PyScaffold Documentation

Release 2.1

Blue Yonder

April 16, 2015

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

```
putup my_project
```

This will create a new subdirectory `my_project` and serve you a project setup with git repository, setup.py, document and test folder ready 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 scaffold of `my_project` provides you with a lot of *features*. PyScaffold is compatible with Python 2.7, 3.3 and 3.4.

CHAPTER 1

Contents

1.1 Features

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

1.1.1 Configuration & Packaging

All configuration can be done in `setup.cfg` like changing the description, url, classifiers and even console scripts of your project. That means in most cases it is not necessary to tamper with `setup.py`.

Run `python setup.py sdist`, `python setup.py bdist` or `python setup.py bdist_wheel` to build a source, binary or wheel distribution. 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 inside your package (`package_data`) or in the root directory of your project (`data_files`) can be configured in `setup.cfg`. To read this data in your code, use:

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

1.1.2 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. To use this feature you need to tag with the format `vMAJOR.MINOR[.REVISION]` , e.g. `v0.0.1` or `v0.1`. The prefix `v` is needed! 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.

1.1.3 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.

In order to use the numpydoc documentation style, the flag `--with-numpydoc` can be specified.

1.1.4 Unittest & Coverage

Run `python setup.py test` to run all unittests defined in the subfolder `tests` with the help of py.test. The py.test plugin pytest-cov is used to automatically generate a coverage report. 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.

1.1.5 Requirements Management

Add the requirements of your project to the `requirements.txt` file 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
```

1.1.6 Licenses

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

1.1.7 Django

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

1.1.8 Cookiecutter

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.

1.1.9 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`.

1.2 Installation

1.2.1 Requirements

The installation of PyScaffold requires:

- 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
```

1.2.2 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]
```

1.2.3 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. This is quite comfortable on the one hand but will also pollute your project with a lot of *egg*-folders. 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

1.3 Configuration

Projects set up with PyScaffold feature an easy package configuration with `setup.cfg`. Here is an example of PyScaffold's own `setup.cfg`:

```
[metadata]
description = Tool for easily putting up the scaffold of a Python project
author = Florian Wilhelm
author_email = Florian.Wilhelm@blue-yonder.com
license = new BSD
url = http://pyscaffold.readthedocs.org/
# Comma separated list of data INSIDE your package to include.
# DO NOT prepend the package name when specifying files and folders.
package_data = data/*
# Add here all kinds of additional classifiers as defined under
# https://pypi.python.org/pypi?%3Aaction=list_classifiers
classifiers = 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 :: 3.3,
              Programming Language :: Python :: 3.4,
              Environment :: Console,
              Intended Audience :: Developers,
              License :: OSI Approved :: BSD License,
              Operating System :: POSIX :: Linux,
              Operating System :: Unix,
              Operating System :: MacOS,
              Operating System :: Microsoft :: Windows

[console_scripts]
# Add here console scripts like:
# hello_world = pyscaffold.module:function
```

```
putup = pyscaffold.runner:run

[data_files]
# Add here data to be included which lies OUTSIDE your package, e.g.
# path/to/destination = files/to/include
# This is equivalent to adding files to MANIFEST.in which is not needed.
# The destination is relative to the root of your virtual environment.
share/pyscaffold = *.rst, *.txt

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

[test]
# html, xml or annotate
cov-report = html
junitxml = junit.xml

[pytest]
# Options for py.test
flakes-ignore =
    doc/conf.py ALL
pep8-ignore =
    doc/conf.py ALL
```

1.4 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.

1.4.1 Chat

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

1.4.2 Bug Reports

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

1.4.3 Code

If you would like to contribute to PyScaffold, fork the main repository on GitHub, 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. Create a branch to hold your changes:

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

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

5. Work on this copy, on your computer, using Git to do the version control. 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
```

6. 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.

1.5 License

Some version-related features in `setup.py` and `_version.py` are taken from Versioneer 0.12 by Brain Warner and were released into the public domain.

1.6 Developers

- Florian Wilhelm <Florian.Wilhelm@blue-yonder.com>

- Felix Wick <Felix.Wick@blue-yonder.com>
- Holger Peters <Holger.Peters@blue-yonder.com>
- Uwe Korn <Uwe.Korn@blue-yonder.com>

1.7 Release Notes

1.7.1 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

1.7.2 Version 2.0.4, 2015-03-17

- Typo and wrong Sphinx usage in the RTD documentation

1.7.3 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

1.7.4 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)

