements_from_file() (in module setupext.pip), 7