pip2 Documentation

Release 0.0.1.dev1

Pip2 Authors

December 30, 2015

CHAPTER 1

Developer Guide

1.1 Developer's Guide

Table of contents:

1.1.1 Contributing to Pip2

Contributing with Code

Supported Python Versions

Pip2 currently only supports Python 3.2.

Prerequisites

The following tools are required:

- Python 3.2
- Git
- pip and virtualenv

You must also have a Github account and basic familiarity with the tools listed above.

These instructions assume a Unix-like operating system (e.g., Mac or Linux). Minor modifications may be required for contributing to pip2 on Windows.

Forking the Repository

Fork the main pip2 repository on Github, and then clone your personal fork:

```
$ git clone https://github.com/<YOUR_USER_NAME>/pip2
```

Installation

Create and activate a virtualenv for pip2 development. For example:

```
$ virtualenv --python=python3.2 pip2-dev
$ source pip2-dev/bin/activate
```

Pip2 depends on Distutils2 which currently doesn't have a version for Python 3 on PyPI (see issue #45). For now, just use pip to install from the `python3` branch of the Distutils2 repository:

```
$ pip install http://hg.python.org/distutils2/archive/python3.tar.bz2
```

Install pip2:

```
$ cd pip2/
$ python setup.py develop
```

Running the Tests

Pip2 uses nose and mock for testing. To install nose:

```
$ pip install nose
```

Mock has been included in Python's standard library since version 3.3. For versions of Python prior to 3.3:

```
$ pip install mock
```

Now, run the unit tests from the root directory of the pip2 repository. You should run these tests frequently as you are modifying the code:

```
$ nosetests
```

If the coverage module is installed (*pip install coverage*), options may be provided to nose so that coverage data is generated:

```
$ nosetests --with-coverage
```

Usually only coverage data for pip2 will be needed. To run the coverage tool on just the pip2 package:

```
$ nosetests --with-coverage --cover-package=pip2
```

To generate HTML coverage data in the ./cover/ directory:

```
$ nosetests --with-coverage --cover-package=pip2 --cover-html
```

Once your changes are working well in your development environment, tox can be used to run these same tests in a clean environment under multiple versions of Python. First, install tox:

```
$ pip install tox
```

The first time you run it, tox will take a while (quite a few minutes) to build virtualenvs and install the required packages:

```
$ tox
```

Subsequent tox runs will reuse the existing virtualenvs and run much faster. Note, however, that you may want to occasionally force the virtualenvs to be recreated by running *tox –recreate* to get the latest versions of pip2's dependencies. Run *tox –help*, visit tox's website, or view the *tox.ini* file in pip2's repository for additional information on using tox.

Contributing with Documentation

Building the Documentation

Install the tools required to build the documentation:

```
$ pip install sphinx
```

Build the HTML version of the documentation:

```
$ cd pip2/docs/
$ make html
```

Launch `pip2/docs/_build/html/index.html` in your browser.

1.1.2 API Reference

Freeze

`pip2.commands.freeze.`**`freeze`**`()`

Get a list of installed projects.

Returns a dictionary where the keys are project names and the values are dictionaries with the following keys and values:

- *'version'* - a string containing the project's version.

For example, the return value may look like this:

```
    {   'TowelStuff': {'version': '0.1.1'},
        'pip2': {'version': '1.0'}}
```

Return type dictionary

Install

`pip2.commands.install.`**`install`**`(`*project_list*`)`

Install a list of projects.

Note that project dependencies are not yet detected and installed.

Returns a dictionary with the following keys and values:

- *'installed'* - a list of strings containing the projects that were successfully installed.
- *'failed'* - a list of strings containing the projects that failed to install.

Parameters **`project_list`** (*iterable of strings*) – the projects to install. May be names of projects on the Python Package Index (PyPI) or paths to local directories and archives (.zip, .tar.gz, .tar.bz2, .tgz, or .tar).

Return type dictionary

Search

pip2.commands.search.**search**(*query*)

Search projects on the Python Package Index (PyPI).

Searches the *name* and *summary* fields for projects that match *query*.

Returns a dictionary containing the search results. The keys are project names and the values are dictionaries with the following keys and values:

- *'summary'* - a string containing the project's summary.
- *'installed_version'* (only present if the project is installed) - a string containing the version of the project currently installed.
- *'latest_version'* (only present if the project is installed) - a string containing the latest version of the project available on the index.

For example, the return value may look like this:

```
{   'TowelStuff': {   'installed_version': '0.1.1',
                      'latest_version': '0.1.1',
                      'summary': 'Useful towel-related stuff.'},
    'towel': {'summary': 'Keeping you DRY since 2010'}}
```

Parameters **query** (*string*) – the search query.

Return type dictionary

Uninstall

pip2.commands.uninstall.**uninstall**(*project_list*)

Uninstall a list of projects.

Returns a dictionary with the following keys and values:

- *'uninstalled'* - a list of strings containing the names of the projects that were successfully uninstalled.
- *'failed'* - a list of strings containing the names of the projects that failed to uninstall.

Parameters **project_list** (*iterable of strings*) – the names of the projects to uninstall.

Return type dictionary