nox-poetry

Claudio Jolowicz

Nov 19, 2023

CONTENTS

Use Poetry inside Nox sessions

This package provides a drop-in replacement for the `nox.session` decorator, and for the `nox.Session` object passed to user-defined session functions. This enables `session.install` to install packages at the versions specified in the Poetry lock file.

```
from nox_poetry import session

@session(python=["3.10", "3.9"])
def tests(session):
    session.install("pytest", ".")
    session.run("pytest")
```

Disclaimer: *This project is not affiliated with Nox, and not an official Nox plugin.*

CHAPTER

ONE

INSTALLATION

Install `nox-poetry` from the Python Package Index:

```
$ pip install nox-poetry
```

Important: This package must be installed into the same environment that Nox is run from. If you installed Nox using pipx, use the following command to install this package into the same environment:

```
$ pipx inject nox nox-poetry
```

CHAPTER

TWO

REQUIREMENTS

- Python 3.8+
- Poetry >= 1.0.0

You need to have a Poetry installation on your system. `nox-poetry` uses Poetry via its command-line interface.

CHAPTER

THREE

USAGE

Import the `@session` decorator from `nox_poetry` instead of `nox`. There is nothing else you need to do. The `session.install` method automatically honors the Poetry lock file when installing dependencies. This allows you to manage packages used in Nox sessions as development dependencies in Poetry.

This works because session functions are passed instances of `nox_poetry.Session`, a proxy for `nox.Session` adding Poetry-related functionality. Behind the scenes, nox-poetry uses Poetry to export a constraints file and build the package.

For more fine-grained control, additional utilities are available under the `session.poetry` attribute:

- `session.poetry.installroot(distribution_format=["wheel"|"sdist"])`
- `session.poetry.build_package(distribution_format=["wheel"|"sdist"])`
- `session.poetry.export_requirements()`

Note that `distribution_format` is a keyword-only parameter.

Here is a comparison of the different installation methods:

- Use `session.install(...)` to install specific development dependencies, e.g. `session.install("pytest")`.
- Use `session.install(".")` (or `session.poetry.installroot()`) to install your own package.
- Use `session.run_always("poetry", "install", external=True)` to install your package with *all* development dependencies.

Please read the next section for the tradeoffs of each method.

CHAPTER

FOUR

WHY?

Let's look at an example:

```
from nox_poetry import session

@session(python=["3.10", "3.9"])
def tests(session):
    session.install("pytest", ".")
    session.run("pytest")
```

This session performs the following steps:

- Build a wheel from the local package.
- Install the wheel as well as the `pytest` package.
- Invoke `pytest` to run the test suite against the installation.

Consider what would happen in this session if we had imported `@session` from `nox` instead of `nox_poetry`:

- Package dependencies would only be constrained by the wheel metadata, not by the lock file. In other words, their versions would not be *pinned*.
- The `pytest` dependency would not be constrained at all.
- Poetry would be installed as a build backend every time.

Unpinned dependencies mean that your checks are not reproducible and deterministic, which can lead to surprises in Continuous Integration and when collaborating with others. You can solve these issues by declaring `pytest` as a development dependency, and installing your package and its dependencies using `poetry install`:

```
@nox.session
def tests(session: Session) -> None:
    """Run the test suite."""
    session.run_always("poetry", "install", external=True)
    session.run("pytest")
```

Unfortunately, this approach comes with its own set of problems:

- Checks run against an editable installation of your package, i.e. your local copy of the code, instead of the installed wheel your users see. In the best case, any mistakes will still be caught during Continuous Integration. In the worst case, you publish a buggy release because you forgot to commit some changes.
- The package is installed, as well as all of its core and development dependencies, no matter which tools a session actually runs. Code formatters or linters, for example, don't need your package installed at all. Besides being wasteful, it goes against the idea of running checks in isolated environments.

`nox-poetry` uses a third approach:

- Installations are performed by pip, via the `session.install` method.
- When installing your own package, Poetry is used to build a wheel, which is passed to pip.
- When installing third-party packages, Poetry is used to export a constraints file, which is passed to pip along with the packages. The constraints file ensures that package versions are pinned by the lock file, without forcing an installation of every listed dependency and sub-dependency.

In summary, this approach brings the following advantages:

- You can manage tools like `pytest` as development dependencies in Poetry.
- Dependencies are pinned by Poetry's lock file, making checks predictable and deterministic.
- You can run checks against an installed wheel, instead of your local copy of the code.
- Every tool can run in an isolated environment with minimal dependencies.
- No need to install your package with all its dependencies if all you need is some linter.

CHAPTER

FIVE

CONTRIBUTING

Contributions are very welcome. To learn more, see the *Contributor Guide*.

CHAPTER

SIX

LICENSE

Distributed under the terms of the *MIT license*, *nox-poetry* is free and open source software.

