
Projy Documentation

Release 0.4.2-dev

Stéphane Péchard

Apr 11, 2017

Contents

1	Content	3
1.1	Installation	3
1.2	Usage	4
1.3	Available templates	7
1.4	Extending Projy	12
1.5	Changelog	16

Projy is a template-based skeleton generator. In one command line, you can generate project skeletons like Django projects, Python packages, LaTeX documents or any files 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.

Installation

If you are familiar with Python, it is strongly suggested that you install Projy in [virtualenv](#).

Pip and Distribute

To install Projy system-wide, just type:

```
$ sudo pip install projy
```

If no pip available, try `easy_install`:

```
$ sudo easy_install projy
```

Play the game

If you want to code, hack, enhance or just understand Projy, you can get the latest code at [Github](#):

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

Then create the local virtualenv and install Projy into it:

```
$ cd projy && make && ./venv/bin/fab install
```

There's also an easy:

```
$ ./venv/bin/fab reinstall
```

to reinstall the local version of Projy, for quick testing.

Read the full manual

To build the manual:

```
$ ./venv/bin/fab build_doc
```

The HTML pages are then in the `projy/docs/_build/html/` directory.

Usage

As an example, let's create some projects as the ones you usually work on.

A Python package example

First is a Python package. The Projy template mostly follows recommendations from [The Hitchhiker's Guide to Packaging](#). Use simply:

```
$ projy PythonPackage TowelStuff
```

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

```
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 component. You can find [further information here](#).

bin/, *docs/* and *towelstuff/* directories

Three directories are created by this project template:

- *bin/* contains your package's scripts ;
- *docs/*, contains the documentation you write for the package. A primary *index.rst* file waits for you to write into it. Yes, it uses [reStructuredText](#) format.
- *towelstuff/*, is where you put the files of your package. It is the lower case version of the project name. By default, it already contains an empty *__init__.py* file.

See the links for more information.

bootstrap

This file is a little treat, not present in [The Hitchhiker's Guide to Packaging](#). Using the `BootstrapScriptFileTemplate` template, it is a simple bash file creating a virtual environment easily. Use it with a simple:

```
$ source bootstrap
```

By default, it installs three packages from [pypi](#):

- [nose](#) is “nicer testing for Python” ;
- [pylint](#), a Python code static checker ;
- [sphinx](#), the Python documentation generator.

Everything you need to write quality code :-). Of course, you can add any other package you may need, it's up to you. You can even externalize this list of package to a [requirement file](#).

CHANGES.txt

The template of the `CHANGES.txt` file simply contains:

```
v<version>, <date> -- Initial release.
```

LICENSE.txt

By default, the Python package template contains the GPL v3 as *LICENSE.txt*. Change it as your convenience.

MANIFEST.in

The [manifest](#) is an important file that contains this:

```
include CHANGES.txt
include LICENSE.txt
include MANIFEST.in
include README.txt
recursive-include bin *
recursive-include docs *
recursive-include towelstuff *
```

README.txt

The usual [README file](#), written in [reStructuredText](#) format.

setup.py

The `setup.py` file created from the template contains:

```
1 # -*- coding: utf-8 -*-
2 """ $project setup.py script """
3
4 # system
```

```

5 from distutils.core import setup
6 from os.path import join, dirname
7
8
9 setup(
10     name='TowelStuff',
11     version='0.1.0',
12     author='Stéphane Péchard',
13     author_email='you@provider.com',
14     packages=['towelstuff', 'towelstuff.test'],
15     url='http://',
16     license='LICENSE.txt',
17     long_description=open(join(dirname(__file__), 'README.txt')).read(),
18     install_requires=[],
19     test_suite='towelstuff.test',
20 )

```

Look at how the content is filled with your own data (names, author, mail).

Customized substitutions

You can modify the substitutions used by the template through the command line:

