Claude Code 완전 가이드: 해커톤 우승자의 70가지 파워 팁

저자: Manus AI

기반 자료: ykdojo (Anthropic 해커톤 우승자) & Ado Kukic (Anthropic DevRel)

서문: 개발의 새로운 시대, AI 에이전트와 함께하는 여정

2025년, 소프트웨어 개발의 패러다임은 근본적인 전환점을 맞이했습니다. 이제 인공지능(AI)은 단순히 코드 조각을 자동 완성해주는 보조 도구를 넘어, 개발자의 파트너로서 전체 개발 라이프사이클에 깊숙이 관여하는 **에이전트(Agent)**로 진화하고 있습니다. 그 혁명의 중심에 바로 **Claude Code**가 있습니다.

이 책은 단순한 기능 나열식 가이드를 넘어, **Anthropic 해커톤 우승자 ykdojo**의 생생한 경험과 검증된 워크플로우, 그리고 **Anthropic DevRel Ado Kukic**의 "Advent of Claude" 챌린지에서 공개된 **70가지가 넘는 실전 팁과 전략을 완전히 집대성**한 결정판 가이드입니다.

ykdojo는 Claude Code를 사용하여 생산성을 극적으로 향상시켰으며, 시스템 프롬프트를 19k 토큰에서 10k 이하로 절반 가까이 줄이고, 자신만의 음성 코딩 시스템을 구축하며, 컨테이너 기반의 멀티 에이전트 오케스트레이션까지 구현한 진정한 파워 유저입니다. 그는 "AI 사용법을 배우는 가장 좋은 방법은 AI를 사용하는 것"이라는 철학 아래, 10억 토큰 이상을 소비하며 축적한 노하우를 GitHub 저장소에 공개했습니다.

Ado Kukic은 Anthropic의 Developer Relations 팀에서 일하며, 2025년 12월 한 달 동안 매일 하나씩 Claude Code의 강력한 기능을 소개하는 "Advent of Claude" 챌린지를 진행했습니다. 그의 팁들은 초보자부터 고급 사용자까지 모두가 활용할 수 있는 실용적인 지식으로 가득합니다.

이 책의 특징:

- **70가지 이상의 완전한 팁**: ykdojo의 43가지 팁과 Ado의 31가지 팁을 하나도 빠짐없이 상세하게 다룹니다.
- **구체적인 예시와 Cookbook**: 각 팁마다 실제로 사용할 수 있는 명령어, 스크립트, 설정 파일을 제공합니다.
- **우승자의 워크플로우**: 해커톤 우승자가 실제로 사용하는 작업 방식을 단계별로 따라갈 수 있습니다.
- **시각 자료**: 다이어그램과 스크린샷으로 복잡한 개념을 명확하게 이해할 수 있습니다.

이 책을 통해 여러분은 생산성을 50%에서 최대 200%까지 끌어올린 개발자들의 비밀을 엿보고, 그들의 사고방식과 기술을 자신의 것으로 만들게 될 것입니다.

목차

Part 2: 기초부터 탄탄하게 - 환경 설정과 필수 명령어

- 2.1. 커스텀 상태 라인으로 모든 것을 한눈에 (ykdojo #0)
- 2.2. 필수 슬래시 명령어 마스터 (ykdojo #1, Ado #4, #16, #17)
- 2.3. CLAUDE.md: AI를 위한 프로젝트 설명서 (Ado #1, #2, ykdojo #30)
- 2.4. 터미널 별칭으로 빠른 접근 (ykdojo #7)
- 2.5. 세션 관리: 대화를 잃지 않는 법 (Ado #9, #10, #11, #12)

Part 3: 생산성을 극대화하는 핵심 기술

- 3.1. 음성으로 코딩하기 (ykdojo #2)
- 3.2. 터미널 출력 추출의 기술 (ykdojo #6)
- 3.3. Cmd+A/Ctrl+A: 전체 선택의 힘 (ykdojo #10)
- 3.4. 마크다운과 Notion 활용 (ykdojo #19, #20)
- 3.5. 키보드 단축키 완전 정복 (Ado #5, #6, #7, #8, ykdojo #38)
- 3.6. Vim 모드로 프롬프트 편집 (Ado #13)
- 3.7. 입력 박스 탐색 및 편집 (ykdojo #38)

