Dependency Management with Poetry - Simple and Effortless, Yet Better

Open Source Hong Kong x Python User Group Meetup, Oct 2024

Who am I

- Data Scientist
- 4 years of experience in Python

Data Science

Software Development

MLOps

Astrophysics

Just a random small potato :D

QR Codes

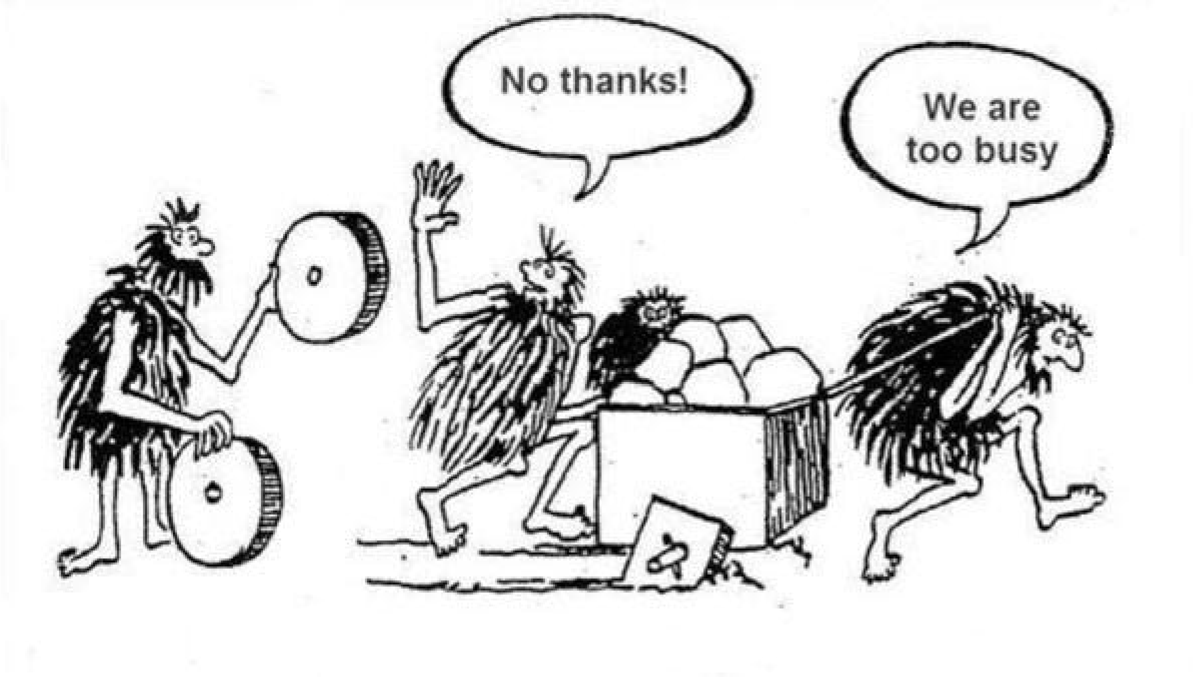
Slides for Today

Code for Today

PyConHK 2024 Ticket

Why Dependency

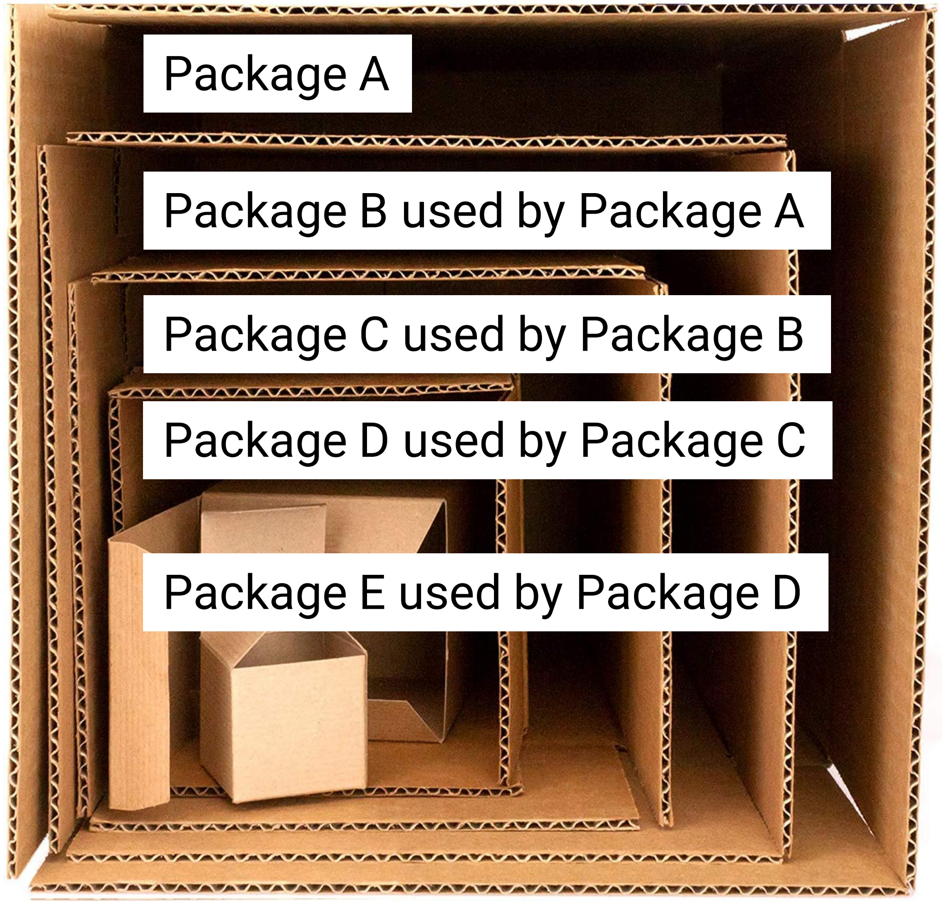
- More efficient development
- Scalability and maintainability
- Leverage developer community



From [Medium]

Why Dependency Management

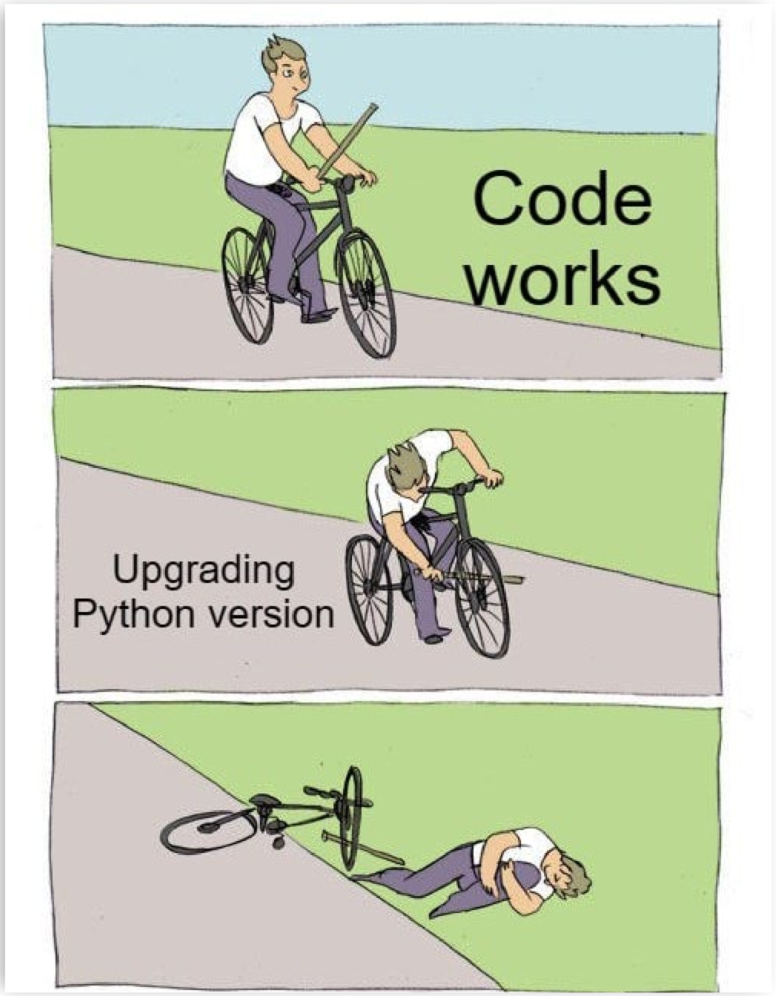
- Maintain consistent development environment
- Handle nested dependencies
- Facilitate reproducibility across machines


Package A
Package B used by Package A
Package C used by Package B
Package D used by Package C
Package E used by Package D

From Amazon

Why Good Dependency Management

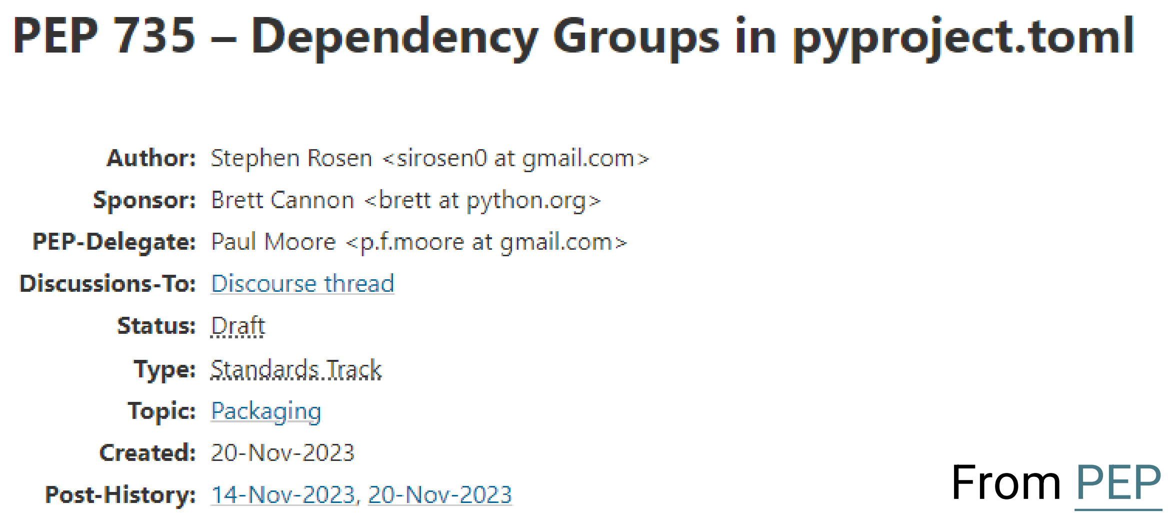
- Avoid breakage during upgrades
- Enhance project stability
- Reduce debugging time
- Simplify collaboration



