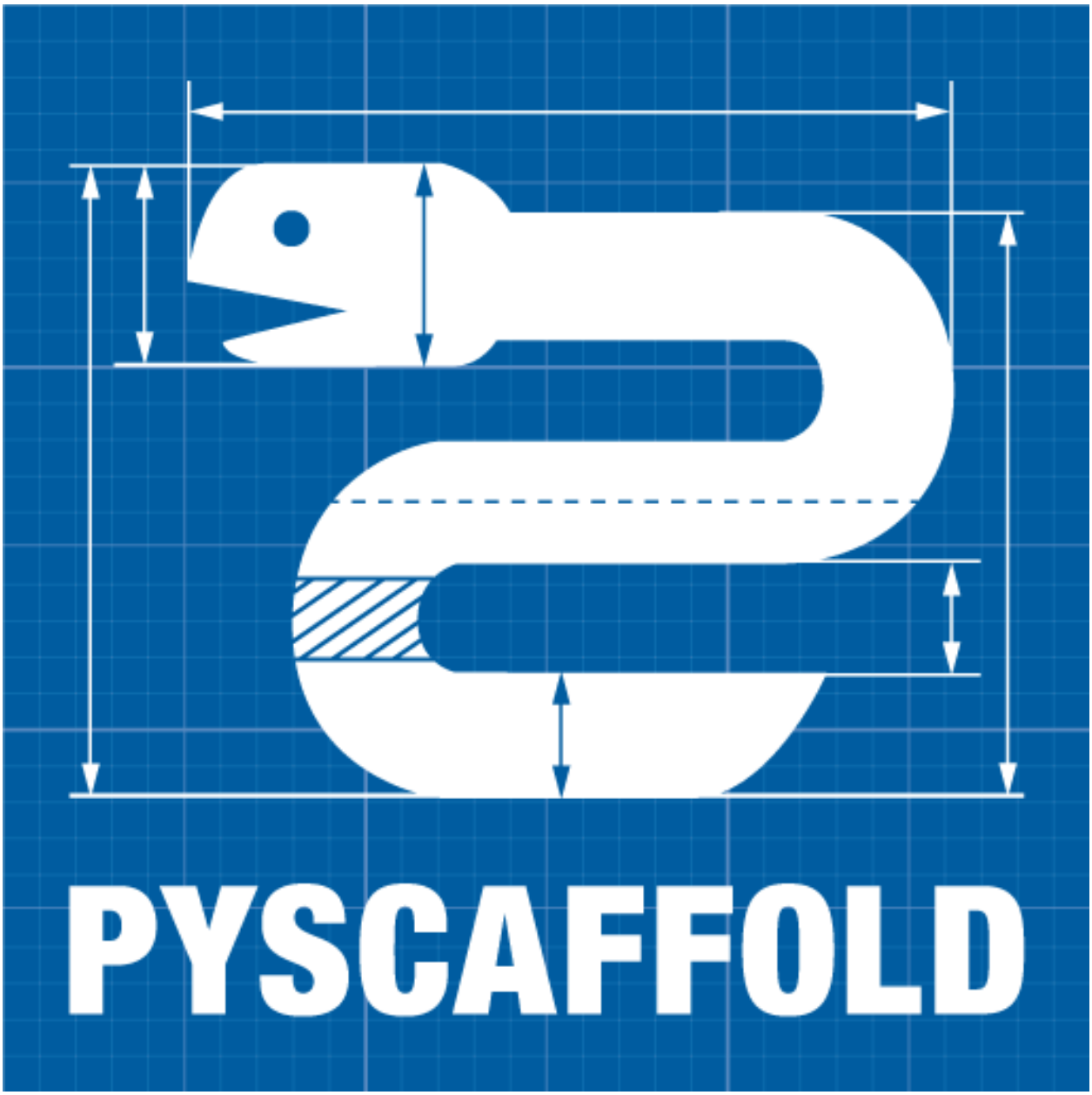


PyScaffold Documentation

Release 2.5.11

Blue Yonder

Apr 14, 2018

Contents

1 Features **3**
- 1.1 Configuration & Packaging . . . 3
- 1.2 Complete Git Integration . . . 4
- 1.3 Sphinx Documentation . . . 4
- 1.4 Unittest & Coverage . . . 4
- 1.5 Management of Requirements & Licenses . . . 5
- 1.6 Django & Cookiecutter . . . 5
- 1.7 Easy Updating . . . 5

2 Installation **7**
- 2.1 Requirements . . . 7
- 2.2 Installation . . . 8
- 2.3 Additional Requirements . . . 8

3 Examples **9**

4 Configuration **11**

5 Contributing **13**
- 5.1 Chat . . . 13
- 5.2 Bug Reports . . . 13
- 5.3 Code . . . 13

6 License **15**

7 Developers **17**

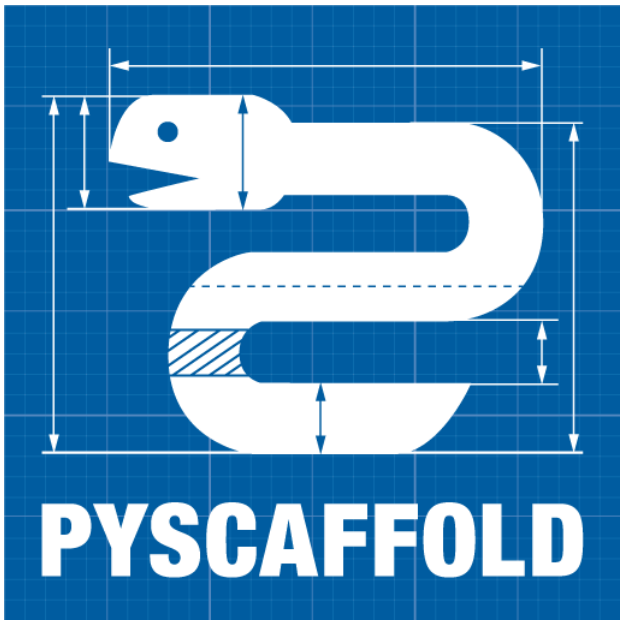
8 Release Notes **19**
- 8.1 Version 2.5.11, 2018-04-12 . . . 19
- 8.2 Version 2.5.10, 2018-03-21 . . . 19
- 8.3 Version 2.5.9, 2018-03-20 . . . 19
- 8.4 Version 2.5.8, 2017-09-10 . . . 19
- 8.5 Version 2.5.7, 2016-10-11 . . . 20
- 8.6 Version 2.5.6, 2016-05-01 . . . 20
- 8.7 Version 2.5.5, 2016-02-26 . . . 20
- 8.8 Version 2.5.4, 2016-02-10 . . . 20
- 8.9 Version 2.5.3, 2016-01-16 . . . 20

