Claude Code Best Practices

Anthropic Engineering 문서 요약 (2025)

📋 개요

Claude Code는 **저수준(low-level)이고 무주관적인(unopinionated)** 설계 철학을 가진 명령줄 기반 에이전틱 코딩 도구입니다. 특정 워크플로우를 강제하지 않고, 개발자에게 원시 모델 접근에 가까운 유연하고 커스터마이징 가능한 도구를 제공합니다.

핵심 특징:

- Shell 네이티브 통합
- 프로그래머가 사용하는 동일한 도구 접근
- 컨텍스트 풍부한 상호작용
- 안전하고 스크립트 가능한 파워 툴

🎯 1. CLAUDE.md: AI 비서의 플레이북

목적

프로젝트별 커스텀 문서로, Claude가 자동으로 읽어 컨텍스트를 이해합니다.

배치 위치

- 프로젝트 루트 디렉토리
- 하위 디렉토리
- 상위 디렉토리
- 글로벌 설정 (`~/.claude/CLAUDE.md`)

포함할 내용

1) 명령어 치트시트

```
# Build Commands
- npm run build: 전체 프로젝트 컴파일
- npm run typecheck: TypeScript 검증
- npm test: 테스트 실행
```

2) 코딩 컨벤션

```
# Code Standards
- ES 모듈 사용 (CommonJS 대신)
- 가능한 경우 import 구조 분해
- 함수형 컴포넌트 선호
```

3) 테스팅 프로토콜

```
# Quality Assurance
- 빠른 반복을 위해 단일 테스트 파일 실행
- null 입력으로 엣지 케이스 검증
- 커밋 전 모든 테스트 통과 확인
```

4) 프로젝트 구조 설명

```
# Project Structure
/src/components: React 컴포넌트
/src/utils: 헬퍼 함수
/tests: 테스트 파일
```

프로 팁

- 세션 중 `#` 명령으로 동적 업데이트
- 프롬프트 엔지니어링처럼 반복적으로 튜닝
- 팀 전체가 동일한 CLAUDE.md 사용하도록 Git 커밋

🔌 2. MCP (Model Context Protocol) 통합

설정 방법

프로젝트별 설정

`.mcp.json` 파일을 프로젝트 루트에 생성:

```
{
  "mcpServers": {
    "puppeteer": {
      "command": "npx",
      "args": ["@modelcontextprotocol/server-puppeteer"]
    },
    "sentry": {
      "command": "npx",
      "args": ["@sentry/mcp-server"]
    }
  }
}
```

글로벌 설정

`~/.config/claude/mcp.json`에 설정하면 모든 프로젝트에서 사용 가능

디버깅

```
claude --mcp-debug
```

활용 사례

- **Puppeteer**: 시각적 테스팅, UI 자동화
- **Sentry**: 텔레메트리 분석, 에러 추적
- **Custom MCP**: REST API를 통한 기능 확장

⚡ 3. Custom Slash Commands

생성 방법

`.claude/commands/` 디렉토리에 Markdown 파일 생성

예시: GitHub Issue 자동 처리

```
<!-- fix-issue.md -->
Please analyze and fix the GitHub issue: $ARGUMENTS

Steps:
1. Fetch issue details using gh cli
2. Identify relevant files
3. Implement fix
4. Create tests
5. Commit and create PR
```

사용 방법

```
/fix-issue 123
```

$ARGUMENTS 키워드

- 명령 실행 시 매개변수 전달
- 디버깅 루프, 로그 분석 등 반복 워크플로우에 유용

🏗️ 4. 구조화된 워크플로우: Explore → Plan → Implement

4.1 Explore (탐색)

```
"이 프로젝트의 인증 로직이 어떻게 구현되어 있는지 읽어줘.
아직 코드는 작성하지 마."
```

목적: 정보 수집, 파일 분석, 컨텍스트 이해

4.2 Plan (계획)

```
"위 정보를 바탕으로 OAuth 2.0을 추가하는 상세한 단계별 계획을 작성해줘."
```

키워드 활용:

- `think hard` : 내부 추론 시간 증가
- `ultrathink` : 더 깊은 추론

4.3 Implement (구현)

```
"좋아, 이제 계획에 따라 구현해줘.
각 단계마다 테스트를 작성하고 커밋해줘."
```

4.4 Verify (검증)

```
"독립적인 에이전트를 사용해서 이 구현이
테스트 케이스에만 맞춘 과적합이 아닌지 확인해줘."
```

🔐 5. 도구 접근 권한 관리

허용 목록 설정

세션별 설정

```
claude --allowedTools "Bash(npm run*),MCP(puppeteer_*)"
```

영구 설정

```
# ~/.claude/config.json
{
  "allowedTools": [
    "Bash(git commit:*)",
    "Bash(npm install)",
    "Edit(*)"
  ]
}
```

