Poetry

Python packaging and dependencies

START HERE

`poetry new <name>`
New project

`poetry update`
Upgrade within constraints

`poetry add <pkg>`
Add a dependency

`poetry show --tree`
Dependency tree

`poetry install`
Install from the lockfile

`poetry build`
Make sdist and wheel

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

`poetry publish`
Upload to PyPI

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 new --flat <name>`
Flat layout instead of src/

`poetry init`
Create pyproject.toml interactively

`poetry check`
Validate pyproject.toml

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

`poetry version`
Show the project version

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

Rules: patch, minor, major, prepatch, preminor, premajor, prerelease or a PEP 440 version.

DEPENDENCIES

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

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

Constraint examples: ^1.2, ~1.2, >=1.2,<2, latest.

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

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

`poetry add --optional <pkg>`
Add as an optional dependency

DEPENDENCIES (cont.)

`poetry add --extras <extra> <pkg>`
Activate extras of the package

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

`poetry add --source <name> <pkg>`
Install from a named source

`poetry add --python <spec> <pkg>`
Restrict by Python version

`poetry add --dry-run <pkg>`
Show what would change

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

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

`poetry source add --priority=<p> <name> <url>`
Add a package source

Priority is primary, supplemental or explicit.

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 --all-groups`
Install every group

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

`poetry install --all-extras`
Install every extra

`poetry install --dry-run`
Show operations without running them

`poetry sync`
Make the env match the lock exactly

Removes packages not in the lock.
Replaces the deprecated install --sync.

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

INSTALL AND LOCK (cont.)

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

`poetry update`
Upgrade all packages within constraints

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

`poetry update --lock`
Update the lock without installing

`poetry update --dry-run`
Preview an update

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)

`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

Takes a version such as 3.12 or a path to an interpreter.

`poetry env list`
List this project's envs

`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

INSPECTING (cont.)

`poetry show --format json`
Machine-readable output

BUILD AND PUBLISH

`poetry build`
Build sdist and wheel into dist/

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

Value is wheel or sdist.

`poetry publish`
Upload dist/ to PyPI

`poetry publish --build`
Build, then upload

`poetry publish --repository <name>`
Upload to another repository

`poetry publish --dry-run`
Do everything except the upload

CONFIG AND MAINTENANCE

`poetry config --list`
Show current settings

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

`poetry config --local <key> <value>`
Store in the project's poetry.toml

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

`poetry self add <plugin>`
Install a plugin

`poetry self update`
Update Poetry itself

RECIPES

Start a project with dev tools

```
poetry new demo && cd demo
poetry add requests
poetry add --group dev pytest ruff
poetry run pytest
```

Install for CI or production

```
poetry check --lock
poetry install --without dev --no-root
poetry run python -m demo
```

Release a new version

```
poetry version minor
poetry build
poetry publish --dry-run
poetry publish
```

Keep the env in the project

```
poetry config virtualenvs.in-project true --local
poetry env use 3.12
poetry install
```