8.10 Version 2.5.2, 2016-01-02 . . . 20
8.11 Version 2.5.1, 2016-01-01 . . . 21
8.12 Version 2.5, 2015-12-09 . . . 21
8.13 Version 2.4.4, 2015-10-29 . . . 21
8.14 Version 2.4.3, 2015-10-27 . . . 21
8.15 Version 2.4.2, 2015-09-16 . . . 21
8.16 Version 2.4.1, 2015-09-09 . . . 21
8.17 Version 2.4, 2015-09-02 . . . 21
8.18 Version 2.3, 2015-08-26 . . . 22
8.19 Version 2.2.1, 2015-06-18 . . . 22
8.20 Version 2.2, 2015-06-01 . . . 22
8.21 Version 2.1, 2015-04-16 . . . 23
8.22 Version 2.0.4, 2015-03-17 . . . 23
8.23 Version 2.0.3, 2015-03-17 . . . 23
8.24 Version 2.0.2, 2015-03-04 . . . 23
8.25 Version 2.0.1, 2015-02-27 . . . 23
8.26 Version 2.0, 2015-02-25 . . . 23
8.27 Version 1.4, 2014-12-16 . . . 24
8.28 Version 1.3.2, 2014-12-02 . . . 24
8.29 Version 1.3.1, 2014-11-24 . . . 24
8.30 Version 1.3, 2014-11-17 . . . 24
8.31 Version 1.2, 2014-10-13 . . . 24
8.32 Version 1.1, 2014-09-29 . . . 25
8.33 Version 1.0, 2014-09-05 . . . 25
8.34 Version 0.9, 2014-07-27 . . . 25
8.35 Version 0.8, 2014-07-25 . . . 25
8.36 Version 0.7, 2014-06-05 . . . 26
8.37 Version 0.6.1, 2014-05-15 . . . 26
8.38 Version 0.6, 2014-05-14 . . . 26
8.39 Version 0.5, 2014-05-02 . . . 26
8.40 Version 0.4.1, 2014-04-27 . . . 26
8.41 Version 0.4, 2014-04-23 . . . 26
8.42 Version 0.3, 2014-04-18 . . . 27
8.43 Version 0.2, 2014-04-15 . . . 27
8.44 Version 0.1.2, 2014-04-10 . . . 27
8.45 Version 0.1.1, 2014-04-10 . . . 27
8.46 Version 0.1, 2014-04-03 . . . 27

9 pyscaffold **29**
9.1 pyscaffold package . . . 29

10 Indices and tables **41**

Python Module Index **43**



PyScaffold helps you to easily setup a new Python project, it is as easy as:

```
putup my_project
```

This will create a new folder `my_project` containing a perfect *project template* with everything you need for some serious coding.

Type `putup -h` to learn about more configuration options. PyScaffold assumes that you have Git installed and set up on your PC, meaning at least your name and email configured. The project template in `my_project` provides you with a lot of *features*. PyScaffold is compatible with Python 2.7, 3.4 and 3.5.

CHAPTER 1

Features

PyScaffold comes with a lot of elaborated features and configuration defaults to make the most common tasks in developing, maintaining and distributing your own Python package as easy as possible.

1.1 Configuration & Packaging

All configuration can be done in `setup.cfg` like changing the description, url, classifiers and even console scripts of your project with the help of pbr. That means in most cases it is not necessary to tamper with `setup.py`. The syntax of `setup.cfg` is pretty much self-explanatory and well commented, check out this *example* or pbr's usage manual.

In order to build a source, binary or wheel distribution, just run `python setup.py sdist`, `python setup.py bdist` or `python setup.py bdist_wheel`.

Namespace Packages

Optionally, namespace packages can be used, if you are planning to distribute a larger package as a collection of smaller ones. For example, use:

```
putup my_project --package my_package --with-namespace com.my_domain
```

to define `my_package` inside the namespace `com.my_domain` in java-style.

Package and Files Data

Additional data, e.g. images and text files, inside your package can be configured under the `[files]` section in `setup.cfg`. It is not necessary to have a `MANIFEST.in` file for this to work. To read this data in your code, use:

```
from pkgutil import get_data
data = get_data('my_package', 'path/to/my/data.txt')
```

Note: Make sure that all files you specify in `[files]` have been added to the repository!

1.2 Complete Git Integration

Your project is already an initialised Git repository and `setup.py` uses the information of tags to infer the version of your project with the help of setuptools_scm. To use this feature you need to tag with the format `MAJOR.MINOR[.PATCH]`, e.g. `0.0.1` or `0.1`. Run `python setup.py --version` to retrieve the current PEP440-compliant version. This version will be used when building a package and is also accessible through `my_project.__version__`.

Unleash the power of Git by using its pre-commit hooks. This feature is available through the `--with-pre-commit` flag. After your project's scaffold was generated, make sure pre-commit is installed, e.g. `pip install pre-commit`, then just run `pre-commit install`.

It goes unsaid that also a default `.gitignore` file is provided that is well adjusted for Python projects and the most common tools.

1.3 Sphinx Documentation

Build the documentation with `python setup.py docs` and run doctests with `python setup.py doctest`. Start editing the file `docs/index.rst` to extend the documentation. The documentation also works with Read the Docs.

The Numpy and Google style docstrings are activated by default. Just make sure Sphinx 1.3 or above is installed.

1.4 Unittest & Coverage

Run `python setup.py test` to run all unittests defined in the subfolder `tests` with the help of py.test and pytest-runner. Some sane default flags for py.test are already defined in the `[pytest]` section of `setup.cfg`. The py.test plugin pytest-cov is used to automatically generate a coverage report. It is also possible to provide additional parameters and flags on the commandline, e.g., type:

```
python setup.py test --addopts -h
```

to show the help of py.test.

JUnit and Coverage HTML/XML

For usage with a continuous integration software JUnit and Coverage XML output can be activated in `setup.cfg`. Use the flag `--with-travis` to generate templates of the Travis configuration files `.travis.yml` and `tests/travis_install.sh` which even features the coverage and stats system Coveralls. In order to use the virtualenv management and test tool Tox the flag `--with-tox` can be specified.

Managing test environments with tox

Run `tox` to generate test virtual environments for various python environments defined in the generated `tox.ini`. Testing and building *sdists* for python 2.7 and python 3.4 is just as simple with tox as:

```
tox -e py27,py34
```

Environments for tests with the the static code analyzers pyflakes and pep8 which are bundled in flake8 are included as well. Run it explicitly with:

```
tox -e flake8
```

With tox, you can use the `--recreate` flag to force tox to create new environments. By default, PyScaffold's tox configuration will execute tests for a variety of python versions. If an environment is not available on the system the tests are skipped gracefully. You can relay on the tox documentation for detailed configuration options.

