HOW BEST TO PACKAGE YOUR LIBRARY?

by Bernat Gabor / @gjbernat / Bloomberg

bit.ly/py-package

Photo by Marcus Cramer / Unsplash - peoples face when they gaze into Python packaging first time

WHO AM I?

- github/gaborbernat
- Maintainer of the virtualenv tool
- PyPa (Python Packaging Authority) member
- Maintainer of the tox tool
- Software Engineer at Bloomberg

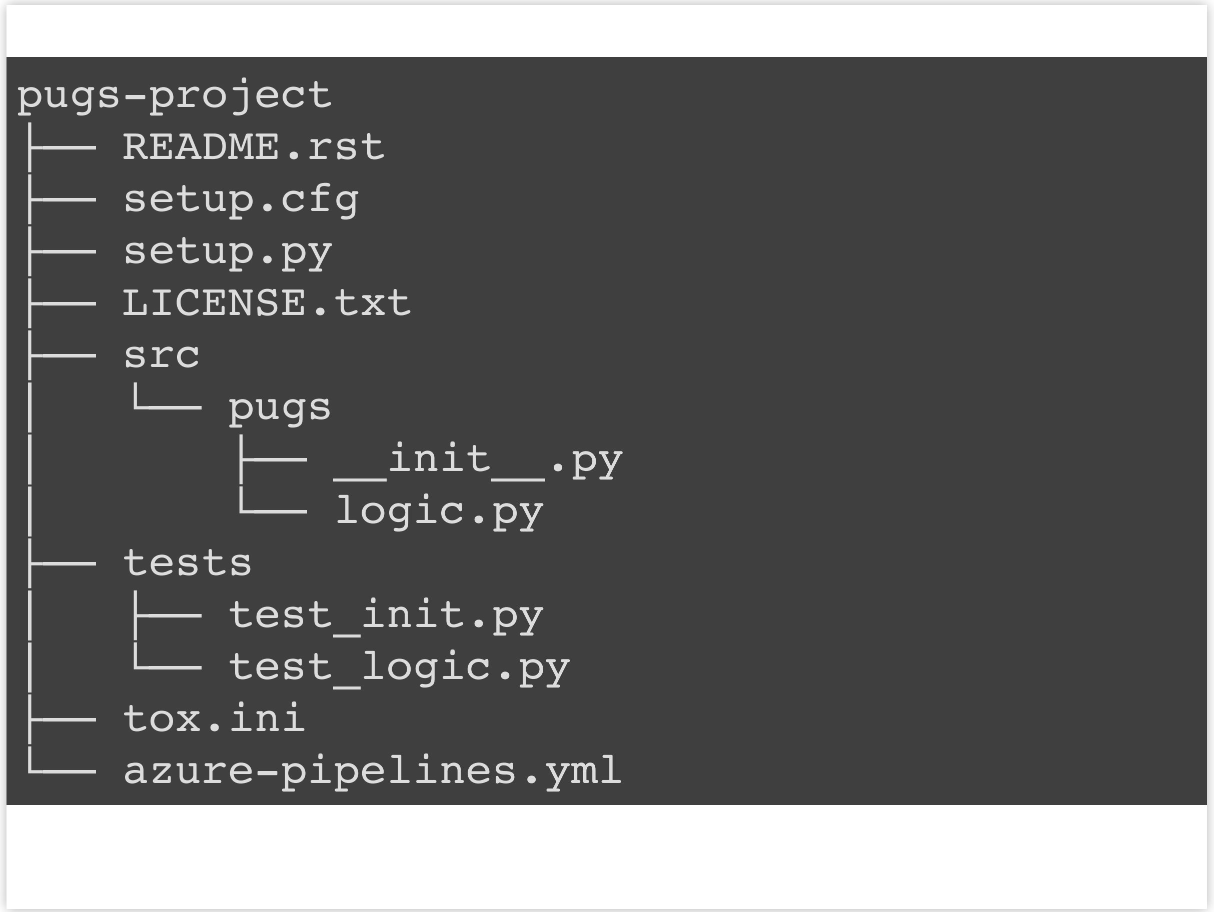
THE STATE OF PYTHON PACKAGING

FROM PYPA POINT OF VIEW

so no tools such as

CONDA

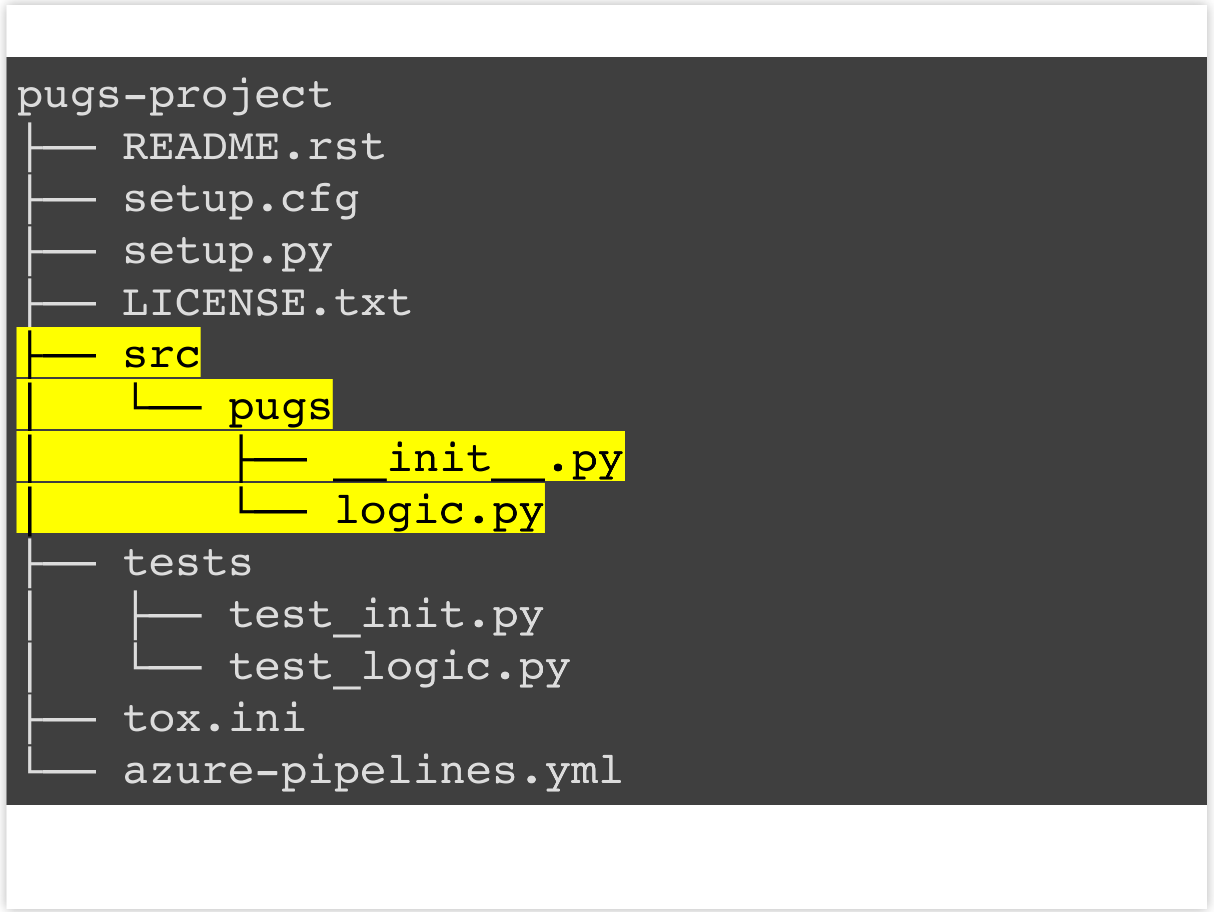
EXAMPLE PROJECT



```
pugs-project
├── README.rst
├── setup.cfg
├── setup.py
├── LICENSE.txt
├── src
│   └── pugs
│       ├── __init__.py
│       └── logic.py
├── tests
│   ├── test_init.py
│   └── test_logic.py
├── tox.ini
└── azure-pipelines.yml
```

