

uvを使ってPython環境を構築する

個人開発集会 2024.12.19 HIDE

自己紹介

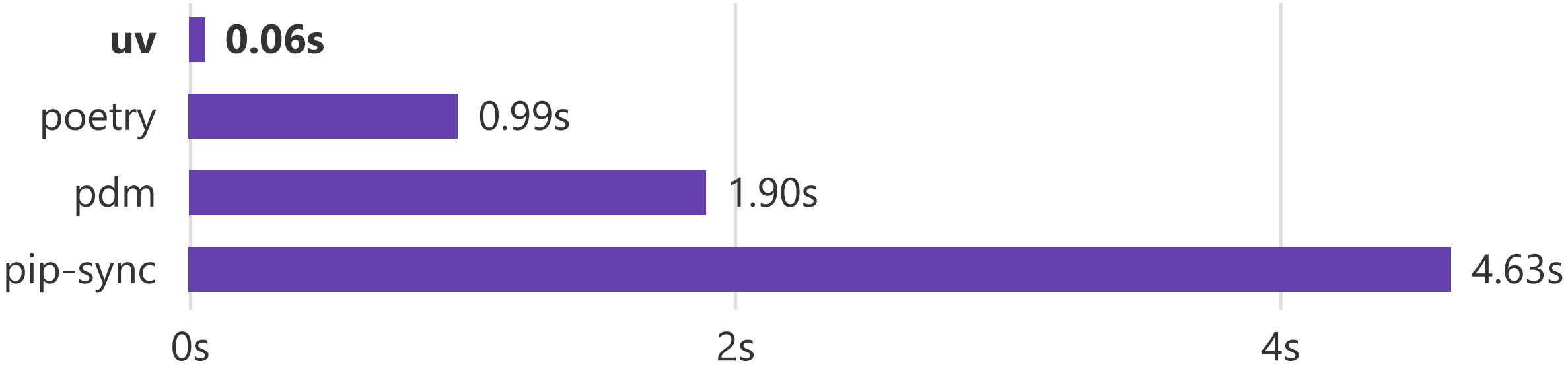
HIDE

- データサイエンス、ソフトウェアの仕事してます
- ハードウェアもわかります
- VRChat: hide3791
- X: @NPComplete00



今日話すこと - uvを使ってみた話

uv - An extremely fast Python package and project manager, written in Rust.



uvを使ってみようと思ったわけ

uvは使い方はとても簡単、環境の不整合を防ぐのにとても有用

pythonを使った開発プロジェクトを複数人でやっていると、

- 自分だけ、なんか動作がおかしい
- いろいろ調べてみると、自分だけが周囲とパッケージのバージョンが違うみたい
- どのパッケージのバージョンが合っていないのだろう？

このような状況に陥って、時間を浪費することがある
→ uvはこういったことを防げる、極論すると`git pull`して`uv sync`すればOK!
膨大な数のパッケージを使うプロジェクトでは効果大きい！

Contents

- uvの特長
- uvのインストール
- Python管理 | `uv python list`, `uv python install`, `uv python find`
- プロジェクト管理と作成 | `uv init`
- 依存関係の管理 | `uv add`, `uv pip list`, `uv sync`
- 作成した環境での実行 | `uv run`
- おわり
- Appendix

uvの特長

- 🚀 A single tool to replace `pip`, `pip-tools`, `pipx`, `poetry`, `pyenv`, `twine`, `virtualenv`, and more.
- ⚡ 10-100x faster than `pip`.
- 🐍 Installs and manages Python versions.
- 🛠️ Runs and installs Python applications.
- ❇️ Runs single-file scripts, with support for inline dependency metadata.
- 🗂️ Provides comprehensive project management, with a universal lockfile.
- 🔩 Includes a pip-compatible interface for a performance boost with a familiar CLI.
- 🏢 Supports Cargo-style workspaces for scalable projects.
- 💾 Disk-space efficient, with a global cache for dependency deduplication.
- ⏬ Installable without Rust or Python via `curl` or `pip`.
- 💻 Supports macOS, Linux, and Windows.

uvのインストール

- 環境
  - ubuntu 24.04.1 LTS on WSL2
- インストール

```
$ curl -LsSf https://astral.sh/uv/install.sh | sh
$ uv --version
uv 0.5.8
```

```
$ echo 'eval "$(uv generate-shell-completion zsh)"' >> ~/.zshrc  # 補完スクリプト
```