1.5 Management of Requirements & Licenses

Add the requirements of your project to `requirements.txt` and `test-requirements.txt` which will be automatically used by `setup.py`. This also allows you to easily customize a plain virtual environment with:

```
pip install -r requirements.txt -r test-requirements.txt
```

Only absolutely necessary requirements of your project should be be stated in `requirements.txt` while the requirements only used for development and especially for running the unittests should go into `test-requirements.txt`.

Since PyScaffold uses pbr it is also possible to define requirements depending on your Python version. Use the environment variable `PBR_REQUIREMENTS_FILES` to define a comma-separated list of requirement files if you want to use non-default names and locations.

All licenses from choosealicense.com can be easily selected with the help of the `--license` flag.

1.6 Django & Cookiecutter

Create a Django project with the flag `--with-django` which is equivalent to `django-admin.py startproject my_project` enhanced by PyScaffold's features.

With the help of Cookiecutter it is possible to customize your project setup. Just use the flag `--with-cookiecutter TEMPLATE` to use a cookiecutter template which will be refined by PyScaffold afterwards.

1.7 Easy Updating

Keep your project's scaffold up-to-date by applying `putput --update my_project` when a new version of PyScaffold was released. An update will only overwrite files that are not often altered by users like setup.py. To update all files use `--update --force`. An existing project that was not setup with PyScaffold can be converted with `putup --force existing_project`. The force option is completely safe to use since the git repository of the existing project is not touched! Also check out if *configuration options* in `setup.cfg` have changed.

Note: If you are updating from a PyScaffold version before 2.0, you must manually remove the files `versioneer.py` and `MANIFEST.in`. If you are updating from a version prior to 2.2, you must remove `${PACKAGE}/_version.py`.

CHAPTER 2

Installation

2.1 Requirements

The installation of PyScaffold requires:

- setuptools
- six

as well as a working installation of Git. Especially Windows users should make sure that the command `git` is available on the command line. Otherwise, check and update your `PATH` environment variable or run PyScaffold from the *Git Bash*.

Note: It is recommended to use virtualenv and pip for Python package management. Make sure *pip*, *six* and *setuptools* are up to date:

```
pip install --upgrade pip setuptools six
```

Additionally, if you want to create a Django project or want to use cookiecutter:

- Django
- cookiecutter

Note: In most cases only Django needs to be installed manually since PyScaffold will download and install its requirements automatically when using `pip`. One exception might be *setuptools* if you are not using a current version of Virtual Environments as development environment. In case you are using the system installation of Python from your Linux distribution make sure *setuptools* is installed. To install it on Debian or Ubuntu:

```
sudo apt-get install python-setuptools
```

In case of Redhat or Fedora:

```
sudo yum install python-setuptools
```

2.2 Installation

If you have `pip` installed, then simply type:

```
pip install --upgrade pyscaffold
```

to get the lastest stable version. The most recent development version can be installed with:

```
pip install --pre --upgrade pyscaffold
```

Using `pip` also has the advantage that all requirements are automatically installed.

If you want to install PyScaffold with all features like Django and cookiecutter support, run:

```
pip install --upgrade pyscaffold[ALL]
```

2.3 Additional Requirements

If you run commands like `python setup.py test` and `python setup.py docs` within your project, some additional requirements like py.test will be installed automatically as *egg*-files inside the `.eggs` folder. This is quite comfortable but can be confusing because these packages won't be available to other packages inside your virtual environment. In order to avoid this just install following packages inside your virtual environment before you run *setup.py* commands like *doc* and *test*:

- Sphinx
- py.test
- pytest-cov

CHAPTER 3

Examples

Just a few examples to get you an idea of how easy PyScaffold is to use:

`putup my_little_project` The simplest way of using PyScaffold. A directory `my_little_project` is created with a Python package named exactly the same. Only a simple copyright statement is used as license.

`putup skynet -l gpl3 -d "Finally, the ultimate AI!" -u http://sky.net` This will create a project and package named *skynet* licensed under the GPL3. The *summary* inside `setup.cfg` is directly set to "Finally, the ultimate AI!" and the homepage to http://sky.net.

`putup Scikit-Gravity -p skgravity -l new-bsd` This will create a project named *Scikit-Gravity* but the package will be named *skgravity* with license new-BSD.

`putup youtub --with-django --with-pre-commit -d "Ultimate video site for hot tub fans"` This will create a web project and package named *youtub* that also includes the files created by Django's `django-admin`. The summary description in `setup.cfg` will be set and a file `.pre-commit-config.yaml` is created with a default setup for pre-commit.

`putup thoroughly_tested --with-tox --with-travis` This will create a project and package *thoroughly_tested* with files `tox.ini` and `.travis.yml` for Tox and Travis.

`putup my_zope_subpackage --with-namespace zope -l gpl3` This will create a project and subpackage named *my_zope_subpackage* in the namespace *zope*. To be honest, there is really only the Zope project that comes to my mind which is using this exotic feature of Python's packaging system. Chances are high, that you will never ever need a namespace package in your life.

CHAPTER 4

Configuration

Projects set up with PyScaffold feature an easy package configuration with `setup.cfg`. Check out the example below as well as pbr's usage manual.

```
[metadata]
name = my_project
summary = A test project that was set up with PyScaffold
author = Florian Wilhelm
author-email = Florian.Wilhelm@blue-yonder.com
license = new BSD
home-page = http://...
description-file = README.rst
# Add here all kinds of additional classifiers as defined under
# https://pypi.python.org/pypi?%3Aaction=list_classifiers
classifier =
    Development Status :: 5 - Production/Stable
    Topic :: Utilities
    Programming Language :: Python
    Programming Language :: Python :: 2
    Programming Language :: Python :: 2.7
    Programming Language :: Python :: 3
    Programming Language :: Python :: 3.4
    Programming Language :: Python :: 3.5
    Environment :: Console
    Intended Audience :: Developers
    License :: OSI Approved :: BSD License
    Operating System :: POSIX :: Linux
    Operating System :: Unix
    Operating System :: MacOS
    Operating System :: Microsoft :: Windows

[entry_points]
# Add here console scripts like:
console_scripts =
    run_my_project = my_project.cli:run
# as well as other entry_points.
```

```
[files]
# Add here 'data_files', 'packages' or 'namespace_packages'.
# Additional data files are defined as key value pairs of source and target:
packages =
    my_project
data_files =
    share/my_project/docs = docs/*

[extras]
# Add here additional requirements for extra features, like:
# PDF =
#    ReportLab>=1.2
#    RXP
ALL =
    django
    cookiecutter

[test]
# py.test options when running `python setup.py test`
addopts = tests

[pytest]
# Options for py.test:
# Specify command line options as you would do when invoking py.test directly.
# e.g. --cov-report html (or xml) for html/xml output or --junitxml junit.xml
# in order to write a coverage file that can be read by Jenkins.
addopts =
    --cov my_project --cov-report term-missing
    --verbose

[aliases]
docs = build_sphinx

[bdist_wheel]
# Use this option if your package is pure-python
universal = 1

[build_sphinx]
# Options for Sphinx build
source_dir = docs
build_dir = docs/_build

[pbr]
# Let pbr run sphinx-apidoc
autodoc_tree_index_modules = True
# autodoc_tree_excludes = ...
# Let pbr itself generate the apidoc
# autodoc_index_modules = True
# autodoc_exclude_modules = ...
# Convert warnings to errors
# warnerrors = True

[devpi:upload]
# Options for the devpi: PyPI server and packaging tool
# VCS export must be deactivated since we are using setuptools-scm
no-vcs = 1
formats = bdist_wheel
```