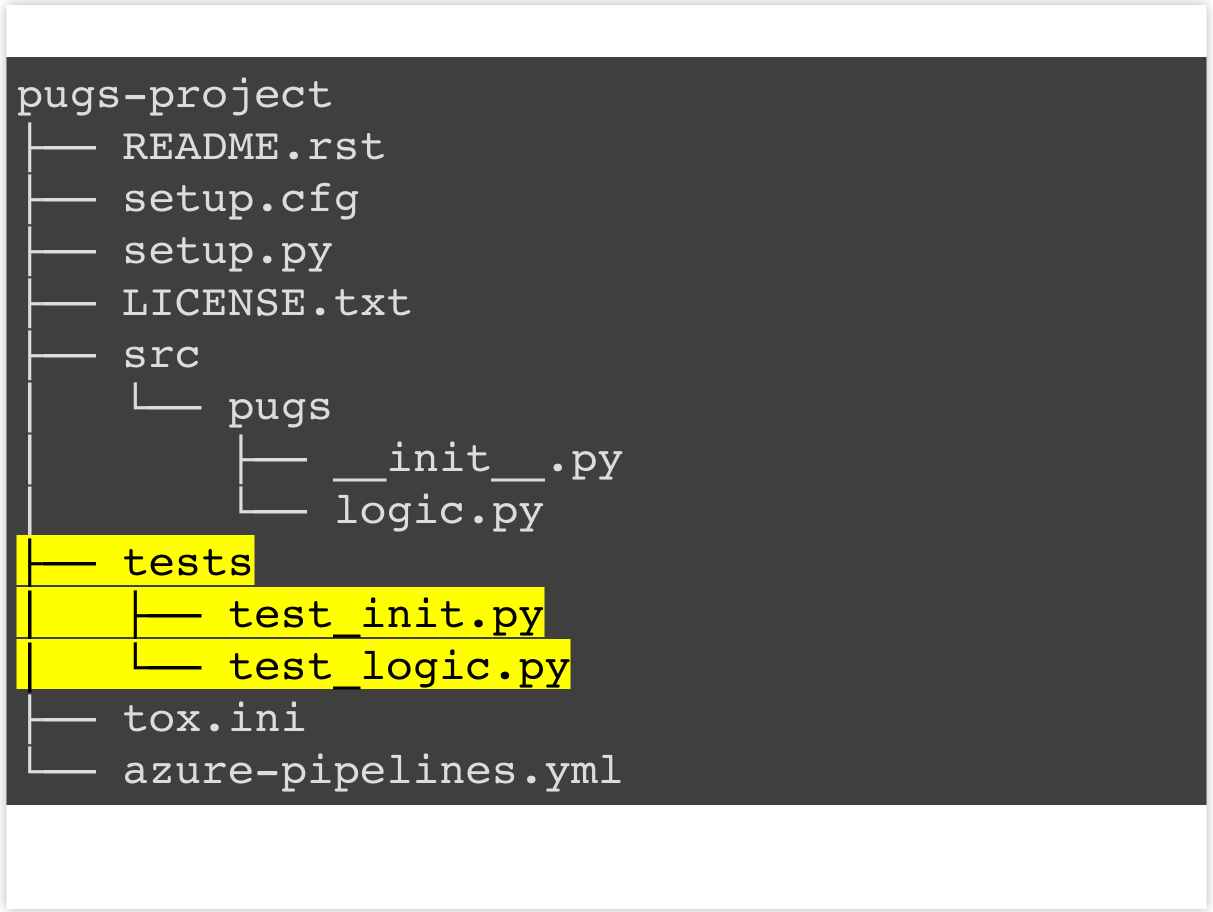
- business logic
- test code
- packaging code
- project management and maintenance: CI/version control

EXAMPLE PROJECT


```
pugs-project
├── README.rst
├── setup.cfg
├── setup.py
├── LICENSE.txt
├── src
│   └── pugs
│       ├── __init__.py
│       └── logic.py
├── tests
│   ├── test_init.py
│   └── test_logic.py
├── tox.ini
└── azure-pipelines.yml
```

- business logic
- test code
- packaging code
- project management and maintenance: CI/version control

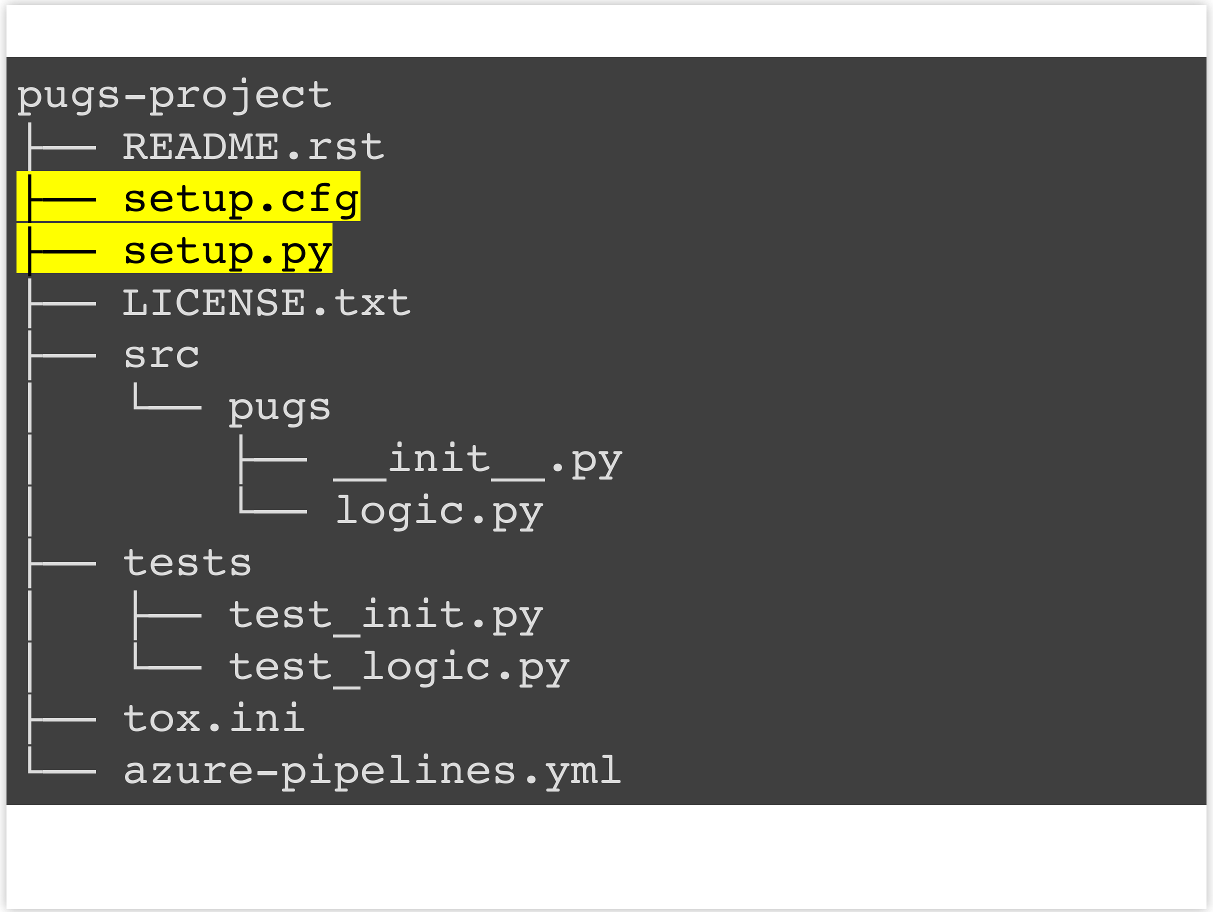
EXAMPLE PROJECT



```
pugs-project
├── README.rst
├── setup.cfg
├── setup.py
├── LICENSE.txt
├── src
│   └── pugs
│       ├── __init__.py
│       └── logic.py
├── tests
│   ├── test_init.py
│   └── test_logic.py
├── tox.ini
└── azure-pipelines.yml
```

- business logic
- test code
- packaging code
- project management and maintenance: CI/version control

