建立 Python 虛擬環境

各種工具比較與使用時機

工具名稱	優點	缺點	適用情境
venv	內建於 `Python 3.*`，無須額外安裝；簡單直接	功能單純、不支援 `Python` 版本管理	單純建立隔離環境， 無需版本控管
virtualenv	支援 `Python 2` 和多版本共存，比 `venv` 更彈性	需額外安裝套件； 語法稍繁瑣	跨版本需求、老舊系統支援
virtualenvwrapper	提供更方便的環境管理（如 `mkvirtualenv`, `workon`, `rmvirtualenv`）	初學者需理解更多命令， 依賴 `virtualenv`	管理多個虛擬環境時更有效率
poetry	一站式套件管理與虛擬環境管理 （整合依賴管理、打包、部署）	學習曲線稍高， 某些套件不完全相容	現代 Python 專案開發、部署
uv	`uv`， 則是用於**極速套件安裝與虛擬環境建立**的工具	還在快速發展中， 與傳統工具不完全相容	對安裝速度與性能有極高需求時

使用 `venv`

1. 產生虛擬環境

   ```
   cd ~/
   python -m venv 虛擬環境名稱
   ```

2. 啟動虛擬環境

   `source test-venv/bin/activate`

3. 若成功啟動，會出現以下畫面:

   ```
   (test-venv) allen@raspberrypi:
   ```

4. 離開虛擬環境

   `deactivate`

使用 `virtualenv` 與 `virtualenvwrapper`

安裝

1. 首先，我們需要找出 `Python` 直譯器的路徑，以便在安裝前設定環境變數，請啟動終端機輸入 `python3` 指令，再加上 `--version` 參數來查詢版本，可以看到版本是 `3.11.2`，如下所示：

   ```
   python3 --version
   ```

2. 然後，使用找到的版本來搜尋 `Python` 直譯器的路徑，使用的是 `which` 指令，可以找出路徑 `「/user/bin/python」`，如下所示：

   ```
   which python 3.11
   ```

3. 接著，設定環境變數 VIRTUALENVWRAPPER_PYTHON 值是找到的 Python 直譯器的路徑，請使用 echo 指令和 「>>」 運算子，將環境變數字串新增至 .bashrc 檔案，如下所示：

```
echo "export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python" >> .bashrc
```

4. 然後，輸入 source 指令讓環境變數的配置生效，如下所示：

```
source ~/.bashrc
```

5. 在安裝前需要更新套件資料庫和升級已安裝套件，如下所示：

```
sudo apt update && sudo apt upgrade -y
```

6. 現在，我們就可以安裝 virtualenv 和 virtualenvwrapper 套件，如下所示：

```
sudo apt install -y python3-virtualenv && sudo apt install -y python3-virtualenvwrapper
```

7. 在成功安裝後，我們還需要設定環境變數 WORKON_HOME，其值是路徑 「$HOME/.virtualenvs」，請使用 echo 指令和 「>>」 運算子新增至 .bashrc 檔案，如下所示:

```
echo "export WORKON_HOME=$HOME/.virtualenvs" >> ~/.bashrc Enter
```

8. 然後設定在每次啟動時，自動執行 virtualenvwrapper.sh 腳本來使用virtualenvwrapper 套件，請使用 echo 指令和 「>>」 運算子新增至 .bashrc 檔案，如下所示：

```
echo "source /usr/share/virtualenvwrapper/virtualenvwrapper.sh" >> ~/.bashrc
```

9. 最後，輸入 source 指令讓我們更改的配置生效，如下所示：

```
source ~/.bashrc
```

使用

1. 建立虛擬環境
   我們準備新增一個名為 test 的 Python 虛擬環境，一個空的 Python 開發環境，使用的是mkvirtualenv 指令，如下所示：

```
cd ~/
mkvirtualenv test
```

> 建立後會直接啟動虛擬環境

建議
在指令後面加上 --system-site-packages 參數，建立的虛擬環境就可以存取樹莓派系統預設 Python 開發環境的安裝套件，如下所示：

```
mkvirtualenv --system-site-packages test
```

2. 關閉目前的 Python 虛擬環境

```
(test) pi@raspberrypi:~ $ deactivate
```

關閉成功，字首的 (test) 會消失。
3. 然後，我們可以執行 lsvirtualenv 指令查詢目前建立了哪些 Python 虛擬環境，可以顯示 Python 虛擬環境 test，如下所示：

```
lsvirtualenv
```