CHAPTER 5

Contributing

PyScaffold is developed by Blue Yonder developers to help automating and standardizing the process of project setups. You are very welcome to join in our effort if you would like to contribute.

5.1 Chat

Join our chat to get in direct contact with the developers of PyScaffold.

5.2 Bug Reports

If you experience bugs or in general issues with PyScaffold, please file a bug report to our Bug Tracker.

5.3 Code

If you would like to contribute to PyScaffold, fork the main repository on GitHub with the help of Git, then submit a "pull request" (PR):

1. Create an account on GitHub if you do not already have one.
2. Fork the project repository: click on the *Fork* button near the top of the page. This creates a copy of the code under your account on the GitHub server.
3. Clone this copy to your local disk:

```
git clone git@github.com:YourLogin/pyscaffold.git
```

4. Run `python setup.py egg_info` after a fresh checkout. This will generate some critically needed files.
5. Create a branch to hold your changes:

```
git checkout -b my-feature
```

and start making changes. Never work on the master branch!

6. Start your work on this branch. When you're done editing, do:

```
git add modified_files
git commit
```

to record your changes in Git, then push them to GitHub with:

```
git push -u origin my-feature
```

7. Go to the web page of your PyScaffold fork, and click "Create pull request" to send your changes to the maintainers for review. Find more detailed information here.