1.7.5 Version 2.0.1, 2015-02-27

- FIX: Installation bug under Windows 7

1.7.6 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

1.7.7 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

1.7.8 Version 1.3.2, 2014-12-02

- Fix: path of Travis install script

1.7.9 Version 1.3.1, 2014-11-24

- Fix: –with-tox tuple bug #28

1.7.10 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

1.7.11 Version 1.2, 2014-10-13

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

1.7.12 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

1.7.13 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

1.7.14 Version 0.9, 2014-07-27

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

1.7.15 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*

1.7.16 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

1.7.17 Version 0.6.1, 2014-05-15

- Fix: Removed six dependency in setup.py

1.7.18 Version 0.6, 2014-05-14

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

1.7.19 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

1.7.20 Version 0.4.1, 2014-04-27

- Problem fixed with pytest-cov installation

1.7.21 Version 0.4, 2014-04-23

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

1.7.22 Version 0.3, 2014-04-18

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

1.7.23 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

1.7.24 Version 0.1.2, 2014-04-10

- Bugfix in Manifest.in
- Python 2.6 problems fixed

1.7.25 Version 0.1.1, 2014-04-10

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

1.7.26 Version 0.1, 2014-04-03

- First release

1.8 pyscaffold

1.8.1 pyscaffold package

Submodules

pyscaffold.info module

Provide general information about the system, user etc.

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

Returns user's email as string

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

Returns boolean

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

Returns boolean

`pyscaffold.info.`**`project`**`(`*args*`)`
Update user settings with the settings of an existing PyScaffold project

Parameters **args** – command line parameters as `argparse.Namespace`

Returns updated command line parameters as `argparse.Namespace`

`pyscaffold.info.`**`read_setup_cfg`**`(`*args*`)`
Read setup.cfg (PyScaffold >= 2.0) for user settings

Parameters **args** – command line parameters as `argparse.Namespace`

Returns updated command line parameters as `argparse.Namespace`

`pyscaffold.info.`**`read_setup_py`**`(`*args*`)`
Read setup.py (PyScaffold < 2.0) for user settings

Parameters **args** – command line parameters as `argparse.Namespace`

Returns updated command line parameters as `argparse.Namespace`

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

Returns user's name as string

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** – path to the project as string
- **tag_name** – name of the tag as string
- **message** – optional tag message as string

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

Parameters

- **struct** – directory structure as dictionary of dictionaries
- **prefix** – prefix for the given directory structure as string

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

Parameters

- **project** – path to the project as string
- **struct** – directory structure as dictionary of dictionaries

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

Parameters **folder** – path as string

pyscaffold.runner module

Command-Line-Interface of PyScaffold

`pyscaffold.runner.`**`main`**`(`*args*`)`
Main entry point of PyScaffold

Parameters **args** – command line parameters as list of strings `pyscaffold.runner.`**`parse_args`**(*args*)

Parse command line parameters

Parameters **args** – command line parameters as list of strings

Returns command line parameters as `argparse.Namespace`

`pyscaffold.runner.`**`prepare_namespace`**(*namespace_str*)

Check the validity of namespace_str and split it up into a list

Parameters **namespace_str** – namespace as string, e.g. "com.blue_yonder"

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

`pyscaffold.runner.`**`run`**(**args*, ***kwargs*)

Entry point for setup.py

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** – command to handle
- **shell** – run the command in the shell
- **cwd** – 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 at 0x7f38e8b337d0>**

Command for django-admin.py

`pyscaffold.shell.`**`get_git_cmd`**(***args*)

Retrieve the git shell command depending on the current platform

All additional parameters are passed to `ShellCommand`

`pyscaffold.shell.`**`git`** **= <pyscaffold.shell.ShellCommand object at 0x7f38e8b332d0>**

Command for git

pyscaffold.structure module

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

`pyscaffold.structure.`**`add_namespace`**(*args*, *struct*)

Prepend the namespace to a given file structure

Parameters

- **args** – command line parameters as `argparse.Namespace`
- **struct** – directory structure as dictionary of dictionaries

Returns directory structure as dictionary of dictionaries

`pyscaffold.structure.`**`check_files_exist`**(*struct*, *prefix=None*)
Checks which files exist in a directory structure

Parameters

- **struct** – directory structure as dictionary of dictionaries
- **prefix** – prefix path for the structure

Returns returns a dictionary of dictionaries where keys representing files exists in the filesystem.

`pyscaffold.structure.`**`create_cookiecutter`**(*args*)

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

Parameters **args** – command line parameters as `argparse.Namespace`

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

Parameters

- **struct** – directory structure as dictionary of dictionaries
- **prefix** – prefix path for the structure
- **update** – update an existing directory structure as boolean

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

Parameters **args** – command line parameters as `argparse.Namespace`

Returns structure as dictionary of dictionaries

`pyscaffold.structure.`**`remove_from_struct`**(*orig_struct*, *del_struct*)
Removes files existing in *del_struct* from structure *orig_struct*

Parameters

- **orig_struct** – directory structure as dictionary of dictionaries
- **del_struct** – directory structure as dictionary of dictionaries

Returns directory structure as dictionary of dictionaries

`pyscaffold.structure.`**`set_default_args`**(*args*)
Set default arguments for some parameters