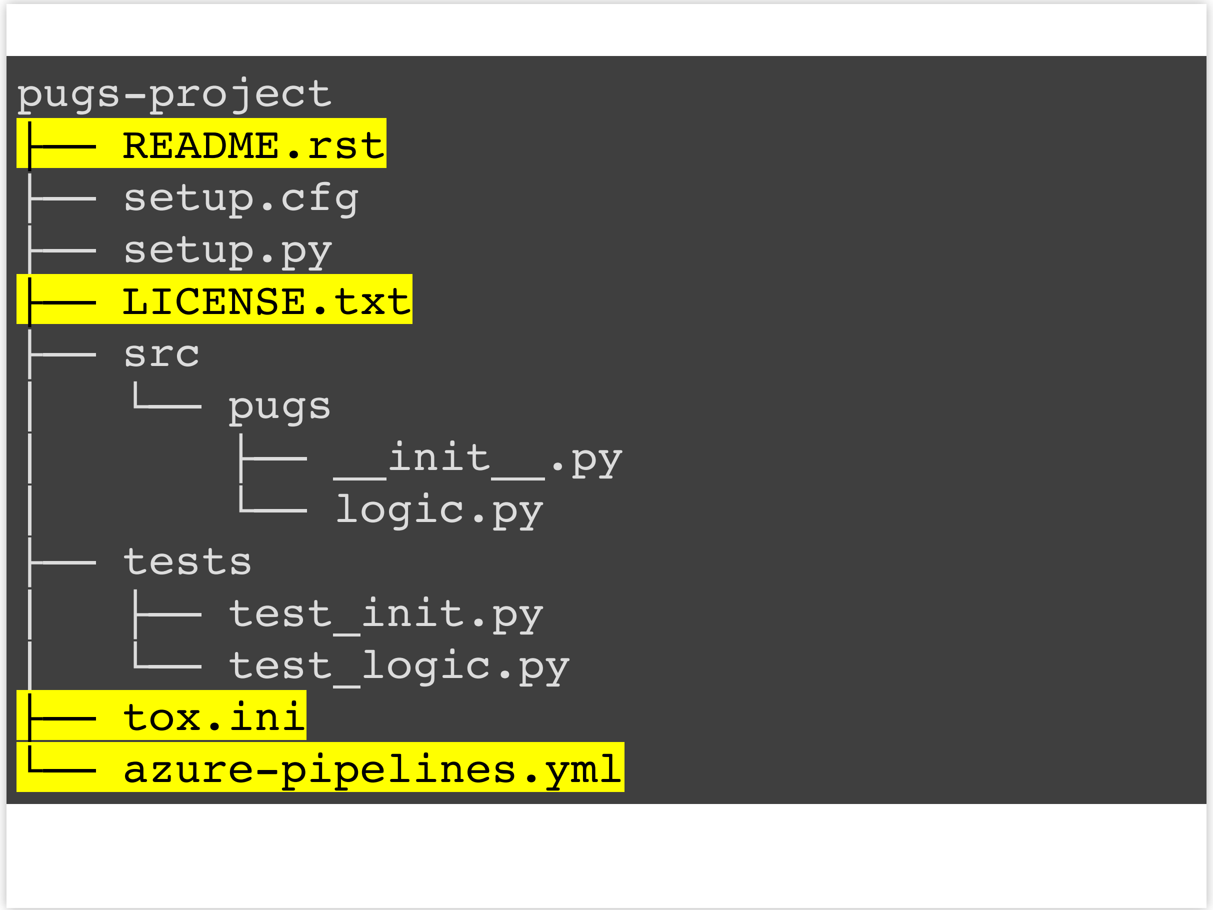
EXAMPLE PROJECT



```
pugs-project
├── README.rst
├── setup.cfg
├── setup.py
├── LICENSE.txt
├── src
│   └── pugs
│       ├── __init__.py
│       └── logic.py
├── tests
│   ├── test_init.py
│   └── test_logic.py
├── tox.ini
└── azure-pipelines.yml
```

- business logic
- test code
- packaging code
- project management and maintenance: CI/version control

EXAMPLE PROJECT



```
pugs-project
├── README.rst
├── setup.cfg
├── setup.py
├── LICENSE.txt
├── src
│   └── pugs
│       ├── __init__.py
│       └── logic.py
├── tests
│   ├── test_init.py
│   └── test_logic.py
├── tox.ini
└── azure-pipelines.yml
```

- business logic
- test code
- packaging code
- project management and maintenance: CI/version control

WHAT DOES IT DO?

```
Python 3.7.2 (v3.7.2:9a3ffc0492, Dec 24 2018, 02:44:43)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pugs
>>> pugs.do_tell()
"An enlightened pug knows how to make the best of whatever he has to work with - A Pug's Guide to Dating -  Gemma Correll"
```

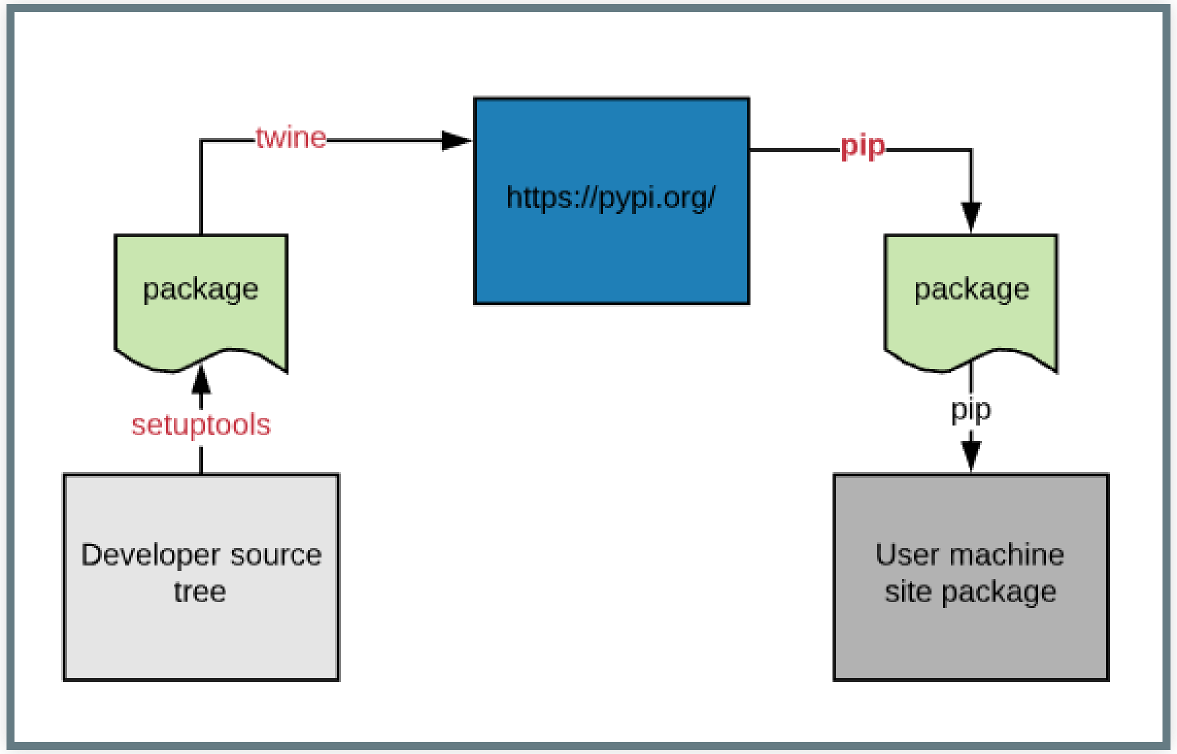
Photo by Charles 🇵🇭 / Unsplash - hmmm

HOW TO MAKE THIS AVAILABLE ON ANOTHER MACHINE?

```
>>> import pugs

>>> pugs
<module 'pugs' from '/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs/__init__.py'>
```

```
>>> import sys
>>> print('\n'.join(sys.path))
/Library/Frameworks/Python.framework/Versions/3.7/lib/python37.zip
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/lib-dynload
/Users/bernat/Library/Python/3.7/lib/python/site-packages
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages
```