CHAPTER 6

License

```
Copyright (c) 2014, Blue Yonder GmbH.
All rights reserved.


Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

  a. Redistributions of source code must retain the above copyright notice,
     this list of conditions and the following disclaimer.
  b. Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in the
     documentation and/or other materials provided with the distribution.
  c. Neither the name of the PyScaffold developers nor the names of
     its contributors may be used to endorse or promote products
     derived from this software without specific prior written
     permission.


THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

CHAPTER 7

Developers

- Florian Wilhelm <Florian.Wilhelm@gmail.com>
- Felix Wick <Felix.Wick@blue-yonder.com>
- Holger Peters <Holger.Peters@blue-yonder.com>
- Uwe Korn <Uwe.Korn@blue-yonder.com>
- Patrick Mühlbauer <Patrick.Muehlbauer@blue-yonder.com>
- Florian Rathgeber <florian.rathgeber@gmail.com>
- Eva Schmücker <email@evaschmuecker.de>
- Tim Werner <tim.werner@blue-yonder.com>

CHAPTER 8

Release Notes

8.1 Version 2.5.11, 2018-04-12

- Updated pbr to version 4.0.2
- Fixes Sphinx version 1.6 regression, issue #152

8.2 Version 2.5.10, 2018-03-21

- Update setuptools_scm to version v1.17.0

8.3 Version 2.5.9, 2018-03-20

- Updated setuptools_scm to version v1.16.1
- Fix error with setuptools version 39.0 and above, issue #148

8.4 Version 2.5.8, 2017-09-10

- Use *sphinx.ext.imgmath* instead of *sphinx.ext.mathjax*
- Added *–with-gitlab-ci* flag for GitLab CI support
- Fix Travis install template dirties git repo, issue #107
- Updated setuptools_scm to version 1.15.6
- Updated pbr to version 3.1.1

8.5 Version 2.5.7, 2016-10-11

- Added encoding to __init__.py
- Few doc corrections in setup.cfg
- [tool:pytest] instead of [pytest] in setup.cfg
- Updated skeleton
- Switch to Google Sphinx style
- Updated setuptools_scm to version 1.13.1
- Updated pbr to version 1.10.0

8.6 Version 2.5.6, 2016-05-01

- Prefix error message with ERROR:
- Suffix of untagged commits changed from {version}-{hash} to {version}-n{hash}
- Check if package identifier is valid
- Added log level command line flags to the skeleton
- Updated pbr to version 1.9.1
- Updated setuptools_scm to version 1.11.0

8.7 Version 2.5.5, 2016-02-26

- Updated pbr to master at 2016-01-20
- Fix sdist installation bug when no git is installed, issue #90

8.8 Version 2.5.4, 2016-02-10

- Fix problem with `fibonacci` terminal example
- Update setuptools_scm to v1.10.1

8.9 Version 2.5.3, 2016-01-16

- Fix classifier metadata (`classifiers` to `classifier` in `setup.cfg`)

8.10 Version 2.5.2, 2016-01-02

- Fix `is_git_installed`

8.11 Version 2.5.1, 2016-01-01

- Fix: Do some sanity checks first before gathering default options
- Updated setuptools_scm to version 1.10.0

8.12 Version 2.5, 2015-12-09

- Usage of `test-requirements.txt` instead of `tests_require` in `setup.py`, issue #71
- Removed `--with-numpydoc` flag since this is now included by default with `sphinx.ext.napoleon` in Sphinx 1.3 and above
- Added small template for unittest
- Fix for the example skeleton file when using namespace packages
- Fix typo in devpi:upload section, issue #82
- Include `pbr` and `setuptools_scm` in PyScaffold to avoid dependency problems, issue #71 and #72
- Cool logo was designed by Eva Schmücker, issue #66

8.13 Version 2.4.4, 2015-10-29

- Fix problem with bad upload of version 2.4.3 to PyPI, issue #80

8.14 Version 2.4.3, 2015-10-27

- Fix problem with version numbering if setup.py is not in the root directory, issue #76

8.15 Version 2.4.2, 2015-09-16

- Fix version conflicts due to too tight pinning, issue #69

8.16 Version 2.4.1, 2015-09-09

- Fix installation with additional requirements `pyscaffold[ALL]`
- Updated pbr version to 1.7

8.17 Version 2.4, 2015-09-02

- Allow different py.test options when invoking with `py.test` or `python setup.py test`
- Check if Sphinx is needed and add it to *setup_requires*
- Updated pre-commit plugins

- Replaced pytest-runner by an improved version
- Let pbr do `sphinx-apidoc`, removed from `conf.py`, issue #65

Note: Due to the switch to a modified pytest-runner version it is necessary to update `setup.cfg`. Please check the *example*.

8.18 Version 2.3, 2015-08-26

- Format of setup.cfg changed due to usage of pbr, issue #59
- Much cleaner setup.py due to usage of pbr, issue #59
- PyScaffold can be easily called from another script, issue #58
- Internally dictionaries instead of namespace objects are used for options, issue #57
- Added a section for devpi in setup.cfg, issue #62

Note: Due to the switch to pbr, it is necessary to update `setup.cfg` according to the new syntax.

8.19 Version 2.2.1, 2015-06-18

- FIX: Removed putup console script in setup.cfg template

8.20 Version 2.2, 2015-06-01

- Allow recursive inclusion of data files in setup.cfg, issue #49
- Replaced hand-written PyTest runner by pytest-runner, issue #47
- Improved default README.rst, issue #51
- Use tests/conftest.py instead of tests/__init__.py, issue #52
- Use setuptools_scm for versioning, issue #43
- Require setuptools>=9.0, issue #56
- Do not create skeleton.py during an update, issue #55

Note: Due to the switch to *setuptools_scm* the following changes apply:

- use `python setup.py --version` instead of `python setup.py version`
- `git archive` can no longer be used for packaging (and was never meant for it anyway)
- initial tag `v0.0` is no longer necessary and thus not created in new projects
- tags do no longer need to start with *v*

8.21 Version 2.1, 2015-04-16

- Use alabaster as default Sphinx theme
- Parameter data_files is now a section in setup.cfg
- Allow definition of extras_require in setup.cfg
- Added a CHANGES.rst file for logging changes
- Added support for cookiecutter
- FIX: Handle an empty Git repository if necessary

8.22 Version 2.0.4, 2015-03-17

- Typo and wrong Sphinx usage in the RTD documentation

8.23 Version 2.0.3, 2015-03-17

- FIX: Removed misleading *include_package_data* option in setup.cfg
- Allow selection of a proprietary license
- Updated some documentations
- Added -U as short parameter for –update

8.24 Version 2.0.2, 2015-03-04

- FIX: Version retrieval with setup.py install
- argparse example for version retrieval in skeleton.py
- FIX: import my_package should be quiet (verbose=False)

8.25 Version 2.0.1, 2015-02-27

- FIX: Installation bug under Windows 7

8.26 Version 2.0, 2015-02-25

- Split configuration and logic into setup.cfg and setup.py
- Removed .pre from version string (newer PEP 440)
- FIX: Sphinx now works if package name does not equal project name
- Allow namespace packages with –with-namespace
- Added a skeleton.py as a console_script template

- Set *v0.0* as initial tag to support PEP440 version inference
- Integration of the Versioneer functionality into setup.py
- Usage of *data_files* configuration instead of *MANIFEST.in*
- Allow configuration of *package_data* in *setup.cfg*
- Link from Sphinx docs to AUTHORS.rst

8.27 Version 1.4, 2014-12-16

- Added numpydoc flag –with-numpydoc
- Fix: Add django to requirements if –with-django
- Fix: Don't overwrite index.rst during update

8.28 Version 1.3.2, 2014-12-02

- Fix: path of Travis install script

8.29 Version 1.3.1, 2014-11-24

- Fix: –with-tox tuple bug #28

8.30 Version 1.3, 2014-11-17

- Support for Tox (https://tox.readthedocs.org/)
- flake8: exclude some files
- Usage of UTF8 as file encoding
- Fix: create non-existent files during update
- Fix: unit tests on MacOS
- Fix: unit tests on Windows
- Fix: Correct version when doing setup.py install

8.31 Version 1.2, 2014-10-13

- Support pre-commit hooks (http://pre-commit.com/)

8.32 Version 1.1, 2014-09-29

- Changed COPYING to LICENSE
- Support for all licenses from http://choosealicense.com/
- Fix: Allow update of license again
- Update to Versioneer 0.12

8.33 Version 1.0, 2014-09-05

- Fix when overwritten project has a git repository
- Documentation updates
- License section in Sphinx
- Django project support with –with-django flag
- Travis project support with –with-travis flag
- Replaced sh with own implementation
- Fix: new *git describe* version to PEP440 conversion
- conf.py improvements
- Added source code documentation
- Fix: Some Python 2/3 compatibility issues
- Support for Windows
- Dropped Python 2.6 support
- Some classifier updates

8.34 Version 0.9, 2014-07-27

- Documentation updates due to RTD
- Added a –force flag
- Some cleanups in setup.py

8.35 Version 0.8, 2014-07-25

- Update to Versioneer 0.10
- Moved sphinx-apidoc from setup.py to conf.py
- Better support for *make html*

8.36 Version 0.7, 2014-06-05

- Added Python 3.4 tests and support
- Flag –update updates only some files now
- Usage of setup_requires instead of six code

8.37 Version 0.6.1, 2014-05-15

- Fix: Removed six dependency in setup.py

8.38 Version 0.6, 2014-05-14

- Better usage of six
- Return non-zero exit status when doctests fail
- Updated README
- Fixes in Sphinx Makefile

8.39 Version 0.5, 2014-05-02

- Simplified some Travis tests
- Nicer output in case of errors
- Updated PyScaffold's own setup.py
- Added –junit_xml and –coverage_xml/html option
- Updated .gitignore file

8.40 Version 0.4.1, 2014-04-27

- Problem fixed with pytest-cov installation

8.41 Version 0.4, 2014-04-23

- PEP8 and PyFlakes fixes
- Added –version flag
- Small fixes and cleanups

8.42 Version 0.3, 2014-04-18

- PEP8 fixes
- More documentation
- Added update feature
- Fixes in setup.py

8.43 Version 0.2, 2014-04-15

- Checks when creating the project
- Fixes in COPYING
- Usage of sh instead of GitPython
- PEP8 fixes
- Python 3 compatibility
- Coverage with Coverall.io
- Some more unittests

8.44 Version 0.1.2, 2014-04-10

- Bugfix in Manifest.in
- Python 2.6 problems fixed

8.45 Version 0.1.1, 2014-04-10

- Unittesting with Travis
- Switch to string.Template
- Minor bugfixes

8.46 Version 0.1, 2014-04-03

- First release

CHAPTER 9

pyscaffold

9.1 pyscaffold package

9.1.1 Subpackages

pyscaffold.contrib package

Module contents

Contribution packages used by PyScaffold

All packages inside `contrib` are external packages that come with their own licences and are not part of the PyScaffold sourcecode itself. The reason for shipping these dependencies directly is to avoid problems in the resolution of `setup_requires` dependencies that occurred more often than not, see issues #71 and #72.

All contribution packages were added with the help of `git subtree` (git version 1.7.11 and above):