From Reddit

Python Enhancement Proposals

- Design document
- Proposal of changes or features

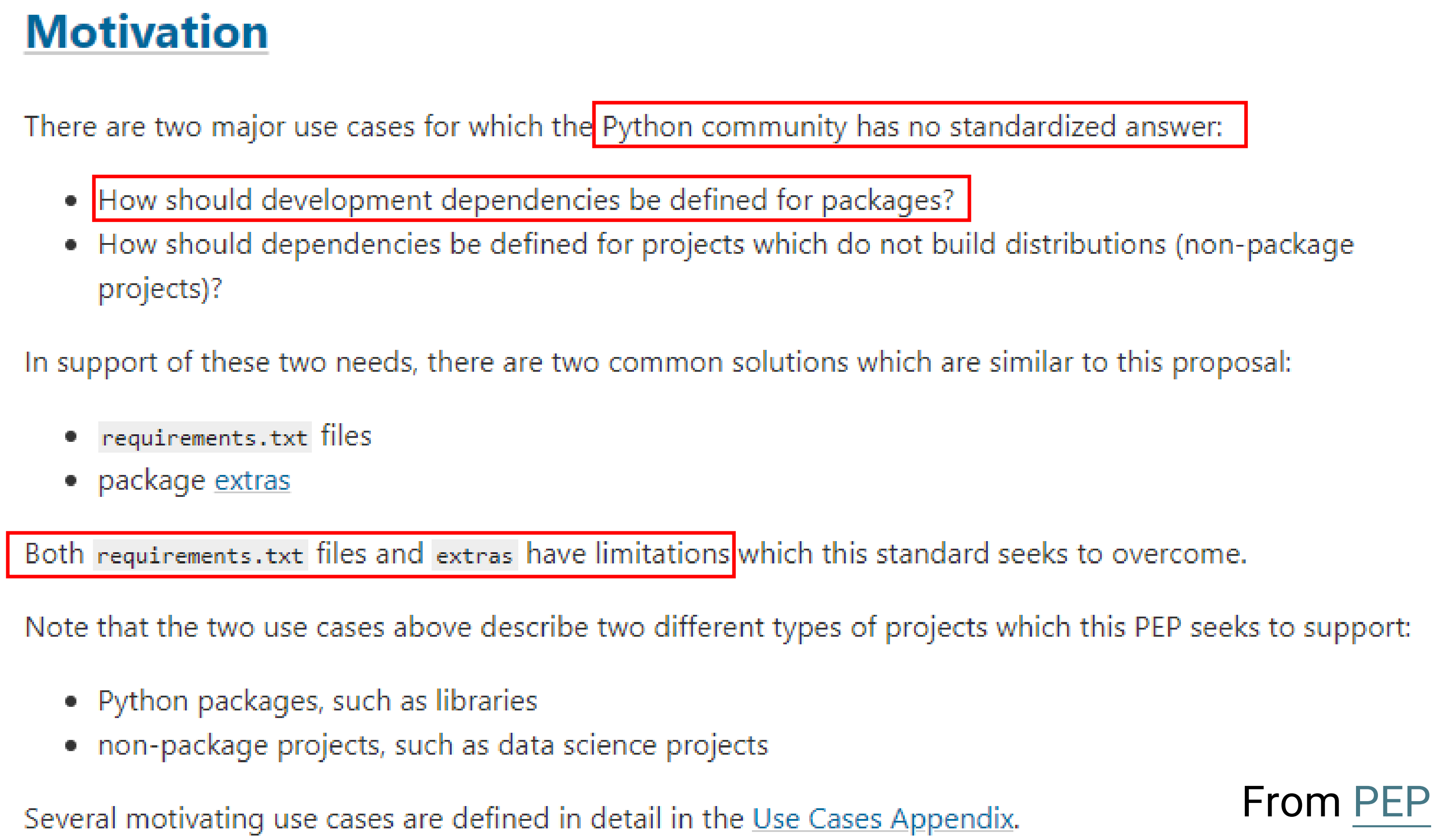


PEP 735 – Dependency Groups in pyproject.toml

Author:	Stephen Rosen <sirosen0 at gmail.com>
Sponsor:	Brett Cannon <brett at python.org>
PEP-Delegate:	Paul Moore <p.f.moore at gmail.com>
Discussions-To:	Discourse thread
Status:	Draft
Type:	Standards Track
Topic:	Packaging
Created:	20-Nov-2023
Post-History:	14-Nov-2023, 20-Nov-2023

From PEP

Dependency Management? How Proper?



Motivation

There are two major use cases for which the Python community has no standardized answer:

- How should development dependencies be defined for packages?
- How should dependencies be defined for projects which do not build distributions (non-package projects)?

In support of these two needs, there are two common solutions which are similar to this proposal:

- `requirements.txt` files
- package extras

Both `requirements.txt` files and `extras` have limitations which this standard seeks to overcome.

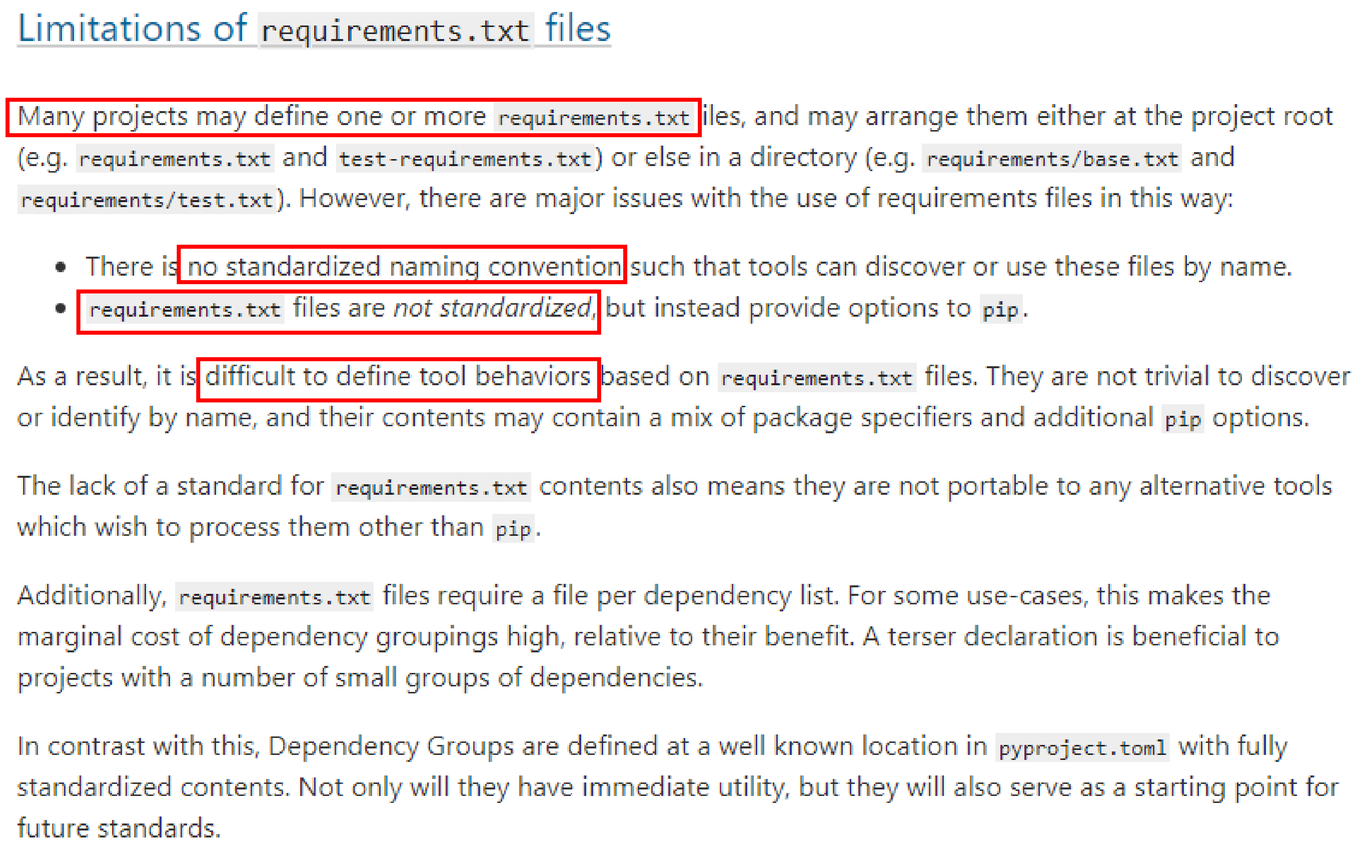
Note that the two use cases above describe two different types of projects which this PEP seeks to support:

- Python packages, such as libraries
- non-package projects, such as data science projects

Several motivating use cases are defined in detail in the Use Cases Appendix.

From PEP

Limitations of requirements.txt



Limitations of `requirements.txt` files

Many projects may define one or more `requirements.txt` files, and may arrange them either at the project root (e.g. `requirements.txt` and `test-requirements.txt`) or else in a directory (e.g. `requirements/base.txt` and `requirements/test.txt`). However, there are major issues with the use of requirements files in this way:

- There is no standardized naming convention such that tools can discover or use these files by name.
- `requirements.txt` files are *not standardized*, but instead provide options to `pip`.

As a result, it is difficult to define tool behaviors based on `requirements.txt` files. They are not trivial to discover or identify by name, and their contents may contain a mix of package specifiers and additional `pip` options.

The lack of a standard for `requirements.txt` contents also means they are not portable to any alternative tools which wish to process them other than `pip`.

Additionally, `requirements.txt` files require a file per dependency list. For some use-cases, this makes the marginal cost of dependency groupings high, relative to their benefit. A terser declaration is beneficial to projects with a number of small groups of dependencies.

In contrast with this, Dependency Groups are defined at a well known location in `pyproject.toml` with fully standardized contents. Not only will they have immediate utility, but they will also serve as a starting point for future standards.

From PEP

TLDR:

- No clear standard for dependency management

Traditional Dependency Management

- pip and requirements.txt

Challenge	Consequence
No built-in version locking	Inconsistent environment across machines
No native dev dependencies	Inseparable development and production dependencies
Inability to handle transitive dependencies	Potential hard-to-debug issues due to package version conflicts
No built-in support for multi-project setup	Incompatible dependency version across projects

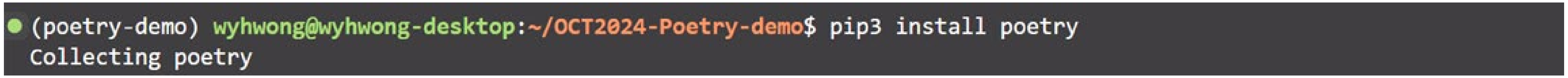
Solution: poetry

Challenge	Poetry
No built-in version locking	poetry.lock
No native dev dependencies	Separate dependencies by group
Inability to handle transitive dependencies	Automatic resolve
No built-in support for multi-project setup	Integration with virtual environment

Installation

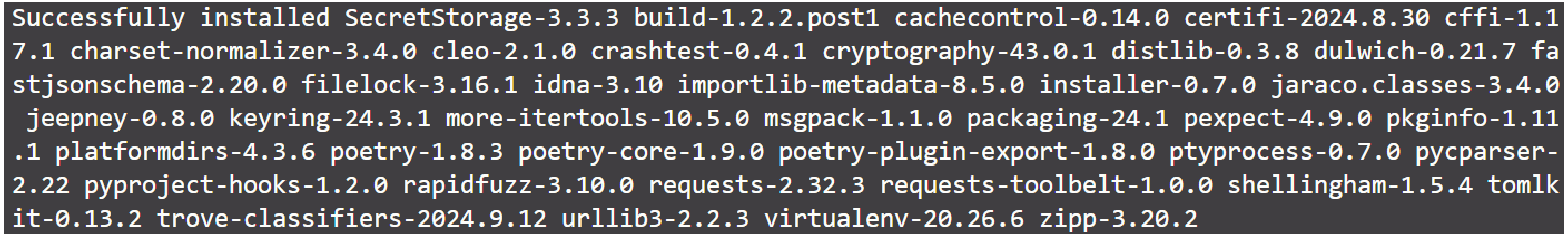
- Initialize your virtual environment
- pip install poetry

