Poetry 치트시트

1. 프로젝트 초기화 및 관리

- `poetry new <project_name>`: 표준 디렉토리 구조(`<project_name>/<project_name>`, `tests/`, `pyproject.toml`)와 함께 새 프로젝트를 생성합니다.
- `poetry init`: 기존 디렉토리에서 대화형으로 `pyproject.toml` 파일을 생성합니다.
- `poetry check`: `pyproject.toml` 파일의 구문 및 종속성 정의의 유효성을 검사합니다.
- `poetry show`: 프로젝트의 모든 종속성을 나열합니다.
  - `--tree`: 종속성을 트리 형태로 시각화하여 보여줍니다.
  - `--outdated`: 설치된 패키지 중 구버전이 있는지 확인합니다.
- `poetry search <package>`: PyPI에서 패키지를 검색합니다.

2. 종속성 관리

- `poetry add <package>`: 프로젝트에 종속성을 추가하고 `pyproject.toml`과 `poetry.lock` 파일을 업데이트합니다.
  - `poetry add "pandas@^1.0"`: 버전 제약 조건을 명시하여 추가합니다.
  - `poetry add --group dev <package>`: 개발용 종속성 그룹에 추가합니다. (예: `pytest`, `black`)
  - `poetry add --group docs <package>`: 문서용 종속성 그룹에 추가할 수 있습니다.
  - `poetry add "git+https://github.com/user/repo.git"`: Git 저장소에서 직접 패키지를 추가합니다.
- `poetry remove <package>`: 종속성을 제거합니다.
- `poetry install`: `poetry.lock` 파일이 있으면 해당 파일에 명시된 정확한 버전의 종속성을 설치합니다. 없으면 `pyproject.toml`을 기반으로 종속성을 해결하고 `poetry.lock`을 생성한 후 설치합니다.
  - `--no-dev`: 기본 `dependencies` 그룹만 설치합니다. (배포 환경용)
  - `--with <group1>,<group2>`: 특정 종속성 그룹을 포함하여 설치합니다.
  - `--sync`: `poetry.lock` 파일과 정확히 일치하도록 환경을 동기화합니다. (불필요한 패키지 제거)
- `poetry update`: `pyproject.toml`에 명시된 버전 제약 조건 내에서 종속성을 최신 버전으로 업데이트하고 `poetry.lock` 파일을 갱신합니다.
  - `poetry update <package1> <package2>`: 특정 패키지만 업데이트합니다.
- `poetry lock`: `pyproject.toml`의 종속성을 해결하여 `poetry.lock` 파일을 생성(또는 업데이트)하지만, 패키지를 설치하지는 않습니다. CI 환경에서 유용합니다.

3. 가상 환경 및 실행

- `poetry run <command>`: 프로젝트의 가상 환경 내에서 명령어를 실행합니다. (예: `poetry run pytest`)
- `poetry shell`: 프로젝트의 가상 환경을 활성화하는 새로운 셸을 시작합니다.
- `poetry config virtualenvs.in-project true`: 프로젝트 디렉토리 내에 `.venv` 폴더를 생성하도록 전역적으로 설정합니다.
- `poetry config --local virtualenvs.in-project true`: 현재 프로젝트에만 적용되도록 설정합니다.
- `poetry env info`: 현재 가상 환경에 대한 정보(경로, Python 버전 등)를 표시합니다.
- `poetry env list`: 이 프로젝트와 관련된 모든 가상 환경을 나열합니다.
- `poetry env use <python_executable_or_version>`: 프로젝트에 사용할 Python 버전을 변경합니다. (예: `python3.10`)
- `poetry env remove <python_version>`: 특정 가상 환경을 삭제합니다.

4. 빌드 및 배포

- `poetry build`: 소스 배포판(sdist)과 휠(wheel)을 `dist/` 디렉토리에 빌드합니다.
- `poetry publish`: 빌드된 패키지를 PyPI 또는 다른 저장소에 게시합니다.
  - `--build`: 게시하기 전에 먼저 빌드합니다.
  - `--repository <name>`: `pyproject.toml`에 설정된 특정 저장소에 게시합니다.
  - `--username <user> --password <pass>`: 사용자 이름과 비밀번호를 직접 입력합니다.
- `poetry config repositories.<name> <url>`: PyPI 외의 다른 패키지 저장소를 설정합니다.

5. `pyproject.toml` 심층 분석

```toml
[tool.poetry]
name = "my-awesome-project"
version = "0.1.0"
description = "A short description of the project."
authors = ["Your Name <you@example.com>"]
license = "MIT"
readme = "README.md"
homepage = "https://poetry.eustace.io/"
repository = "https://github.com/eustace/poetry"
documentation = "https://poetry.eustace.io/docs"

# ^: 호환되는 버전 (예: ^1.2.3은 >=1.2.3, <2.0.0)
# ~: 근사 버전 (예: ~1.2.3은 >=1.2.3, <1.3.0)
[tool.poetry.dependencies]
python = "^3.9"
requests = "^2.25.1"
# 선택적 종속성 (extras)
scipy = { version = "^1.7.3", optional = true }

# 개발용 종속성 그룹
[tool.poetry.group.dev.dependencies]
pytest = "^6.2.2"
black = {version = "^22.3.0", allow-prereleases = true}

# 문서 빌드용 종속성 그룹
[tool.poetry.group.docs.dependencies]
mkdocs = "^1.2.3"

# Extras 정의
[tool.poetry.extras]
science = ["scipy"]

# 프로젝트에서 사용할 스크립트 정의
# poetry run <script_name>으로 실행 가능
[tool.poetry.scripts]
my-script = "my_awesome_project.main:app"

[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"
```

- Extras 설치: `poetry install --extras "science"`

6. Poetry 워크플로우

1. 프로젝트 생성: `poetry new my-project`
2. 가상환경 설정: `poetry config virtualenvs.in-project true` (권장)
3. 의존성 추가: `poetry add pandas`, `poetry add --group dev pytest`
4. 개발: `poetry run python my_project/main.py` 또는 `poetry shell` 후 `python ...`
5. 테스트: `poetry run pytest`
6. 버전 관리: `git add .`, `git commit ...` (`poetry.lock` 파일 포함)
7. 배포:
   - 다른 프로젝트에서 사용: `pyproject.toml`에 `my-project = { path = "../my-project", develop = true }` 추가
   - PyPI 배포: `poetry publish --build`