appypi Documentation

Release 0.2.0

Stéphane Péchard

October 17, 2013

CONTENTS

appypi is a terminal-based Python Package Index package manager. Each app installed through appypi is sandboxed in an individual virtualenv and usable from the user space. No root access is required to install an app. appypi in **NOT** a replacement for pip or easy_install. Actually, it uses pip intensively to manage package installations.

appypi creates launchers into the user ~/bin directory. These launchers mimic the package behavior in terms of what commands can be called. For example, appypi will create a *django-admin.py* launcher when installing the django package, and a *fab* launcher when installing the Fabric package. This way, you can use the package as if it was installed with pip directly.

appypi won't install a package if it finds it in your path already. It is your duty to take care of these external installations before using appypi.

Installing a package:

```
$ appypi install django
Looking for django...
Found Django version 1.4.2
Installing...
Install successful!
```

Upgrading a package:

```
$ appypi upgrade django
Upgrading: Django (1.3 => 1.4.2)
```

Removing a package:

```
$ appypi remove django
These packages will be REMOVED:

    Django

Do you want to continue? [y/n]y
Package Django has been removed.
```

CHAPTER

ONE

INSTALLATION INSTRUCTIONS

It is usually suggested to install a package like appypi in a virtualenv. It is still possible here, but it is recommended to use the *meta-style* installation process.

1.1 Meta-style

As appypi is an app installer, you can use it to install itself. This way, your installation is consistent and you keep everything in the dedicated database. Here is a little script to do this in one shot. You need at least pip version 0.8.1 to do that:

```
virtualenv -q /tmp/appypi-venv &&
source /tmp/appypi-venv/bin/activate &&
pip -q install appypi &&
appypi install appypi &&
deactivate
```

After that, you can begin to *use appypi*, as it is callable in your PATH.

1.2 Usual suspects: pip and distribute

To install appypi for your user only, type:

```
$ pip install appypi --user
```

To install appypi system-wide, just type:

```
$ sudo pip install appypi
```

If no pip available, try `easy_install`:

```
$ sudo easy_install appypi
```

1.3 Play the game

If you want to code, hack, enhance or just understand appypi, you can get the latest code at Github:

```
$ git clone http://github.com/stephanepechard/appypi
```

Then create the local virtualenv and install appypi:

```
$ cd appypi && source bootstrap && fab install
```

CHAPTER

TWO

USAGE

As a simple package manager, Appypi is used to install, remove, upgrade Pypi packages you may need. Here are the ways to do it.

2.1 Install a package

Easy peasy:

```
$ appypi install <package>
```

It may be a bit long the first time, as it constructs the package list. You can install multiple packages in one line, each one will be installed in an independant virtualenv:

```
$ appypi install <package1> <package2> ...
```

Of course, if you try to install a package that is not on Pypi, appypi will gently decline your proposition. Same thing, if you try to install a package that don't have any launcher to be used with, like in a library, appypi will stop the installation. Unhappily, there is no way to know it before going far into the installation process.

2.2 Install a list of package

As with pip, you can use a requirements file to install many packages at once:

```
$ appypi install --requirements=requirements.txt
```

But contrary to pip, you can not specify the version of a package like `django==1.4`. You can not install stuff from git or any other VCS as well. Maybe one day...

Note that each package will be installed in an independant virtualenv. They can't be installed in conjunction in a unique one.

2.3 Remove a package

Sad, but true:

```
$ appypi remove <package>
```

A confirmation for deletion is asked to the user. Again, you can remove multiple packages at the same time:

```
$ appypi remove <package1> <package2> ...
```

2.4 Upgrade a package

You guessed right:

```
$ appypi upgrade <package>
```

The given package is upgraded to the last version available on Pypi.

2.5 Upgrade all installed packages

Yes, it's easy:

```
$ appypi upgrade
```

All packages are upgraded to their last versions available on Pypi.