- アップデート

```
$ uv self update  # 新しいバージョンがあればアップデートされる
```

Python管理 | `uv python list`

インストールされているpythonを確認する。3.12がインストールされているのがわかる。

```
$ uv python list
cpython-3.13.1+freethreaded-linux-x86_64-gnu    <download available>
cpython-3.13.1-linux-x86_64-gnu                 <download available>
cpython-3.12.8-linux-x86_64-gnu                 <download available>
cpython-3.12.3-linux-x86_64-gnu                 /usr/bin/python3.12
cpython-3.12.3-linux-x86_64-gnu                 /usr/bin/python3 -> python3.12
cpython-3.12.3-linux-x86_64-gnu                 /bin/python3.12
cpython-3.12.3-linux-x86_64-gnu                 /bin/python3 -> python3.12
cpython-3.11.11-linux-x86_64-gnu                <download available>
cpython-3.10.16-linux-x86_64-gnu                <download available>
cpython-3.9.21-linux-x86_64-gnu                 <download available>
cpython-3.8.20-linux-x86_64-gnu                 <download available>
cpython-3.7.9-linux-x86_64-gnu                  <download available>
pypy-3.10.14-linux-x86_64-gnu                   <download available>
pypy-3.9.19-linux-x86_64-gnu                    <download available>
pypy-3.8.16-linux-x86_64-gnu                    <download available>
pypy-3.7.13-linux-x86_64-gnu                    <download available>
```

Python管理 | `uv python install`, `uv python find`

python 3.13をインストールする。

```
$ uv python install 3.13
Installed Python 3.13.1 in 1.69s
 + cpython-3.13.1-linux-x86_64-gnu
```

3.13がインストールされているかを確認する。

```
$ uv python find 3.13
/home/foo/.local/share/uv/python/cpython-3.13.1-linux-x86_64-gnu/bin/python3.13
```

プロジェクト管理 | `uv init`

プロジェクトを新規作成する。

```
$ uv init project1 --python 3.12  # 3.12を指定
Initialized project `project1` at `/home/foo/workspace/project1`
```

作成したフォルダ構成は下記のようになる。デフォルトでGitリポジトリが作られる。

```
$ ls -a project1/
.  ..  .git  .gitignore  .python-version  README.md  hello.py  pyproject.toml
```

プロジェクトのPythonバージョンを見てみる。

```
$ cd project1/
$ cat .python-version
3.12
```

プロジェクトの作成 | `uv init` (cont'd)

プロジェクトの情報を見てみる。
pythonのバージョンは3.12以上になっている。
パッケージを何も追加していないので、`dependencies`は空になっている。

```
$ cat pyproject.toml
[project]
name = "project1"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.12"
dependencies = []
```

プロジェクトの作成 | `uv init` (cont'd)

Gitリポジトリの状態を見てみる。

```
$ git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        .python-version
        README.md
        hello.py
        pyproject.toml
nothing added to commit but untracked files present (use "git add" to track)
```

まずはコミットする。

```
$ git add .gitignore .python-version README.md hello.py pyproject.toml
$ git commit -m "Initial"
```

依存関係の管理 | `uv add`, `uv pip list`

プロジェクトにpandasパッケージをインストールしてみる。

```
$ uv add pandas==2.2.2  # 2.2.2を指定
Resolved 7 packages in 289ms
Prepared 2 packages in 1.49s
Installed 6 packages in 16ms
 + numpy==2.2.0
 + pandas==2.2.2
 + python-dateutil==2.9.0.post0
 + pytz==2024.2
 + six==1.17.0
 + tzdata==2024.2
```

依存関係の管理 | `uv add`, `uv pip list` (cont'd)

インストールされているパッケージを確認する。

```
$ uv pip list
Package          Version
---------------- -----------
numpy            2.2.1
pandas           2.2.2
python-dateutil 2.9.0.post0
pytz             2024.2
six              1.17.0
tzdata           2024.2
```

依存関係の管理｜`uv add`, `uv pip list` (cont`d)

Gitリポジトリの状態を見てみる。`pyproject.toml` が変更されて、`uv.lock` ができている。

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   pyproject.toml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        uv.lock
no changes added to commit (use "git add" and/or "git commit -a")
```

pandas 2.2.2のインストールをコミットする。

```
$ git add pyproject.toml uv.lock
$ git commit -m "Add pandas 2.2.2"
```

依存関係の管理｜`uv add`, `uv pip list` (cont`d)

`pyproject.toml` の変更箇所を見てみる。pandasがインストールされている。

```
$ git diff pyproject.toml
diff --git a/pyproject.toml b/pyproject.toml
index 16cdb56..35c15a9 100644
--- a/pyproject.toml
+++ b/pyproject.toml
@@ -4,4 +4,6 @@ version = "0.1.0"
 description = "Add your description here"
 readme = "README.md"
 requires-python = ">=3.12"
-dependencies = []
+dependencies = [
+    "pandas==2.2.2",
+]
```

依存関係の管理｜`uv add`, `uv pip list` (cont`d)

branchを切り替えて、pandasを2.2.2から2.2.3にアップデートしてみる。

