uv 치트시트

uv 사용 가이드: 파이썬 프로젝트를 위한 차세대 초고속 도구

uv는 Rust로 작성된 초고속 파이썬 패키지 설치 및 환경 관리 도구입니다. 기존 `pip`, `pip-tools`, `venv`의 기능을 하나로 통합하여, 파이썬 개발 워크플로우의 속도를 획기적으로 개선합니다.

프로젝트 초기화 및 가상 환경 관리 (`uv init`, `uv venv`)

`uv`는 새로운 파이썬 프로젝트를 시작하거나 기존 프로젝트에 도입할 때 매우 유용합니다. 특히 가상 환경 생성 속도가 압도적으로 빠릅니다.

- `uv init`: 현재 디렉토리에 새로운 `uv` 프로젝트를 초기화합니다. `pyproject.toml`과 `uv.lock` 파일을 생성하여 의존성 관리를 시작할 준비를 합니다.
- `uv venv`: `.venv` 폴더에 가상 환경을 생성합니다.
  - `uv venv my-env`: 지정한 이름(`my-env`)으로 가상 환경을 생성합니다.
- `uv venv --python <version>`: 특정 파이썬 버전을 지정하여 가상 환경을 만듭니다.
  - 시스템에 설치된 파이썬 버전들을 자동으로 탐색하여 사용합니다.
- `uv venv --seed`: 가상 환경 생성 시 `pip`, `setuptools`, `wheel` 패키지를 미리 설치(seed)합니다.
- 가상 환경 활성화:
  - Linux/macOS: `source .venv/bin/activate`
  - Windows: `.venv\Scripts\activate`
- 가상 환경 비활성화: `deactivate` 입력

패키지 추가 및 제거 (`uv add`, `uv remove`)

`uv`는 `pyproject.toml` 파일을 직접 수정하지 않고도 의존성을 관리할 수 있는 직관적인 명령어를 제공합니다. 패키지 설치와 파일 업데이트가 동시에 처리됩니다.

- `uv add <package>`: 패키지를 의존성 목록에 추가하고 즉시 설치합니다.
  - `uv add requests`: `requests` 패키지 추가.
  - `uv add 'requests==2.31.0'`: 특정 버전 명시.
  - `uv add ".[dev]"`: 현재 프로젝트를 개발용 선택적 의존성(extra)과 함께 설치.
  - `uv add -r requirements.txt`: 기존 텍스트 파일에 명시된 모든 패키지 일괄 추가.
- `uv remove <package>`: 의존성 목록에서 패키지를 제거하고 환경에서도 삭제합니다.

환경 동기화 (`uv sync`)

`uv sync`는 설정 파일(`pyproject.toml` 또는 `uv.lock`)의 내용과 실제 설치된 가상 환경의 상태를 완벽하게 일치시키는 핵심 명령어입니다.

- `uv sync`: 현재 활성화된 가상 환경을 설정 파일 상태와 동일하게 동기화합니다.
  - 목록에 없는 패키지는 자동으로 제거됩니다.
  - 팀원 간에 동일한 개발 환경을 유지하는 데 매우 효과적입니다.

의존성 버전 고정 및 해결 (`uv lock`)

`uv lock`은 의존성 관계를 계산하여 버전을 고정합니다. `add`나 `remove` 시 자동으로 실행되지만, 명시적으로 호출하여 갱신할 수도 있습니다.

- `uv lock`: 모든 직접 및 간접 의존성의 정확한 버전을 `uv.lock` 파일에 기록합니다.
- `uv lock --upgrade-package <package>`: 다른 패키지는 그대로 두고 특정 패키지만 최신 버전으로 업데이트하며 락 파일을 갱신합니다.
- `uv lock --strict`: 의존성 충돌이 발생할 경우 경고 대신 에러를 발생시켜 엄격하게 관리합니다.

설치된 패키지 확인 및 검증

기존 `pip` 명령어와 유사한 인터페이스를 통해 현재 환경을 점검할 수 있습니다.

- `uv pip list`: 설치된 모든 패키지 목록을 출력합니다.
- `uv pip freeze`: `requirements.txt` 형식으로 현재 설치된 패키지 정보를 출력합니다.
- `uv pip check`: 설치된 패키지 간의 의존성 호환성에 문제가 없는지 검증합니다.

캐시 관리 (`uv cache`)

`uv`는 압도적인 속도를 위해 로컬 캐시를 적극적으로 사용합니다. 문제가 발생하거나 용량을 확보해야 할 때 사용합니다.

- `uv cache clean`: 저장된 모든 빌드 및 다운로드 캐시를 삭제합니다.
- `uv cache dir`: 캐시 데이터가 저장되는 실제 경로를 확인합니다.

효율적인 uv 개발 워크플로우 예시

1. **프로젝트 시작**:
   ```
   uv init my-project
   cd my-project
   ```
2. **필요한 의존성 추가**:
   ```
   uv add fastapi uvicorn
   ```
3. **개발 도구 추가**:
   ```
   uv add pytest --group dev
   ```
4. **환경 동기화**:
   ```
   uv sync
   ```
5. **애플리케이션 실행**:
   ```
   uv run uvicorn main:app --reload
   ```

`pyproject.toml` 설정 파일 가이드

```toml
[build-system]
# 프로젝트 빌드 방식 및 도구 지정
requires = ["setuptools≥61.0", "wheel"]
build-backend = "setuptools.build_meta"

[project]
# 프로젝트 메타데이터 정의 (PyPI 배포 시 사용)
name = "my-awesome-package"
version = "0.1.0"
description = "초고속으로 동작하는 파이썬 프로젝트 예시"
readme = { file = "README.md", content-type = "text/markdown" }
requires-python = "≥3.8"
license = { text = "MIT License" }
authors = [
    { name = "홍길동", email = "gildong@example.com" },
]
keywords = ["awesome", "utility", "python"]

# 런타임에 필요한 필수 의존성 목록
dependencies = [
    "requests≥2.25.1",
    "beautifulsoup4<5.0.0",
]

[project.optional-dependencies]
# 개발, 테스트, 문서화 등 특정 목적을 위한 추가 의존성 그룹
dev = [
    "pytest≥7.0",
    "black",
    "mypy",
]

[project.scripts]
# 명령줄에서 바로 실행할 스크립트(CLI) 정의
my-cli = "my_package.cli:main"
```