Part 4: 컨텍스트 관리의 예술

- 4.1. 선제적 컨텍스트 압축 (ykdojo #8)
- 4.2. 터미널 탭으로 멀티태스킹 (ykdojo #14)
- 4.3. 대화 복제 및 반복제 (ykdojo #23)
- 4.4. /context로 X-Ray 비전 (Ado #15)
- 4.5. realpath로 절대 경로 얻기 (ykdojo #24)

Part 5: Git과 GitHub 워크플로우 완전 정복

- 5.1. Git과 GitHub CLI 프로 활용 (ykdojo #4)
- 5.2. Git worktrees로 병렬 브랜치 작업 (ykdojo #16)
- 5.3. 대화형 PR 리뷰 (ykdojo #26)
- 5.4. 승인된 명령어 감사 (ykdojo #33)

Part 6: 고급 기능 - MCP, Hooks, Agents

- 6.1. MCP: 외부 세계와의 연결 (Ado #22, #23, #24, #25)
- 6.2. Hooks: 규칙의 강제 (Ado #26)
- 6.3. Skills: 재사용 가능한 지식 (Ado #27)
- 6.4. Agents: 전문화된 서브에이전트 (Ado #28)
- 6.5. Plugins: 기능 번들 (Ado #29)
- 6.6. CLAUDE.md vs Skills vs Slash Commands vs Plugins (ykdojo #25)

Part 7: 시스템 최적화와 자동화

- 7.1. 시스템 프롬프트 슬림화 (ykdojo #15)

- 7.2. 장시간 작업을 위한 수동 지수 백오프 (ykdojo #17)
- 7.3. 백그라운드에서 bash 명령 및 에이전트 실행 (ykdojo #36)
- 7.4. 자동화의 자동화 (ykdojo #41)
- 7.5. Headless 모드로 CI/CD 통합 (Ado #30)

Part 8: 컨테이너와 샌드박스

- 8.1. 컨테이너로 위험한 작업 격리 (ykdojo #21)
- 8.2. Sandbox 모드와 권한 관리 (Ado #20)
- 8.3. YOLO 모드: 위험을 감수할 때 (Ado #20)

Part 9: 브라우저 통합과 웹 자동화

- 9.1. 네이티브 브라우저 통합 (Ado #21)
- 9.2. Playwright MCP (Ado #22)
- 9.3. Gemini CLI를 대체 수단으로 (ykdojo #11)

Part 10: 실전 활용 사례

- 10.1. 작성-테스트 사이클 완성 (ykdojo #9)
- 10.2. 자신만의 워크플로우에 투자 (ykdojo #12)
- 10.3. 대화 기록 검색 (ykdojo #13)
- 10.4. 글쓰기 도우미로 활용 (ykdojo #18)
- 10.5. 연구 도구로 활용 (ykdojo #27)
- 10.6. 출력 검증 방법 마스터 (ykdojo #28)
- 10.7. DevOps 엔지니어로 활용 (ykdojo #29)
- 10.8. 범용 인터페이스로 활용 (ykdojo #31)
- 10.9. 테스트 많이 작성 (TDD) (ykdojo #34)
- 10.10. 복잡한 코드 단순화 (ykdojo #40)

Part 11: 고급 패턴과 철학

- 11.1. 계획과 빠른 프로토타이핑의 균형 (ykdojo #39)
- 11.2. 개인화된 소프트웨어 시대 (ykdojo #37)
- 11.3. 사용이 최고의 학습 (ykdojo #22)
- 11.4. 지식 공유 및 기여 (ykdojo #42)
- 11.5. 계속 학습하기 (ykdojo #43)

Part 12: 고급 기능과 SDK

- 12.1. Extended Thinking (Ado #19)
- 12.2. LSP 통합 (Ado #31)
- 12.3. Claude Agent SDK (Ado #31)
- 12.4. 팀 설정과 공유 워크플로우 (Ado #30)

Part 13: 학습 로드맵과 다음 단계

- 13.1. 초급자를 위한 로드맵 (1-3개월)
- 13.2. 중급자를 위한 로드맵 (3-12개월)
- 13.3. 고급자를 위한 로드맵 (1년 이상)
- 13.4. 필독 참고 자료

맺음말: AI는 부조종사, 주인공은 당신입니다

Part 1: 에이전틱 개발자의 사고방식

Claude Code를 효과적으로 사용하기 위해 가장 먼저 필요한 것은 기술적인 스킬이 아니라 **사고방식의 전환**입니다. 우리는 더 이상 혼자 코드를 작성하는 개발자가 아닙니다. 우리는 AI라는 강력한 파트너와 함께 일하는 **에이전틱 개발자(Agentic Developer)**입니다.