HIGH LEVEL PACKAGE ACQUISITION



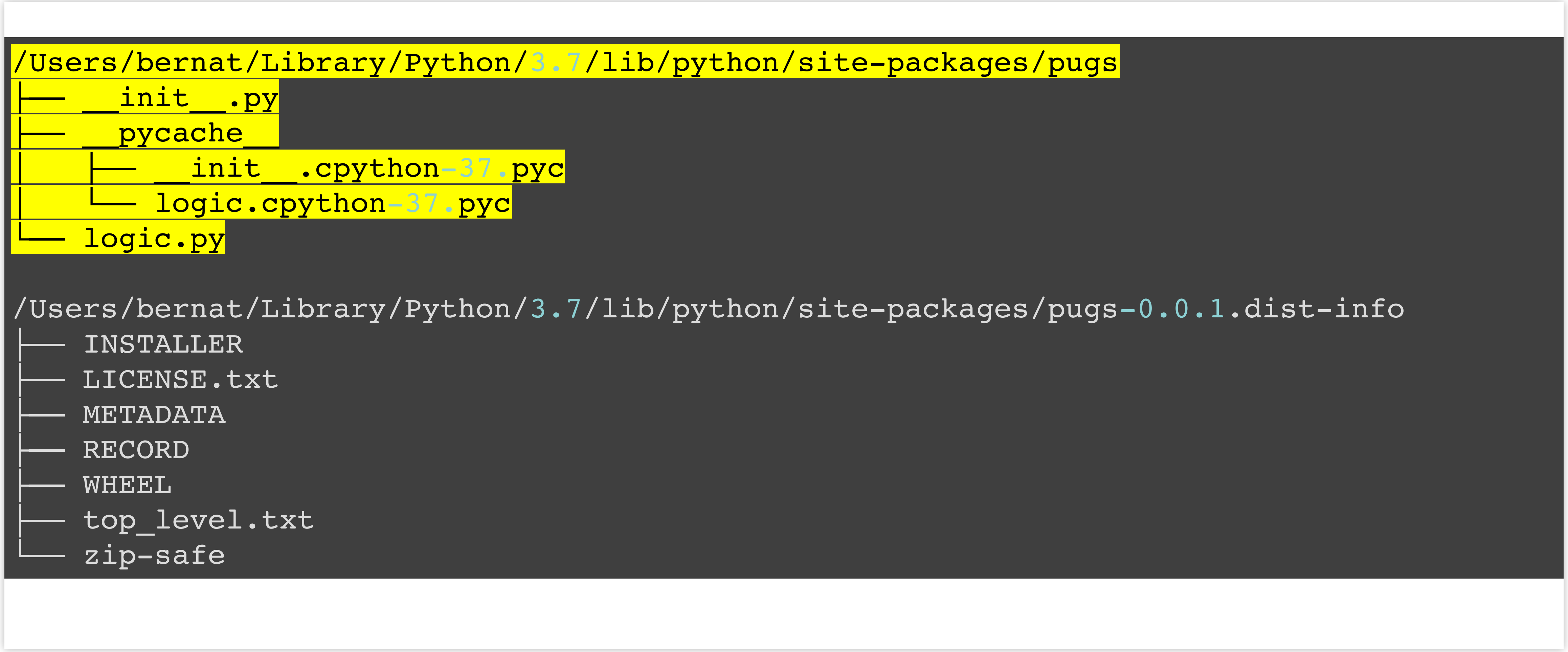
INSTALLED PACKAGE STRUCTURE

- package files
- package metadata: {package}-{version}.dist-info - PEP-427

```
/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-37.pyc
│   └── logic.cpython-37.pyc
└── logic.py

/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs-0.0.1.dist-info
├── INSTALLER
├── LICENSE.txt
├── METADATA
├── RECORD
├── WHEEL
├── top_level.txt
└── zip-safe
```

INSTALLED PACKAGE STRUCTURE

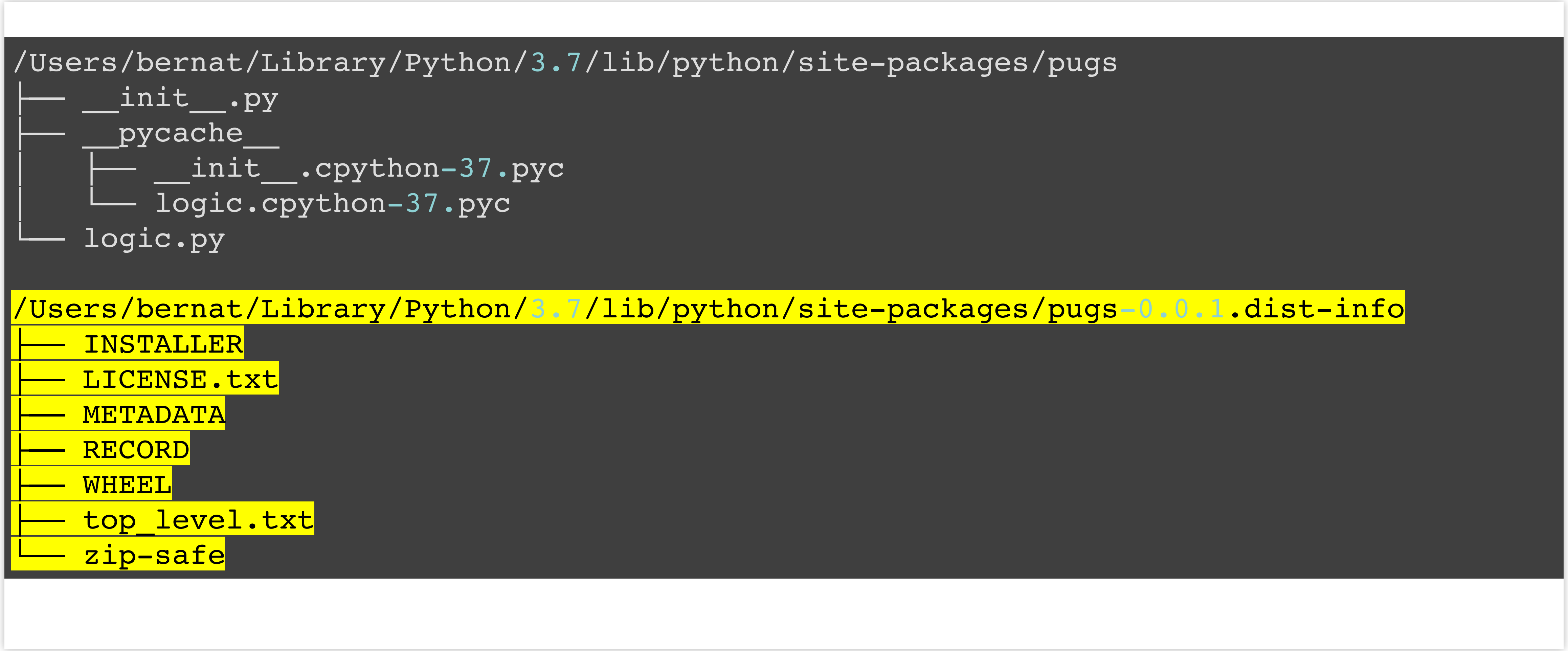
- package files
- package metadata: {package}-{version}.dist-info - PEP-427



```
/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-37.pyc
│   └── logic.cpython-37.pyc
└── logic.py

/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs-0.0.1.dist-info
├── INSTALLER
├── LICENSE.txt
├── METADATA
├── RECORD
├── WHEEL
├── top_level.txt
└── zip-safe
```

INSTALLED PACKAGE STRUCTURE

- package files
- package metadata: {package}-{version}.dist-info - PEP-427



```
/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-37.pyc
│   └── logic.cpython-37.pyc
└── logic.py

/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs-0.0.1.dist-info
├── INSTALLER
├── LICENSE.txt
├── METADATA
├── RECORD
├── WHEEL
├── top_level.txt
└── zip-safe
```

HOW TO GENERATE THE PACKAGES?

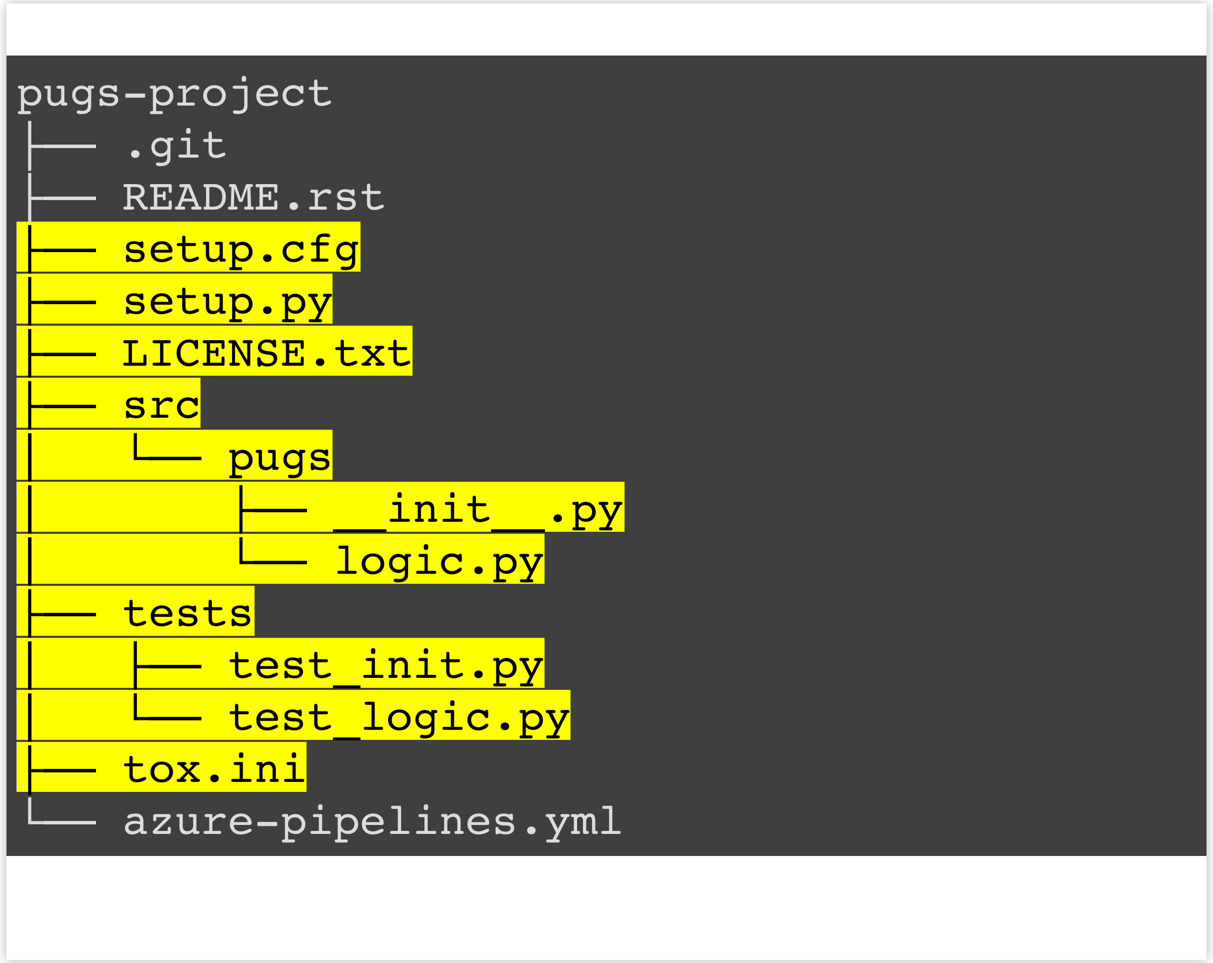
- source distribution - sdist
- wheel distribution

Photo by Ryan Antooa / Unsplash - let's get started, excited!

