uv

uv at a glance, the all-in-one, fast Python project manager for versions, environments, dependencies, and publishing.

Install & Self-Management

Install uv once, then keep it current.

PURPOSE	COMMAND	RESULT
Install uv on macOS or Linux.	`curl -LsSf astral.sh/uv/install.sh | sh`	$ curl → uv
Install with Homebrew (or pipx).	`brew install uv`	brew → uv
Update uv to the latest release.	`uv self update`	v0.x → latest
Print the installed uv version.	`uv self version`	uv 0.x.y

pin uv in CI with: uv self update <version>

Python Versions

Install, pin, and manage Python interpreters.

PURPOSE	COMMAND	RESULT
List available and installed versions.	`uv python list`	3.14.6 ✓ 3.13.14 3.12.13
Install a specific Python version.	`uv python install 3.14`	↓ 3.14
Pin the project to a version.	`uv python pin 3.14`	.python-version 3.14
Find an installed interpreter.	`uv python find 3.14`	3.14 → .../bin/python
Upgrade a managed Python.	`uv python upgrade`	3.13 → 3.14
Uninstall a managed Python.	`uv python uninstall 3.11`	3.11 → 🗑

uv-managed Pythons live in uv's data dir, independent of the system Python

Projects & Dependencies

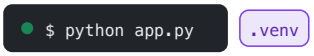
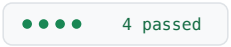
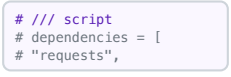
Create a project, manage dependencies, lock, and sync.

PURPOSE	COMMAND	RESULT
Create a new project.	`uv init myapp`	
Add a dependency.	`uv add requests`	
Add a development dependency.	`uv add --dev pytest`	
Add with a version constraint.	`uv add 'httpx>=0.27'`	
Remove a dependency.	`uv remove requests`	
Resolve and write the lockfile.	`uv lock`	
Install exactly from the lockfile.	`uv sync`	
Inspect the dependency tree.	`uv tree`	

pyproject.toml declares · uv.lock pins · .venv installs

Running Code & Scripts

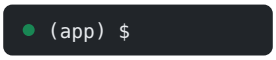
Run commands and scripts in managed environments.

PURPOSE	COMMAND	RESULT
Run a command in the project env.	`uv run python app.py`	
Run an installed tool or module.	`uv run pytest`	
Run a standalone script.	`uv run script.py`	
Add one-off deps for a single run.	`uv run --with rich script.py`	
Declare inline (PEP 723) script deps.	`uv add --script script.py requests`	

inline deps live in a # /// script ... # /// block

Virtual Environments

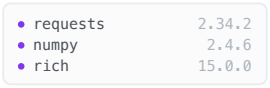
Create and activate a standalone `.venv` with uv.

PURPOSE	COMMAND	RESULT
Create a .venv environment in the project.	`uv venv`	
Create it with a specific Python version.	`uv venv --python 3.14`	
Create the environment at a custom path.	`uv venv .venvs/app`	
Activate it in your shell.	`source .venv/bin/activate`	

uv run and uv sync manage the project .venv for you

The pip Interface

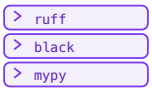
Drop-in pip-compatible commands, uv-fast.

PURPOSE	COMMAND	RESULT
Install packages (pip-compatible).	`uv pip install requests`	
Compile requirements to a lockfile.	`uv pip compile requirements.in -o requirements.txt`	
Sync the env exactly to a lockfile.	`uv pip sync requirements.txt`	
List or show installed packages.	`uv pip list`	
Freeze the environment.	`uv pip freeze > requirements.txt`	
Check for dependency conflicts.	`uv pip check`	

the pip interface targets the active env, not pyproject.toml

Tools (uvx)

Run and install command-line tools.

PURPOSE	COMMAND	RESULT
Run a tool once, ephemerally.	`uvx ruff check`	
The explicit form of uvx.	`uv tool run ruff`	
Install a tool user-wide.	`uv tool install ruff`	
List installed tools.	`uv tool list`	
Upgrade a tool to the latest.	`uv tool upgrade ruff`	
Uninstall a tool.	`uv tool uninstall ruff`	

uvx NAME is shorthand for uv tool run NAME

Build, Publish & Cache

Package, publish, export, and manage the cache.

PURPOSE	COMMAND	RESULT
Build a wheel and sdist.	`uv build`	
Publish to a package index.	`uv publish`	
Export the lock to requirements.	`uv export -o requirements.txt`	
Clean or prune the cache.	`uv cache clean`	
Show cache directory and size.	`uv cache dir`	

uv cache prune removes only unused entries · uv cache size shows total size