Guides

Working on projects

uv supports managing Python projects, which define their dependencies in a `pyproject.toml` file.

Creating a new project

You can create a new Python project using the `uv init` command:

```
$ uv init hello-world
$ cd hello-world
```

Alternatively, you can initialize a project in the working directory:

```
$ mkdir hello-world
$ cd hello-world
$ uv init
```

uv will create the following files:

```
├── .gitignore
├── .python-version
├── README.md
├── main.py
└── pyproject.toml
```

The `main.py` file contains a simple "Hello world" program. Try it out with `uv run`:

```
$ uv run main.py
Hello from hello-world!
```

Project structure

A project consists of a few important parts that work together and allow uv to manage your project. In addition to the files created by `uv init`, uv will create a virtual environment and `uv.lock` file in the root of your project the first time you run a project command, i.e., `uv run`, `uv sync`, or `uv lock`.

A complete listing would look like:

```
.
├── .venv
│   ├── bin
│   ├── lib
│   └── pyvenv.cfg
├── .python-version
├── README.md
├── main.py
├── pyproject.toml
└── uv.lock
```

`pyproject.toml`

The `pyproject.toml` contains metadata about your project:

pyproject.toml

```toml
[project]
name = "hello-world"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
dependencies = []
```

You'll use this file to specify dependencies, as well as details about the project such as its description or license. You can edit this file manually, or use commands like `uv add` and `uv remove` to manage your project from the terminal.

> **Tip**
>
> See the official `pyproject.toml` guide for more details on getting started with the `pyproject.toml` format.

You'll also use this file to specify uv configuration options in a `[tool.uv]` section.

`.python-version`

The `.python-version` file contains the project's default Python version. This file tells uv which Python version to use when creating the project's virtual environment.

`.venv`

The `.venv` folder contains your project's virtual environment, a Python environment that is isolated from the rest of your system. This is where uv will install your project's dependencies.

See the project environment documentation for more details.

`uv.lock`

`uv.lock` is a cross-platform lockfile that contains exact information about your project's dependencies. Unlike the `pyproject.toml` which is used to specify the broad requirements of your project, the lockfile contains the exact resolved versions that are installed in the project environment. This file should be checked into version control, allowing for consistent and reproducible installations across machines.

`uv.lock` is a human-readable TOML file but is managed by uv and should not be edited manually.

See the lockfile documentation for more details.

Managing dependencies

You can add dependencies to your `pyproject.toml` with the `uv add` command. This will also update the lockfile and project environment:

```
$ uv add requests
```

You can also specify version constraints or alternative sources:

```
$ # Specify a version constraint
$ uv add 'requests==2.31.0'

$ # Add a git dependency
$ uv add git+https://github.com/psf/requests
```

If you're migrating from a `requirements.txt` file, you can use `uv add` with the `-r` flag to add all dependencies from the file:

```
$ # Add all dependencies from `requirements.txt`.
$ uv add -r requirements.txt -c constraints.txt
```

To remove a package, you can use `uv remove`:

```
$ uv remove requests
```

To upgrade a package, run `uv lock` with the `--upgrade-package` flag:

```
$ uv lock --upgrade-package requests
```

The `--upgrade-package` flag will attempt to update the specified package to the latest compatible version, while keeping the rest of the lockfile intact.

See the documentation on managing dependencies for more details.

Managing version

The `uv version` command can be used to read your package's version.

To get the version of your package, run `uv version`:

```
$ uv version
hello-world 0.7.0
```

To get the version without the package name, use the `--short` option:

```
$ uv version --short
0.7.0
```

To get version information in a JSON format, use the `--output-format json` option:

```
$ uv version --output-format json
{
    "package_name": "hello-world",
    "version": "0.7.0",
    "commit_info": null
}
```

See the publishing guide for details on updating your package version.

Running commands

`uv run` can be used to run arbitrary scripts or commands in your project environment.

Prior to every `uv run` invocation, uv will verify that the lockfile is up-to-date with the `pyproject.toml`, and that the environment is up-to-date with the lockfile, keeping your project in-sync without the need for manual intervention. `uv run` guarantees that your command is run in a consistent, locked environment.

For example, to use `flask`:

```
$ uv add flask
$ uv run -- flask run -p 3000
```

Or, to run a script:

example.py

```
# Require a project dependency
import flask

print("hello world")
```

```
$ uv run example.py
```

Alternatively, you can use `uv sync` to manually update the environment then activate it before executing a command:

macOS and Linux

```
$ uv sync
$ source .venv/bin/activate
$ flask run -p 3000
$ python example.py
```

Windows

```
PS> uv sync
PS> .venv\Scripts\activate
PS> flask run -p 3000
PS> python example.py
```

Note

The virtual environment must be active to run scripts and commands in the project without `uv run`. Virtual environment activation differs per shell and platform.

See the documentation on running commands and scripts in projects for more details.

Building distributions

`uv build` can be used to build source distributions and binary distributions (wheel) for your project.

By default, `uv build` will build the project in the current directory, and place the built artifacts in a `dist/` subdirectory:

```
$ uv build
$ ls dist/
hello-world-0.1.0-py3-none-any.whl
hello-world-0.1.0.tar.gz
```

See the documentation on building projects for more details.

Next steps

To learn more about working on projects with uv, see the projects concept page and the command reference.

Or, read on to learn how to build and publish your project to a package index.