CLI 명령으로 추가

```
claude allow "git commit"
claude allow edit
```

보안 고려사항

- 안전하고 자주 사용하는 작업만 사전 승인

- 파일 수정이나 특정 명령 실행 전 권한 요청
- 민감한 작업은 수동 승인 유지

🧪 6. Test-Driven Development (TDD)

워크플로우

1. **실패하는 테스트 먼저 작성**

```
"tests/test_features.py에 새 엔드포인트에 대한
실패하는 테스트를 작성해줘. 아직 구현은 하지 마."
```

2. **테스트 커밋**

```
git add tests/ && git commit -m "Add failing tests"
```

3. **구현 코드 작성**

```
"이제 테스트를 통과하도록 구현해줘."
```

4. **반복 개선**

```
"엣지 케이스를 추가하고 리팩토링해줘."
```

Anthropic 팀의 경험

Security Engineering 팀:

- **이전:** 디자인 문서 → 불안정한 코드 → 리팩토링 → 테스트 포기
- **현재:** 의사코드 요청 → TDD 가이드 → 주기적 체크인
- **결과:** 더 신뢰할 수 있고 테스트 가능한 코드

📚 7. Context 관리 및 최적화

7.1 Compact 기능

```
/compact
```

- 컨텍스트 제한 도달 시 자동 요약
- 이전 메시지를 압축하여 컨텍스트 유지
- 장기 실행 에이전트에 필수

7.2 Reference Quotes 기법

```
<instructions>
문서를 바탕으로 질문에 답변하세요.
최종 답변 전에 <scratchpad>를 사용해
관련 있는 정확한 인용구를 찾으세요.
</instructions>
```

7.3 In-Context Examples

- 올바르게 답변된 질문 예시 제공
- 긴 문서에서 정보 회상 능력 향상

🔄 8. Git 워크플로우 통합

GitHub CLI (gh) 활용

```
# 설치 후 자동으로 사용 가능
gh issue list
gh pr create
gh pr review
```

Claude에게 요청 예시

```
"방금 완료한 작업에 대해 'feat: Add user login functionality'
제목으로 PR을 생성해줘."

"이 프로젝트의 열린 버그 이슈 목록을 보여주고,
가장 최근 것을 기반으로 새 브랜치를 만들어줘."
```

Clean Git State 유지

- 프로토타이핑 시작 전 항상 clean git state 유지
- 자율 루프 실행 시 정기적으로 커밋
- 80% 완성된 솔루션 검토 후 최종 개선

🚀 9. 자동화 및 Auto-Accept Mode

Auto-Accept 활성화

```
# Shift + Tab
```

활용 시나리오

- **빠른 프로토타이핑**: 익숙하지 않은 추상적 문제에 적용
- **자율 루프**: 코드 작성 → 테스트 실행 → 반복
- **80/20 접근**: 80% 자동 완성 후 수동으로 마무리

주의사항

- Clean git state에서 시작
- 중요한 프로덕션 코드에는 신중히 사용
- 정기적으로 결과 검토

🔍 10. 반복적 검증 및 개선

독립 에이전트 검증

```
"별도의 에이전트를 사용해서 이 코드가:
1. 테스트 케이스에만 과적합되지 않았는지
2. 일반적인 입력에 대해 올바르게 작동하는지
3. 엣지 케이스를 처리하는지 확인해줘."
```

규칙 기반 피드백

Code Linting이 최고의 피드백:

- TypeScript 〉 JavaScript (추가 피드백 레이어)
- 명확한 규칙 정의 → 실패한 규칙 설명
- 예: 이메일 생성 시 유효성 검사, 기존 연락 여부 확인

시각적 목업 활용

- 스크린샷 도구나 MCP 통합
- UI 코드를 디자인에 맞추도록 지시
- "이 목업과 일치하도록 컴포넌트를 구현해줘"

🎓 Anthropic 팀 사용 사례

Data Infrastructure Team

- **문제:** 대규모 코드베이스 탐색 어려움
- **해결:** CLAUDE.md 파일로 데이터 파이프라인 의존성 설명
- **결과:** 전통적인 데이터 카탈로그 도구 대체

