blue yonder

Forward looking. Forward thinking.

How to set up a new Python Project

Felix Wick

EuroPython 2014 Berlin

How to set up your Environment?

- you usually have several projects with different dependencies
- need for project-specific Python environments
  - *virtualenv* and *pip*

blue yonder

virtualenv

- tool to create isolated Python environments
- i.e. separate Python interpreter and site-packages
- prerequisite: install *virtualenv* itself globally via *pip* or sources
- create a new virtual environment called `venv`:

```
$ virtualenv venv
```

- this also installs *setuptools* and *pip* into `venv`
- activate `venv` (adds its bin directory at the front of `$PATH`):

```
$ source venv/bin/activate
```

- restore `$PATH`:

```
(venv)$ deactivate
```


blue yonder

pip

- tool to install and manage Python packages
- offers installation from Package Index, source or binary distributions, …
- get and install latest version of package *pyscaffold* from public *PyPI*:

```
(venv)$ pip install pyscaffold
```

- use of requirement specifiers, e.g.:

```
(venv)$ pip install 'pyscaffold>=0.7'
```

- specification of whole environment by requirements file, e.g.:

```
(venv)$ pip install –r requirements.txt
```

- get installed packages in requirements format:

```
(venv)$ pip freeze
pyscaffold==0.7
…
```


blue yonder

How to organize your Project?

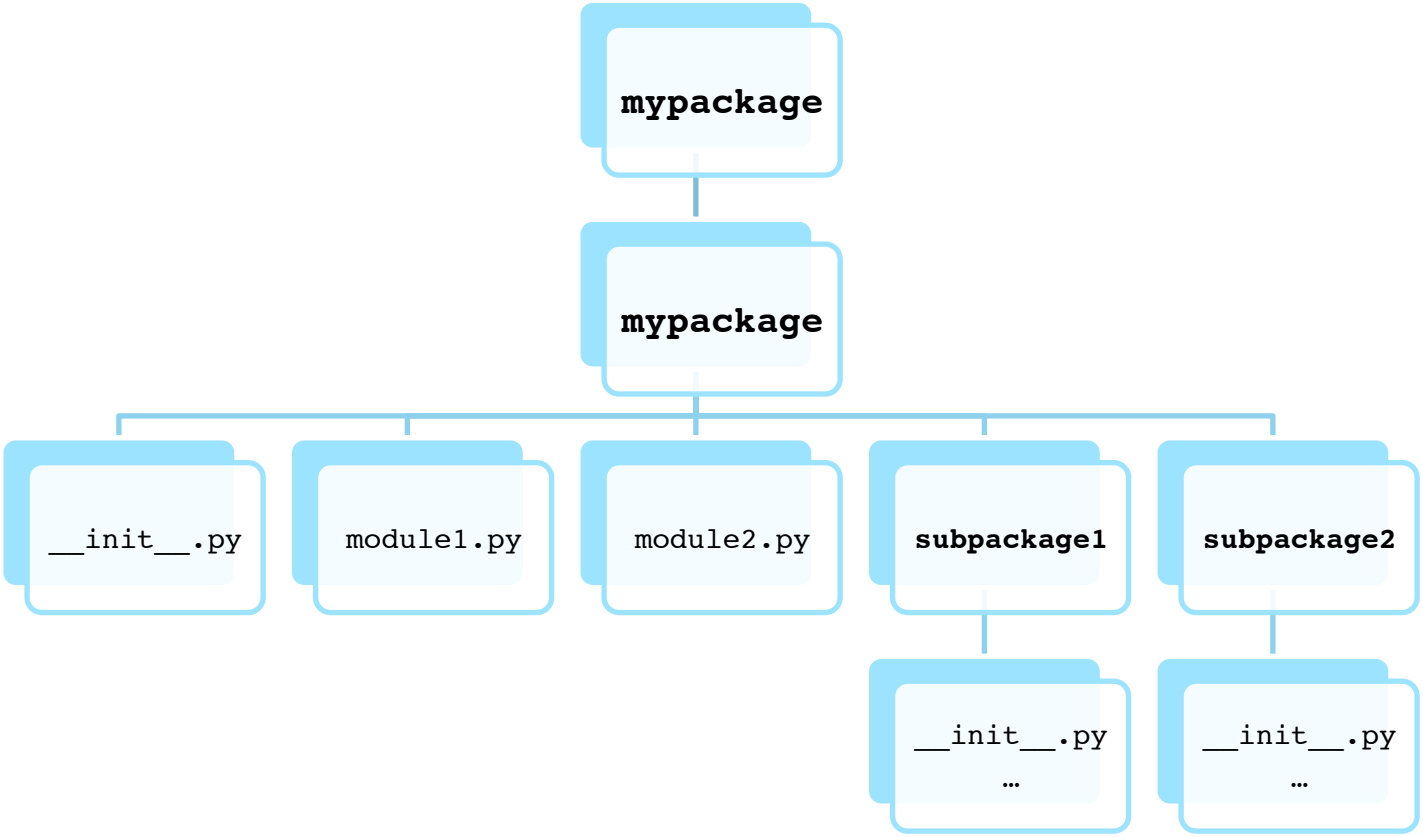
- need for a version control system
  - *git* is a good choice

