# uv 치트시트

## uv 사용 가이드: Python 프로젝트를 위한 초고속 도구

uv는 Rust로 작성된 초고속 Python 패키지 설치 및 환경 관리 도구입니다. pip, pip-tools, venv의 기능을 통합하여 놀라운 속도로 Python 개발 워크플로우를 혁신합니다.

## 1. 프로젝트 초기화 및 가상 환경 관리 (uv init, uv venv)

uv는 새로운 Python 프로젝트를 시작하거나 기존 프로젝트를 uv와 함께 사용하도록 설정하는 데 탁월합니다. 가상 환경을 매우 빠르게 생성하고 관리할 수 있습니다.

- uv init: 현재 디렉토리에 새로운 uv 프로젝트를 초기화합니다. pyproject.toml과 uv.lock 파일을 생성하여 종속성 관리를 시작합니다.
- uv venv: .venv라는 이름의 가상 환경을 생성합니다.
  - uv venv my-env: my-env라는 이름으로 가상 환경을 생성합니다.
- uv venv --python <version>: 특정 Python 버전을 사용하여 가상 환경을 생성합니다.
  - 시스템에 설치된 python3.11, python3.10 등을 자동으로 찾아 사용합니다.
- uv venv --seed: pip, setuptools, wheel을 가상 환경에 미리 설치합니다.
- 가상 환경 활성화:
  - Linux/macOS: source .venv/bin/activate
  - Windows: .venv\Scripts\activate
- 가상 환경 비활성화: deactivate

## 2. 패키지 추가 및 제거 (uv add, uv remove)

uv는 pyproject.toml 파일에 의존성을 추가하거나 제거하는 직관적인 명령어를 제공합니다. 이 명령어들은 내부적으로 필요한 패키지 설치 및 환경 동기화를 자동으로 처리합니다.

- uv add <package>: 패키지를 pyproject.toml의 의존성 목록에 추가하고 설치합니다.
  - uv add requests: requests 패키지를 추가합니다.
  - uv add 'requests==2.31.0': 특정 버전의 requests를 추가합니다.
  - uv add ".[dev]": 개발(dev)용 추가 종속성 그룹을 포함하여 현재 프로젝트를 설치합니다.
  - uv add -r requirements.txt: 기존 requirements.txt 파일의 모든 종속성을 pyproject.toml에 추가하고 설치합니다.
- uv remove <package>: 패키지를 pyproject.toml의 의존성 목록에서 제거하고 환경에서 삭제합니다.
  - uv remove requests: requests 패키지를 제거합니다.
  - uv remove -r requirements.txt: 파일에 명시된 모든 패키지를 환경에서 제거합니다.

## 3. 환경 동기화 (uv sync)

uv sync는 uv 프로젝트의 핵심 명령어로, pyproject.toml 또는 uv.lock 파일에 정의된 대로 현재 환경을 정확하게 일치시킵니다.

- uv sync: pyproject.toml 또는 uv.lock 파일에 따라 현재 활성화된 환경을 동기화합니다.
  - 파일에 명시된 패키지는 설치/업데이트됩니다.
  - 파일에 없는 패키지는 환경에서 제거됩니다.
  - 이 기능은 재현 가능한 환경을 유지하고 팀원 간의 환경 불일치를 방지하는 데 매우 유용합니다.

## 4. 종속성 고정 및 해결 (uv lock)

uv lock은 uv add나 uv remove를 실행할 때 자동으로 작동하지만, 명시적으로 실행하여 의존성 버전을 고정하거나 업데이트할 수 있습니다.

- uv lock: pyproject.toml의 의존성을 기반으로 uv.lock 파일을 업데이트합니다. 이 파일은 모든 직접 및 전이 종속성에 대한 정확한 버전을 기록합니다.
- uv lock --upgrade-package <package>: 특정 패키지를 최신 호환 버전으로 업그레이드하면서 uv.lock 파일을 업데이트합니다.
- uv lock --strict: 의존성 충돌이 발생하면 오류를 발생시킵니다.
- uv lock --generate-hashes: uv.lock에 파일 해시를 포함하여 보안을 강화합니다.

## 5. 설치된 패키지 확인 (uv pip list, uv pip freeze, uv pip check)

uv는 현재 환경에 설치된 패키지를 쉽게 확인하고 검증할 수 있는 명령어를 제공합니다.

