Pixi 치트시트

1. 프로젝트 초기화 및 구성

- `pixi init`: 현재 디렉토리에 `pixi.toml` 파일을 생성하여 새 프로젝트를 초기화합니다.
  - `--conda-channels <channel1>,<channel2>`: 사용할 conda 채널을 지정합니다. (예: `conda-forge`)
  - `--platform <p1>,<p2>`: 지원할 플랫폼(os-arch)을 지정합니다. (예: `win-64`, `linux-aarch64`)
- `pixi init --env-file <file>`: `environment.yml` 같은 환경 파일로부터 프로젝트를 초기화합니다.

2. 종속성 관리

- `pixi add <package>`: 프로젝트에 종속성을 추가합니다.
  - `pixi add "numpy≥1.20,<2.0"`: 버전 제약 조건을 명시하여 추가합니다.
  - `pixi add --platform <platform> <package>`: 특정 플랫폼을 위한 종속성을 추가합니다.
  - `pixi add --build <package>`: 빌드 타임에만 필요한 종속성을 추가합니다.
  - `pixi add --host <package>`: 호스트 타임에만 필요한 종속성을 추가합니다.
- `pixi remove <package>`: 프로젝트에서 종속성을 제거합니다.
- `pixi list`: 프로젝트의 종속성 목록을 보여줍니다.
  - `--platform <platform>`: 특정 플랫폼의 종속성을 보여줍니다.
- `pixi install`: `pixi.lock` 파일에 명시된 모든 종속성을 설치하여 환경을 구성합니다.
- `pixi update`: `pixi.toml` 파일의 제약 조건에 따라 종속성을 최신 버전으로 업데이트하고 `pixi.lock` 파일을 갱신합니다.

3. 환경 및 작업 실행

- `pixi run <task_name> [args...]`: `pixi.toml`에 정의된 작업을 실행합니다. 추가 인수를 작업에 전달할 수 있습니다.
- `pixi shell`: 프로젝트의 환경이 활성화된 새로운 셸을 시작합니다. 이 셸 안에서는 `python`, `pip` 등의 명령어가 프로젝트 환경에 맞게 설정됩니다.
- `pixi task list`: 사용 가능한 모든 작업을 나열합니다.
- `pixi task add <name> <command>`: `pixi.toml`에 새 작업을 추가합니다.
  - `pixi task add test "pytest -v"`
- `pixi task remove <name>`: 작업을 제거합니다.

4. 다중 환경 (Features)

`pixi`는 "features"라는 개념을 통해 여러 환경을 관리할 수 있습니다. 예를 들어, 기본 종속성 외에 테스트나 문서 빌드에만 필요한 추가적인 종속성을 그룹화할 수 있습니다.

- `pixi.toml`에서 feature 정의하기:

```toml
[feature.test.dependencies]
pytest = "*"
pytest-cov = "*"

[feature.docs.dependencies]
mkdocs = "*"
```

- Feature 환경 활성화:
  - `pixi shell --env test`: `test` feature 환경이 활성화된 셸을 시작합니다. 기본 종속성과 `test` 종속성이 모두 설치됩니다.
- Feature 환경에서 작업 실행:
  - `pixi run --env test pytest`
  - 또는 `pixi.toml`의 작업에 직접 환경을 지정할 수 있습니다:

```toml
[tasks]
test = { cmd = "pytest", env = "test" }
```

이 경우 `pixi run test`만 실행하면 됩니다.

5. `pixi.toml` 파일 심층 분석

```toml
# 프로젝트의 기본 정보를 정의합니다.
[project]
name = "my-project"
version = "0.1.0"
description = "A pixi project."
authors = ["Your Name <you@example.com>"]
channels = ["conda-forge"] # 사용할 conda 채널 목록
platforms = ["linux-64", "osx-64", "win-64"] # 지원할 플랫폼

# 실행할 수 있는 작업(스크립트)을 정의합니다.
[tasks]
start = "python main.py"
lint = "ruff check ."
# 작업에 대한 설명 추가 가능
format = { cmd = "ruff format .", description = "Format the code." }
# 특정 환경에서 실행되는 작업 정의
test = { cmd = "pytest", env = "test" }


# 모든 환경에 기본적으로 포함되는 종속성입니다.
[dependencies]
python = "≥3.9"
numpy = "≥1.20"
pandas = "*"

# 빌드 시에만 필요한 종속성입니다.
[build-dependencies]
cmake = "*"

# 호스트(pixi를 실행하는 시스템)에 필요한 종속성입니다.
[host-dependencies]


# 'test'라는 이름의 추가 환경(feature)을 정의합니다.
[feature.test.dependencies]
pytest = "*"
pytest-cov = "*"

# 'docs'라는 이름의 추가 환경(feature)을 정의합니다.
[feature.docs.dependencies]
mkdocs = "*"
mkdocs-material = "*"
```

6. 기타 유용한 명령어

- `pixi info`: 현재 프로젝트와 시스템에 대한 정보를 표시합니다.
- `pixi search <package>`: 사용 가능한 패키지를 검색합니다.
- `pixi self-update`: pixi 자체를 최신 버전으로 업데이트합니다.
- `pixi auth login`: private conda 채널에 대한 인증을 설정합니다.
- `pixi project channels add <channel>`: 프로젝트에 conda 채널을 추가합니다.
- `pixi project platforms add <platform>`: 프로젝트에 지원 플랫폼을 추가합니다.

왜 Pixi를 사용하는가?

Pixi는 `conda`의 강력한 패키지 관리 능력과 `pip`의 유연성을 결합하고, `cargo`나 `npm`과 같은 현대적인 도구의 사용자 경험을 제공합니다.

- 재현성: `pixi.lock` 파일을 통해 모든 플랫폼에서 동일한 환경을 정확하게 재현할 수 있습니다.
- 속도: 병렬 다운로드와 효율적인 해결(resolution)을 통해 환경을 빠르게 구성합니다.
- 통합된 도구: 종속성 관리, 작업 실행, 환경 관리를 하나의 도구로 해결합니다.
- 크로스 플랫폼: Windows, macOS, Linux에서 동일한 방식으로 작동합니다.