

# Managing a Python project with uv in 2026

Isaac Bythewood · 2026-04-05 · 3 min read

coding



```
~/my-project
$ uv init my-project
Initialized project 'my-project'
$ uv add flask requests
Resolved 10 packages in 42ms
Installed 10 packages in 18ms
+ flask==3.1.1
+ requests==2.32.3
$ uv add --dev ruff pytest
Resolved 14 packages in 31ms
Installed 4 packages in 9ms
$ uv run ruff check .
All checks passed!
$
```

How I set up and manage Python projects in 2026 using uv and ruff. Two tools and one config file replace everything.

Setting up a Python project used to mean juggling multiple tools for package management, Python versions, and virtual environments. I used to use pipenv, pyenv, and Black but I've since moved on. In 2026 I use `uv` for project management and `ruff` for linting and formatting. Two tools, one config file.

## Installing uv

[uv](#) is a single binary that replaces pip, pipenv, pyenv, and virtualenv. It's written in Rust by Astral and it's extremely fast.

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

## Starting a new project

Instead of manually creating a `Pipfile` or `requirements.txt`, run `uv init` and you get a project scaffold with a `pyproject.toml` ready to go.

```
uv init my-project
cd my-project
```

This gives you:

```
my-project/
├── .python-version
├── README.md
├── main.py
└── pyproject.toml
```

No `Pipfile`, no `setup.cfg`, no `requirements.txt`. Just `pyproject.toml`.

## Managing Python versions

You don't need `pyenv` anymore. `uv` handles downloading and managing Python versions for you.

```
uv python pin 3.13
uv python install
```

This writes `3.13` to `.python-version` and installs it if you don't already have it. That's it.

## Adding dependencies

Adding dependencies updates your `pyproject.toml` and `uv.lock` in one step.

```
uv add flask requests
uv add --dev pytest ruff
```

When you clone the project on another machine or set up CI, `uv sync` installs everything from the lockfile.

## Running things

`uv run` executes inside the project's virtual environment without you needing to activate anything. No more `pipenv shell` or `source .venv/bin/activate`.

```
uv run python main.py
uv run pytest
```

## Linting and formatting with Ruff

I used to use `Black`, `Flake8`, and `isort` separately but [Ruff](#) handles all of that now. It's from the same team that makes `uv` and it's just as fast. The formatter is designed as a drop-in replacement for

Black with near-identical output, so switching won't mean a massive reformatting commit across your codebase.

```
uv run ruff check .
uv run ruff format .
```

`ruff check` handles linting and import sorting. `ruff format` handles code formatting. You configure both in `pyproject.toml`:

```
[tool.ruff]
line-length = 88
target-version = "py313"

[tool.ruff.lint]
select = ["E", "F", "I", "UP"]

[tool.ruff.format]
quote-style = "double"
```

`E` and `F` are the pycodestyle and pyflakes rules that Flake8 used by default. `I` is isort-compatible import sorting. `UP` is pyupgrade which modernizes your syntax automatically. You can browse the full [rule list](#) and add what makes sense for your project.

## The full `pyproject.toml`

Here's what a complete `pyproject.toml` looks like. This replaces the `Pipfile`, `Pipfile.lock`, `.flake8`, and `.isort.cfg` I used to have scattered across my projects.

```
[project]
name = "my-project"
version = "0.1.0"
description = ""
readme = "README.md"
requires-python = "≥3.13"
dependencies = [
    "flask",
    "requests",
]

[dependency-groups]
dev = [
    "pytest",
    "ruff",
]

[tool.ruff]
line-length = 88
target-version = "py313"

[tool.ruff.lint]
select = ["E", "F", "I", "UP"]

[tool.ruff.format]
quote-style = "double"
```

That's all you need to get a Python project off the ground in 2026. If you're still juggling multiple tools and config files give uv and ruff a try, I don't think you'll go back.