[![logo](../../../assets/logo-letter.svg)](../../.. "uv") uv

[uv](https://github.com/astral-sh/uv "Go to repository")

The pip interface

    * [Using environments](../../../pip/environments/)
    * [Managing packages](../../../pip/packages/)
    * [Inspecting packages](../../../pip/inspection/)
    * [Declaring dependencies](../../../pip/dependencies/)
    * [Locking environments](../../../pip/compile/)
    * [Compatibility with pip](../../../pip/compatibility/)
  * [Reference](../../../reference/)

Reference

    * [Commands](../../../reference/cli/)
    * [Settings](../../../reference/settings/)
    * [Troubleshooting](../../../reference/troubleshooting/)

Troubleshooting

      * [Build failures](../../../reference/troubleshooting/build-failures/)
      * [Reproducible examples](../../../reference/troubleshooting/reproducible-examples/)
    * [Resolver](../../../reference/resolver-internals/)
    * [Benchmarks](../../../reference/benchmarks/)
    * [Policies](../../../reference/policies/)

Policies

Table of contents

# Running commands in projects

When working on a project, it is installed into the virtual environment at `.venv`. This environment is isolated from the current shell by default, so invocations that require the project, e.g., `python -c "import example"`, will fail. Instead, use `uv run` to run commands in the project environment:

```
$ uv run python -c "import example"
```

When using `run`, uv will ensure that the project environment is up-to-date before running the given command.

The given command can be provided by the project environment or exist outside of it, e.g.:

```
    $ # Presuming the project provides `example-cli`
    $ uv run example-cli foo

    $ # Running a `bash` script that requires the project to be available
    $ uv run bash scripts/foo.sh
```

## Requesting additional dependencies

Additional dependencies or different versions of dependencies can be requested per invocation.

The `--with` option is used to include a dependency for the invocation, e.g., to request a different version of `httpx`:

```
    $ uv run --with httpx==0.26.0 python -c "import httpx; print(httpx.__version__)"
    0.26.0
    $ uv run --with httpx==0.25.0 python -c "import httpx; print(httpx.__version__)"
```

```
0.25.0
```

The requested version will be respected regardless of the project's requirements. For example, even if the project requires `httpx==0.24.0`, the output above would be the same.

## Running scripts

Scripts that declare inline metadata are automatically executed in environments isolated from the project. See the [scripts guide](../../../guides/scripts/#declaring-script-dependencies) for more details.

For example, given a script:

example.py

```
# /// script
# dependencies = [
#   "httpx",
# ]
# ///

import httpx
```

```
    resp = httpx.get("https://peps.python.org/api/peps.json")

    data = resp.json()

    print([(k, v["title"]) for k, v in data.items()][:10])
```

The invocation `uv run example.py` would run _isolated_ from the project with only the given dependencies listed.

## Signal handling

uv does not cede control of the process to the spawned command in order to provide better error messages on failure. Consequently, uv is responsible for forwarding some signals to the child process the requested command runs in.

On Unix systems, uv will forward SIGINT and SIGTERM to the child process. Since shells send SIGINT to the foreground process group on Ctrl-C, uv will only forward a SIGINT to the child process if it is seen more than once or the child process group differs from uv's.

On Windows, these concepts do not apply and uv ignores Ctrl-C events, deferring handling to the child process so it can exit cleanly.

January 29, 2025

Back to top [Previous Managing dependencies](../dependencies/) [Next Locking and syncing](../sync/)