WHAT IS A SOURCE DISTRIBUTION?

- source tree minus
  - project management files
  - maintainer files
- has business logic, packaging, tests
- all the files needed to install a package from raw source

WHAT IS A SOURCE DISTRIBUTION?

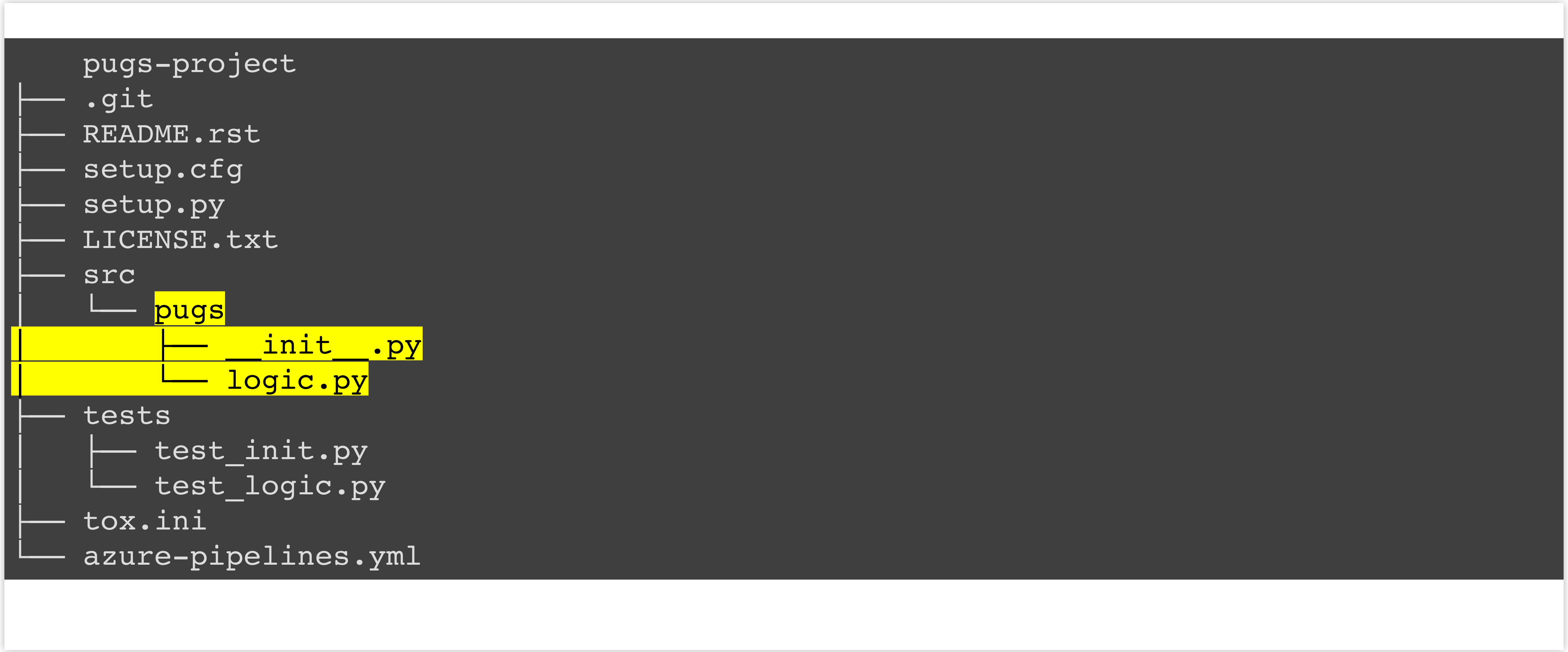


```
pugs-project
├── .git
├── README.rst
├── setup.cfg
├── setup.py
├── LICENSE.txt
├── src
│   └── pugs
│       ├── __init__.py
│       └── logic.py
├── tests
│   ├── test_init.py
│   └── test_logic.py
├── tox.ini
└── azure-pipelines.yml
```

WHAT IS A WHEEL?

- source tree minus
  - project management files
  - maintainer files
  - tests
  - packaging files
- the installed binary files with some meta data

WHAT IS A WHEEL?

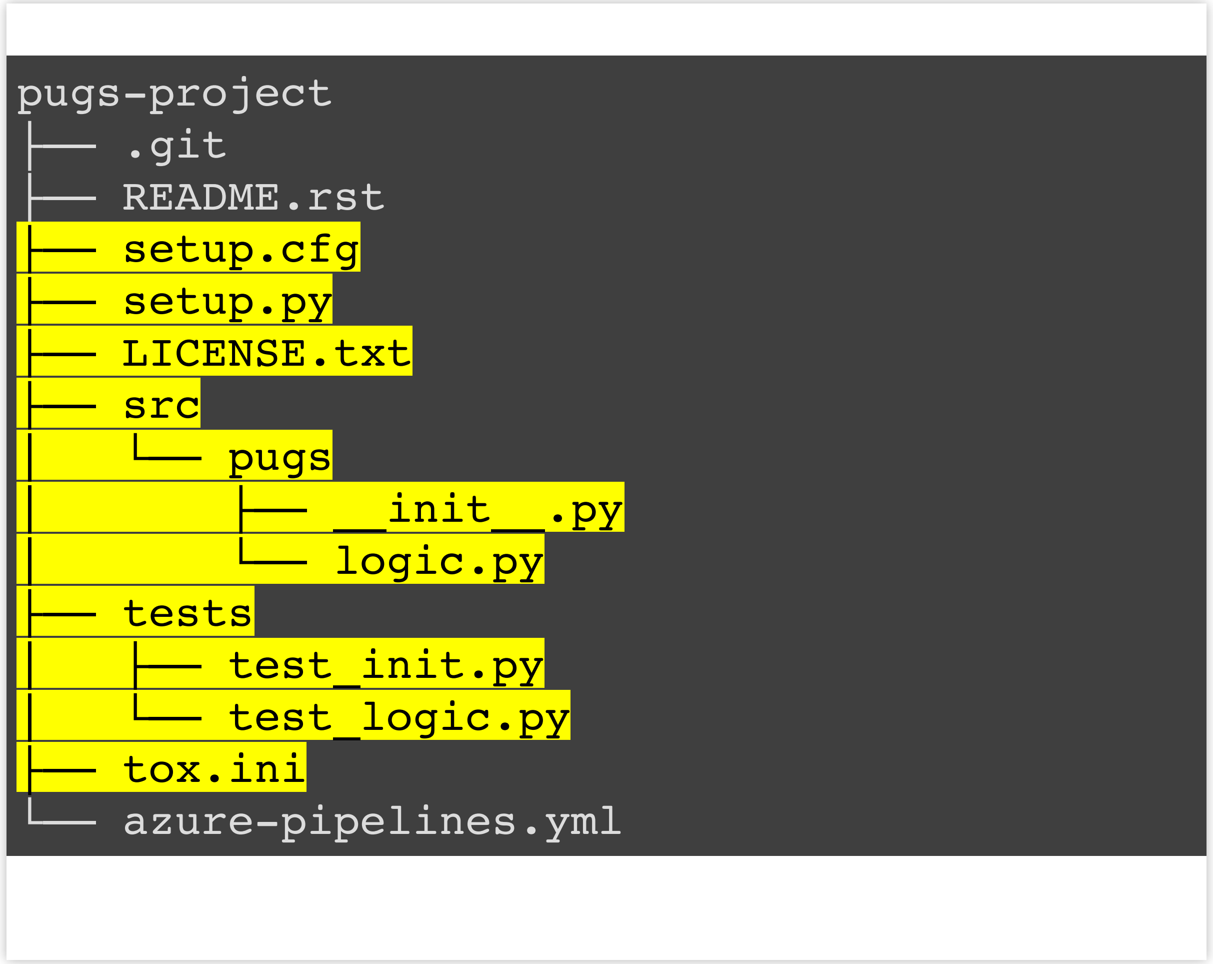


```
    pugs-project
├── .git
├── README.rst
├── setup.cfg
├── setup.py
├── LICENSE.txt
├── src
│   └── pugs
│       ├── __init__.py
│       └── logic.py
├── tests
│   ├── test_init.py
│   └── test_logic.py
├── tox.ini
└── azure-pipelines.yml
```

WHAT IS A WHEEL?