結果

```
pi@raspberrypi:~ $ lsvirtualenv
test
====
```

4. 退出 `test` 虛擬環境後，我們可以使用 `workon` 指令來再度啟動 `test` 虛擬環境，如下所示：

```
workon test
```

5. 進入 `test` 虛擬環境後，請輸入 `pip list` 指令來檢視 `test` 虛擬環境已經有安裝的套件有哪些，如下所示：

```
(test) pi@raspberrypi:~ $ pip list
```

上述指令的執行結果可以看到虛擬環境安裝的套件清單，如下所示：

```
Package                       Version
----------------------------- -------------------
appdirs                        1.4.4
asgiref                        3.6.0
astroid                        2.14.2
asttokens                      2.2.1
attrs                         22.2.0
av                            12.3.0
:                             :
```

6. 刪除虛擬環境
先以 `deactivate` 指令關閉 `test` 虛擬環境，在使用 `rmvirtualenv` 指令移除 `test` 虛擬環境，如下所示：

```
(test) pi@raspberrypi:~ deactivate
pi@raspberrypi:~ rmvirtualenv test
```

7. 指令集

指令	用途
`workon`	啟動 `Python` 虛擬環境
`deactivate`	關閉目前的 `Python` 虛擬環境
`lsvirtualenv`	查詢目前建立了哪些 `Python` 虛擬環境
`pip list`	檢視 `Python` 虛擬環境安裝的套件清單
`rmvirtualenv`	移除 `Python` 虛擬環境

使用 `conda`

如果您使用 `Anaconda` 或 `Miniconda`，可以使用 `conda` 建立指定 `Python` 版本的虛擬環境：

1. 下載 `Miniconda` 安裝腳本（for `ARM64`）
如果你用的是 `Raspberry Pi 5（64-bit OS）`，請執行：

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-aarch64.sh
```

2. 執行安裝腳本
`bash Miniconda3-latest-Linux-aarch64.sh`

```
接著按提示操作：
- 按 Enter 閱讀授權
- 輸入 yes 同意
- 預設安裝目錄通常是 ~/miniconda3，可以直接按 Enter
- 安裝完成後選擇是否將 conda init 初始化動作加到 ~/.bashrc 執行（建議選擇 `yes`）
```

3. 重新載入環境變數
   `source ~/.bashrc`
4. 測試 `conda` 是否安裝成功
   `conda --version`
5. 建立 `Conda` 虛擬環境

```
conda create -n tf-model-env3.9 python=3.9
```

6. 啟動虛擬環境：

```
conda activate tf-model-env3.9
```

7. 查詢已安裝虛擬環境

```
conda env list
```

8. 關閉虛擬環境

```
conda deactivate
```

> 用的是 32-bit Raspberry Pi OS，Miniconda 沒有官方支援；請改用 virtualenv。
> Conda 管理的環境與 virtualenvwrapper 無衝突，可以同時使用，但請避免在同一個 shell 中混用

Poetry

之前 python 開發者通常使用 virtualenv或是 venv 等方式來建立虛擬環境並使用 pip 來安裝套件 但是這樣的一個方法最大的問題就是相依的問題。 pip 在解決套件版本衝突時常碰到比較多的麻煩。因為它缺乏依賴解析算法，導致環境不穩定與出現一些不可預測的錯誤，此外 pip 在移除套件本身，並不會自動移除其相依的其他套件。
這些問題，促使了，像 pipenv 和 poetry 這樣的工具發展，他們能更好處理套件相依性。
pipenv 因為有一些問題存在所以停滯發展了一個階段。 至於 poetry 目前已經是相當穩定且成熟的一個方案，建議使用它來操作。

Poetry 不單單只是套件管理工具而已。 Poetry 也內建了虛擬環境的管理功能 當在專案中使用 Poetry 進行初始化和安裝套件時，Poetry 會自動在專案目錄中建立一個虛擬環境。

使用 Poetry 來安裝與管理虛擬環境

1. 安裝 Poetry
   安裝 Poetry 時，Python 版本必須在 3.8 以上。

```
curl -sSL https://install.python-poetry.org | python3 -
```

   安裝完畢，請加入系統環境變數到 $HOME/.bashrc，內容如下:
   `nano ~/.bashrc`

```
 :
export PATH="/home/allen/.local/bin:$PATH"
```

編輯完成，請執行 `.bashrc`， 使用指令: `source ~/.bashrc`。

2. 檢查是否安裝成功

