Single-File Dependency Management in Python Script Using UV

Wednesday January 1, 2025

UV deserves a comprehensive overview post, but today I’m going to discuss a specific convenience feature that I *love*.

In the past, whenever I’ve had a Python script with a dependency that I want to manage in an isolated environment, I’ve set up a separate Bash script to bootstrap it by activating the virtual environment and then running the Python script. Ugly and bloated, but it works. With UV, however, you can streamline this by using inline dependency management.

Let’s walk through an example. (Make sure you already have uv installed.)

I don’t have Numpy installed globally, so we’ll use that in our example script. Create a file named **numpy_version.py** with the following contents:

```
import numpy as np

def show_numpy_version():
    print(np.__version__)

if __name__ == '__main__':
    show_numpy_version()
```

If we try to run it as-is (`python3 numpy_version.py`) we see this:

```
Traceback (most recent call last):
  File "numpy_version.py", line 1, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
```

We can use uv to add inline metadata declaring the dependencies:

```
uv add --script numpy_version.py numpy
```

The script will be updated to look like this:

```
# /// script
# requires-python = ">=3.12"
# dependencies = [
#     "numpy",
# ]
# ///
import numpy as np

def show_numpy_version():
    print(np.__version__)

if __name__ == '__main__':
    show_numpy_version()
```

Then, we use uv to run it:

```
uv run numpy_version.py
```

And we get this output:

```
Reading inline script metadata from `numpy_version.py`
2.2.1
```

We can make our script directly runnable by adding a shebang line at the top of the file:

```
#!/usr/bin/env -S uv run
```

...and making the script executable:

```
chmod u+x numpy_version.py
```

The -S argument to env is required in order for the run argument to be handled correctly.

Now we can run the script as `./numpy_version.py`.

When we run the script like this, uv automatically manages the dependencies in an isolated environment. We can test this by trying to run the updated script directly with Python again:

```
python3 numpy_version.py
```

We get this, indicating that in a global context Numpy is still not installed:

```
Traceback (most recent call last):
  File "numpy_version.py", line 10, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
```

Our end result is a nice, clean, directly runnable, isolated script whose only execution requirement is that uv be installed.

The portability that this provides is *really* nice. I installed uv on a Le Potato and I was able to immediately run my script on the device with no changes. UV even automatically installed a CPython version to meet the Python version requirement. (The inline metadata specifies a Python version >= 3.12, but the global Python interpreter on the SBC is 3.11)