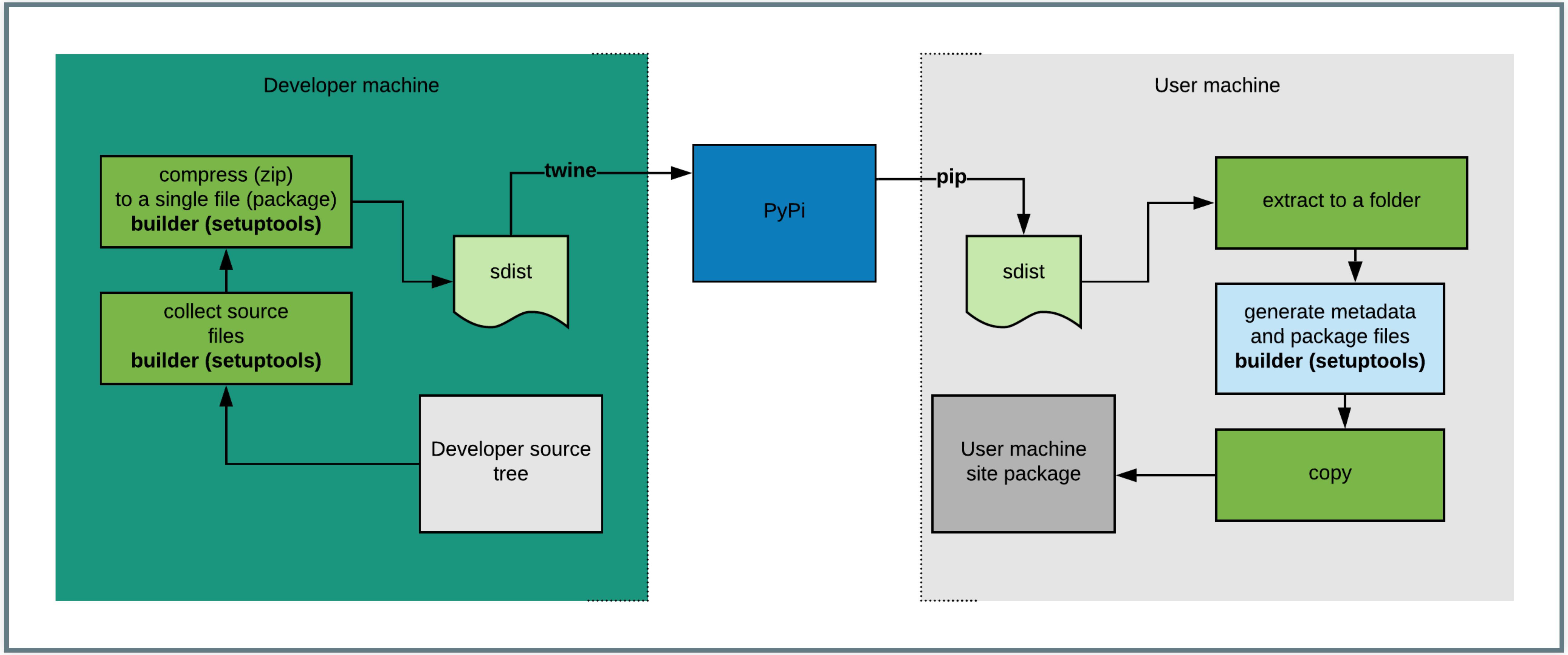
```
/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-37.pyc
│   └── logic.cpython-37.pyc
└── logic.py

/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs-0.0.1.dist-info
├── INSTALLER
├── LICENSE.txt
├── METADATA
├── RECORD
├── WHEEL
├── top_level.txt
└── zip-safe
```

HOW TO SHIP A SOURCE DISTRIBUTION?



```
pugs-project
├── .git
├── README.rst
├── setup.cfg
├── setup.py
├── LICENSE.txt
├── src
│   └── pugs
│       ├── __init__.py
│       └── logic.py
├── tests
│   ├── test_init.py
│   └── test_logic.py
├── tox.ini
└── azure-pipelines.yml
```

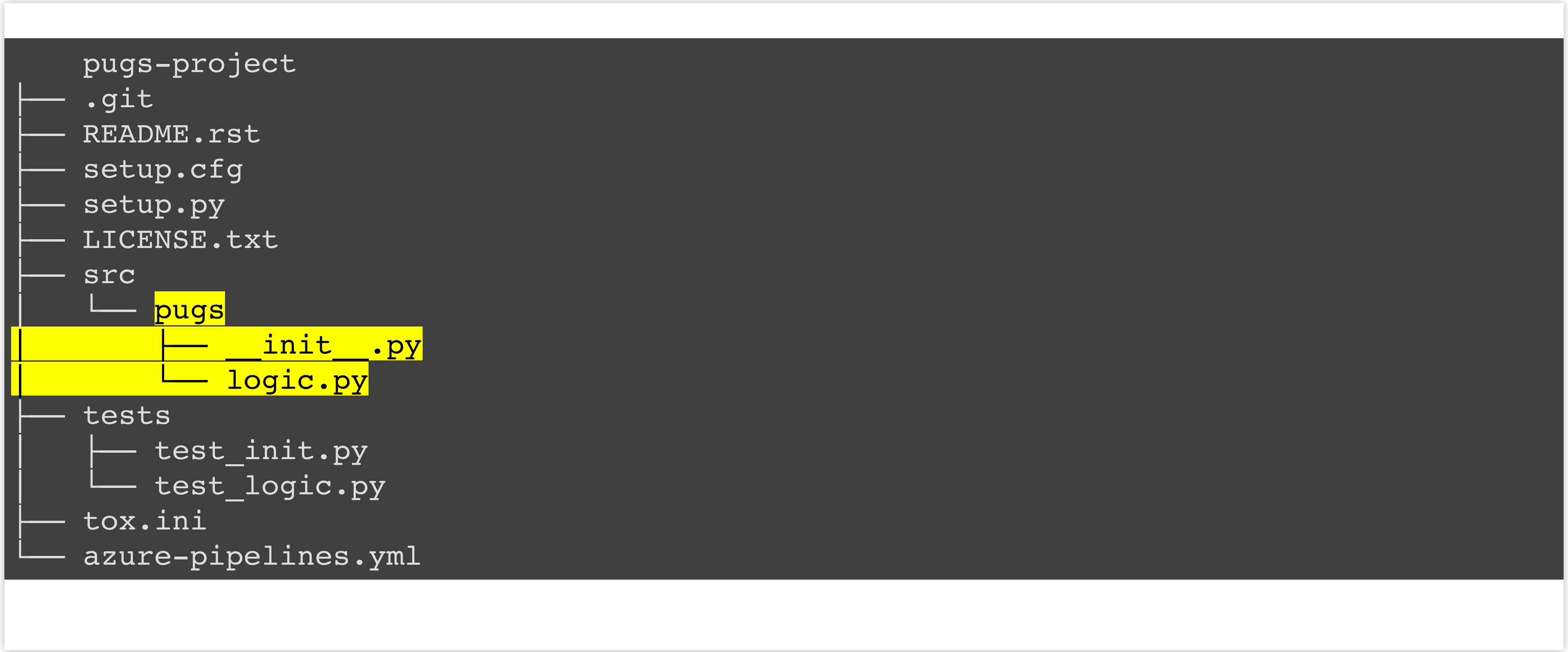
HOW TO SHIP A SOURCE DISTRIBUTION?



WHAT IS A WHEEL?

```
/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-37.pyc
│   └── logic.cpython-37.pyc
└── logic.py

/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs-0.0.1.dist-info
├── INSTALLER
├── LICENSE.txt
├── METADATA
├── RECORD
├── WHEEL
├── top_level.txt
└── zip-safe
```

HOW TO SHIP A WHEEL?

