# uv Fast Python package manager

## CONCEPTS

uv run and uv add lock and sync automatically, so you rarely need uv lock or uv sync by hand. Commit uv.lock; keep .venv out of git.

**pyproject.toml** your dependencies
↓ uv lock
**uv.lock** exact pinned versions
↓ uv sync
**.venv** installed packages
↓ uv run
**Your command** runs inside .venv
Project workflow

uv pip ... is a drop-in style interface for existing venvs and requirements files; it does not touch pyproject.toml or uv.lock. Use uv add for projects.

## PROJECTS

**uv init** *<name>*
Create a project in a new directory

**uv init**
Initialise the current directory

**uv init --lib** *<name>*
Library layout with src/ and build system

**uv add** *<pkg>*...
Add dependencies and install them

**uv add** '*<pkg>* ⩾ *<ver>*'
Add with a version constraint

**uv add --dev** *<pkg>*
Add to the dev dependency group

**uv add --group** *<name>* *<pkg>*
Add to a named dependency group

**uv add -r requirements.txt**
Import dependencies from a requirements file

**uv remove** *<pkg>*...
Remove dependencies

**uv sync**
Make .venv match the lockfile exactly

**uv sync --locked**
Fail if uv.lock is out of date (CI)

## PROJECTS (cont.)

**uv sync --frozen**
Install from uv.lock without re-locking

**uv sync --no-dev**
Skip the dev group

**uv lock**
Write or update uv.lock

**uv lock --upgrade**
Re-resolve to the newest allowed versions

**uv lock --upgrade-package** *<pkg>*
Upgrade one package only

**uv tree**
Show the dependency tree

**uv export -o requirements.txt**
Export the lock as a requirements file

**uv build**
Build sdist and wheel into dist/

## RUNNING

**uv run** *<cmd>* **[args]**
Run a command in the project environment

**uv run python**
Python REPL with project deps

**uv run** *<script.py>*
Run a script (inline metadata is honoured)

**uv run --with** *<pkg>* *<cmd>*
Add a temporary dependency for this run

**uv run --no-project** *<script.py>*
Ignore the surrounding project

**uv run --python** *<ver>* *<cmd>*
Run on a specific Python version

**uv init --script** *<file.py>*
Create a script with inline metadata

**uv add --script** *<file.py>* *<pkg>*
Declare a script's dependencies

## TOOLS

**uvx** *<tool>* **[args]**
Run a tool in a throwaway env

## TOOLS (cont.)

**uvx** *<tool>*@*<ver>* **[args]**
Run an exact version

**uvx** *<tool>*@latest **[args]**
Run the newest version

**uvx --from** *<pkg>* *<cmd>*
Command name differs from package name

**uv tool install** *<pkg>*
Install a tool onto PATH

**uv tool list**
List installed tools

**uv tool upgrade** *<pkg>*
Upgrade within its original constraint

## PYTHON VERSIONS

**uv python install** *<ver>*
Download a managed Python

**uv python list**
Show available and installed Pythons

**uv python pin** *<ver>*
Write .python-version for this directory

## VIRTUAL ENVS AND PIP

**uv venv** [*<path>*]
Create a venv (default .venv)

**uv venv --python** *<ver>*
Create with a specific Python

**uv pip install** *<pkg>*
Install into the active or local .venv

**uv pip install -r requirements.txt**
Install from a requirements file

**uv pip compile** *<in>* **-o requirements.txt**
Resolve to a pinned requirements file

## CACHE AND SELF

**uv --version**
Print the uv version

**uv help** *<command>*
Show help for a command