Caution:
Poetry should always be installed in a dedicated virtual environment to isolate it from the rest of your system.



```
● (poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo$ pip3 install poetry
  Collecting poetry
```

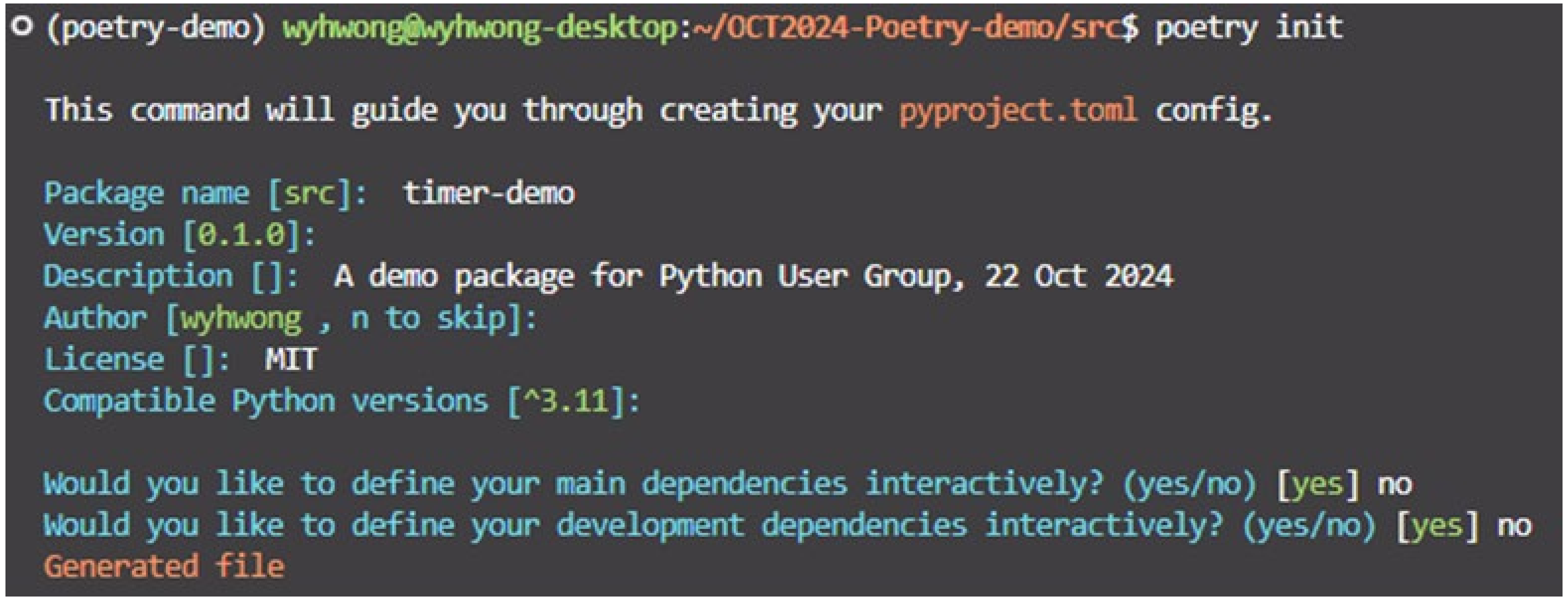
⋮



```
Successfully installed SecretStorage-3.3.3 build-1.2.2.post1 cachecontrol-0.14.0 certifi-2024.8.30 cffi-1.1
7.1 charset-normalizer-3.4.0 cleo-2.1.0 crashtest-0.4.1 cryptography-43.0.1 distlib-0.3.8 dulwich-0.21.7 fa
stjsonschema-2.20.0 filelock-3.16.1 idna-3.10 importlib-metadata-8.5.0 installer-0.7.0 jaraco.classes-3.4.0
 jeepney-0.8.0 keyring-24.3.1 more-itertools-10.5.0 msgpack-1.1.0 packaging-24.1 pexpect-4.9.0 pkginfo-1.11
.1 platformdirs-4.3.6 poetry-1.8.3 poetry-core-1.9.0 poetry-plugin-export-1.8.0 ptyprocess-0.7.0 pycparser-
2.22 pyproject-hooks-1.2.0 rapidfuzz-3.10.0 requests-2.32.3 requests-toolbelt-1.0.0 shellingham-1.5.4 tomlk
it-0.13.2 trove-classifiers-2024.9.12 urllib3-2.2.3 virtualenv-20.26.6 zipp-3.20.2
```

Project Initialization

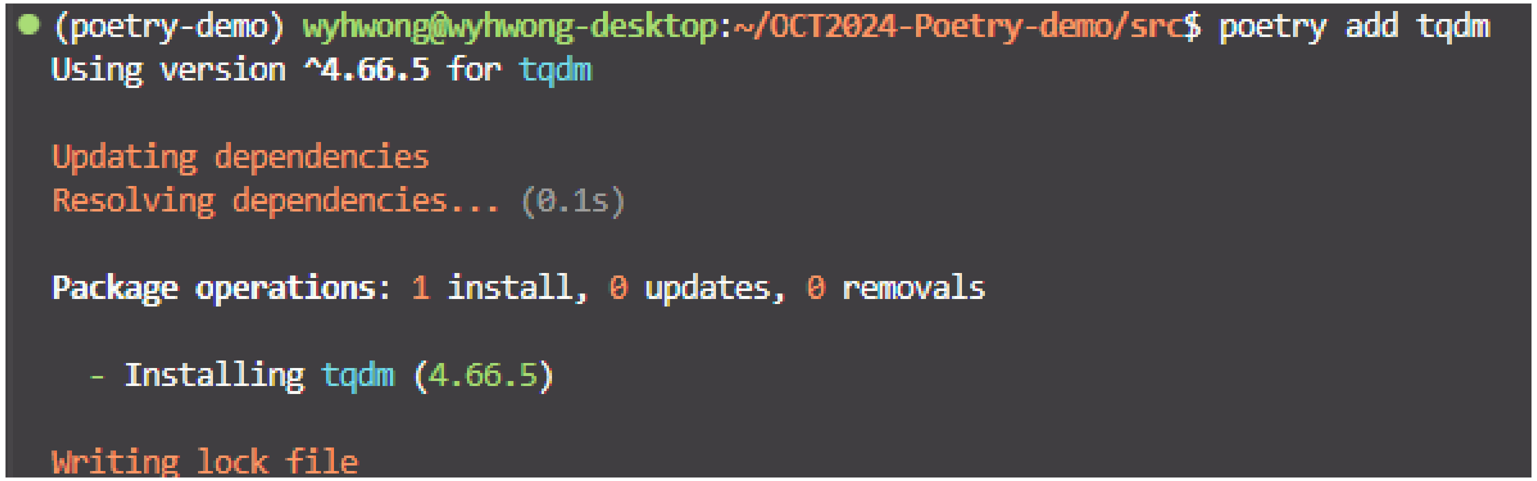
- poetry init



```
o (poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ poetry init

  This command will guide you through creating your pyproject.toml config.

  Package name [src]:  timer-demo
  Version [0.1.0]:
  Description []:  A demo package for Python User Group, 22 Oct 2024
  Author [wyhwong , n to skip]:
  License []:  MIT
  Compatible Python versions [^3.11]:

  Would you like to define your main dependencies interactively? (yes/no) [yes] no
  Would you like to define your development dependencies interactively? (yes/no) [yes] no
  Generated file
```

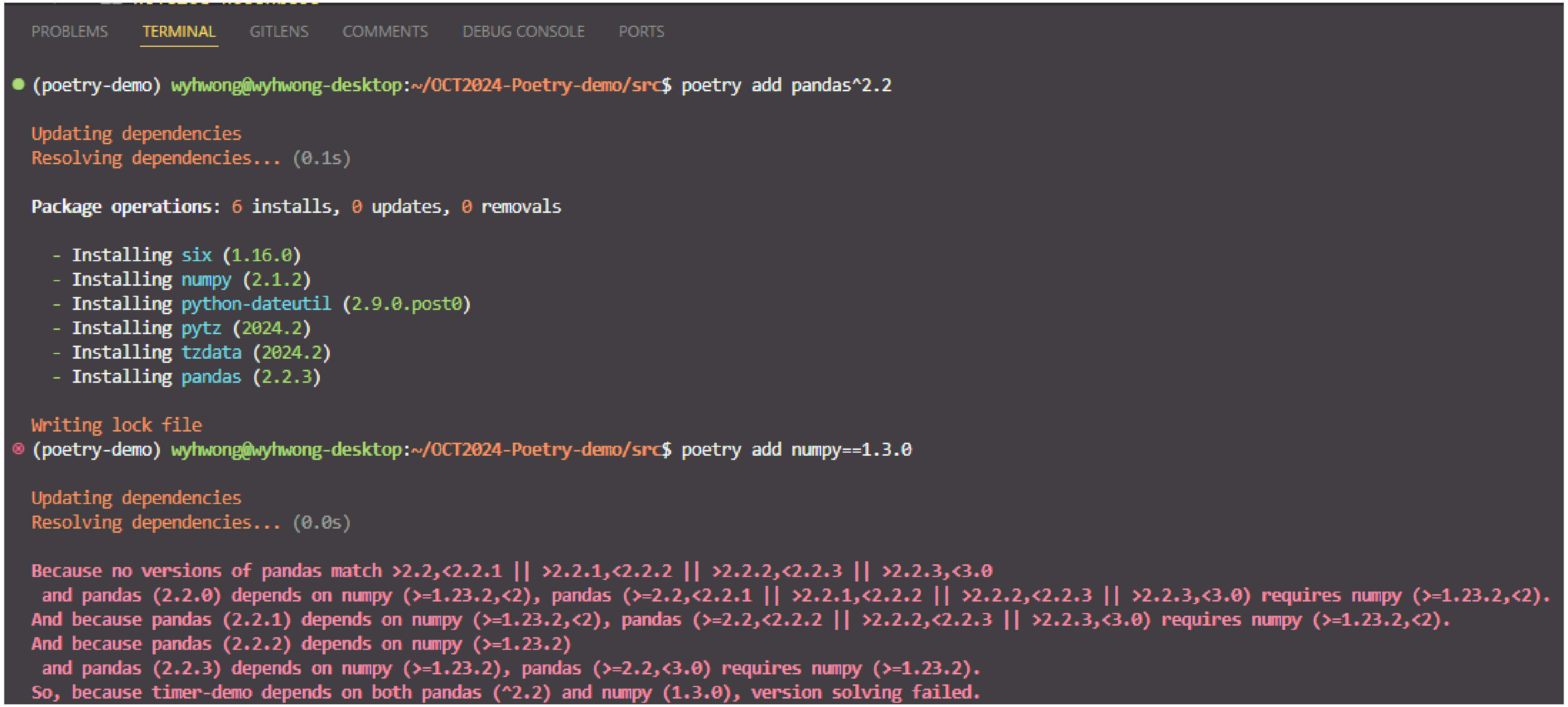
Add dependency

- poetry add <package>



```
● (poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ poetry add tqdm
  Using version ^4.66.5 for tqdm

  Updating dependencies
  Resolving dependencies... (0.1s)

  Package operations: 1 install, 0 updates, 0 removals

    - Installing tqdm (4.66.5)

  Writing lock file
```