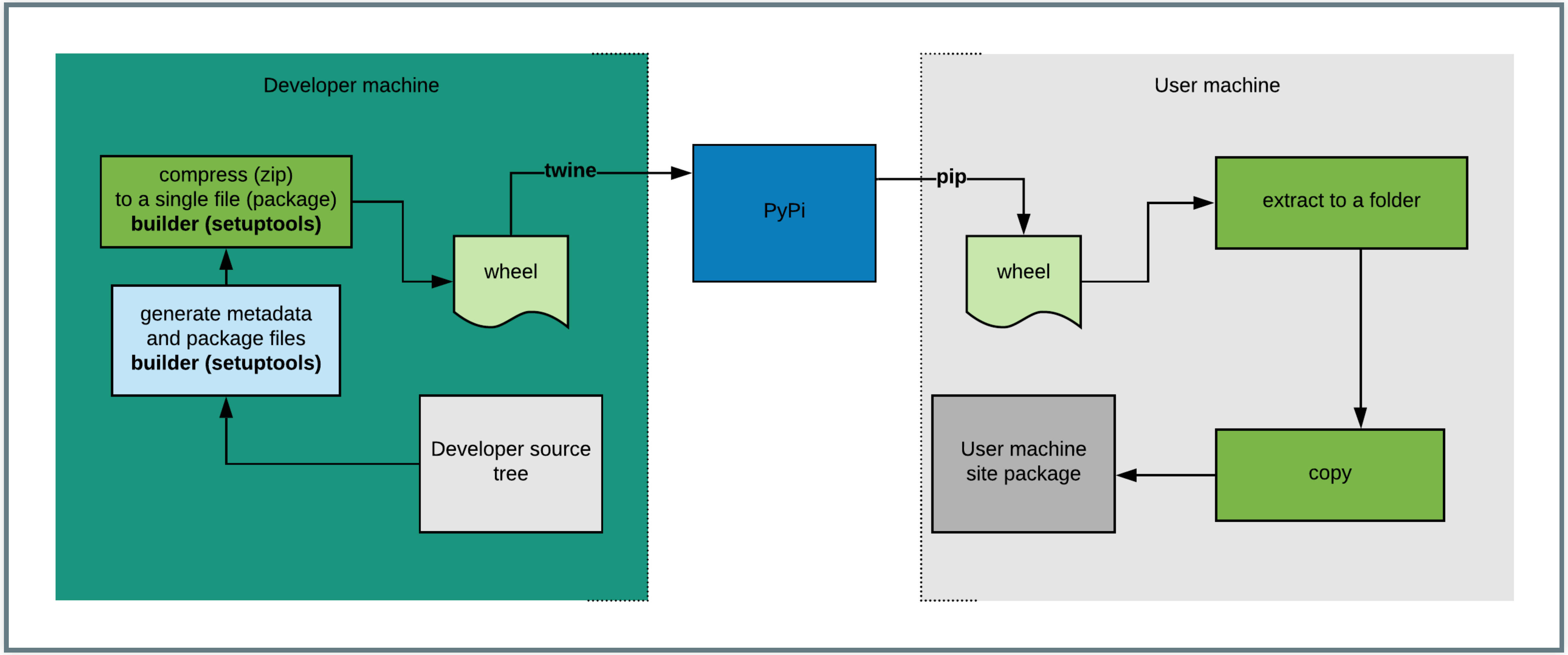


```
    pugs-project
├── .git
├── README.rst
├── setup.cfg
├── setup.py
├── LICENSE.txt
├── src
│   └── pugs
│       ├── __init__.py
│       └── logic.py
├── tests
│   ├── test_init.py
│   └── test_logic.py
├── tox.ini
└── azure-pipelines.yml
```

WHAT IS A WHEEL?

```
/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-37.pyc
│   └── logic.cpython-37.pyc
└── logic.py

/Users/bernat/Library/Python/3.7/lib/python/site-packages/pugs-0.0.1.dist-info
├── INSTALLER
├── LICENSE.txt
├── METADATA
├── RECORD
├── WHEEL
├── top_level.txt
└── zip-safe
```

WHEEL


Developer machine
compress (zip)
to a single file (package)
builder (setuptools)
twine
wheel
generate metadata
and package files
builder (setuptools)
Developer source
tree
PyPi
pip
User machine
wheel
extract to a folder
User machine
site package
copy

GENERATE WHEEL - PEP 427

- select and copy python files
- generate metadata - PEP 376
- generate pyc files
- generate binary library if c-extension package (C/CPP files)
  - get C/C++ toolchain (compiler, dependencies, libraries, etc)
- compress it into a file named PEP 425

```
psutil-5.6.2-cp37-cp37m-win_amd64.whl
```

GENERATE WHEEL - PEP 427

- select and copy python files - requires correct setuptools version
- generate metadata - PEP 376 - requires correct wheels version
- generate pyc files - requires correct python version
- generate binary library if c-extension package (C/CPP files)
  - get C/C++ toolchain (compiler, dependencies, libraries, etc)
- compress it into a file named PEP 425 - requires correct wheels version

```
psutil-5.6.2-cp37-cp37m-win_amd64.whl
```

WHY?

if version 41.0.0 setuptools adds *collect_magic_files*

```
from setuptools import setup
setup(collect_magic_files=True)
```

running with version 40.0.0 will not work

BUCKLE UP FOR A BIT OF HISTORY

Photo by Charles 🇵🇭 / Unsplash - ehhh

HISTORY OF PACKAGING

- 2000 - Python 1.6 - distutils - setup.py introduced
- 2004 - setuptools - quickly became de facto standard
- 2008 - pip (replaces easy_install)
- 2014 - wheels
- setuptools/wheels - setup.py arbitrary user code
- 2015 - flit - declarative over dynamic (arbitrary code run)
  - easier to understand
  - harder to get wrong

HOW DOES A BUILD WORK? - SDIST - DEVELOPER MACHINE

- on the source tree invoke the command

```
python setup.py sdist
```

- upload to pypi

```
python setup.py upload
# http connection - post
```

HOW DOES A BUILD WORK? - SDIST - DEVELOPER MACHINE

- on the source tree invoke the command

```
python setup.py sdist
```

- ~~upload to pypi~~

```
~~python setup.py upload~~
~~# http connection - post~~
```

HOW DOES A BUILD WORK? - SDIST - DEVELOPER MACHINE

- on the source tree invoke the command

```
python setup.py sdist
```

- upload to pypi

```
twine upload package-sdist.targ.gz
# https connection - post - guaranteed
```

HOW DOES A BUILD WORK? - SDIST - USER MACHINE

```
pip install pugs==1.0.0
```

- discover the package from PyPi
- download the sdist and extract it
- invoke setuptools to perform the install

```
python setup.py install
```

PROBLEM 1 - BUILD DEPENDENCIES

- setuptools pulled in from the builder python

```
python setup.py sdist
python setup.py install
```

- cryptic error if build dependencies are missing

```
File "setup_build.py", line 99, in run
      from Cython.Build import cythonize
  ImportError: No module named Cython.Build
```

- or even worse no error - incorrect version
- or even worse - user can hijack setuptools and try to inject their own code

PROBLEM 1 - BUILD DEPENDENCIES

Idea: declarative build environment provision

```
python setup.py install
```

- create temporary folder
- create an isolated python environment

```
python -m virtualenv our_build_env
```

- install build dependencies

```
our_build_env/bin/python -m pip dep1 dep2
```

- generate a wheel, that we can simply install afterwards

```
our_build_env/bin/python setup.py bdist_wheel
```

PROBLEM 1 - BUILD DEPENDENCIES

- PEP-518
- specify deps via pyproject.toml

```
[build-system]
requires = [
    "setuptools >= 40.8.0",
    "wheel >= 0.30.0",
    "cython >= 0.29.4",
]
```

- TOML - Tom's Obvious, Minimal Language.
  - standardized
  - not overly complex
  - does not conflict with other files out there - setup.cfg

PROBLEM 1 - BUILD DEPENDENCIES

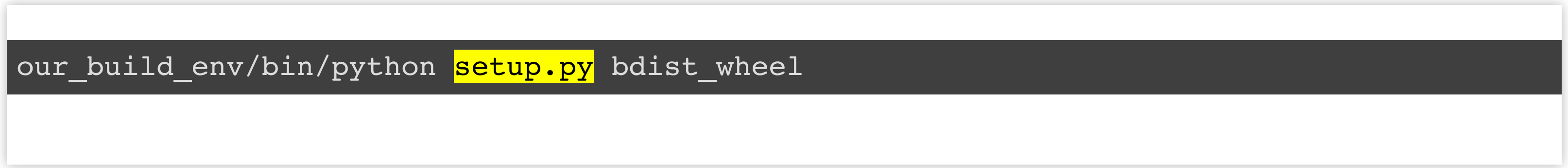
- PEP-518
- **build backend** - performs the package generation in an isolated environment
    - setuptools
    - flit
    - poetry
- **build frontend** - prepares the isolated environment, and invokes the backend
    - pip - PEP-518 support with version 18.0
    - tox - PEP-518 support with version 3.3.0
    - poetry

YAY

Photo by Bruce Galpin / Unsplash - yay!

