UV Tutorial pt2

Let’s start by creating a new project using Python 3.10. It’s super easy with uv. Just run the following command:

```
uv init --python 3.10 project_name
```

This command does a lot for you behind the scenes. It creates a new directory called project_name and sets up the necessary files and directories. Here's what your project structure will look like:

```
project_name/
├─── .git/
├─── .gitignore
├─── .python-version
├─── README.md
├─── pyproject.toml
├─── hello.py
```

Adding Packages to Your Project

Now, let’s add some packages to our project. Packages are the backbone of any Python project as they provide the functionality you need.

To add new packages, simply run:

```
uv add fastapi sqlalchemy alembic
```

This command does the following:

- Creates a new virtual environment called venv.
- Installs the specified packages (fastapi, sqlalchemy, and alembic) in the virtual environment.
- Updates the pyproject.toml file with the new package information.
- Creates a uv.lock file, which records the exact versions of all dependencies. This ensures consistent installations across different environments and machines.

If you need to add more packages later, you can do so easily:

```
uv add uvicorn
```

Editing Your Main Python File

Let’s edit the hello.py file to create a simple FastAPI application. Open hello.py and add the following code:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def main():
    return "hello from FastAPI"

if __name__ == "__main__":
    main()
```

This code sets up a basic FastAPI application that returns a simple “hello from FastAPI” message when accessed.

Running Your Project

To see your application in action, let’s run the project. Use the following command:

```
uv run uvicorn hello:app --port 8000 --reload
```

This command starts a Uvicorn server on port 8000 with automatic reloading enabled. You can now open your web browser and go to http://localhost:8000 to see your FastAPI application in action.

Syncing Your Project

When you’re done with your project and have pushed it to a Git repository, others can clone the repository and set it up easily. They just need to run:

```
uv sync
```

This command will create the virtual environment and install all the required packages listed in the uv.lock file.

Managing Dependencies

Sometimes you might want to add optional dependencies. For example, to add httpx as an optional dependency under a group called network, run:

```
uv add httpx --optional network
```

This will update the pyproject.toml file and add httpx to the network group.

To add development dependencies, such as pytest, use:

```
uv add pytest --dev
```

To remove a dependency, for example, sqlalchemy, run:

```
uv remove sqlalchemy
```

To see a detailed view of your project’s dependencies, use:

```
uv tree
```

This will output a tree view of all the packages and their dependencies.

Running Inline Scripts with Dependencies

You can also specify dependencies directly within your Python scripts. For instance, if you have a file test.py and it requires certain packages, you can add those dependencies at the top of the script:

```
# /// script
# requires-python = ">=3.13"
# dependencies = [
#     "requests",
#     "rich",
# ]
# ///

import requests
from rich.pretty import pprint

resp = requests.get("https://peps.python.org/api/peps.json")
data = resp.json()
pprint([(k, v["title"]) for k, v in data.items()][:10])
```

To run this script without installing the dependencies in your virtual environment, use:

```
uv run test.py
```

Checking for Dependency Conflicts

To check for conflicts and missing dependencies:

```
uv pip check
```