Conflict of Dependencies

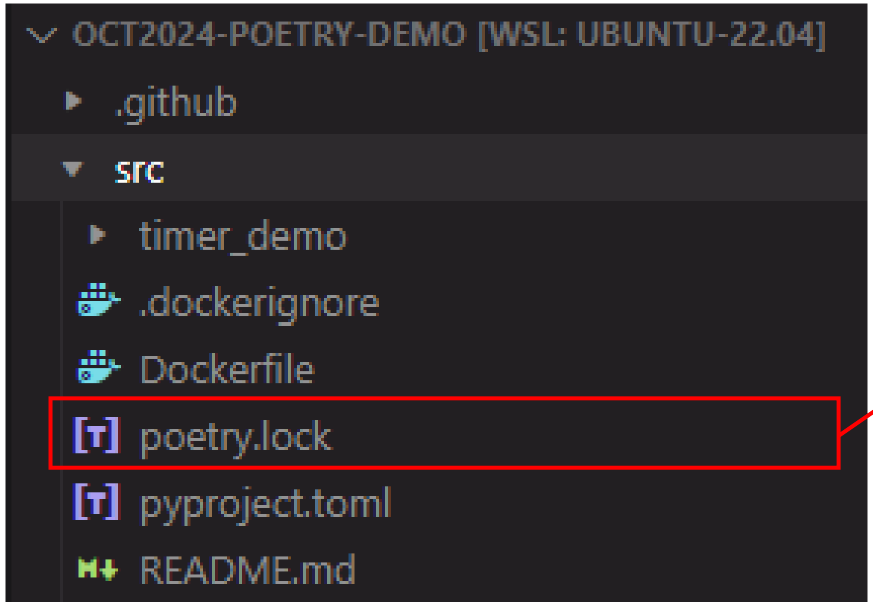

PROBLEMS TERMINAL GITLENS COMMENTS DEBUG CONSOLE PORTS

```
● (poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ poetry add pandas^2.2

  Updating dependencies
  Resolving dependencies... (0.1s)

  Package operations: 6 installs, 0 updates, 0 removals

    - Installing six (1.16.0)
    - Installing numpy (2.1.2)
    - Installing python-dateutil (2.9.0.post0)
    - Installing pytz (2024.2)
    - Installing tzdata (2024.2)
    - Installing pandas (2.2.3)

  Writing lock file
⊗ (poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ poetry add numpy==1.3.0

  Updating dependencies
  Resolving dependencies... (0.0s)

  Because no versions of pandas match >2.2,<2.2.1 || >2.2.1,<2.2.2 || >2.2.2,<2.2.3 || >2.2.3,<3.0
   and pandas (2.2.0) depends on numpy (>=1.23.2,<2), pandas (>=2.2,<2.2.1 || >2.2.1,<2.2.2 || >2.2.2,<2.2.3 || >2.2.3,<3.0) requires numpy (>=1.23.2,<2).
  And because pandas (2.2.1) depends on numpy (>=1.23.2,<2), pandas (>=2.2,<2.2.2 || >2.2.2,<2.2.3 || >2.2.3,<3.0) requires numpy (>=1.23.2,<2).
  And because pandas (2.2.2) depends on numpy (>=1.23.2)
   and pandas (2.2.3) depends on numpy (>=1.23.2), pandas (>=2.2,<3.0) requires numpy (>=1.23.2).
  So, because timer-demo depends on both pandas (^2.2) and numpy (1.3.0), version solving failed.
```

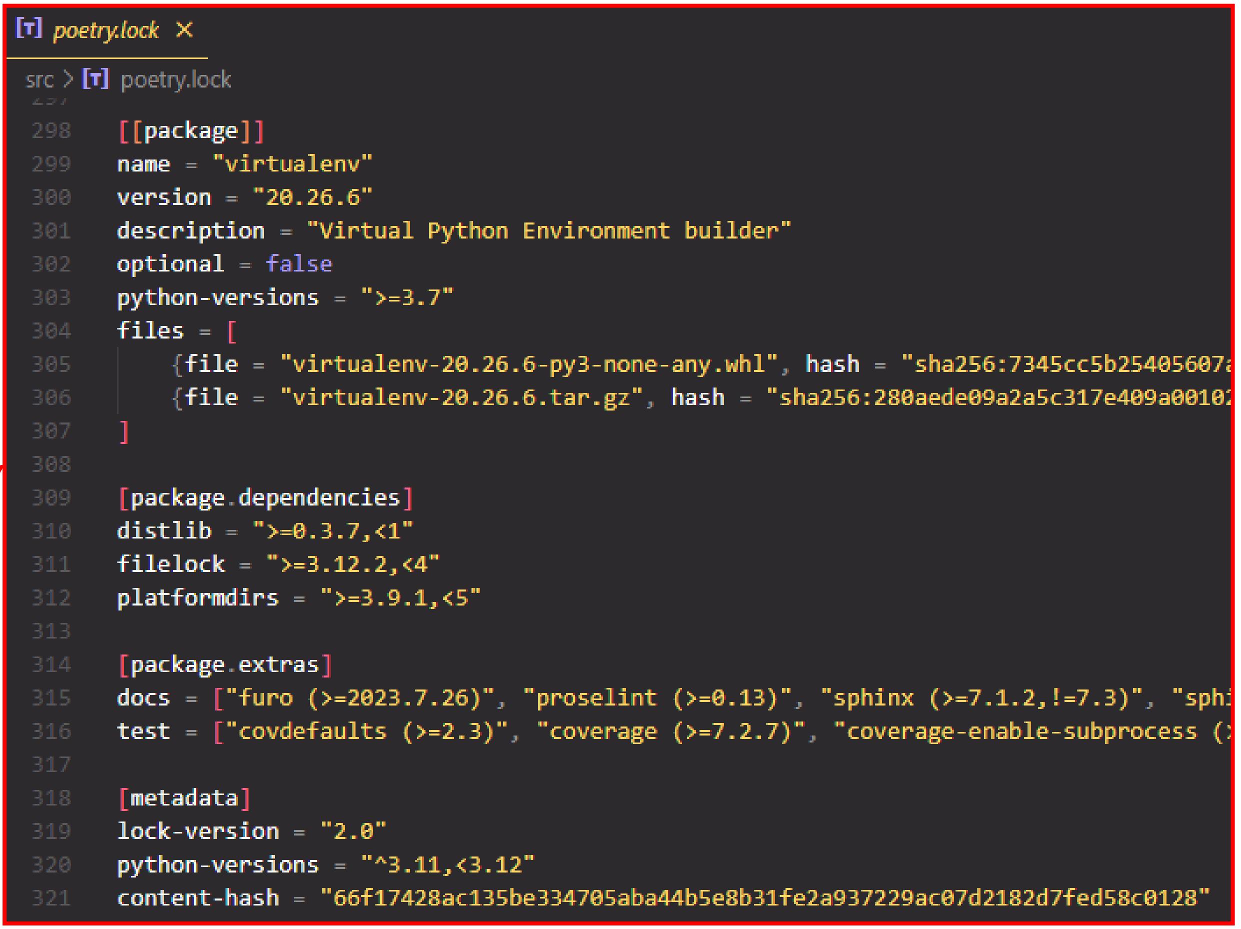
Lock File

Created by poetry after poetry add

- Manage transitive dependencies
- Check integrity of pyproject.toml



```
OCT2024-POETRY-DEMO [WSL: UBUNTU-22.04]
  .github
  src
    timer_demo
    .dockerignore
    Dockerfile
    poetry.lock
    pyproject.toml
    README.md
```



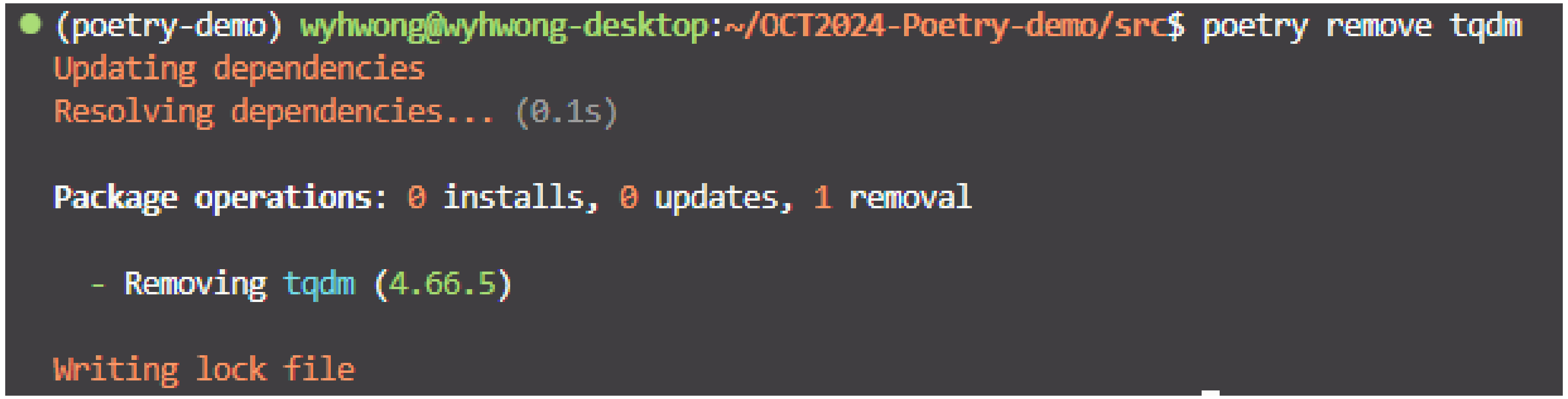
poetry.lock

src > poetry.lock

```
[[package]]
name = "virtualenv"
version = "20.26.6"
description = "Virtual Python Environment builder"
optional = false
python-versions = ">=3.7"
files = [
    {file = "virtualenv-20.26.6-py3-none-any.whl", hash = "sha256:7345cc5b25405607a
    {file = "virtualenv-20.26.6.tar.gz", hash = "sha256:280aede09a2a5c317e409a00102
]

[package.dependencies]
distlib = ">=0.3.7,<1"
filelock = ">=3.12.2,<4"
platformdirs = ">=3.9.1,<5"

[package.extras]
docs = ["furo (>=2023.7.26)", "proselint (>=0.13)", "sphinx (>=7.1.2,!=7.3)", "sphi
test = ["covdefaults (>=2.3)", "coverage (>=7.2.7)", "coverage-enable-subprocess (

[metadata]
lock-version = "2.0"
python-versions = "^3.11,<3.12"
content-hash = "66f17428ac135be334705aba44b5e8b31fe2a937229ac07d2182d7fed58c0128"
```