CHAPTER

SEVEN

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

EIGHT

CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

8.1 Reference

Using Poetry in Nox sessions.

This package provides a drop-in replacement for the *session()* decorator, and for the *Session* object passed to user-defined session functions. This enables *session.install* to install packages at the versions specified in the Poetry lock file.

Example

```
>>> @session(python=["3.8", "3.9"])
... def tests(session: Session) -> None:
...     session.install("pytest", ".")
...     session.run("pytest")
```

It also provides helper functions that allow more fine-grained control:

- *session.poetry.installroot*
- *session.poetry.build_package*
- *session.poetry.export_requirements*

Two constants are defined to specify the format for distribution archives:

- *WHEEL*
- *SDIST*

8.1.1 Functions

nox_poetry.**session**(**args*, ***kwargs*)

Drop-in replacement for the nox.session() decorator.

Use this decorator instead of `@nox.session`. Session functions are passed *Session* instead of nox.sessions.Session; otherwise, the decorators work exactly the same.

Parameters

- **args** (*Any*) – Positional arguments are forwarded to `nox.session`.

- **kwargs** (*Any*) – Keyword arguments are forwarded to `nox.session`.

Returns
The decorated session function.

Return type
Any

8.1.2 Classes

class nox_poetry.**Session**(*session*)

Proxy for `nox.sessions.Session`, passed to session functions.

This class overrides *session.install*, and provides Poetry-related utilities:

- *Session.poetry.installroot*
- *Session.poetry.build_package*
- *Session.poetry.export_requirements*

Parameters
session (*Session*) –

_PoetrySession.**install**(**args*, ***kwargs*)

Install packages into a Nox session using Poetry.

This function installs packages into the session's virtual environment. It is a wrapper for `nox.sessions.Session.install()`, whose positional arguments are command-line arguments for pip install, and whose keyword arguments are the same as those for `nox.sessions.Session.run()`.

If a positional argument is ".", a wheel is built using *build_package()*, and the argument is replaced with the file URL returned by that function. Otherwise, the argument is forwarded unchanged.

In addition, a constraints file is generated for the package dependencies using *export_requirements()*, and passed to `pip install` via its `--constraint` option. This ensures that any package installed will be at the version specified in Poetry's lock file.

Parameters

- **args** (*str*) – Command-line arguments for `pip install`.
- **kwargs** (*Any*) – Keyword-arguments for `session.install`. These are the same as those for `nox.sessions.Session.run()`.

Return type
None

_PoetrySession.**installroot**(*, *distribution_format=DistributionFormat.WHEEL*, *extras=()*)

Install the root package into a Nox session using Poetry.

This function installs the package located in the current directory into the session's virtual environment.

A constraints file is generated for the package dependencies using *export_requirements()*, and passed to pip install via its `--constraint` option. This ensures that core dependencies are installed using the versions specified in Poetry's lock file.

Parameters

- **distribution_format** (*str*) – The distribution format, either wheel or sdist.
- **extras** (*Iterable[str]*) – Extras to install for the package.

Return type
None

_PoetrySession.**export_requirements**()

Export a requirements file from Poetry.

This function uses poetry export to generate a requirements file containing the project dependencies at the versions specified in `poetry.lock`. The requirements file includes both core and development dependencies.

The requirements file is stored in a per-session temporary directory, together with a hash digest over `poetry.lock` to avoid generating the file when the dependencies have not changed since the last run.

Returns
The path to the requirements file.

Return type
Path

_PoetrySession.**build_package**(*, *distribution_format=DistributionFormat.WHEEL*)

Build a distribution archive for the package.

This function uses poetry build to build a wheel or sdist archive for the local package, as specified via the `distribution_format` parameter. It returns a file URL with the absolute path to the built archive.

Parameters
distribution_format (*str*) – The distribution format, either wheel or sdist.

Returns
The file URL for the distribution package.

Return type
str

8.1.3 Constants

nox_poetry.**WHEEL: str = DistributionFormat.WHEEL**

A wheel archive.

nox_poetry.**SDIST: str = DistributionFormat.SDIST**

A source archive.

8.2 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

8.2.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.2.2 How to request a feature

Request features on the Issue Tracker.

8.2.3 How to set up your development environment

You need Python 3.8+ and Poetry.

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or Nox with nox-poetry:

```
$ poetry run python
$ poetry run nox
```

8.2.4 How to test the project

Run the full test suite:

```
$ poetry run nox
```

List the available Nox sessions:

```
$ poetry run nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ poetry run nox --session=tests
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

8.2.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ poetry run nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.3 Contributor Covenant Code of Conduct

8.3.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

8.3.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.3.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

8.3.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

8.3.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at mail@claudiojolowicz.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

8.3.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

8.3.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

8.4 License

PYTHON MODULE INDEX

INDEX