```
git subtree add --prefix pyscaffold/contrib/setuptools_scm --squash     https://
↪github.com/pypa/setuptools_scm.git v1.10.1

git subtree add --prefix pyscaffold/contrib/pbr --squash     https://github.com/
↪openstack-dev/pbr.git 1.8.1
```

Updating works with:

```
git subtree pull --prefix pyscaffold/contrib/setuptools_scm     https://github.com/
↪pypa/setuptools_scm.git NEW_TAG --squash

git subtree pull --prefix pyscaffold/contrib/pbr     https://github.com/openstack-dev/
↪pbr.git NEW_TAG --squash
```

Using `subtree` instead of git's `submodule` had several advantages.

Note: Updating pbr like described above only works if there was no change in the pbr directory but in most cases we remove *test-requirements.txt* files since otherwise Travis complains about them. In order to update it's best to completely remove *contrib/pbr* first and then use the command above.

pyscaffold.contrib.**add_dir_to_syspath**(*path*)
Contextmanager to temporarily prepend a path the `sys.path`

Parameters **path** – path as string

pyscaffold.contrib.**import_mod**(*module*, *path*)
Imports a module from a directory path

Parameters

- **module** – module name as string
- **path** – path as string

Returns module

pyscaffold.templates package

Module contents

Templates for all files of a project's scaffold

pyscaffold.templates.**authors**(*opts*)
Template of AUTHORS.rst

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**changes**(*opts*)
Template of CHANGES.rst

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**conftest_py**(*opts*)
Template of conftest.py

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**coveragerc**(*opts*)
Template of .coveragerc

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**get_template**(*name*)
Retrieve the template by name

Parameters **name** – name of template

Returns template as `string.Template`

pyscaffold.templates.**gitignore**(*opts*)
Template of .gitignore

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**gitignore_empty**(*opts*)
Template of empty .gitignore

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**gitlab_ci**(*opts*)
Template of .gitlab-ci.yml

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**init**(*opts*)
Template of __init__.py

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**license**(*opts*)
Template of LICENSE.txt

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**namespace**(*opts*)
Template of __init__.py defining a namespace package

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**pre_commit_config**(*opts*)
Template of .pre-commit-config.yaml

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**readme**(*opts*)
Template of README.rst

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**requirements**(*opts*)
Template of requirements.txt

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**setup_cfg**(*opts*)
Template of setup.cfg

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**setup_py**(*opts*)
Template of setup.py

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**skeleton**(*opts*)
Template of skeleton.py defining a basic console script

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**sphinx_authors**(*opts*)
Template of authors.rst

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**sphinx_changes**(*opts*)
Template of changes.rst

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**sphinx_conf**(*opts*)
Template of conf.py

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**sphinx_index**(*opts*)
Template of index.rst

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**sphinx_license**(*opts*)
Template of license.rst

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**sphinx_makefile**(*opts*)
Template of Sphinx's Makefile

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**test_requirements**(*opts*)
Template of test-requirements.txt

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**test_skeleton**(*opts*)
Template of unittest for skeleton.py

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**tox**(*opts*)
Template of tox.ini

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**travis**(*opts*)
Template of .travis.yml

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

pyscaffold.templates.**travis_install**(*opts*)
Template of travis_install.sh

Parameters **opts** – mapping parameters as dictionary

Returns file content as string

9.1.2 Submodules

9.1.3 pyscaffold.cli module

Command-Line-Interface of PyScaffold

pyscaffold.cli.**create_project**(*opts*)
Create the project's directory structure

Parameters **opts** (*dict*) – options of the project

pyscaffold.cli.**get_default_opts**(*project_name*, ***aux_opts*)
Creates default options using auxiliary options as keyword argument

Use this function if you want to use PyScaffold from another application in order to generate an option dictionary that can than be passed to *create_project*.

Parameters

- **project_name** (*str*) – name of the project
- ****aux_opts** – auxiliary options as keyword parameters

Returns options with default values set

Return type dict

pyscaffold.cli.**main**(*args*)
PyScaffold is a tool for putting up the scaffold of a Python project.

pyscaffold.cli.**make_sanity_checks**(*opts*)
Perform some sanity checks, e.g., if git is installed.

Parameters **opts** (*dict*) – options of the project

pyscaffold.cli.**parse_args**(*args*)
Parse command line parameters

Parameters **args** (*[str]*) – command line parameters as list of strings

Returns command line parameters

Return type dict

pyscaffold.cli.**run**()
Entry point for setup.py

9.1.4 pyscaffold.info module

Provide general information about the system, user etc.

pyscaffold.info.**email**()
Retrieve the user's email

Returns user's email

Return type str

pyscaffold.info.**is_git_configured**()
Check if user.name and user.email is set globally in git

Returns True if it is set globally, False otherwise

Return type bool

pyscaffold.info.**is_git_installed**()
Check if git is installed

Returns True if git is installed, False otherwise

Return type bool

pyscaffold.info.**project**(*opts*)
Update user options with the options of an existing PyScaffold project

Params: opts (dict): options of the project

Returns options with updated values

Return type dict

pyscaffold.info.**username**()
Retrieve the user's name

Returns user's name

Return type str

9.1.5 pyscaffold.integration module

Integration part for hooking into distutils/setuptools

Rationale: The `use_pyscaffold` keyword is unknown to setuptools' setup(...) command, therefore the `entry_points` are checked for a function to handle this keyword which is `pyscaffold_keyword` below. This is where we hook into setuptools and apply the magic of setuptools_scm and pbr.

pyscaffold.integration.**build_cmd_docs**()
Return Sphinx's BuildDoc if available otherwise a dummy command

Returns command object

Return type Command

pyscaffold.integration.**deactivate_pbr_authors_changelog**()
Deactivate automatic generation of AUTHORS and ChangeLog file

