Poetry Python packaging and dependencies

CONCEPTS

> Commit poetry.lock for applications so everyone installs the same versions. poetry install needs a lock that matches pyproject.toml; run poetry lock after editing dependencies by hand.

pyproject.toml your constraints
↓ `poetry lock`
poetry.lock exact pinned versions
↓ `poetry install`
Virtualenv installed packages
↓ `poetry run`
Your command runs inside the env

Dependency workflow

> poetry shell and poetry export are plugins (poetry-plugin-shell, poetry-plugin-export) and are not installed by default in Poetry 2. Install them with poetry self add.

PROJECTS

`poetry new <name>`
Create a project in a new directory

`poetry init`
Create pyproject.toml interactively

`poetry check`
Validate pyproject.toml

`poetry check --lock`
Also verify poetry.lock matches

`poetry version <rule>`
Bump patch, minor, major or set a version

DEPENDENCIES

`poetry add <pkg>...`
Add dependencies and install them

`poetry add <pkg>@<constraint>`
Add with a version constraint

`poetry add --group <name> <pkg>`
Add to a dependency group

`poetry add --dev <pkg>`
Shortcut for the dev group

`poetry add --editable <path>`
Add a local path in editable mode

DEPENDENCIES (cont.)

`poetry remove <pkg>...`
Remove dependencies

`poetry remove --group <name> <pkg>`
Remove from a group

INSTALL AND LOCK

`poetry install`
Install the project and its locked dependencies

`poetry install --no-root`
Skip installing the project itself

`poetry install --with <group>`
Include an optional group

`poetry install --without <group>`
Exclude a group

`poetry install --only <group>`
Install just that group

`poetry install --extras <extra>`
Install an extra

`poetry sync`
Make the env match the lock exactly

`poetry lock`
Refresh poetry.lock for pyproject.toml

`poetry lock --regenerate`
Discard the lock and resolve afresh

`poetry update`
Upgrade all packages within constraints

`poetry update <pkg>...`
Upgrade only these packages

RUNNING AND ENVIRONMENTS

`poetry run <cmd>`
Run a command in the project env

`poetry run python`
Python REPL with project deps

`poetry run pytest`
Run a tool from the dev group

`poetry shell`
Spawn a shell in the env (plugin)

RUNNING AND ENVIRONMENTS (cont.)

`poetry env info`
Show env details

`poetry env info --path`
Print only the env path

`poetry env use <python>`
Pick or create the env for a Python

`poetry env remove <python>`
Delete an env

INSPECTING

`poetry show`
List installed packages

`poetry show <pkg>`
Details for one package

`poetry show --tree`
Dependencies as a tree

`poetry show --outdated`
Only packages with newer versions

`poetry show --latest`
Show the latest version of each

`poetry show --top-level`
Only explicitly declared packages

`poetry show --why <pkg>`
Why a package is installed

BUILD AND PUBLISH

`poetry build`
Build sdist and wheel into dist/

`poetry build --format wheel`
Build only a wheel or an sdist

`poetry publish`
Upload dist/ to PyPI

`poetry publish --build`
Build, then upload

CONFIG AND MAINTENANCE

`poetry config --list`
Show current settings

`poetry config <key> <value>`
Set a setting

`poetry config virtualenvs.in-project true`
Create .venv inside the project