Remove Dependency

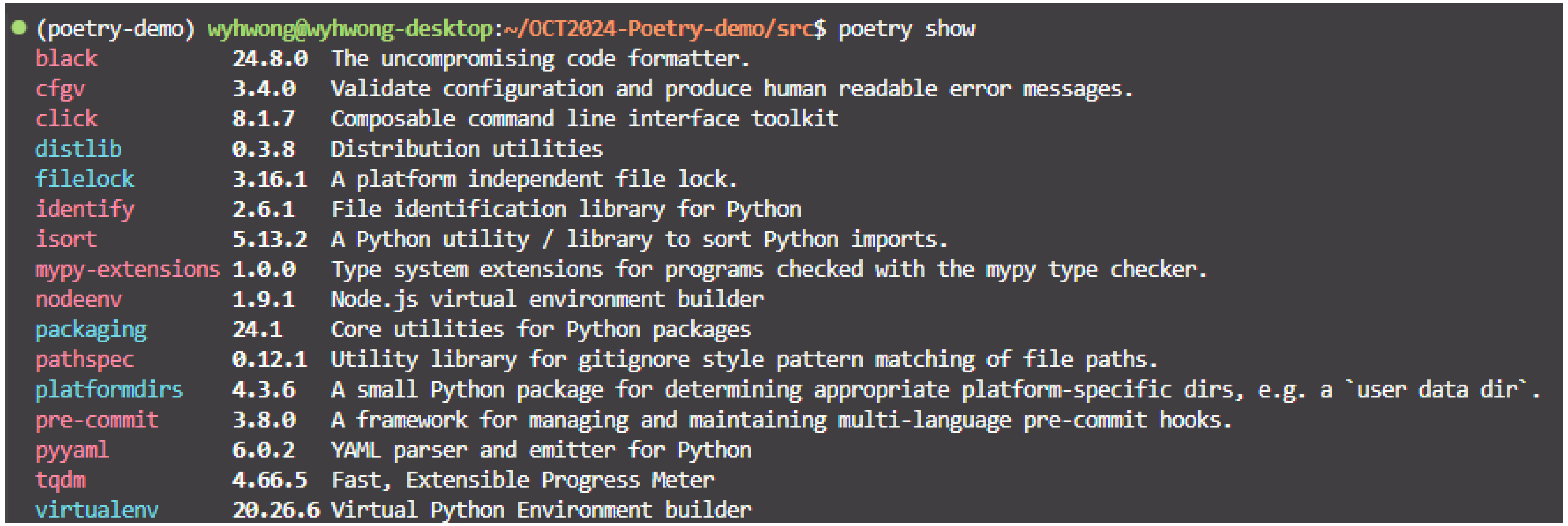
- poetry remove <package>



```
● (poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ poetry remove tqdm
  Updating dependencies
  Resolving dependencies... (0.1s)

  Package operations: 0 installs, 0 updates, 1 removal

    - Removing tqdm (4.66.5)

  Writing lock file
```

Show Project-specific Dependencies

- poetry show

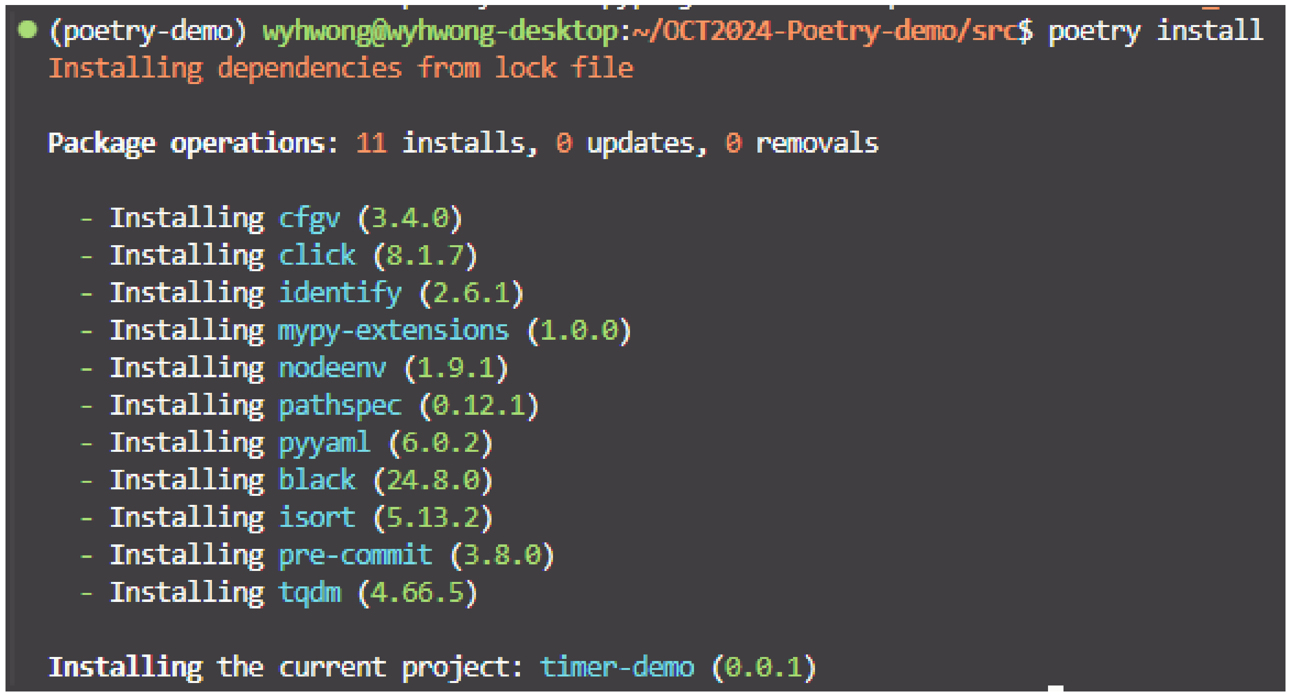


```
● (poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ poetry show
  black           24.8.0  The uncompromising code formatter.
  cfgv            3.4.0   Validate configuration and produce human readable error messages.
  click           8.1.7   Composable command line interface toolkit
  distlib         0.3.8   Distribution utilities
  filelock        3.16.1  A platform independent file lock.
  identify        2.6.1   File identification library for Python
  isort           5.13.2  A Python utility / library to sort Python imports.
  mypy-extensions 1.0.0   Type system extensions for programs checked with the mypy type checker.
  nodeenv         1.9.1   Node.js virtual environment builder
  packaging       24.1    Core utilities for Python packages
  pathspec        0.12.1  Utility library for gitignore style pattern matching of file paths.
  platformdirs    4.3.6   A small Python package for determining appropriate platform-specific dirs, e.g. a `user data dir`.
  pre-commit      3.8.0   A framework for managing and maintaining multi-language pre-commit hooks.
  pyyaml          6.0.2   YAML parser and emitter for Python
  tqdm            4.66.5  Fast, Extensible Progress Meter
  virtualenv      20.26.6 Virtual Python Environment builder
```

Red: Not installed yet
Blue: Installed

Project Installation

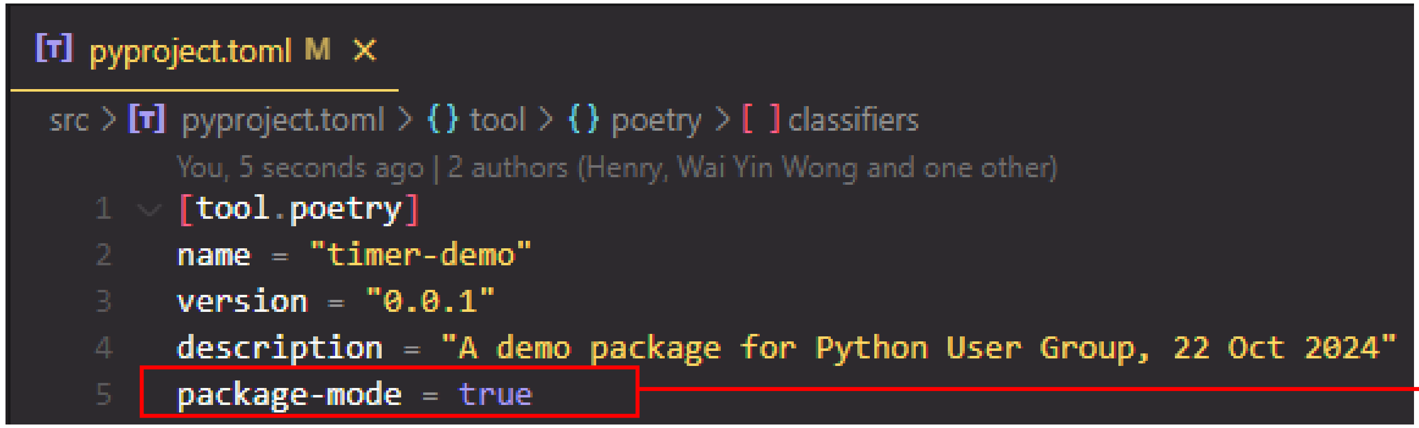
- poetry install



```
● (poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ poetry install
  Installing dependencies from lock file

  Package operations: 11 installs, 0 updates, 0 removals

    - Installing cfgv (3.4.0)
    - Installing click (8.1.7)
    - Installing identify (2.6.1)
    - Installing mypy-extensions (1.0.0)
    - Installing nodeenv (1.9.1)
    - Installing pathspec (0.12.1)
    - Installing pyyaml (6.0.2)
    - Installing black (24.8.0)
    - Installing isort (5.13.2)
    - Installing pre-commit (3.8.0)
    - Installing tqdm (4.66.5)

  Installing the current project: timer-demo (0.0.1)
```

Install as a package

- Configurable in pyproject.toml

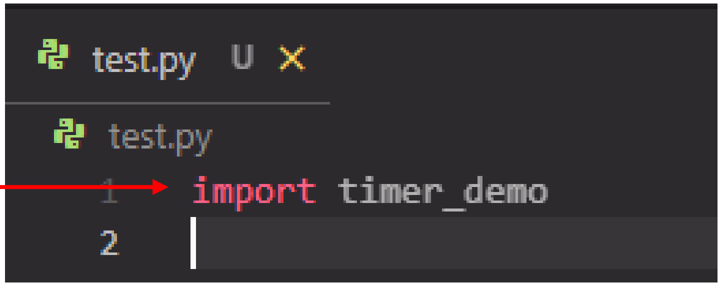


pyproject.toml M ×

src > pyproject.toml > {} tool > {} poetry > [ ] classifiers

You, 5 seconds ago | 2 authors (Henry, Wai Yin Wong and one other)

```toml
[tool.poetry]
name = "timer-demo"
version = "0.0.1"
description = "A demo package for Python User Group, 22 Oct 2024"
package-mode = true
```