- uv pip list: 설치된 패키지 목록을 pip list와 유사한 형식으로 표시합니다.
- uv pip freeze: 설치된 패키지를 requirements.txt 형식으로 출력합니다.
- uv pip check: 설치된 패키지 간의 종속성이 호환되는지 확인하여 잠재적인 문제를 보고합니다.

## 6. 캐시 관리 (uv cache)

uv는 빠른 속도를 위해 로컬 캐시를 적극적으로 활용합니다. 캐시를 효율적으로 관리하여 디스크 공간을 확보하거나 문제를 해결할 수 있습니다.

- uv cache clean: uv의 모든 캐시를 지웁니다.
- uv cache clean --name <package>: 특정 패키지의 캐시만 지웁니다.
- uv cache dir: 캐시 디렉토리의 경로를 보여줍니다.

## uv를 활용한 효율적인 Python 워크플로우 예시

uv는 개발자가 재현 가능하고 효율적인 Python 환경을 설정하고 유지할 수 있도록 돕습니다. 다음은 일반적인 워크플로우 예시입니다.

1. 새 프로젝트 시작 또는 기존 프로젝트 초기화:

   ```
   uv init my-new-project
   cd my-new-project
   ```

   이 명령은 my-new-project 디렉토리를 생성하고 pyproject.toml, uv.lock 파일을 초기화합니다. 가상 환경도 자동으로 생성됩니다.
2. 기본 의존성 추가:

   ```
   uv add fastapi uvicorn
   ```

   이 명령은 fastapi와 uvicorn을 pyproject.toml에 추가하고, uv.lock 파일을 업데이트하며, .venv 가상 환경에 해당 패키지들을 설치합니다.
3. 개발용 의존성 추가 (예: 테스트 프레임워크):

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

   pyproject.toml의 [dependency-groups.dev] 섹션에 pytest를 추가하고 설치합니다.
4. 환경 동기화 (협업 시 또는 pyproject.toml 수동 변경 후):

   ```
   uv sync
   ```

   이 명령은 pyproject.toml 또는 uv.lock에 정의된 대로 현재 가상 환경의 패키지들을 정확히 일치시킵니다. 다른 사람이 프로젝트를 클론할 때도 uv sync만으로 모든 의존성을 쉽고 빠르게 설치할 수 있습니다.
5. 스크립트 실행:

   ```
   uv run uvicorn main:app --reload
   ```

   uv run은 프로젝트의 가상 환경을 자동으로 활성화하고 지정된 명령어를 실행합니다.

이 워크플로우는 uv add, uv remove, uv sync 및 uv lock (자동 또는 수동)을 통해 pyproject.toml과 uv.lock 파일을 기반으로 일관되고 재현 가능한 환경을 유지하도록 보장합니다.