```
$ git checkout -b test  # branchを切り替える
$ uv add pandas==2.2.3  # 2.2.3を指定
Resolved 7 packages in 258ms
Prepared 1 package in 1.27s
Uninstalled 1 package in 9ms
Installed 1 package in 15ms
 - pandas==2.2.2
 + pandas==2.2.3
```

`pyproject.toml` と `uv.lock` が変更されている。

```
$ git status
On branch test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   pyproject.toml
        modified:   uv.lock
```

依存関係の管理 | `uv add`, `uv pip list` (cont`d)

`pyproject.toml` を見ると、pandas 2.2.3になっているのがわかる。

```
$ cat pyproject.toml
[project]
name = "project1"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.12"
dependencies = [
    "pandas==2.2.3",
]
```

pandas 2.2.3のインストールをコミットする。

```
$ git add pyproject.toml uv.lock
$ git commit -m "Update pandas 2.2.3"
```

依存関係の管理｜`uv add`, `uv pip list` (cont`d)

branchを戻すと`pyproject.toml`内の記載は2.2.2に戻る。但し、実際は2.2.3のまま。

```
$ git checkout master
$ cat pyproject.toml
[project]
name = "project1"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.12"
dependencies = [
    "pandas==2.2.2",
]

$ uv pip list
Package         Version
--------------- -----------
numpy           2.2.0
pandas          2.2.3       # 2.2.3のまま
--以下略-
```

依存関係の管理｜uv sync

`uv sync` すると2.2.2が実際にインストールされる。

```
$ uv sync
Resolved 7 packages in 0.46ms
Uninstalled 1 package in 9ms
Installed 1 package in 15ms
 - pandas==2.2.3
 + pandas==2.2.2
```

```
$ uv pip list
Package         Version
--------------- -----------
numpy           2.2.0
pandas          2.2.2       # 2.2.2に戻る
--以下略-
```

作成した環境での実行｜uv run

pytorchの超簡単な実行例

```
$ uv add torch torchvision torchaudio
```

インストールしたパッケージが `pyproject.toml` に反映されている。

```
$ git diff pyproject.toml
diff --git a/pyproject.toml b/pyproject.toml
index 35c15a9..bfcf437 100644
--- a/pyproject.toml
+++ b/pyproject.toml
@@ -6,4 +6,7 @@ readme = "README.md"
 requires-python = ">=3.12"
 dependencies = [
     "pandas==2.2.2",
+    "torch>=2.5.1",
+    "torchaudio>=2.5.1",
+    "torchvision>=0.20.1",
 ]
```

作成した環境での実行｜uv run (cont'd)

```
# バージョンとcuDNNを確認するだけのスクリプト
import torch
print(torch.__version__)
print(torch.backends.cudnn.is_available())
```

作成したpython仮想環境で実行する。

```
$ uv run sample_pytorch.py
2.5.1+cu124
True
```

python仮想環境を明示的に切り替えてから実行することもできる。

```
$ source .venv/bin/activate
(project1) $ python sample_pytorch.py
2.5.1+cu124
True
```

作成した環境での実行｜uv run (cont'd)

パッケージを入れたときに仮想環境のフォルダ `.venv` が作られる。

```
$ ls -a
.   .git          .python-version  README.md  pyproject.toml     uv.lock
..  .gitignore    .venv            hello.py   sample_pytorch.py
```

また自動的に作られる `.gitignore` は `.venv` をコミット対象から除外している。

```
$ cat .gitignore
# Python-generated files
__pycache__/
*.py[oc]
build/
dist/
wheels/
*.egg-info

# Virtual environments
.venv
```

おわり

- uvの機能はもっといっぱいあって、いろいろなことができます
- 今回は、pythonを使う開発プロジェクトで発生しがちなpythonバージョンやパッケージの不整合の防止にフォーカスして話してみました
- もっと使い込んでるよ！って人は、ぜひ自分に使い方を教えてください

Appendix | `uv tree`

(LTでは触れていなかったのですが) `uv tree` でパッケージの依存関係を確認できます。

```
$ uv pip list
Package         Version
--------------- -----------
numpy           2.2.1
pandas          2.2.2
python-dateutil 2.9.0.post0
pytz            2024.2
six             1.17.0
tzdata          2024.2
$ uv tree
Resolved 7 packages in 6ms
project1 v0.1.0
└── pandas v2.2.2
    ├── numpy v2.2.1
    ├── python-dateutil v2.9.0.post0
    │   └── six v1.17.0
    ├── pytz v2024.2
    └── tzdata v2024.2
```