test.py U ×

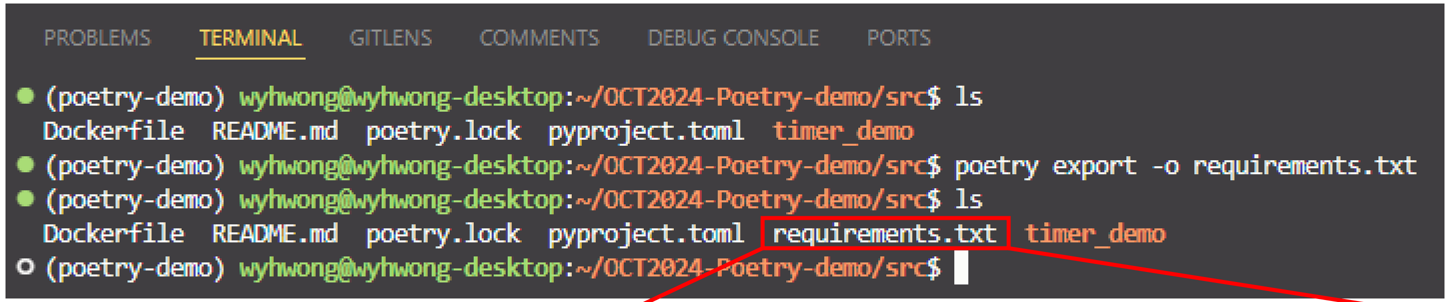
test.py

```python
import timer_demo

```

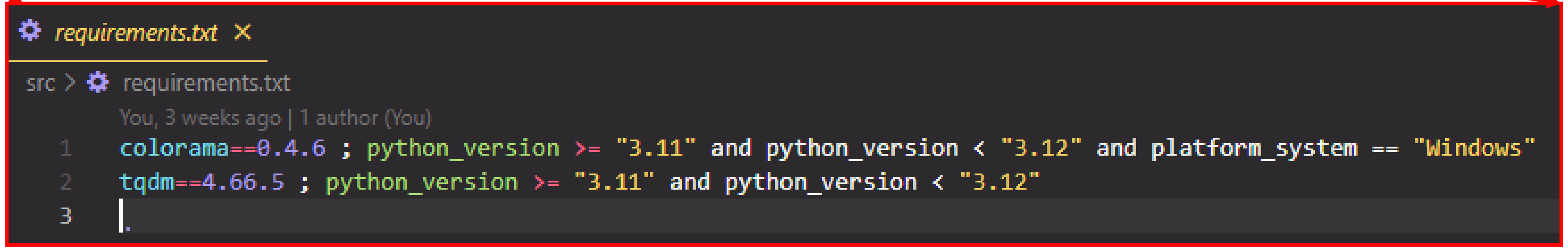
Possible to import in other directories

From poetry to requirements.txt

- poetry export



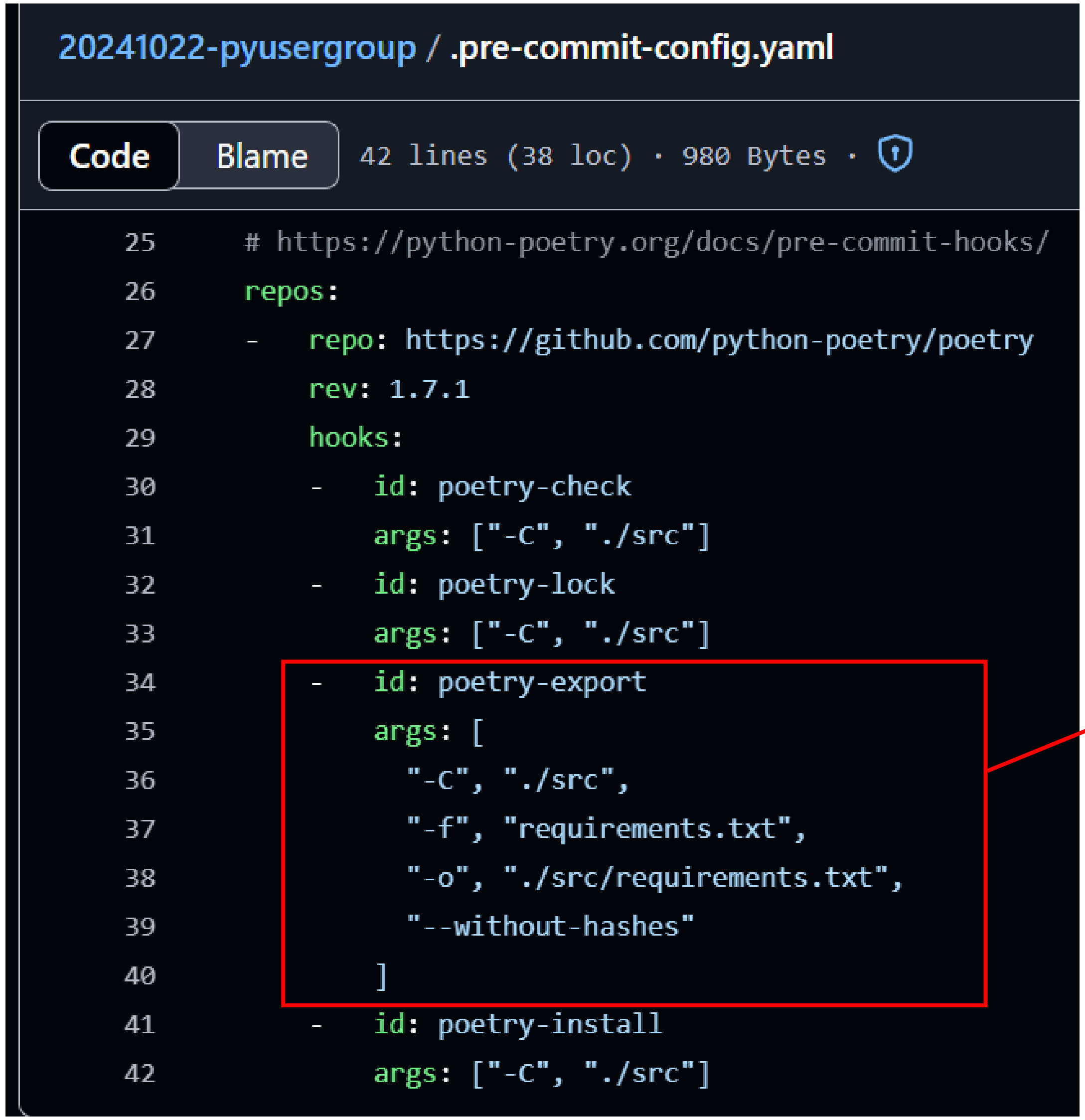
```
(poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ ls
Dockerfile  README.md  poetry.lock  pyproject.toml  timer_demo
(poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ poetry export -o requirements.txt
(poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$ ls
Dockerfile  README.md  poetry.lock  pyproject.toml  requirements.txt  timer_demo
(poetry-demo) wyhwong@wyhwong-desktop:~/OCT2024-Poetry-demo/src$
```



```
colorama==0.4.6 ; python_version >= "3.11" and python_version < "3.12" and platform_system == "Windows"
tqdm==4.66.5 ; python_version >= "3.11" and python_version < "3.12"
```

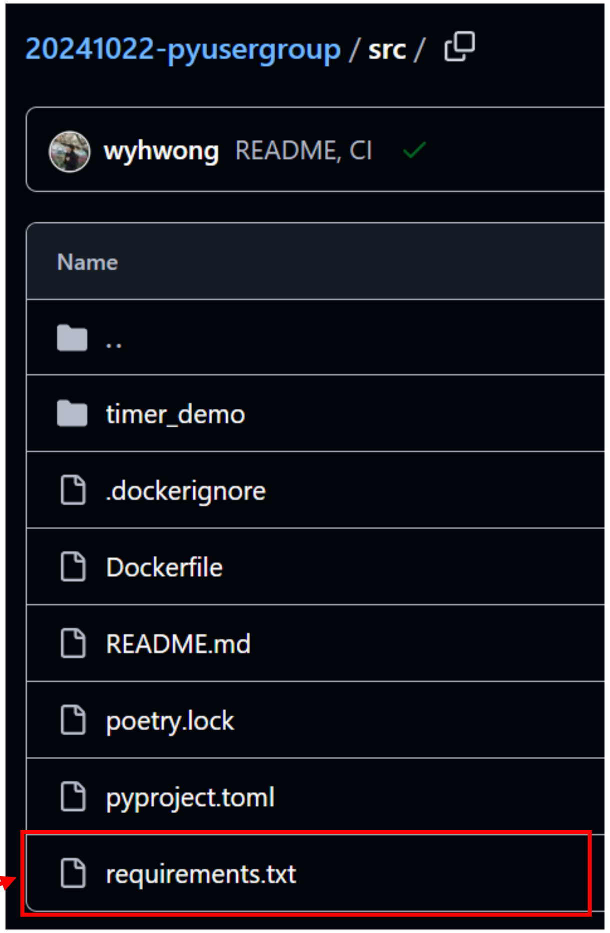
Users do not need to install poetry to run your project

Pre-commit hooks



20241022-pyusergroup / .pre-commit-config.yaml

```yaml
# https://python-poetry.org/docs/pre-commit-hooks/
repos:
-   repo: https://github.com/python-poetry/poetry
    rev: 1.7.1
    hooks:
    -   id: poetry-check
        args: ["-C", "./src"]
    -   id: poetry-lock
        args: ["-C", "./src"]
    -   id: poetry-export
        args: [
          "-C", "./src",
          "-f", "requirements.txt",
          "-o", "./src/requirements.txt",
          "--without-hashes"
        ]
    -   id: poetry-install
        args: ["-C", "./src"]
```

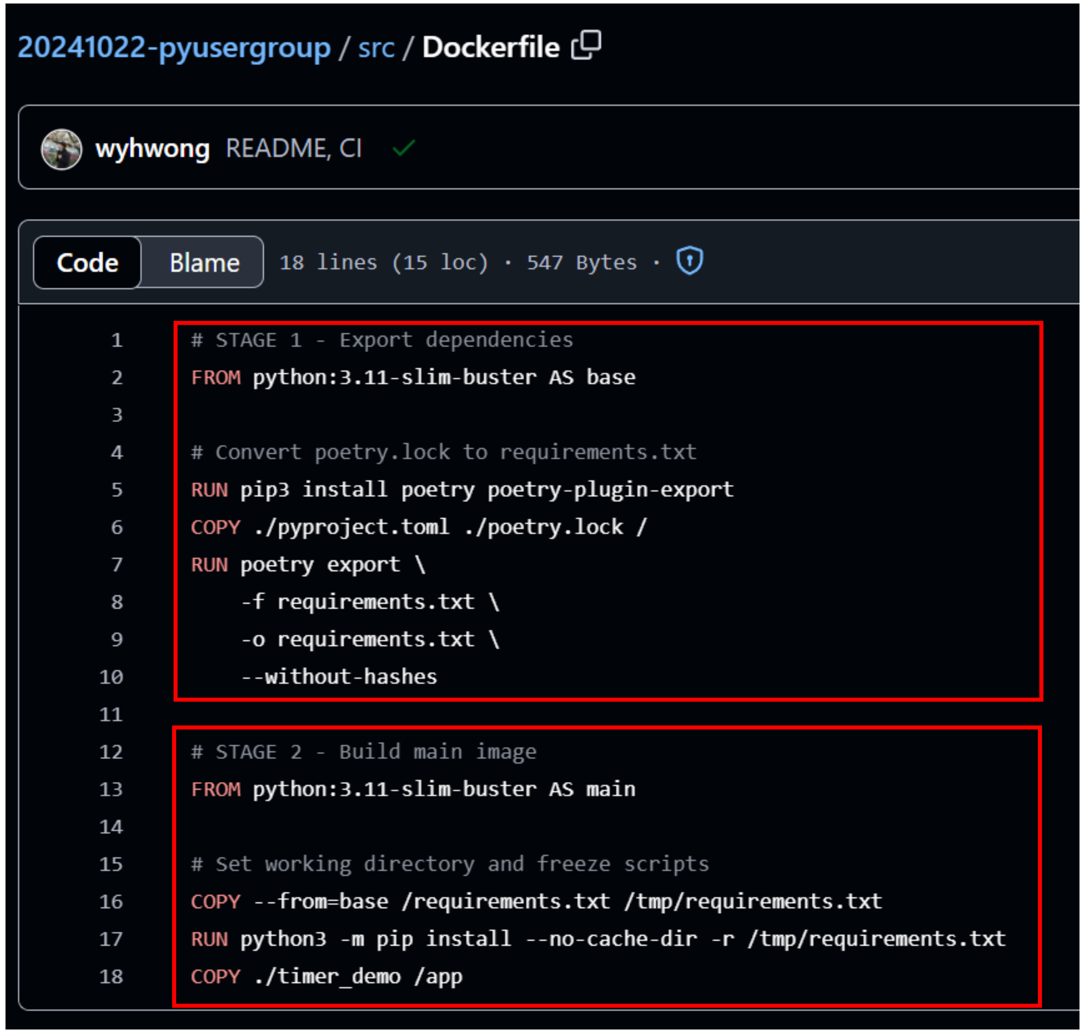


20241022-pyusergroup / src /

```
..
timer_demo
.dockerignore
Dockerfile
README.md
poetry.lock
pyproject.toml
requirements.txt
```