This is an automatism of pbr and we rather keep track of our own AUTHORS.rst and CHANGES.rst files.

pyscaffold.integration.**local_version2str**(*version*)
Create the local part of a PEP440 version string

Parameters **version** (setuptools_scm.version.ScmVersion) – version object

Returns local version

Return type str

pyscaffold.integration.**pyscaffold_keyword**(*dist*, *keyword*, *value*)
Handles the *use_pyscaffold* keyword of the setup(...) command

Parameters

- **dist** (setuptools.dist) – distribution object as
- **keyword** (*str*) – keyword argument = 'use_pyscaffold'
- **value** – value of the keyword argument

pyscaffold.integration.**version2str**(*version*)
Creates a PEP440 version string

Parameters **version** (setuptools_scm.version.ScmVersion) – version object

Returns version string

Return type str

9.1.6 pyscaffold.pytest_runner module

This module provides a test runner for setup.py copied over from https://bitbucket.org/pytest-dev/pytest-runner/ in order to make some improvements.

This file is MIT licensed:

Copyright (c) 2011 Jason R. Coombs <jaraco@jaraco.com>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

class pyscaffold.pytest_runner.**PyTest**(*dist*, ***kw*)
Bases: setuptools.command.test.test

finalize_options()

initialize_options()

static marker_passes(*marker*)
Given an environment marker, return True if the marker is valid and matches this environment.

run()
Override run to ensure requirements are available in this session (but don't install them anywhere).

run_tests()

user_options = [('extras', None, 'Install (all) setuptools extras when running tests')

9.1.7 pyscaffold.repo module

Functionality for working with a git repository

pyscaffold.repo.**add_tag**(*project*, *tag_name*, *message=None*)
Add an (annotated) tag to the git repository.

Parameters

- **project** (*str*) – path to the project
- **tag_name** (*str*) – name of the tag
- **message** (*str*) – optional tag message

pyscaffold.repo.**get_git_root**(*default=None*)
Return the path to the top-level of the git repository or *default*.

Parameters **default** (*str*) – if no git root is found, default is returned

Returns top-level path or *default*

Return type str

pyscaffold.repo.**git_tree_add**(*struct*, *prefix=''*)
Adds recursively a directory structure to git

Parameters

- **struct** (*dict*) – directory structure as dictionary of dictionaries
- **prefix** (*str*) – prefix for the given directory structure

pyscaffold.repo.**init_commit_repo**(*project*, *struct*)
Initialize a git repository

Parameters

- **project** (*str*) – path to the project
- **struct** (*dict*) – directory structure as dictionary of dictionaries

pyscaffold.repo.**is_git_repo**(*folder*)
Check if a folder is a git repository

Parameters **folder** (*str*) – path

9.1.8 pyscaffold.shell module

Shell commands like git, django-admin.py etc.

class pyscaffold.shell.**ShellCommand**(*command*, *shell=True*, *cwd=None*)
Bases: object

Shell command that can be called with flags like git('add', 'file')

Parameters

- **command** (*str*) – command to handle
- **shell** (*bool*) – run the command in the shell
- **cwd** (*str*) – current working dir to run the command

pyscaffold.shell.**called_process_error2exit_decorator**(*func*)
Decorator to convert given CalledProcessError to an exit message

This avoids displaying nasty stack traces to end-users

pyscaffold.shell.**django_admin = <pyscaffold.shell.ShellCommand object>**
Command for django-admin.py

pyscaffold.shell.**get_git_cmd**(***args*)
Retrieve the git shell command depending on the current platform

Parameters ****args** – additional keyword arguments to *ShellCommand*

pyscaffold.shell.**git = <pyscaffold.shell.ShellCommand object>**
Command for git

9.1.9 pyscaffold.structure module

Functionality to generate and work with the directory structure of a project

class pyscaffold.structure.**FileOp**
Bases: object

Namespace for file operations during an update:

- NO_OVERWRITE: Do not overwrite an existing file during update
- NO_CREATE: Do not create the file during an update

NO_CREATE = 1

NO_OVERWRITE = 0

pyscaffold.structure.**add_namespace**(*opts*, *struct*)
Prepend the namespace to a given file structure

Parameters

- **opts** (*dict*) – options of the project
- **struct** (*dict*) – directory structure as dictionary of dictionaries

Returns directory structure as dictionary of dictionaries

Return type dict

pyscaffold.structure.**apply_update_rules**(*rules*, *struct*, *prefix=None*)
Apply update rules using *FileOp* to a directory structure

Parameters

- **rules** (*dict*) – directory structure as dictionary of dictionaries with *FileOp* keys. The structure will be modified.

- **struct** (*dict*) – directory structure as dictionary of dictionaries
- **prefix** (*str*) – prefix path for the structure

Returns directory structure with keys removed according to the rules

Return type dict

pyscaffold.structure.**create_cookiecutter**(*opts*)
Create a cookie cutter template

Parameters **opts** (*dict*) – options of the project

pyscaffold.structure.**create_django_proj**(*opts*)
Creates a standard Django project with django-admin.py

Parameters **opts** (*dict*) – options of the project

Raises RuntimeError – raised if django-admin.py is not installed

pyscaffold.structure.**create_structure**(*struct*, *prefix=None*, *update=False*)
Manifests a directory structure in the filesystem

Parameters

- **struct** (*dict*) – directory structure as dictionary of dictionaries
- **prefix** (*str*) – prefix path for the structure
- **update** (*bool*) – update an existing directory structure

Raises RuntimeError – raised if content type in struct is unknown

pyscaffold.structure.**make_structure**(*opts*)
Creates the project structure as dictionary of dictionaries

Parameters **opts** (*dict*) – options of the project

Returns structure as dictionary of dictionaries

Return type dict

9.1.10 pyscaffold.utils module

Miscellaneous utilities and tools

pyscaffold.utils.**best_fit_license**(*txt*)
Finds proper license name for the license defined in txt

Parameters **txt** (*str*) – license name

Returns license name

Return type str

pyscaffold.utils.**chdir**(*path*)
Contextmanager to change into a directory

Parameters **path** (*str*) – path to change current working directory to

pyscaffold.utils.**check_setuptools_version**()
Check that setuptools has all necessary capabilities for setuptools_scm

Raises RuntimeError – raised if necessary capabilities are not met

pyscaffold.utils.**exceptions2exit**(*exception_list*)
Decorator to convert given exceptions to exit messages

This avoids displaying nasty stack traces to end-users

Parameters **[Exception]** (*exception_list*) – list of exceptions to convert

pyscaffold.utils.**get_files**(*pattern*)
Retrieve all files in the current directory by a pattern.

Use ** as greedy wildcard and * as non-greedy wildcard.

Parameters **pattern** (*str*) – pattern as used by distutils.filelist.Filelist

Returns list of files

Return type [str]

pyscaffold.utils.**is_valid_identifier**(*string*)