Parameters **args** – command line parameters as `argparse.Namespace`

Returns command line parameters as `argparse.Namespace`

pyscaffold.templates module

Templates for all files of a project's scaffold

`pyscaffold.templates.`**`authors`**(*args*)
Template of AUTHORS.rst

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

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

Parameters **txt** – license name as string

Returns license name as string

`pyscaffold.templates.`**`changes`**(*args*)
Template of CHANGES.rst

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`coveragerc`**(*args*)
Template of .coveragerc

Parameters **args** – command line parameters as `argparse.Namespace`

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.`**`gitattributes`**(*args*)
Template of .gitattributes

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`gitignore`**(*args*)
Template of .gitignore

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`gitignore_empty`**(*args*)
Template of empty .gitignore

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`init`**(*args*)
Template of __init__.py

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`license`**(*args*)
Template of LICENSE.txt

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

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

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

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

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`readme`**`(`*args*`)`
Template of README.rst

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`requirements`**`(`*args*`)`
Template of requirements.txt

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`setup_cfg`**`(`*args*`)`
Template of setup.cfg

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`setup_py`**`(`*args*`)`
Template of setup.py

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

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

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`sphinx_authors`**`(`*args*`)`
Template of authors.rst

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`sphinx_changes`**`(`*args*`)`
Template of changes.rst

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`sphinx_conf`**`(`*args*`)`
Template of conf.py

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`sphinx_index`**`(`*args*`)`
Template of index.rst

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`sphinx_license`**`(`*args*`)`
Template of license.rst

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

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

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`tox`**`(`*args*`)`
Template of tox.ini

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`travis`**`(`*args*`)`
Template of .travis.yml

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`travis_install`**`(`*args*`)`
Template of travis_install.sh

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

`pyscaffold.templates.`**`version`**`(`*args*`)`
Template of _version.py

Parameters **args** – command line parameters as `argparse.Namespace`

Returns file content as string

pyscaffold.utils module

Miscellaneous utilities and tools

class `pyscaffold.utils.`**`ObjKeeper`**`(`*name*, *bases*, *dct*`)`
Bases: `type`

Metaclass to keep track of generated instances of a class

`instances` = {}

`pyscaffold.utils.`**`capture_objs`**`(`*cls*`)`
Captures the instances of a given class during runtime

param cls class to capture

return dynamic list with references to all instances of `cls`

`pyscaffold.utils.`**`chdir`**`(`**args*, ***kwds*`)`
Contextmanager to change into a directory

Parameters **path** – path to change into as string `pyscaffold.utils.`**`exceptions2exit`**(*exception_list*)

Decorator to convert given exceptions to exit messages

This avoids displaying nasty stack traces to end-users

Parameters **exception_list** – list of exceptions to convert

`pyscaffold.utils.`**`git2pep440`**(*ver_str*)

Converts a git description to a PEP440 conforming string

Parameters **ver_str** – git version description

Returns PEP440 version description

`pyscaffold.utils.`**`is_valid_identifier`**(*string*)

Check if string is a valid package name

Parameters **string** – package name as string

Returns boolean

`pyscaffold.utils.`**`levenshtein`**(*s1*, *s2*)

Calculate the Levenshtein distance between two strings

Parameters

- **s1** – first string
- **s2** – second string

Returns distance between s1 and s2 as integer

`pyscaffold.utils.`**`list2str`**(*lst*, *indent=0*, *brackets=True*, *quotes=True*)

Generate a Python syntax list string with an indention

Parameters

- **lst** – list
- **indent** – indention as integer
- **brackets** – surround the list expression by brackets as boolean
- **quotes** – surround each item with quotes

Returns string

`pyscaffold.utils.`**`make_valid_identifier`**(*string*)

Try to make a valid package name identifier from a string

Parameters **string** – invalid package name as string

Returns valid package name as string or `RuntimeError`

`pyscaffold.utils.`**`safe_get`**(*namespace*, *attr*)

Safely retrieve the value of a namespace's attribute

Parameters

- **namespace** – namespace as `argparse.Namespace` object
- **attr** – attribute name as string

Returns value of the attribute or None

`pyscaffold.utils.`**`safe_set`**(*namespace*, *attr*, *value*)

Safely set an attribute of a namespace object

The new attribute is set only if the attribute did not exist or was None.

Parameters

- **namespace** – namespace as `argparse.Namespace` object
- **attr** – attribute name as string
- **value** – value for new attribute

`pyscaffold.utils.`**`stash`**`(`**args*, ***kwds*`)`

Stashes a file away inside the context and restores it when leaving.

Parameters **filename** – file name as string

`pyscaffold.utils.`**`utf8_decode`**`(`*string*`)`

Decode a Python 2 str object to unicode for compatibility with Python 3

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

Returns Python 2 unicode object or Python 3 str object

`pyscaffold.utils.`**`utf8_encode`**`(`*string*`)`

Encode a Python 2 unicode object to str for compatibility with Python 3

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

Returns Python 2 str object or Python 3 str object

Module contents

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index