Automatic export of requirements.txt
Compatibility for non-poetry users

Ship to Production



20241022-pyusergroup / src / Dockerfile

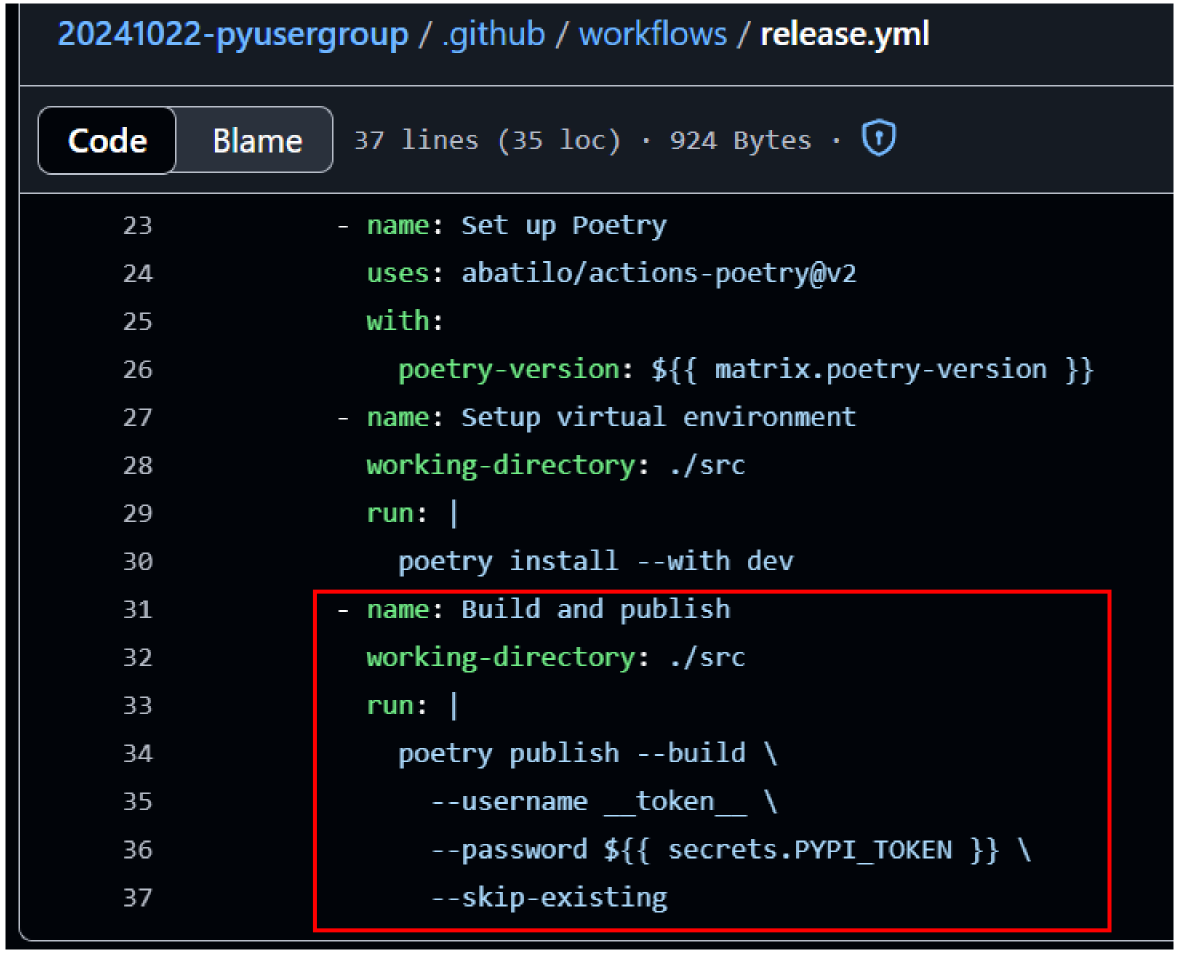
wyhwong README, CI

Code Blame 18 lines (15 loc) · 547 Bytes ·

```dockerfile
# STAGE 1 - Export dependencies
FROM python:3.11-slim-buster AS base

# Convert poetry.lock to requirements.txt
RUN pip3 install poetry poetry-plugin-export
COPY ./pyproject.toml ./poetry.lock /
RUN poetry export \
    -f requirements.txt \
    -o requirements.txt \
    --without-hashes

# STAGE 2 - Build main image
FROM python:3.11-slim-buster AS main

# Set working directory and freeze scripts
COPY --from=base /requirements.txt /tmp/requirements.txt
RUN python3 -m pip install --no-cache-dir -r /tmp/requirements.txt
COPY ./timer_demo /app
```

Stage 1: Dependency export
Stage 2: Build image

Release with GitHub Actions

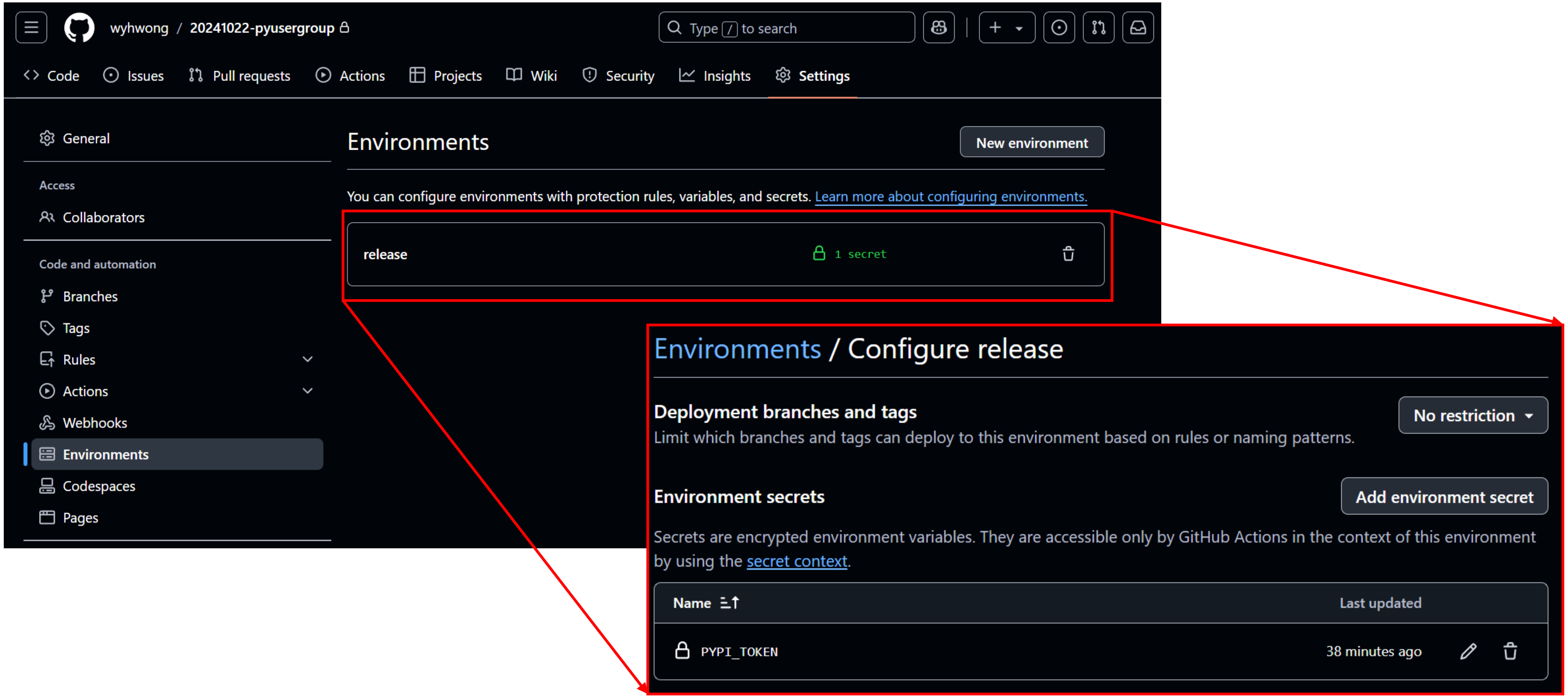


20241022-pyusergroup / .github / workflows / release.yml

```yaml
      - name: Set up Poetry
        uses: abatilo/actions-poetry@v2
        with:
          poetry-version: ${{ matrix.poetry-version }}
      - name: Setup virtual environment
        working-directory: ./src
        run: |
          poetry install --with dev
      - name: Build and publish
        working-directory: ./src
        run: |
          poetry publish --build \
            --username __token__ \
            --password ${{ secrets.PYPI_TOKEN }} \
            --skip-existing
```

Simple setup, install, and publish

Simple Setup of Environment



Environments

You can configure environments with protection rules, variables, and secrets. Learn more about configuring environments.

release 1 secret

Environments / Configure release

Deployment branches and tags

Limit which branches and tags can deploy to this environment based on rules or naming patterns.

No restriction

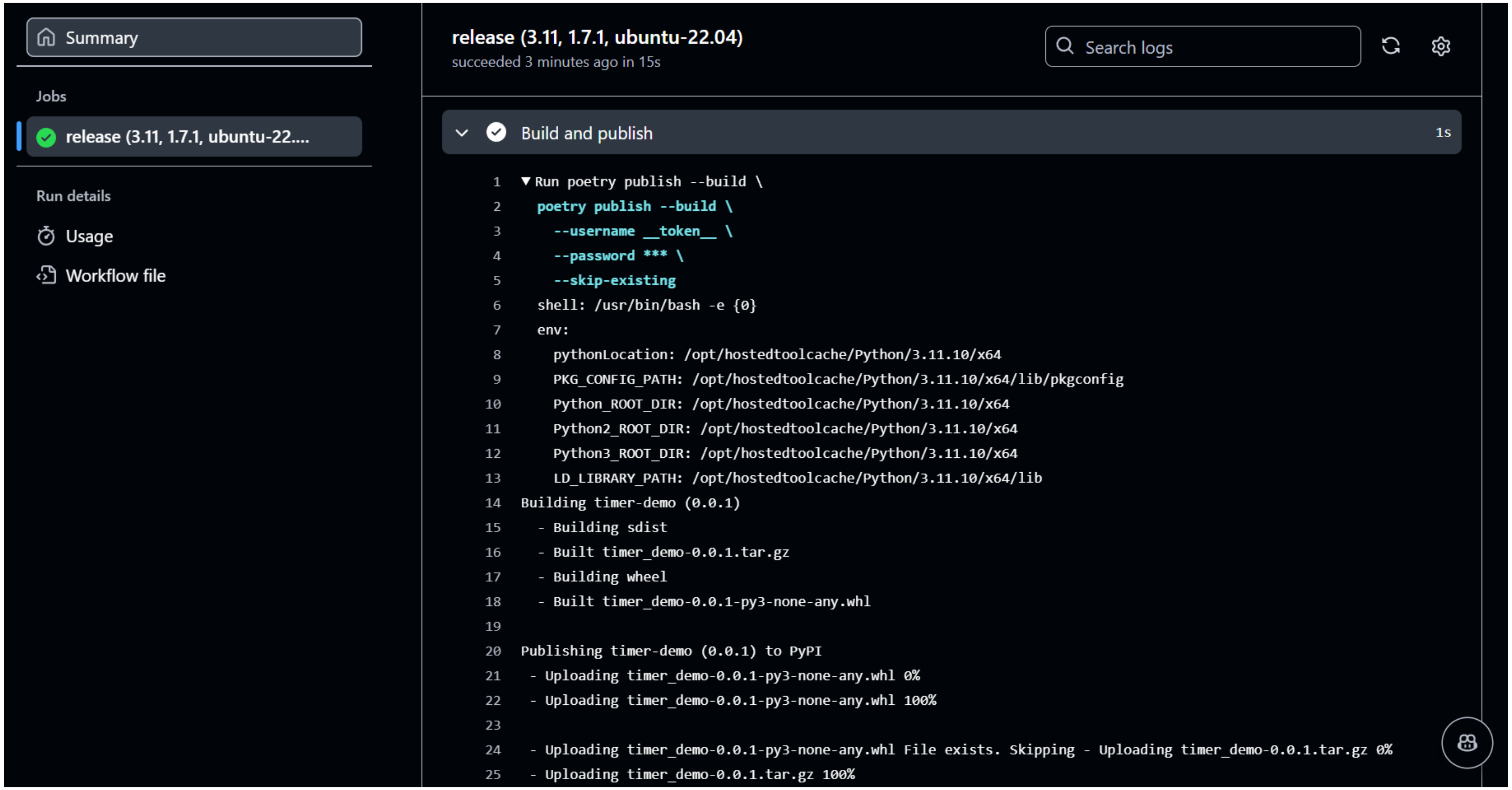
Environment secrets

Add environment secret

Secrets are encrypted environment variables. They are accessible only by GitHub Actions in the context of this environment by using the secret context.