## pyproject.toml 파일의 예시

```toml
[build-system]
# [build-system] 테이블은 프로젝트를 빌드하는 데 필요한 도구와 방법을 지정합니다.
# 이 섹션은 프로젝트가 어떻게 빌드되어야 하는지(예: 휠 파일 생성) 빌드 프론트엔드(pip 등)에 알려줍니다.

requires = ["setuptools>=61.0", "wheel"]
# requires: 프로젝트를 빌드하기 위해 필요한 패키지 목록을 정의합니다.
# 일반적으로 setuptools, hatchling, poetry-core 등 빌드 백엔드 패키지가 포함됩니다.

build-backend = "setuptools.build_meta"
# build-backend: 프로젝트를 빌드하는 데 사용될 Python 모듈을 지정합니다.
# 'setuptools.build_meta'는 setuptools를 빌드 백엔드로 사용함을 의미합니다.

[project]
# [project] 테이블은 프로젝트 자체에 대한 표준 메타데이터를 정의합니다.
# 이는 PyPI(Python Package Index)에 패키지를 게시할 때 사용되는 정보이기도 합니다.
# name: 패키지의 이름입니다. PyPI에 게시될 이름이며, PEP 508에 따라 유효해야 합니다.
name = "my-awesome-package"
# version: 패키지의 현재 버전입니다. PEP 440에 따라 유효해야 합니다.
version = "0.1.0"
# description: 패키지에 대한 짧은 한 줄 설명입니다.
description = "이것은 제가 만든 아주 멋진
```

```
Python 패키지입니다."
# readme: README 파일의 경로를 지정합니다.
# PyPI 페이지에 프로젝트 설명을 표시하는 데 사용됩니다.
# content-type을 명시하여 PyPI가 README를 올바르게 렌더링하도록 합니다.
readme = { file = "README.md", content-type = "text/markdown" }
# requires-python: 이 패키지가 호환되는 Python 버전 범위입니다.
requires-python = "≥3.8"
# license: 프로젝트의 라이선스를 지정합니다.
# 텍스트 형식으로 직접 명시하거나, 파일 경로를 지정할 수 있습니다.
license = { text = "MIT License" } # 또는 { file = "LICENSE" }
# authors: 프로젝트의 저자 정보를 포함하는 테이블 배열입니다.
# 각 테이블은 이름과 이메일을 포함할 수 있습니다.
authors = [
    { name = "홍길동", email = "hong.gildong@example.com" },
    { name = "김철수", email = "kim.chulsoo@example.com" },
]
# keywords: 패키지와 관련된 키워드 목록입니다. 검색 가능성을 높여줍니다.
keywords = ["awesome", "utility", "python", "example"]
# classifiers: PyPI 트로브 분류자 목록입니다.
# 프로젝트의 상태, 라이선스, 지원되는 Python 버전 등을 설명합니다.
# PyPI에서 프로젝트를 필터링하고 검색하는 데 도움이 됩니다.
# 전체 목록은 https://pypi.org/classifiers/ 에서 확인할 수 있습니다.
classifiers = [
    "Development Status :: 3 - Alpha",
    "Intended Audience :: Developers",
    "License :: OSI Approved :: MIT License",
    "Programming Language :: Python :: 3",
    "Programming Language :: Python :: 3.8",
    "Programming Language :: Python :: 3.12",
    "Operating System :: OS Independent",
    "Topic :: Software Development :: Libraries :: Python Modules",
]
# dependencies: 프로젝트의 런타임 종속성 목록입니다.
# 이 패키지가 작동하기 위해 필요한 다른 패키지들을 명시합니다.
dependencies = [
    "requests≥2.25.1",
    "beautifulsoup4<5.0.0",
    "lxml; platform_system == 'Linux'",
# 특정 플랫폼에만 적용되는 종속성
]

[project.urls]
# [project.urls] 테이블은 프로젝트 관련 URL을 정의합니다.
# 일반적으로 문서, 홈 페이지, 소스 코드 저장소 링크 등을 포함합니다.
Homepage = "https://github.com/your-username/my-awesome-package"
Documentation = "https://my-awesome-package.readthedocs.io/"
Repository = "https://github.com/your-username/my-awesome-package.git"
"Bug Tracker" = "https://github.com/your-username/my-awesome-package/issues"

[project.optional-dependencies]
# [project.optional-dependencies] 테이블은 선택적 종속성을 정의합니다.
# 'dev', 'test', 'docs' 등과 같이 특정 목적을 위한 추가 종속성 그룹을 만듭니다.
# dev: 개발 환경 설정에 필요한 패키지입니다. (예: 린터, 포맷터)
dev = [
    "pytest≥7.0",
    "flake8",
    "black",
    "isort",
    "mypy",
]
# docs: 문서 생성에 필요한 패키지입니다.
docs = [
    "sphinx≥4.0",
    "sphinx-rtd-theme",
]

[project.scripts]
# [project.scripts] 테이블은 명령줄 스크립트 엔트리 포인트를 정의합니다.
# 이 섹션에 정의된 스크립트는 패키지 설치 시 시스템 PATH에 추가되어 명령줄에서 바로 실행할 수 있게 됩니다.
my-cli = "my_package.cli:main" # 'my-cli'라는 명령어를 'my_package/cli.py' 파일의 'main' 함수에 연결합니다.

[project.gui-scripts]
# [project.gui-scripts] 테이블은 GUI 애플리케이션 엔트리 포인트를 정의합니다.
my-gui = "my_package.gui:start_app"

# [tool] 테이블은 특정 도구에 대한 설정을 포함합니다.
# PyPA 표준에 속하지 않는 모든 도구별 설정은 이 아래에 정의됩니다.
# 각 도구는 자체 이름으로 하위 테이블을 가집니다 (예: [tool.black], [tool.mypy]).
```