PROBLEM 2 - DIVERSITY - BETTER USER INTERFACE

- PEP-518 involves still calling to setup.py



```
our_build_env/bin/python setup.py bdist_wheel
```

- setup.py allows arbitrary python code to run - cannot change due to backward compatibility
- flit implementation still involves generating setup.py and adhering to those limitations

PROBLEM 2 - DIVERSITY - BETTER USER INTERFACE

- PEP-517
- So instead of:

```
our_build_env/bin/python setup.py bdist_wheel
```

- Allow build backends to expose invocation end-points:

```
[build-system]
requires = ["flit"]
build-backend = "flit.api:main"
```

- And the build frontend will now do:

```
import flit.api
backend = flit.api.main

backend.build_wheel()  # build wheel via
backend.build_sdist()  # build source distribution via
```

PROBLEM 2 - DIVERSITY - BETTER USER INTERFACE

- PEP-517
- So instead of:

```
our_build_env/bin/python setup.py bdist_wheel
```

- Allow build backends to expose invocation end-points:

```
[build-system]
requires = ["flit"]
build-backend = "flit.api:main"
```

- And the build frontend will now do:

```
import flit.api
backend = flit.api.main

backend.build_wheel()  # build wheel via
backend.build_sdist()  # build source distribution via
```

PROBLEM 2 - DIVERSITY - BETTER USER INTERFACE

- PEP-517
- So instead of:

```
our_build_env/bin/python setup.py bdist_wheel
```

- Allow build backends to expose invocation end-points:

```
[build-system]
requires = ["flit"]
build-backend = "flit.api:main"
```

- And the build frontend will now do:

```
import flit.api
backend = flit.api.main

backend.build_wheel()  # build wheel via
backend.build_sdist()  # build source distribution via
```

PROBLEM 2 - DIVERSITY - BETTER USER INTERFACE

- PEP-517 backend support
  - setuptools - *setuptools.build_meta* - version 40.8+
  - flit - *flit.buildapi*
  - poetry - *poetry.masonry.api* - version 0.12.11
- PEP-517 frontend support
  - pip 19.0
  - tox 3.3.0

CAVEATS

- No editable install support yet
- requires fairly new pip
    - on developer machine - if wheel distribution
    - on user machine - if source distribution

BUILD DEPENDENCIES - DEVELOPER MACHINE

- pip build command planned, for now pep517 package
- to build a ***wheel*** going forward use the following command:

```
python -m pep517.build --binary --out-dir /tmp/build-to .
# old -> python setup.py bdist_wheel
```

- to build a ***sdist*** going forward use the following command:

```
python -m pep517.build --source --out-dir /tmp/build-to .
# old -> python setup.py sdist
```

BENEFITS

- Reproducible declarative builds
- No more need for *setup.py*
- simpler packaging that fails less often

TOX

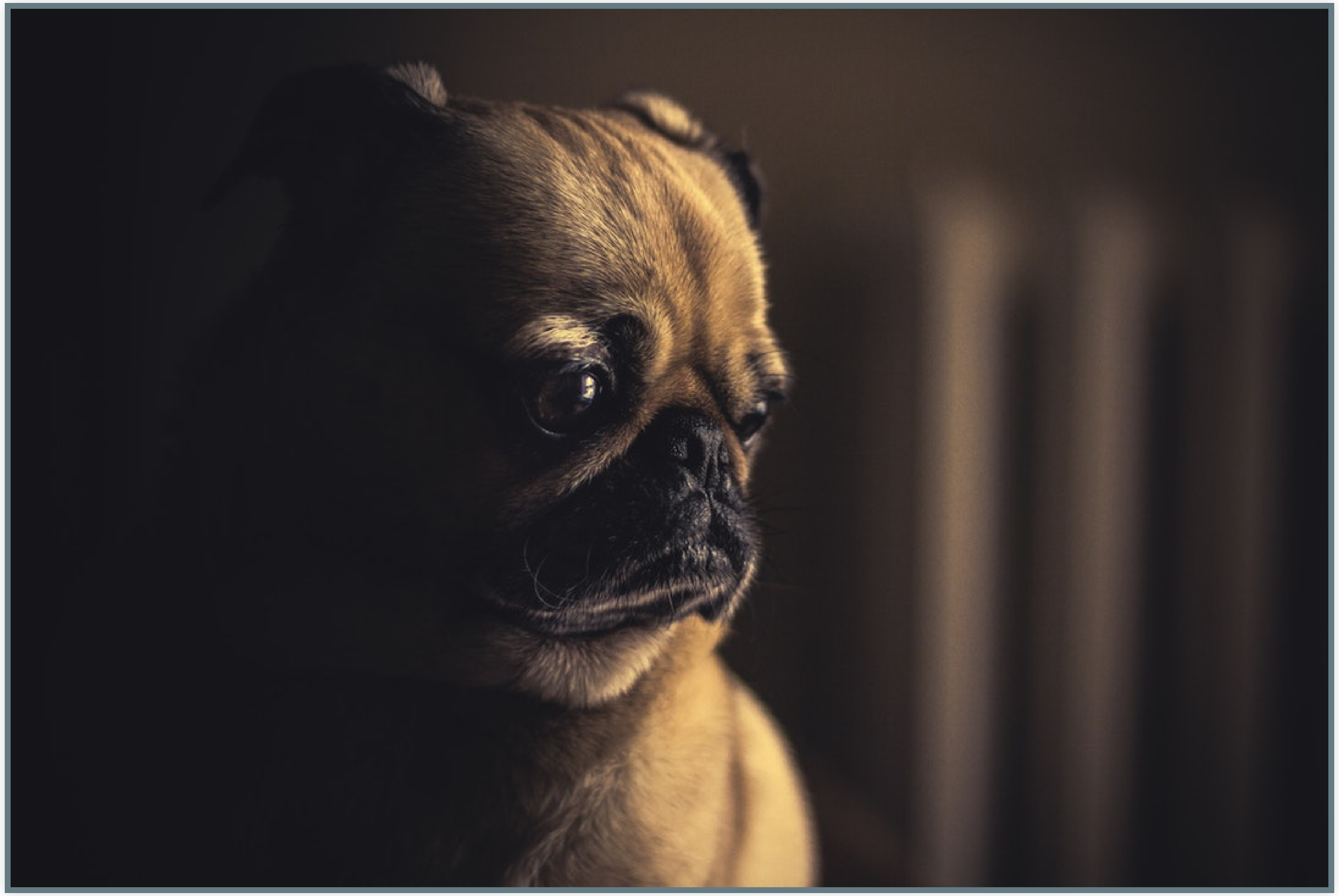


Photo by Matthew Henry / Unsplash - no free lunch here, yet!

TOX

- Not implicitly activated (version 4 probably will have it)
- add

```
[tox]
isolated_build = True
```

- during the packaging phase the source distribution will be built inside an isolated env
- RIP

```
python setup.py sdist
```

SUMMARY

- PEP-518 - ensures declarative and reproducible build environments
- PEP-517 - ensures we no longer build on top of old/legacy setup.py infrastructure

Photo by Yoad Shejtman / Unsplash

SUMMARY

- check-out the new ways to package your code
  - setuptools - general purpose (c-extensions) - now provides setup.cfg only
  - flit - simple pure python apps - pyproject.toml config
  - poetry - one tool to rule them all - pyproject.toml config
  - to build a ***wheel***:

```
python -m pep517.build --binary --out-dir /tmp/build-to .
```

  - to build a ***sdist***:

```
python -m pep517.build --source --out-dir /tmp/build-to .
```

THANK YOU

https://www.bernat.tech/pep-517-and-python-packaging/

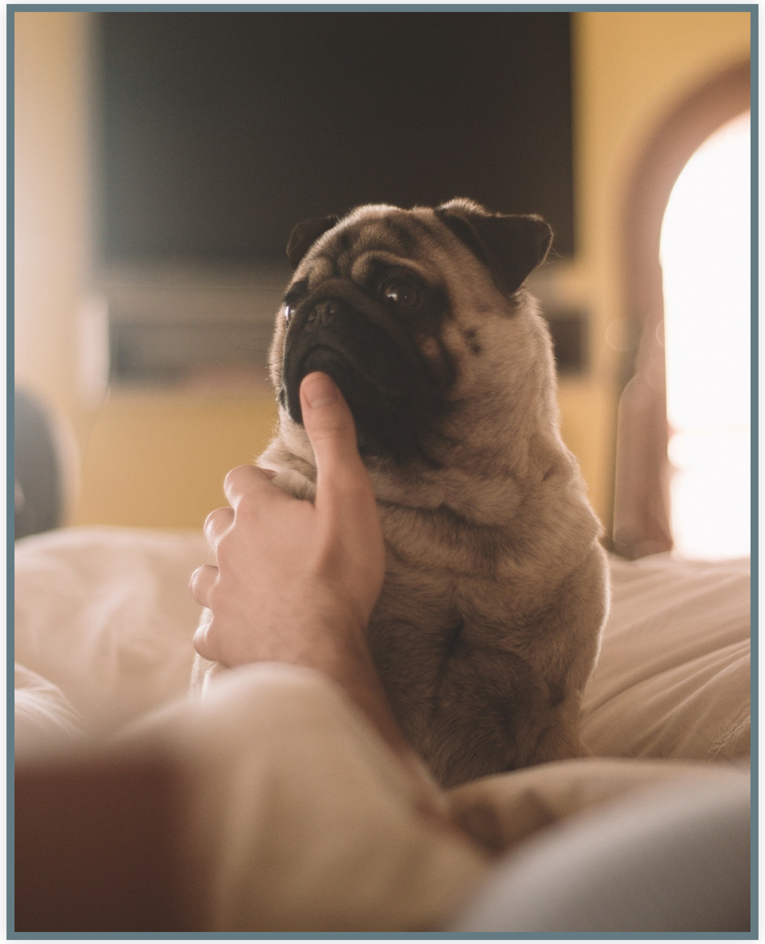


Photo by Milan Popovic / Unsplash - the end