Name	Last updated
PYPI_TOKEN	38 minutes ago

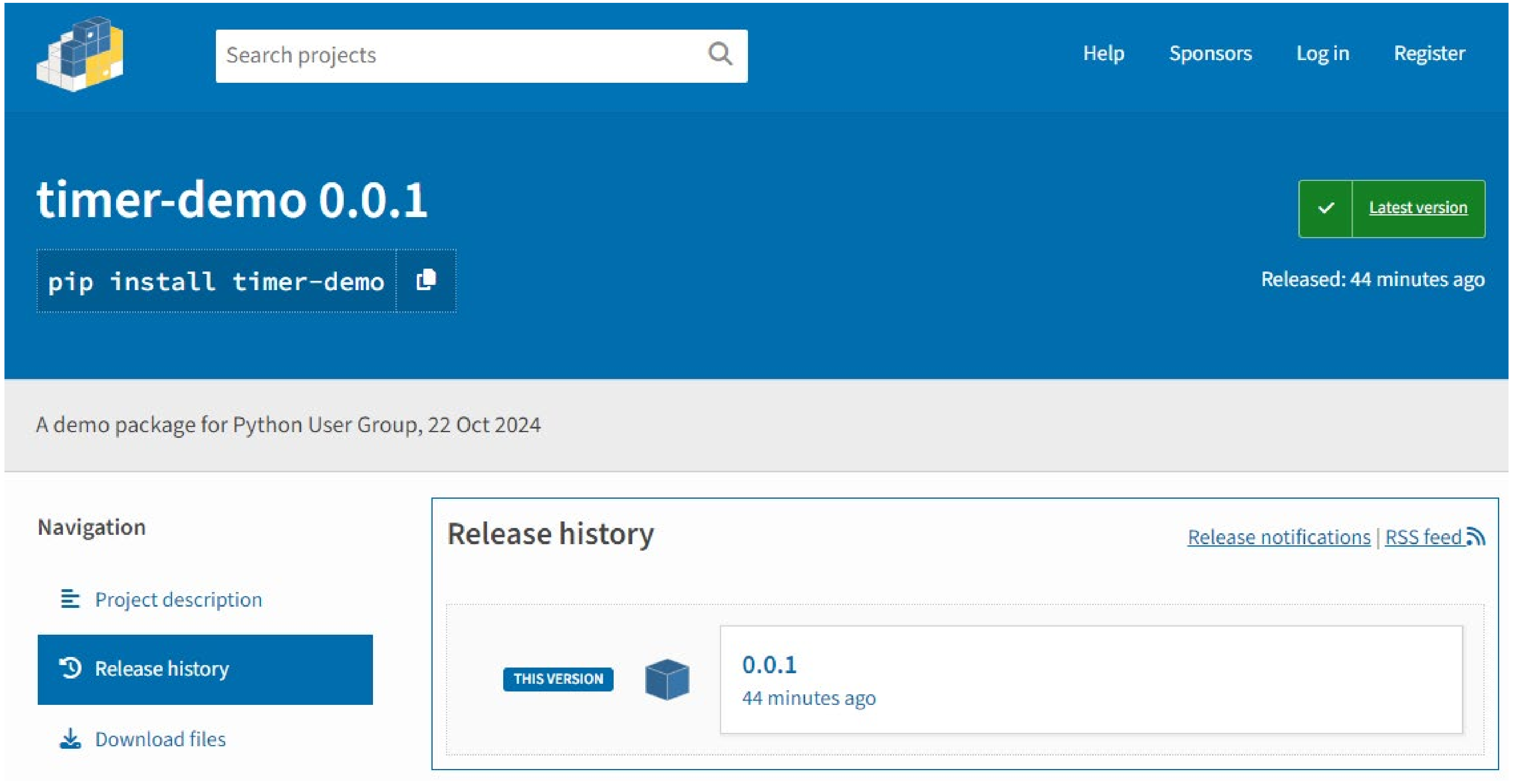
Logs on GitHub Actions



release (3.11, 1.7.1, ubuntu-22.04)
succeeded 3 minutes ago in 15s

```
▼Run poetry publish --build \
  poetry publish --build \
    --username __token__ \
    --password *** \
    --skip-existing
  shell: /usr/bin/bash -e {0}
  env:
    pythonLocation: /opt/hostedtoolcache/Python/3.11.10/x64
    PKG_CONFIG_PATH: /opt/hostedtoolcache/Python/3.11.10/x64/lib/pkgconfig
    Python_ROOT_DIR: /opt/hostedtoolcache/Python/3.11.10/x64
    Python2_ROOT_DIR: /opt/hostedtoolcache/Python/3.11.10/x64
    Python3_ROOT_DIR: /opt/hostedtoolcache/Python/3.11.10/x64
    LD_LIBRARY_PATH: /opt/hostedtoolcache/Python/3.11.10/x64/lib
Building timer-demo (0.0.1)
  - Building sdist
  - Built timer_demo-0.0.1.tar.gz
  - Building wheel
  - Built timer_demo-0.0.1-py3-none-any.whl

Publishing timer-demo (0.0.1) to PyPI
 - Uploading timer_demo-0.0.1-py3-none-any.whl 0%
 - Uploading timer_demo-0.0.1-py3-none-any.whl 100%

 - Uploading timer_demo-0.0.1-py3-none-any.whl File exists. Skipping - Uploading timer_demo-0.0.1.tar.gz 0%
 - Uploading timer_demo-0.0.1.tar.gz 100%
```

View on PyPI


Search projects

Help Sponsors Log in Register

timer-demo 0.0.1

```
pip install timer-demo
```

Latest version

Released: 44 minutes ago

A demo package for Python User Group, 22 Oct 2024

Navigation

Project description

Release history

Download files

Release history

Release notifications | RSS feed

THIS VERSION

0.0.1
44 minutes ago

PyCon Hong Kong 2024

Schedule Sessions Speakers

Time to Skip Tedious Steps - Spare Efforts with PyTorch Lightning ☆

2024-11-16 16:00–16:30, LT8
Language: English

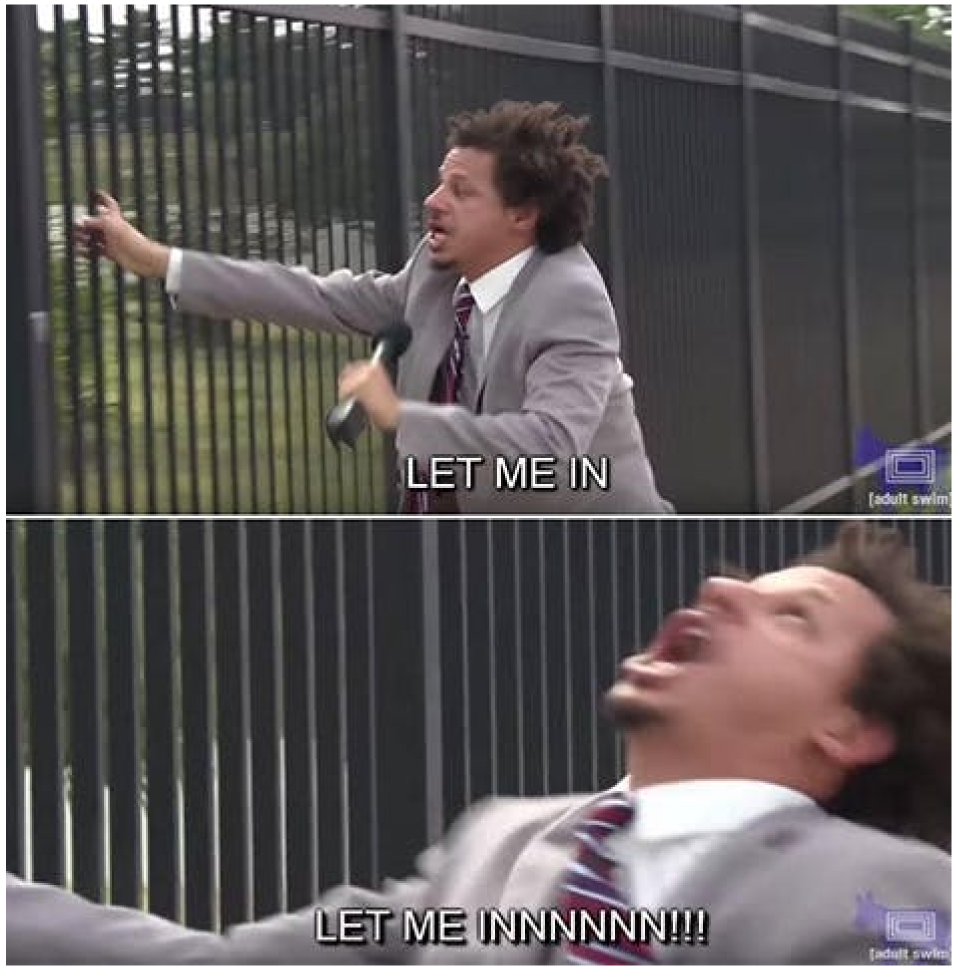
With the rapid advancement in deep learning, models become super large and consume significant resources, making efficiency and simplicity more critical than ever. In this talk, we introduce PyTorch Lightning, a deep learning framework that emerges as a powerful tool that streamlines the process of building, training, and scaling models, allowing researchers and practitioners to focus on what truly matters: innovation.

We will begin with an overview of PyTorch Lightning, discussing the key benefits it offers over traditional PyTorch. We will explore how PyTorch Lightning abstracts away the boilerplate code associated with model training, making it easier to implement and experiment with complex models. Then, we walk through the process of migrating traditional PyTorch to PyTorch Lightning and setting up distributed training.

Henry, Wai Yin Wong

Henry is a data scientist with 4 years of experience in Python. With broad exposure to classical and modern statistical approaches, he has been developing solutions for HVAC energy optimization, object detection and classification, predictive maintenance, physics-guided machine learning, and survival analysis. As a member of LIGO Scientific Collaboration, Henry has also contributed to academic research in the field of black-hole physics under the Bayesian framework.

Secure Your Ticket



QnA