Pixi 치트시트

프로젝트 시작 및 구성

- `pixi init`: 현재 디렉토리에 `pixi.toml` 파일을 생성하여 새로운 프로젝트를 시작합니다.
  - `--conda-channels <채널1>,<채널2>`: 사용할 Conda 채널을 지정합니다. (예: `conda-forge`)
  - `--platform <p1>,<p2>`: 지원할 플랫폼(OS 및 아키텍처)을 지정합니다. (예: `win-64`, `linux-aarch64`)
- `pixi init --env-file <파일>`: `environment.yml`과 같은 기존 환경 설정 파일을 기반으로 프로젝트를 초기화합니다.

의존성(Dependencies) 관리

- `pixi add <패키지>`: 프로젝트에 새로운 패키지를 추가합니다.
  - `pixi add "numpy⩾1.20,<2.0"`: 버전을 명시하여 특정 범위 내의 패키지를 추가합니다.
  - `pixi add --platform <플랫폼> <패키지>`: 특정 플랫폼 전용 의존성을 추가합니다.
  - `pixi add --build <패키지>`: 빌드 시점에만 필요한 패키지를 추가합니다.
  - `pixi add --host <패키지>`: 실행 환경(호스트)에 필요한 패키지를 추가합니다.
- `pixi remove <패키지>`: 프로젝트에서 의존성 패키지를 제거합니다.
- `pixi list`: 현재 프로젝트에 설치된 의존성 목록을 확인합니다.
  - `--platform <플랫폼>`: 특정 플랫폼용 의존성 구성을 보여줍니다.
- `pixi install`: `pixi.lock` 파일을 기반으로 모든 의존성을 설치하여 환경을 구축합니다.
- `pixi update`: `pixi.toml`의 제약 조건 내에서 패키지를 최신 버전으로 업데이트하고 락 파일을 갱신합니다.

작업 실행 및 환경 제어

- `pixi run <작업명> [인자...]`: `pixi.toml`에 정의된 작업을 실행합니다. 실행 시 추가 인자를 전달할 수 있습니다.
- `pixi shell`: 프로젝트 환경이 활성화된 새로운 셸 세션을 시작합니다. 이 셸에서는 `python`, `pip` 등이 프로젝트 설정에 맞춰 동작합니다.
- `pixi task list`: 사용 가능한 모든 작업(Task) 목록을 나열합니다.
- `pixi task add <이름> <명령어>`: `pixi.toml`에 새로운 실행 작업을 추가합니다.
  - 예: `pixi task add test "pytest -v"`
- `pixi task remove <이름>`: 등록된 작업을 삭제합니다.

다중 환경 관리 (Features)

`pixi`는 "features" 기능을 통해 용도별로 여러 환경을 관리할 수 있습니다. 예를 들어, 기본 개발 환경과 별개로 테스트나 문서 빌드용 의존성을 그룹화할 수 있습니다.

- `pixi.toml`에서 feature 정의 예시:

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

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

- 특정 Feature 환경 활성화:
  - `pixi shell --env test`: `test` 기능이 포함된 환경으로 셸을 시작합니다. 기본 의존성과 테스트용 의존성이 함께 활성화됩니다.
- Feature 환경에서 작업 실행:
  - `pixi run --env test pytest`
  - 또는 `pixi.toml` 설정에서 작업별로 환경을 미리 지정할 수 있습니다:

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

    이후에는 간단히 `pixi run test`만 실행하면 됩니다.

`pixi.toml` 설정 파일 상세 분석

```
# 프로젝트 기본 정보 정의
[project]
name = "my-project"
version = "0.1.0"
description = "Pixi 프로젝트 예시"
authors = ["홍길동 <gildong@example.com>"]
channels = ["conda-forge"] # 사용할 Conda 채널 목록
platforms = ["linux-64", "osx-64", "win-64"] # 지원 플랫폼

# 실행 가능한 작업(스크립트) 정의
[tasks]
start = "python main.py"
lint = "ruff check ."
# 작업에 대한 상세 설명 추가 가능
format = { cmd = "ruff format .", description = "코드 포맷팅 실행" }
# 특정 feature 환경에서 구동될 작업 정의
test = { cmd = "pytest", env = "test" }

# 모든 환경에 공통으로 포함되는 기본 의존성
[dependencies]
python = "⩾3.9"
numpy = "⩾1.20"
pandas = "*"

# 빌드 과정에서만 필요한 의존성
[build-dependencies]
cmake = "*"

# 호스트(실행 시스템) 환경에 필요한 의존성
[host-dependencies]

# 'test' 전용 추가 환경(feature) 정의
[feature.test.dependencies]
pytest = "*"
pytest-cov = "*"

# 'docs' 전용 추가 환경(feature) 정의
[feature.docs.dependencies]
mkdocs = "*"
mkdocs-material = "*"
```

기타 유용한 명령어

- `pixi info`: 현재 프로젝트와 시스템 환경 정보를 상세히 표시합니다.
- `pixi search <패키지>`: 사용 가능한 패키지를 Conda 채널에서 검색합니다.
- `pixi self-update`: Pixi 도구 자체를 최신 버전으로 업데이트합니다.
- `pixi auth login`: 비공개(Private) Conda 채널 접근을 위한 인증을 수행합니다.
- `pixi project channels add <채널>`: 프로젝트 설정에 새로운 Conda 채널을 추가합니다.
- `pixi project platforms add <플랫폼>`: 프로젝트가 지원할 플랫폼을 추가합니다.

왜 Pixi인가?

Pixi는 `conda`의 강력한 패키지 관리 기능과 `pip`의 유연함을 결합한 도구로, `cargo`나 `npm`과 같은 현대적인 사용자 경험을 제공합니다.

- 완벽한 재현성: `pixi.lock` 파일을 통해 모든 플랫폼에서 동일한 환경을 보장합니다.
- 압도적인 속도: 병렬 다운로드와 효율적인 의존성 해결 알고리즘으로 환경을 빠르게 구축합니다.
- 통합 관리: 의존성 관리, 작업 실행, 가상 환경 제어를 하나의 도구로 처리합니다.
- 진정한 크로스 플랫폼: Windows, macOS, Linux 어디서나 일관되게 작동합니다.