Quick Guide to uv

Python versions

Installing and managing Python itself.

- **uv python install**: Install Python versions.
- **uv python list**: View available Python versions.
- **uv python find**: Find an installed Python version.
- **uv python pin**: Pin the current project to use a specific Python version.
- **uv python uninstall**: Uninstall a Python version.

Scripts

Executing standalone Python scripts, e.g., example.py.

- **uv run**: Run a script.
- **uv add --script**: Add a dependency to a script (uv add pydantic)
- **uv remove --script**: Remove a dependency from a script

Projects

Creating and working on Python projects, i.e., with a pyproject.toml.

- **uv init**: Create a new Python project.
- **uv add**: Add a dependency to the project.
- **uv remove**: Remove a dependency from the project.
- **uv sync**: Sync the project's dependencies with the environment.
- **uv lock**: Create a lockfile for the project's dependencies.
- **uv run**: Run a command in the project environment.
- **uv tree**: View the dependency tree for the project.
- **uv build**: Build the project into distribution archives.
- **uv publish**: Publish the project to a package index.

Tools

- **uvx / uv tool run**: Run a tool in a temporary environment.
- **uv tool install**: Install a tool user-wide.
- **uv tool uninstall**: Uninstall a tool.
- **uv tool list**: List installed tools.
- **uv tool update-shell**: Update the shell to include tool executables.

Virtual Environments & Packages

- **uv venv**: Create a new virtual environment.
- **.venv\Scripts\activate**: Activate the virtual environment (Windows).
- **uv venv <env_name>**: Create a new virtual environment with `env_name (`uv venv fastapi`)
- **<name>\Scripts\activate** : Activate the virtual environment with name (`fastapi\Scripts\activate`)

Managing packages in an environment (replacing pip and pipdeptree)

- **uv pip install**: Install packages into the current environment.
- **uv pip show**: Show details about an installed package.
- **uv pip freeze**: List installed packages and their versions.
- **uv pip check**: Check that the current environment has compatible packages.
- **uv pip list**: List installed packages.
- **uv pip uninstall**: Uninstall packages.
- **uv pip tree**: View the dependency tree for the environment.

Locking packages in an environment

- **uv pip compile**: Compile requirements into a lockfile.
- **uv pip sync**: Sync an environment with a lockfile.

Utility

Managing and inspecting uv's state, such as the cache, storage directories, or performing a self-update:

- **uv cache clean**: Remove cache entries.
- **uv cache prune**: Remove outdated cache entries.
- **uv cache dir**: Show the uv cache directory path.
- **uv tool dir**: Show the uv tool directory path.
- **uv python dir**: Show the uv installed Python versions path.
- **uv self update**: Update uv to the latest version.

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:

```
.
├── .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
```

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
```

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

```
[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.

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
```

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

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

```
# 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.

Help menus

The `--help` flag can be used to view the help menu for a command, e.g., for `uv`:

```
uv --help
```

To view the help menu for a specific command, e.g., for `uv init`:

```
uv init --help
```

When using the `--help` flag, uv displays a condensed help menu. To view a longer help menu for a command, use `uv help`:

```
uv help
```

To view the long help menu for a specific command, e.g., for `uv init`:

```
uv help init
```

When using the long help menu, uv will attempt to use less or more to "page" the output so it is not all displayed at once. To exit the pager, press q.

Viewing the version

To check the installed version:

```
uv version
```

The following are also valid:

```
uv --version      # Same output as `uv version`
uv -V             # Will not include the build commit and date
uv pip --version  # Can be used with a subcommand
```

Follow Rahul Kumar Sah