```
$ projy PythonPackage TowelStuff "author, Monty" "date, 2012-06-18"
```

Then the substitutes `author` (normally get from `git`) 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` option but **they won't have any effect if the template file does not consider them.**

A full “scooped” Django project

Starting with version 0.4, Projy includes some post-project creation capabilities, that allows you to make almost anything you want after the files structure has been generated. Let see this with the first template using this, the [Django](#) project template. Nothing changed in the command, you type:

```
$ projy DjangoTemplate FunWebsite
```

Now, not only the Django template is used to create the project structure, but once it is done, a hook is called to do the following:

- build the [virtual environment](#) through the use of the Makefile file;
- create the FunWebsite Django project with the `django-admin.py startproject` command and the Django binary installed in the virtual environment ;
- separate the original `settings.py` file into three different settings files, following recommendations from [Two scoops of Django](#) ;
- initiate an empty [git](#) repository ;
- do some minor adjustments to get the project as I'm used to.

This hook helps crafting a very customized Django project tree structure. It is made of Python commands and automatically run into the project directory. What it does is defined into the Django project template.

Options

Projy comes also with some useful command line option.

Listing templates

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 on the next command.

What's inside a template

Type:

```
$ projy -i PythonPackage
```

and you'll see the detailed structure of the `PythonPackage` template. It shows the created directories and files, with the substitutions included in the template.

Available templates

Here is a list of all the templates, but also collectors, integrated into Projy at the moment. Of course, you can propose new templates, they'll be integrated into Projy.

Project templates

Project templates are used to create a files/directories structure. That's the second argument of the command line. For this list, the projects we create are all called `TowelStuff`. They are somewhat ordered by the programming language they use.

Python

These are [Python](#) or [Django](#) templates.

Django project

The command:

```
$ projy DjangoProject TowelStuff
```

produces:

```
TowelStuff/
  CHANGES.txt          - PythonPackageCHANGESFileTemplate
  fabfile.py            - DjangoFabfileTemplate
  LICENSE.txt           - GPL3FileTemplate
  Makefile              - DjangoMakefileTemplate
  manage.py             - **generated by Django**
  README.txt            - READMEReSTFileTemplate
  .gitignore            - DjangoGitignoreTemplate

TowelStuff/conf/
  nginx.conf            - DjangoNginxConfTemplate
  requirements_base.txt - DjangoRequirementsBaseTemplate
  requirements_dev.txt  - DjangoRequirementsDevTemplate
  requirements_production.txt - DjangoRequirementsProdTemplate
  supervisord.conf      - DjangoSupervisorConfTemplate

TowelStuff/static/

TowelStuff/TowelStuff/
  __init__.py           - **empty file**
  urls.py               - **generated by Django**
  wsgi.py               - **generated by Django**

TowelStuff/TowelStuff/settings/
  __init__.py           - **empty file**
  base.py               - **generated by Django as settings.py and renamed**
  dev.py                - DjangoSettingsDevTemplate
  prod.py               - DjangoSettingsProdTemplate

TowelStuff/TowelStuff/templates/

TowelStuff/TowelStuff/venv/
  ...
```

Directories `TowelStuff/static/` and `TowelStuff/TowelStuff/templates/` are created but empty. They are here to make a coherent and usable structure. The directory `TowelStuff/TowelStuff/venv/` contains the virtual environment created for the project.

Python package

The command:

```
$ projy PythonPackage TowelStuff
```

produces:

```
TowelStuff/
  bootstrap             - BootstrapScriptFileTemplate
  CHANGES.txt          - PythonPackageCHANGESFileTemplate
  LICENSE.txt           - GPL3FileTemplate
  MANIFEST.in           - PythonPackageMANIFESTFileTemplate
  README.txt            - READMEReSTFileTemplate
  setup.py              - PythonPackageSetupFileTemplate

TowelStuff/docs/
  index.rst
```

```
TowelStuff/towelstuff/  
    __init__.py
```

Python script

The command:

```
$ projy PythonScript TowelStuff
```

produces:

```
TowelStuff/  
    TowelStuff.py          - PythonScriptFileTemplate
```

LaTeX

These are [LaTeX](#) templates.

LaTeX book

The command:

```
$ projy LaTeXBook TowelStuff
```

produces:

```
TowelStuff/  
    TowelStuff.tex          - LaTeXBookFileTemplate  
    references.bib          - BibTeXFileTemplate  
    Makefile                - LaTeXMakefileFileTemplate
```

Note: the Makefile uses [Latexmk](#).

Fabric file

The command:

```
$ projy Fabfile TowelStuff
```

produces:

```
/  
    fabfile.py              - FabfileFileTemplate
```

This one is probably not generic enough, I added some stuff I use. Feel free to customize it.

Bootstrap

The command:

```
$ projy Bootstrap TowelStuff
```

produces:

```
/
bootstrap                - BootstrapScriptFileTemplate
```

Yes, the name has no impact on the produced file. Don't hesitate to make it short!

Projy itself!

Finally, a bit of a special template, which lets you create a Projy template and an empty file template from Projy itself. Call it meta if you want :-). See [Extending Projy](#) to know how such templates are meant to be written.

The command:

```
$ projy ProjyTemplate TowelStuff
```

produces:

```
/
TowelStuffTemplate.py      - ProjyTemplateFileTemplate
TowelStuffFileTemplate.txt
```

File templates

Python files

- ProjyTemplateFileTemplate
- PythonPackageCHANGESFileTemplate
- PythonPackageMANIFESTFileTemplate
- PythonPackageSetupFileTemplate
- PythonScriptFileTemplate
- FabfileFileTemplate

Django files

- DjangoFabfileTemplate
- DjangoMakefileTemplate
- DjangoNginxConfTemplate
- DjangoProjectTemplate
- DjangoRequirementsBaseTemplate
- DjangoRequirementsDevTemplate

- DjangoRequirementsProdTemplate
- DjangoSettingsDevTemplate
- DjangoSettingsProdTemplate
- DjangoSupervisorConfTemplate

Bash files

- BootstrapScriptFileTemplate

Text files

- DjangoGitignoreTemplate
- READMEReSTFileTemplate

LaTeX files

- BibTeXFileTemplate
- LaTeXBookFileTemplate
- LaTeXMakefileFileTemplate

Licenses

- AGPL3FileTemplate
- ApacheLicenseFileTemplate
- BSDLicenseFileTemplate
- DWTIFYWTPLFileTemplate
- GPL2FileTemplate
- GPL3FileTemplate
- LaTeX3LicenseFileTemplate
- LGPL3FileTemplate
- MITLicenseFileTemplate
- MPL2FileTemplate
- PythonLicense2FileTemplate

Collectors

Here is the list of currently available collectors:

- AuthorCollector
- AuthorMailCollector

Extending Projy

Writing new templates and data collectors is easy. Let's continue reviewing our example.

Project templates

Here is the project template used to create a Python package:

```
1  # -*- coding: utf-8 -*-
2  """ Projy template for PythonPackage. """
3
4  # system
5  from datetime import date
6  # parent class
7  from projy.templates.ProjyTemplate import ProjyTemplate
8  # collectors
9  from projy.collectors.AuthorCollector import AuthorCollector
10 from projy.collectors.AuthorMailCollector import AuthorMailCollector
11
12
13 class PythonPackageTemplate(ProjyTemplate):
14     """ Projy template class for PythonPackage. """
15
16     def __init__(self):
17         ProjyTemplate.__init__(self)
18
19
20     def directories(self):
21         """ Return the names of directories to be created. """
22         directories_description = [
23             self.project_name,
24             self.project_name + '/' + self.project_name.lower(),
25             self.project_name + '/docs',
26         ]
27         return directories_description
28
29
30     def files(self):
31         """ Return the names of files to be created. """
32         files_description = [
33             [ self.project_name,
34               'bootstrap',
35               'BootstrapScriptFileTemplate' ],
36             [ self.project_name,
37               'CHANGES.txt',
38               'PythonPackageCHANGESFileTemplate' ],
39             [ self.project_name,
40               'LICENSE.txt',
41               'GPL3FileTemplate' ],
42             [ self.project_name,
43               'MANIFEST.in',
44               'PythonPackageMANIFESTFileTemplate' ],
45             [ self.project_name,
46               'README.txt',
47               'READMEReSTFileTemplate' ],
48             [ self.project_name,
```