```
poetry --version
```

3. 檢查 `Poetry` 中的組態(設定)檔

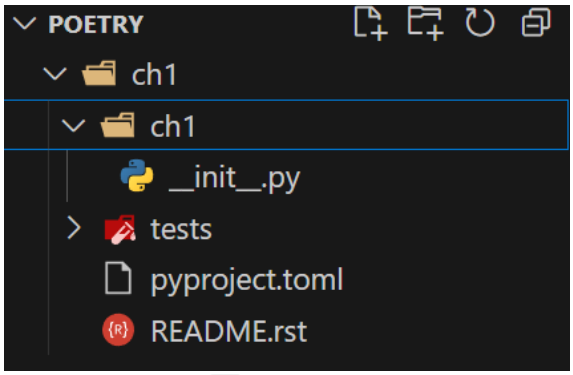
```
poetry config --list
```

4. 設定虛擬環境在專案中，透過**指令**開啟允許專案的虛擬環境設置功能。

```
poetry config virtualenvs.in-project true
```

5. 用 `Poerty` 建新專案
   使用下面指令，用 `Poetry` 來建一個新專案，這裡用 `ch1` 來命名

```
poetry new myFirstProject
```



6. 開始安裝虛擬環境 `env`

```
cd myFirstProject
poetry env use /home/allen/.pyenv/versions/3.13.1/bin/python3.13
poetry install
```

> 安裝完成會多了一個資料夾 `.venv`

```
allen@raspberrypi:~/python/myFirstProject $ ls  -al
total 28
drwxr-xr-x 5 allen allen 4096 Jun 18 15:29 .
drwxr-xr-x 3 allen allen 4096 Jun 18 14:21 ..
drwxr-xr-x 4 allen allen 4096 Jun 18 15:29 .venv
-rw-r--r-- 1 allen allen    0 Jun 18 14:21 README.md
-rw-r--r-- 1 allen allen  246 Jun 18 15:29 poetry.lock
-rw-r--r-- 1 allen allen  372 Jun 18 14:21 pyproject.toml
drwxr-xr-x 3 allen allen 4096 Jun 18 14:21 src
drwxr-xr-x 2 allen allen 4096 Jun 18 14:21 tests
```

7. 檢查 `python` 版本

```
cd myFirstProject
poetry run python --version
```

```
Python 3.13.1
```

若出現 `3.13` 以上的版本，就代表順利安裝成功

8. 安裝套件
   使用 `Poetry` 安裝套件指令是 `poetry add package_name`

```
# 安裝 langchain ^0.3.14 需要 `3.13.1` 以上版本至 < `4`
poetry add langchain
poetry add numpy
poetry add black
```

> Poetry 自動幫我解析好套件與安裝好其依賴的套件。

安裝完成，打開 `toml` 檔，會發現到多了一行程式碼:

```
dependencies = [
    "numpy (>=2.2.1,<3.0.0)",
    "blank (>=0.1.0,<0.2.0)",
    "langchain (>=0.3.14,<0.4.0)"
]
```

`poetry.lock` 文件是由 `Poetry` 自動生成的一個檔案，它會精確記錄了專案的依賴套件及其精確版本號。當在專案中使用 `portry` 添加一個套件時，`portry` 會在 `pyproject.toml` 文件中記錄這個套件，還會在 `poetry,lock` 文件中鎖定該檔案鎖定該套件的當前版本和其所相依的套件版本。

9. 啟動虛擬環境

```
# 切換到專案下
cd ~/myFirstProject
source .venv\bin\activate
```

結果

```
(myfirstproject-py3.13)
```

3.2 詳解 `pyproject.toml`

`pyproject.toml` 文件跟 `package.json` 是好相似的概念，是專案設定和依賴管理的核心，提供了一個中心化的地方來儲存專案的 `metadata` 和相依性，使得開發流程更加順暢一致。

- 運行時依賴相依

```
[tool.poetry.dependencies]
python = "^3.11"
langchain = "0.3.14"
numpy = ">=1.26.2,<3"
```

- 開發 `dev` 模式依賴相依

```
[tool.poetry.group.dev.dependencies]
 pytest = "^5.2"
 black = "^24.10.0"
```

> 只要安裝過 `poetry add package_name`, 異動過 `pyproject.toml`, 再次更新需使用指令 `poetry update package`

在 `pyproject.toml` 中，可以指定專案的**依賴**及其**版本**，指定版本的方法有以下常用方法:

1. 精確版本

```
[tool.poetry.dependencies]
numpy = "1.21.0"
```

> 指定一個明確的版本，代表只有該特定版本會接受。

2. 指定範圍