2.6 Show information about a package

```
$ appypi show <package>
```

This prints details found on Pypi about the installed version of the package. The full description is printed, so it may be a bit verbose. For example, for the Projy package:

```
Package: Projy
Version: 0.2
Author: Stéphane Péchard
Homepage: https://github.com/stephanepechard/projy
Summary: Projy is a template-based skeleton generator.

Description: Projy
=====
**Projy is a template-based skeleton generator**.
In one command line, you can generate project skeletons
like Python packages, LaTeX document or any file structure
composed of directories and files.

Each file is generated by a different template.
It uses the simple core templating system from Python,
nothing fancy on that part. You can easily add new templates
and new ways to collect data to insert in the created files.
As much as possible, Projy tries to be simple to use and extend.

See [the complete documentation](http://projy.readthedocs.org/).


Installation
============
If you are familiar with Python, it is strongly suggested that you
install Projy in [virtualenv](http://pypi.python.org/pypi/virtualenv).
```

```
Pip and Distribute
------------------
If you are on Linux or Mac OS X, just type:

    $ sudo pip install projy

If no pip available, try ``easy_install``:

    $ sudo easy_install projy


Install from git
----------------
If you prefer git, that is ok too. You can get the very latest code at
Github:

    $ git clone http://github.com/stephanepechard/projy

Usage
=====
As an example, let’s create a Python package. The Projy template mostly
follows recommendations from
[The Hitchhiker’s Guide to Packaging](http://guide.python-distribute.org/).


A simple example
----------------
Use simply:

    $ projy PythonPackage TowelStuff

In the same directory as you typed this command, you now have a
*TowelStuff* directory, with the following structure:


    TowelStuff/
        bin/
        bootstrap
        CHANGES.txt
        docs/
            index.rst
        LICENSE.txt
        MANIFEST.in
        README.txt
        setup.py
        towelstuff/
            __init__.py


Each file has been created with a specific template, so the package is
fully functional, yet empty. Now, let’s give a little explanation
on each file. You can find [further
information here](http://guide.python-distribute.org/creation.html).


Specify substitutions
---------------------
You can specify template substitutions directly from the command line.
```

```
Add them at the end of the previous example:

    $ projy PythonPackage TowelStuff "author,Emmett Brown" "date,1851-09-24"

Then the substitutes *author* and *date* (defaulted to the current day)
are defined by the given values, not those computed by Projy.
The format of such substitutions should be "key,value". Neither the
key or the value should therefore include a comma.
Leading and trailing spaces are removed from both key and value.

To know which substitutions can be overwritten this way, use the *-i*
option as described in the dedicated section. You can add substitutions
that are not listed with the *-i* but they won't have any effect if the
template file does not consider them.


Options
-------
Projy comes with some command line options. Type:

    $ projy -l

and you'll see the list of available templates in your installation.
That's an easy way to copy/paste the name of the template you want
to use next.

Type

    $ projy -i PythonPackage

and you'll see the details of the Python package template. It shows
the created directories and files, with the substitutions included in
the template.

Available templates
===================
The currently available templates are:

 * a [Fabric](http://fabfile.org) file ;
 * a [LaTeX](http://www.latex-project.org/) book ;
 * a [Python](http://python.org/) package ;
 * a Python script ;
 * a [Projy](https://github.com/stephanepechard/projy) template, meta-style.
 * a bootstrap file, to manage your virtualenv happiness ;

See the official doc for more details on created files into these
templates. Soon to come, more templates around Django. Of course,
anyone can propose some templates, they'll be integrated into Projy.
```

2.7 List installed packages

Obviously:

```
$ appypi list
```

The output shows the name, the version and the summary of every installed package. Here is an example output:

```
appypi - 4 installed packages
------------------------------
Name       - Version - Summary
------------------------------
Projy      - 0.2     - Projy is a template-based skeleton generator.
Fabric     - 1.5.0   - Fabric is a simple, Pythonic tool for remote execution and deployment.
subliminal - 0.6.2   - Subtitles, faster than your thoughts
Glances    - 1.5.1   - CLI curses-based monitoring tool
```

2.8 Update the local list of Pypi packages

You guessed right:

```
$ appypi update
```

Note that this update is forced programmatically every 7 days.

CHAPTER

THREE

SOME TECHNICAL DETAILS

3.1 Commands

You may have noticed that appypi commands were almost the same as in aptitude , the Debian packages manager. It is not done unpurposely. Actually, you can find the same *install*, *remove*, *update* and *upgrade* in aptitude. The *list* command is not available in aptitude but in dpkg The *show* command is not available in aptitude but in apt-get.

3.2 What does appypi create on your disk?

Except for the launchers, everything appypi will create on your disk is confined to the *.appypi* directory.

3.2.1 Global directory

That directory contains all files and directories appypi need to work properly:

- the appypi database (SQLite format) ;
- the Pypi packages list dump file ;
- the appypi cache directory ;
- for each installed app, a directory containing the *bootstrap* file, the current freeze state of the app virtualenv put in the *freeze.txt* and the virtualenv in itself put in the *venv* directory.

3.2.2 Launchers

For each app installed, any launchable script located in the package is translated by appypi as a launchable script. Any of these file is situated in the user *bin* directory. It creates it if it does not exist.

3.3 Caches

3.3.1 Pypi packages list

You don't want to query the list of **ALL** Pypi packages at the installation of any package. That's why appypi uses a cache of this Pypi list, to be fast at installing several packages in a short interval. This cache is written on disk in a single file, using Python's pickle functionnalitie

Of course, Pypi packages list grows over time, and appypi should keep it up to date. It is done automatically if the list file is more than 7 days old. At the moment, there is no way of modifying this duration, but it seems reasonable. If this does not suit you, you can manually trigger the update with a simple:

```
$ appypi update
```

3.3.2 Packages cache

appypi maintains its own local cache for Pypi packages. It may be redundant with your own pip cache if you set it already (and you should), but this way appypi is kept independent from pip. Then, each single package comes either directly from Pypi or from the appypi local cache.

The annoying thing is that if pip find a package in the cache, it asks the user what to do with it:

```
The file /home/user/.appypi/appypi_cache/glances-1.5.1.tar.gz exists. (i)gnore, (w)ipe, (b)ackup
```

There is no way to tell pip what to do programmatically before its version 1.3. For previous versions of pip, you have to manually choose **(i)gnore** anytime it is asked. Shitty, I know...

In case it is too disturbing for you, you can delete the full cache with:

```
$ rm -rf ~/.appypi/appypi_cache
```

3.4 Slowness?

As you can imagine, entering in a virtualenv for each command is a bit time-consuming. Your package may therefore be take a little more time to start than usually. It is therefore not recommended to use appypi with particularly time-sensitive usage. For any other use of common packages, appypi is the way to go :-)

CHAPTER

FOUR

CHANGELOG

4.1 0.2 (2012-12-16)

- add requirement file handling
- full pylint-compliant cleaning, not perfect but better

4.2 0.1.4 (2012-11-17)

- fix setup.py, MANIFEST.IN for README.txt

4.3 0.1.3 (2012-11-15)

- fix README file for official release

4.4 0.1.2 (2012-11-15)

- fix setup.py and bootstrap, better documentation

4.5 0.1.1 (2012-11-11)

- fix erroneous MANIFEST

4.6 0.1 (2012-11-10)

- Initial release