Product Engineering Team

- **사용:** 버그 수정과 기능 개발의 "첫 번째 정거장"
- **접근:** 어떤 파일을 검토할지 Claude에게 물어봄
- **효과:** 수동 컨텍스트 수집 시간 제거

Security Engineering Team

- **활용:** 스택 트레이스와 문서로 프로덕션 이슈 진단
- **워크플로우:** 의사코드 → TDD → 주기적 체크인
- **개선:** 더 신뢰할 수 있고 테스트 가능한 코드

Inference Team

- **문제:** ML 백그라운드 없는 팀원들의 학습 곡선
- **해결:** Claude로 모델 특정 함수 설명
- **절약:** 리서치 시간 80% 감소 (1시간 → 10-20분)

📊 실전 워크플로우 예시

1. 새 코드베이스 온보딩

```
"이 프로젝트의 전체 구조를 설명해줘.
주요 컴포넌트와 그들 간의 관계를 알려줘."

"사용자 인증이 어떻게 구현되어 있는지 설명해줘."
```

2. 버그 수정

```
"이 버그를 고칠 수 있어? 내가 보는 동작은 이래: [설명]
관련 파일들을 찾아서 수정 방안을 제시해줘."
```

3. Kubernetes 장애 대응

```
"이 대시보드 스크린샷을 보고 문제를 진단해줘."
[Claude가 메뉴별로 가이드]
"Pod IP 주소 고갈이네. 새 IP 풀을 만들고 클러스터에 추가하는
정확한 명령어를 알려줘."
```

결과: 시스템 장애 중 20분 절약

4. 로그 분석

```
<!-- analyze-logs.md -->
다음 로그를 분석해서:
1. 에러 패턴 식별
2. 빈도 계산
3. 가능한 원인 제시
4. 해결 방안 추천

$ARGUMENTS
```

💡 핵심 원칙 요약

1. **컨텍스트가 왕(Context is King)**

    - CLAUDE.md를 충실히 작성하고 유지
    - 팀 전체가 동일한 컨텍스트 공유
2. **계획 우선 (Plan First)**
    - Explore → Plan → Implement 순서 준수
    - 구현 전에 항상 계획 검토
3. **반복적 개선 (Iterate and Verify)**
    - 작은 단계로 나누어 진행
    - 각 단계마다 검증
    - TDD 접근법 활용
4. **도구 통합 (Tool Integration)**
    - 기존 개발 도구와 자연스럽게 통합
    - MCP로 기능 확장
    - GitHub CLI 등 CLI 도구 적극 활용
5. **자동화와 안전 (Automate Safely)**
    - 반복 작업은 슬래시 커맨드로 자동화
    - 안전한 작업은 사전 승인
    - Clean git state 유지

🎯 성공을 위한 체크리스트

프로젝트 시작 시

- ☐ CLAUDE.md 파일 작성
- ☐ .mcp.json 설정 (필요시)
- ☐ .claude/commands/ 디렉토리 생성
- ☐ 허용 목록 설정
- ☐ GitHub CLI 설치 및 인증

작업 시작 전

- ☐ Git state가 clean한지 확인
- ☐ 관련 CLAUDE.md 섹션 확인

- [ ] 필요한 도구들이 접근 가능한지 확인

개발 중

- [ ] Explore → Plan → Implement 순서 따르기
- [ ] 주기적으로 커밋
- [ ] 테스트 작성 및 실행
- [ ] 독립 검증 수행

완료 후

- [ ] 코드 리뷰
- [ ] 문서 업데이트
- [ ] CLAUDE.md에 새로운 패턴 추가
- [ ] 슬래시 커맨드 추가 (반복 작업의 경우)

📖 추가 리소스

- **공식 문서**: claude.ai/code
- **Anthropic Academy**: 인증서 발급 가능한 코스
- **GitHub 샘플**: 커뮤니티 공유 CLAUDE.md 및 워크플로우

🚀 결론

Claude Code는 단순한 코드 생성 도구가 아닌 **적극적인 개발 파트너**입니다. 명확한 컨텍스트 제공(CLAUDE.md), 반복 작업 자동화(슬래시 커맨드), 검증된 워크플로우(Explore→Plan→Implement)를 체계적으로 적용하면, Claude Code를 혁신과 생산성의 강력한 엔진으로 변모시킬 수 있습니다.

핵심은 Claude에게 프로그래머가 사용하는 동일한 도구를 제공하는 것입니다.