```

49         'setup.py',
50         'PythonPackageSetupFileTemplate' ],
51     [ self.project_name + '/' + self.project_name.lower(),
52       '__init__.py',
53       None ],
54     [ self.project_name + '/docs',
55       'index.rst',
56       None ],
57 ]
58 return files_description
59
60
61 def substitutes(self):
62     """ Return the substitutions for the templating replacements. """
63     author_collector = AuthorCollector()
64     mail_collector = AuthorMailCollector()
65     substitute_dict = dict(
66         project = self.project_name,
67         project_lower = self.project_name.lower(),
68         date = date.today().isoformat(),
69         author = author_collector.collect(),
70         author_email = mail_collector.collect(),
71     )
72     return substitute_dict

```

To write a new template, you have to specify five parts:

- the name of the template, which is the name of the class ;
- the directories, files, substitutes and posthook functions.

When writing a new template, you can use the `self.project_name` variable which contains the name of the project as you typed it. In our example, it is `TowelStuff`.

Name of the template

Here it is simply `PythonPackageTemplate`. This is the name you type in the command line concatenated with `Template` at the end. The created template inherits from the father of all templates, the `ProjyTemplate` class.

The *directories* function

directories()

Returns a tuple containing all the names of the directories to be created.

Return type list of directory names

In our example, the created directories are `TowelStuff`, `TowelStuff/towelstuff` and `TowelStuff/docs`.

The *files* function

files()

This function should return a tuple containing three informations for each file:

- the directory the file is in. It is defined as in *the directories function* ;

- the name of the file ;
- the template of the file, which is not the same as the project template. See [File templates](#).

Return type list of file names

In our example, eight files are created:

- `bootstrap` created from `BootstrapScriptFileTemplate` ;
- `CHANGES.txt` created from `PythonPackageCHANGESFileTemplate` ;
- `LICENSE.txt` created from `GPL3FileTemplate` ;
- `MANIFEST.in` created from `PythonPackageMANIFESTFileTemplate` ;
- `README.txt` created from `READMEReSTFileTemplate` ;
- `setup.py` created from `PythonPackageSetupFileTemplate` ;
- `__init__.py` into the `TowelStuff/towelstuff` directory, created from `PythonPackageSetupFileTemplate` ;
- `index.rst` into the `TowelStuff/docs` directory, created empty.

Details on the content of each file is given on [Usage](#).

The *substitutes* function

substitutes()

This function should return a dictionary containing the string substitutions used in the template.

Return type dictionary of substitutions

In our example, the substitutions made in all the created files are:

- `$project` is replaced by `TowelStuff` ;
- `$project_lower` is replaced by `towelstuff` ;
- `$date` is replaced by the current date, in the format `2013-11-23` ;
- `$author` is replaced by what returns the `AuthorCollector` ;
- `$author_email` is replaced by what returns the `AuthorMailCollector` ;

The *posthook* function

posthook()

This function contains any post commands you need to be done on your project. Its implementation is not mandatory.

File templates

From all the templated files we created, let's see how the `PythonPackageSetupFileTemplate` is made. Here is its content:

```

1  # -*- coding: utf-8 -*-
2  """ $project setup.py script """
3
4  # $project
5  from $project_lower import __version__
6
7  # system
8  try:
9      from setuptools import setup
10 except ImportError:
11     from distutils.core import setup
12 from os.path import join, dirname
13
14
15 setup(
16     name=__version__,
17     version='0.1.0',
18     description='My $project project',
19     author='$author',
20     author_email='$author_email',
21     packages=['$project_lower', '$project_lower.test'],
22     url='http://stephanepechard.github.com/projy',
23     long_description=open('README.txt').read(),
24     install_requires=[],
25     test_suite='$project_lower.test',
26     classifiers=[
27         'Development Status :: 3 - Alpha',
28         'License :: OSI Approved :: GNU General Public License v3 (GPLv3)',
29         'Programming Language :: Python',
30     ],
31 )

```

It is simply the file you want to create with the variables that will be substitute in the creation process. Each variable should begin by \$ as described in the [Template](#) mechanism. Nothing fancy on this side, as you can see.

Data collectors

A data collector, as its name suggest, collects data. It is used by Projy to complete the *File templates*. Here is the data collector for the author data:

```

1  # -*- coding: utf-8 -*-
2  """ AuthorCollector class
3      Tries to find the program user name, as accuratly as possible.
4
5      Put the functions alphabetical order in the same order as their importance.
6      For example here, author_from_git should be taken before author_from_system
7      as it is probably better.
8  """
9
10 # system
11 import getpass
12 import locale
13 import os
14 from subprocess import Popen, PIPE, CalledProcessError
15 # parent class
16 from projy.collectors.Collector import Collector

```

```

17
18
19 class AuthorCollector(Collector):
20     """ The AuthorCollector class. """
21
22     def __init__(self):
23         self.author = None
24
25
26     def author_from_git(self):
27         """ Get the author name from git information. """
28         self.author = None
29         try:
30             encoding = locale.getdefaultlocale()[1]
31             # launch git command and get answer
32             cmd = Popen(["git", "config", "--get", "user.name"], stdout=PIPE)
33             stdoutdata = cmd.communicate().decode(encoding)
34             if (stdoutdata[0]):
35                 import ipdb; ipdb.set_trace()
36                 author = stdoutdata[0].rstrip(os.linesep)
37                 self.author = author#.decode('utf8')
38         except ImportError:
39             pass
40         except CalledProcessError:
41             pass
42         except OSError:
43             pass
44
45         return self.author
46
47
48     def author_from_system(self):
49         """ Get the author name from system information.
50             This is just the user name, not the real name.
51         """
52         self.author = getpass.getuser()
53         return self.author

```

A data collector defines as many functions as necessary. In the case of the author, two ways of finding it are written. The first uses `git`. As many users of Projy would probably use it, chances are that its configuration will reflect the author's information. As a fallback in case `git` does not return the wanted data, the user name is taken as the system current user name. There are probably other methods to find it, so feel free to propose some more.

Functions are treated in the alphabetical order, which means that the most accurate functions should come before the least accurate ones. Of course, one may not always know what the most accurate way of finding a particular data is. Be smart then!

Changelog

0.4.1 (2013-10-21)

- Fix crash with UTF-8 username
- Fix crash with some templates asking for a post hook

0.4 (2013-10-15)

- Add post hook to run python commands after template completion
- Add full Django project template following most recommendations of ‘Two Scoops of Django’

0.3 (2012-11-18)

- Update templates
- Use blessings to print text
- Update project to same file organisation as appypi
- Add requirement files
- New README.txt file
- Add tests suite, to be continued...

0.2 (2012-08-14)

- More documentation

0.1.2 (2012-07-03)

- New Bootstrap template

0.1.1 (2012-06-24)

- Minor fix

0.1 (2012-06-24)

- Initial release

D

`directories()` (built-in function), [13](#)

F

`files()` (built-in function), [13](#)

P

`posthook()` (built-in function), [14](#)

S

`substitutes()` (built-in function), [14](#)