Check if string is a valid package name

Parameters **string** (*str*) – package name

Returns True if string is valid package name else False

Return type bool

pyscaffold.utils.**levenshtein**(*s1*, *s2*)
Calculate the Levenshtein distance between two strings

Parameters

- **s1** (*str*) – first string
- **s2** (*str*) – second string

Returns distance between s1 and s2

Return type int

pyscaffold.utils.**list2str**(*lst*, *indent=0*, *brackets=True*, *quotes=True*, *sep=', '*)
Generate a Python syntax list string with an indention

Parameters

- **lst** (*[str]*) – list of strings
- **indent** (*int*) – indention
- **brackets** (*bool*) – surround the list expression by brackets
- **quotes** (*bool*) – surround each item with quotes
- **sep** (*str*) – separator for each item

Returns string representation of the list

Return type str

pyscaffold.utils.**make_valid_identifier**(*string*)
Try to make a valid package name identifier from a string

Parameters **string** (*str*) – invalid package name

Returns valid package name as string or RuntimeError

Return type str

Raises RuntimeError – raised if identifier can not be converted

pyscaffold.utils.**prepare_namespace**(*namespace_str*)
Check the validity of namespace_str and split it up into a list

Parameters **namespace_str** (*str*) – namespace, e.g. “com.blue_yonder”

Returns list of namespaces, e.g. [“com”, “com.blue_yonder”]

Return type [str]

Raises RuntimeError – raised if namespace is not valid

pyscaffold.utils.**utf8_decode**(*string*)
Decode a Python 2 str object to unicode for compatibility with Python 3

Parameters **string** (*str*) – Python 2 str object or Python 3 str object

Returns Python 2 unicode object or Python 3 str object

Return type str

pyscaffold.utils.**utf8_encode**(*string*)
Encode a Python 2 unicode object to str for compatibility with Python 3

Parameters **string** (*str*) – Python 2 unicode object or Python 3 str object

Returns Python 2 str object or Python 3 str object

Return type str

9.1.11 Module contents

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pyscaffold, 40
pyscaffold.cli, 33
pyscaffold.contrib, 29
pyscaffold.info, 34
pyscaffold.integration, 34
pyscaffold.pytest_runner, 35
pyscaffold.repo, 36
pyscaffold.shell, 36
pyscaffold.structure, 37
pyscaffold.templates, 30
pyscaffold.utils, 38

Index

A

add_dir_to_syspath() (in module pyscaffold.contrib), 30
add_namespace() (in module pyscaffold.structure), 37
add_tag() (in module pyscaffold.repo), 36
apply_update_rules() (in module pyscaffold.structure), 37
authors() (in module pyscaffold.templates), 30

B

best_fit_license() (in module pyscaffold.utils), 38
build_cmd_docs() (in module pyscaffold.integration), 34

C

called_process_error2exit_decorator() (in module pyscaffold.shell), 37
changes() (in module pyscaffold.templates), 30
chdir() (in module pyscaffold.utils), 38
check_setuptools_version() (in module pyscaffold.utils), 38
conftest_py() (in module pyscaffold.templates), 30
coveragerc() (in module pyscaffold.templates), 30
create_cookiecutter() (in module pyscaffold.structure), 38
create_django_proj() (in module pyscaffold.structure), 38
create_project() (in module pyscaffold.cli), 33
create_structure() (in module pyscaffold.structure), 38

D

deactivate_pbr_authors_changelog() (in module pyscaffold.integration), 34
django_admin (in module pyscaffold.shell), 37

E

email() (in module pyscaffold.info), 34
exceptions2exit() (in module pyscaffold.utils), 38

F

FileOp (class in pyscaffold.structure), 37
finalize_options() (pyscaffold.pytest_runner.PyTest method), 35

G

get_default_opts() (in module pyscaffold.cli), 33
get_files() (in module pyscaffold.utils), 39
get_git_cmd() (in module pyscaffold.shell), 37
get_git_root() (in module pyscaffold.repo), 36
get_template() (in module pyscaffold.templates), 30
git (in module pyscaffold.shell), 37
git_tree_add() (in module pyscaffold.repo), 36
gitignore() (in module pyscaffold.templates), 30
gitignore_empty() (in module pyscaffold.templates), 31
gitlab_ci() (in module pyscaffold.templates), 31

I

import_mod() (in module pyscaffold.contrib), 30
init() (in module pyscaffold.templates), 31
init_commit_repo() (in module pyscaffold.repo), 36
initialize_options() (pyscaffold.pytest_runner.PyTest method), 35
is_git_configured() (in module pyscaffold.info), 34
is_git_installed() (in module pyscaffold.info), 34
is_git_repo() (in module pyscaffold.repo), 36
is_valid_identifier() (in module pyscaffold.utils), 39

L

levenshtein() (in module pyscaffold.utils), 39
license() (in module pyscaffold.templates), 31
list2str() (in module pyscaffold.utils), 39
local_version2str() (in module pyscaffold.integration), 35

M

main() (in module pyscaffold.cli), 33
make_sanity_checks() (in module pyscaffold.cli), 33
make_structure() (in module pyscaffold.structure), 38
make_valid_identifier() (in module pyscaffold.utils), 39
marker_passes() (pyscaffold.pytest_runner.PyTest static method), 35

N

namespace() (in module pyscaffold.templates), 31

NO_CREATE (pyscaffold.structure.FileOp attribute), 37
NO_OVERWRITE (pyscaffold.structure.FileOp attribute), 37

P

parse_args() (in module pyscaffold.cli), 33
pre_commit_config() (in module pyscaffold.templates), 31
prepare_namespace() (in module pyscaffold.utils), 39
project() (in module pyscaffold.info), 34
pyscaffold (module), 40
pyscaffold.cli (module), 33
pyscaffold.contrib (module), 29
pyscaffold.info (module), 34
pyscaffold.integration (module), 34
pyscaffold.pytest_runner (module), 35
pyscaffold.repo (module), 36
pyscaffold.shell (module), 36
pyscaffold.structure (module), 37
pyscaffold.templates (module), 30
pyscaffold.utils (module), 38
pyscaffold_keyword() (in module pyscaffold.integration), 35
PyTest (class in pyscaffold.pytest_runner), 35

R

readme() (in module pyscaffold.templates), 31
requirements() (in module pyscaffold.templates), 31
run() (in module pyscaffold.cli), 34
run() (pyscaffold.pytest_runner.PyTest method), 36
run_tests() (pyscaffold.pytest_runner.PyTest method), 36

S

setup_cfg() (in module pyscaffold.templates), 31
setup_py() (in module pyscaffold.templates), 32
ShellCommand (class in pyscaffold.shell), 36
skeleton() (in module pyscaffold.templates), 32
sphinx_authors() (in module pyscaffold.templates), 32
sphinx_changes() (in module pyscaffold.templates), 32
sphinx_conf() (in module pyscaffold.templates), 32
sphinx_index() (in module pyscaffold.templates), 32
sphinx_license() (in module pyscaffold.templates), 32
sphinx_makefile() (in module pyscaffold.templates), 32

T

test_requirements() (in module pyscaffold.templates), 32
test_skeleton() (in module pyscaffold.templates), 32
tox() (in module pyscaffold.templates), 33
travis() (in module pyscaffold.templates), 33
travis_install() (in module pyscaffold.templates), 33

U

user_options (pyscaffold.pytest_runner.PyTest attribute), 36
username() (in module pyscaffold.info), 34
utf8_decode() (in module pyscaffold.utils), 40
utf8_encode() (in module pyscaffold.utils), 40

V

version2str() (in module pyscaffold.integration), 35