- need for a reasonable directory structure, unit tests, documentation

blue yonder

Directory Structure

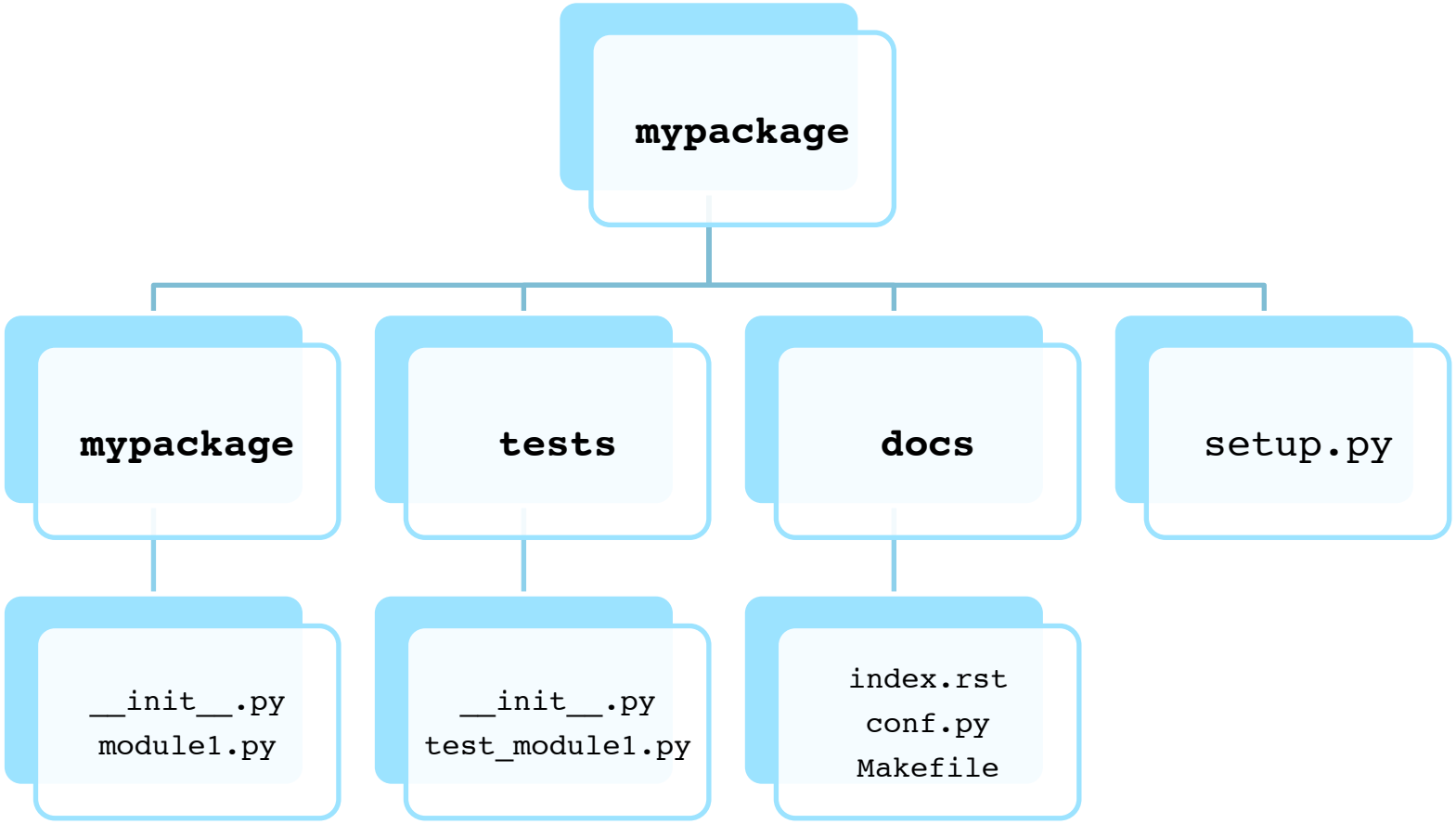
- module: a file containing Python code
- package: a folder containing an `__init__.py` file and Python modules



blue yonder

Tests and Documentation

- unit tests in `tests` folder
- *Sphinx* documentation in `docs` folder



blue yonder

How to distribute your Package?

- use *distutils*, or better *setuptools*
- package installation:

```
(venv)$ python setup.py install
```

- helpful during development (creates links in `venv`'s site-packages):

```
(venv)$ python setup.py develop
```

- source or binary distribution:

```
(venv)$ python setup.py sdist
(venv)$ python setup.py bdist
```


blue yonder

Versioning

- for a new version you need to update:
  - `__version__` attribute in `__init__.py` for self-identification
  - `version` argument in `setup.py` for distribution naming
- cumbersome to do this manually

- *versioneer* manages versions by *git*'s version identifier
- simply use *git* tags to specify releases

blue yonder

Tests and Documentation with *setuptools*

- run the unit tests by means of *unittest* package:

```
(venv)$ python setup.py test
```

- definition of customized commands (`cmdclass` argument in `setup.py`)
- use *pytest* instead of *unittest* when running `setup.py test`
- build *Sphinx* documentation by:

```
(venv)$ python setup.py docs
```


blue yonder

Sample Content of `setup.py`

```
from setuptools import setup, find_packages
setup(name="mypackage",
      version="0.1",
      packages=find_packages(exclude=['tests', 'tests.*']),
      install_requires=['numpy==1.8.1', 'scikit-learn==0.14.1'],
      test_suite="tests",
      entry_points={'console_scripts': ['run = mypackage.module1:run']}
     )
```

- `packages`: to be included in distribution
- `install_requires`: package dependencies
- `test_suite`: tests to be run by `setup.py test`
- `entry points`: functions made available as command-line tools

blue yonder

So…

- to set up a proper Python project, you should prepare:
  - project structure
  - *git* repository
  - `setup.py`
  - versioning

- tough and boring…

blue yonder

Better use *pyscaffold*

- tool to prepare a Python project scaffold
- can be installed via *PyPI,* sources available on *github*
- requires only *virtualenv* and *git* to be installed
- setting up a project as simple as:

```
(venv)$ putup myproject
```

- short demonstration…

blue yonder

Blue Yonder is hiring!

Please visit our booth at EuroPython 2014