```
[tool.poetry.dependencies]
numpy = ">=1.21.0, <2.0.0"
```

任何版本從 `1.21.0` (含) 到 `2.0.0` (不含) 都是可以接受的。

3. `*` (所有)

```
[tool.poetry.dependencies]
numpy = "1.21.*"
```

任何兼容的版本， 使用 `1.21` 系列，例如: `1.21.1`、`1.21.2`。

4. `caret` 指定版本字首數字

```
[tool.poetry.dependencies]
numpy = "^1.21.0"
```

> 允許版本為 `1` 開頭的數字，但需低於 `2.0.0` 的版本

5. `~` 特定範圍

```
[tool.poetry.dependencies]
numpy = "~1.21.0"
```

> 允許 `1.21.0` 以及任何更高的版本，但是低於 `1.22.0` 的版本

6. 多版本指定 `[ , , ,...]`

```
[tool.poetry.dependencies]
numpy =[ ">=1.20.0, <1.21.0", ">=1.26.2,<3"]
```

3.3 使用 Python-dotenv 來管理環境變數

在開發軟體時，管理環境變數是一個重要的步驟。環境變數常用來儲存敏感資訊，如資料庫連接字串、API 金鑰等，將這些資訊與程式碼分離，能夠提高安全性並使專案易於維護。 `Python-dotenv` 是一個常用的工具，可以幫助你輕鬆管理環境變數。

`Python-dotenv` 可以從 `.env` 文件中讀取環境變數並將它們載入到運行中的應用程式。這樣就可以將敏感資訊存放到 `.env` 文件中，而不是直接寫在程式碼裡。

1. 安裝 `Python-dotenv`

```
cd myFirstProject
poetry add python-dotenv
```

2. 建立 `.env` 文件
   - 在專案裡面的資料夾 `src` 有一個檔案 `__init__.py`，同層級目錄裡建立一支 `Python` 檔: `01_dotenv.py`

```
from dotenv import load_dotenv
import os

# 載入 .env 文件
load_dotenv()

# 讀取環境變數
aoai_key = os.getenv("AOAI_API_KEY")

print(f"AOAI key: {aoai_key}")
```

- 在專案裡，與 `pyproject.toml` 同層位置，建立一個 `.env` 檔，然後編輯該檔案內容，如下:

```
AOAI_API_KEY=5566
```

3. 指定路徑的 `.env` 檔
   可以指定要哪個路徑下的 `.env` 檔，甚至多個 `.env` 檔。
   修改程式碼如下即可:

```
#load_dotenv()
load_dotenv(dotenv_path='/home/allen/python/myFirstProject/.env')
```

4. 使用 `dotenv_values`
   除了上面的方法，還可以用 `dotenv_values` 來把環境變數載入，這樣子載入的方式是把環境變數變成 `JSON` 的方式，所以取值就不用 `os.getenv`。

```
from dotenv import dotenv_values

config = dotenv_values(".env")

print(config.get("AOAI_API_KEY"))
```

5. `poetry 命令`

命令	**說明**
`poetry config --list`	列出 `Poetry` 中的設定
`poetry --version`	列出 `Poetry` 安裝的版本
`poetry new projectName`	產生 `Poetry` 專案
`poetry install`	相依套件安裝
`poetry add package_name`	安裝套件
`poetry remove package_name`	移除套件
`poetry env remove <環境名稱>`	移除虛擬環境
`poetry env use <環境名稱>`	使用虛擬環境
`poetry show`	列出已安裝的套件
`poetry update`	依賴套件的升級
`poetry shell`	激活虛擬環境

使用 uv 建立 Python 虛擬環境

1. 安裝 uv

```
curl -Ls https://astral.sh/uv/install.sh | bash
```

> 安裝於 $HOME/.local/bin

2. 將 uv 加入 PATH（假設安裝在 $HOME/.local/bin）

nano ~/.bashrc

```
export PATH="$HOME/.local/bin:$PATH"
```

執行

```
source ~/.bashrc
```

3. 建立虛擬環境（例如建立在 ~/uv-env）

```
uv venv ~/uv-env
```

4. 啟用虛擬環境

```
source ~/uv-env/bin/activate
```

5. 測試：確認 Python 與 pip 版本

```
(uv-env) pi@raspberrypi:~ python --version
(uv-env) pi@raspberrypi:~ pip --version
```

6. 安裝套件範例（例如安裝 requests）

```
(uv-env) pi@raspberrypi:~  pip install requests
```

7. 關閉虛擬環境

```
(uv-env) pi@raspberrypi:~ deactivate
```