

Python Packaging

for users and devs

2024-05-17

Guilhem Saurel

Available at

[https://homepages.laas.fr/gsaurel/
python-packaging.pdf](https://homepages.laas.fr/gsaurel/python-packaging.pdf)

Under License



<https://creativecommons.org/licenses/by-sa/4.0/>

Source

[https://gitlab.laas.fr/gsaurel/talks :
python-packaging.md](https://gitlab.laas.fr/gsaurel/talks : python-packaging.md)

Discussions

<https://matrix.to/#/room/#allo-pi2:laas.fr>

Introduction

- 1 Use python packages from other people
- 2 Provide your own python packages to other people
- 3 Get an overview of different Package Managers for that

- 1 linux
- 2 macos, *BSD
- 3 windows

- 1 linux
- 2 macos, *BSD
- 3 windows
 - in pure python
 - with WSL

Part 1: Use python packages

Dependency \approx Addiction

Dependency \approx Addiction

- is this dependency essential ?
- can it be made optional ?
- what about its own dependencies ?

- is it pure python ?
- are you confident in its future ?
- are you sure you will be able to handle its updates ?

Either:

- 1 README.md
- 2 requirements.txt
- 3 pyproject.toml

Either:

- 1 README.md
- 2 requirements.txt
- 3 pyproject.toml

So that:

- 1 you won't forget
- 2 you can `pip install -r requirements.txt`
- 3 you can `pip install .`

- Use a venv
- Troubleshooting: \$PYTHONPATH

- Use a venv
- Troubleshooting: \$PYTHONPATH

Also troubleshooting:

```
import sys  
print(sys.path)
```

- keep them up to date
- document your needs
- document what won't work

- constraint graphs grow quickly
- solutions can change over time
- use a lock file with your current working solution

- constraint graphs grow quickly
- solutions can change over time
- use a lock file with your current working solution
- `pip freeze > requirements.lock`

```
Django==4.2.11  
httplib==0.27.0  
ipython==8.23.0  
jedi==0.19.1  
Jinja2==3.1.3  
matplotlib-inline==0.1.6  
numpy==1.26.4  
pandas==2.2.1  
tqdm==4.66.2
```

Part 2: Distribute your packages

- `ruff format`
- `ruff check`

- `ruff format`
- `ruff check`

Use those in your IDE, git hooks, and/or CI

<https://spdx.org/licenses/>

eg.: BSD / MIT / Apache / GPL

- setuptools
- poetry
- flit

- name
- version
- authors
- license
- urls
- dependencies
- entrypoints
- tooling configuration

ref. your builder docs

- `python -m build`
- `pip install .`
- in your CI

- `python -m build`
- `pip install .`
- in your CI

This is enough for other people to use eg.:

```
pip install \  
    git+https://gitlab.laas.fr/gsaurel/ndh
```

- decide a version number: <https://semver.org/>
- document changes between versions:
<https://keepachangelog.com/>
- publish a git tag (bonus points if signed)
- publish package archives (bonus points if signed)

- twine
- flit publish
- poetry publish
- github.com/pypa/gh-action-pypi-publish

Part 3: Some python package managers

- apt
- pacman
- rpm

This is the most simple and most stable solution.

This is the incontournable standard solution.

<https://github.com/jazzband/pip-tools>

Simple dependency constraint declaration + solution pinning

<https://python-poetry.org/>

Full feature and widest adoption.

Should be deprecated in favor of poetry.

<https://pdm-project.org/latest/>

A bit more modern than poetry, but narrower adoption and support.

This will eat your home.

<https://github.com/astral-sh/uv>

The new cool kid.

The perfection you didn't ask for, yet.

The perfection you didn't ask for, yet.

Come to the next formations to know more !

Questions ?

Prior art

- Managing Python Packages (2019)
- Python Tooling (2022)

This presentation

<https://homepages.laas.fr/gsaurel/python-packaging.pdf>

<https://matrix.to/#/room/#allo-pi2:laas.fr>