uv Fast Python package manager

START HERE

`uv init <name>`
New project

`uv lock --upgrade`
Refresh all locked versions

`uv add <pkg>`
Add a dependency

`uvx <tool>`
Run a tool, no install

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

`uv python install <ver>`
Get a Python

`uv sync`
Install from the lockfile

`uv pip install <pkg>`
pip-style install

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 init --app <name>`
Application project

`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 <pkg> --optional <extra>`
Add to an optional extra

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

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

`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Fail if uv.lock is out of date (CI)

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

PROJECTS (cont.)

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

`uv sync --all-extras`
Include every optional extra

`uv sync --all-groups`
Include every dependency 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 lock --check`
Verify the lockfile is current

`uv tree`
Show the dependency tree

`uv tree --outdated`
Mark packages with newer versions

`uv version`
Show the project version

`uv version --bump <part>`
Bump major, minor or patch

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

`uv build`
Build sdist and wheel into dist/

`uv publish`
Upload dist/ files to PyPI

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 run --no-sync <cmd>`
Skip the auto-sync step

`uv run --locked <cmd>`
Error if the lockfile needs changes

`uv run --group <name> <cmd>`
Include a dependency group

RUNNING (cont.)

`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
Alias for uv tool run.

`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

`uvx --with <pkg> <tool>`
Add extra packages to the tool env

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

`uv tool install '<pkg>≥<ver>'`
Install with a version constraint

`uv tool list`
List installed tools

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

`uv tool upgrade --all`
Upgrade every installed tool

`uv tool uninstall <pkg>`
Remove an installed tool

`uv tool update-shell`
Add the tool bin dir to your PATH

PYTHON VERSIONS

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

`uv python list`
Show available and installed Pythons

`uv python find [<ver>]`
Print the path of a matching interpreter

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

`uv python upgrade [<ver>]`
Move to the latest patch release

`uv python uninstall <ver>`
Remove a managed Python

VIRTUAL ENVS AND PIP

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

VIRTUAL ENVS AND PIP (cont.)

`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 install -e .`
Editable install of the current project

`uv pip uninstall <pkg>`
Uninstall a package

`uv pip freeze`
List installed packages pinned

`uv pip list`
List installed packages

`uv pip show <pkg>`
Show package details

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

`uv pip sync requirements.txt`
Make the env exactly match the file
Unlike install, sync removes packages that are not listed.

CACHE AND SELF

`uv cache prune`
Drop unused cache entries

`uv cache clean`
Wipe the whole cache

`uv self update`
Update uv (standalone installs)

`uv --version`
Print the uv version

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

RECIPES

Start a new app with dev tools

```
uv init myapp && cd myapp
uv add requests
uv add --dev pytest ruff
uv run pytest
```

Reproducible CI install

```
uv sync --locked --no-dev
uv run --no-sync python -m
myapp
```

Single-file script with dependencies

```
uv init --script fetch.py --
python 3.12
uv add --script fetch.py
requests rich
uv run fetch.py
```

Try a tool and pin a Python

```
# run once, nothing installed
uvx ruff check .
uv python pin 3.12
uv venv && uv pip install -r
requirements.txt
```