1.1. 큰 문제 정복의 첫걸음: 분해하고 정복하라 (ykdojo Tip #3)

해커톤 우승자 ykdojo는 성공적인 AI 협업의 핵심이 **"큰 문제를 잘게 쪼개는 능력"**에 있다고 반복해서 강조합니다. 이는 전통적인 소프트웨어 엔지니어링에서도 중요한 원칙이지만, AI와 협업할 때 더욱 결정적인 역할을 합니다. AI는 명확하고 구체적인 지시를 받았을 때 가장 뛰어난 성능을 발휘하기 때문입니다.

잘못된 접근 방식:

> *"로그인 페이지를 만들어줘."*

이런 막연한 요청은 AI에게 너무 많은 결정권을 주어, 여러분이 원하지 않는 방향으로 갈 가능성이 높습니다.

에이전틱 개발자의 접근 방식:

1. "사용자 모델을 위한 데이터베이스 스키마를 설계해줘. 필드는 username, email, password_hash, created_at, updated_at이 필요해."
2. "이 스키마를 바탕으로 Drizzle ORM 마이그레이션 파일을 생성해줘."
3. "React와 TailwindCSS를 사용하여 로그인 폼 UI 컴포넌트를 만들어줘. 이메일과 비밀번호 입력 필드, 그리고 '로그인' 버튼이 필요해."
4. "사용자명과 비밀번호를 받아 `/api/auth/login` 엔드포인트로 POST 요청을 보내는 로직을 작성해줘."
5. "로그인 성공 시 토큰을 localStorage에 저장하고 대시보드로 리다이렉트하는 로직을 추가해줘."
6. "로그인 실패 시나리오(잘못된 비밀번호, 존재하지 않는 사용자)에 대한 테스트 코드를 작성해줘."

ykdojo의 실제 사례: 음성 전사 시스템 구축

ykdojo는 자신만의 음성 전사 시스템을 만들 때 이 원칙을 적용했습니다. 처음부터 "음성 전사 앱을 만들어줘"라고 하지 않고, 다음과 같이 작업을 분해했습니다.

1. 첫 번째 실행 파일: 모델을 다운로드하는 기능만 구현
2. 두 번째 실행 파일: 음성을 녹음하는 기능만 구현
3. 세 번째 실행 파일: 미리 녹음된 오디오를 전사하는 기능만 구현
4. 네 번째 단계: 키보드 단축키를 감지하는 기능 추가
5. 다섯 번째 단계: 전사된 텍스트를 사용자의 커서 위치에 삽입하는 기능 추가
6. 마지막 단계: 모든 기능을 하나의 UI로 통합

이렇게 하나씩 완성하고 검증한 후 다음 단계로 넘어가는 방식으로, 복잡한 시스템을 안정적으로 구축할 수 있었습니다.

시각화: A에서 B로 가는 두 가지 방법

- **직접 경로 (실패 확률 높음)**: A → B
- **단계적 경로 (성공 확률 높음)**: A → A1 → A2 → A3 → B

각 단계에서 검증하고, 문제가 있으면 그 단계만 수정하면 되므로 디버깅이 훨씬 쉽습니다.

1.2. 계획 모드 vs. 욜로 모드: 언제 생각하고 언제 실행할 것인가 (Ado Tip #18, #19)

Claude Code는 두 가지 작업 방식을 제공합니다. 상황에 따라 적절한 모드를 선택하는 것은 생산성을 극대화하는 중요한 기술입니다.

계획 모드 (Plan Mode)

`Shift+Tab`을 두 번 누르면 계획 모드로 진입합니다. 이 모드에서 Claude는:

- 코드베이스를 읽고 분석합니다
- 아키텍처를 파악합니다
- 의존성을 탐색합니다
- 구현 계획을 단계별로 작성합니다

하지만 여러분이 승인하기 전까지는 아무것도 편집하지 않습니다. 이는 "두 번 생각하고, 한 번 실행하라(Think twice, execute once)"는 철학을 구현한 것입니다.

Ado의 조언:

> *"저는 90%의 시간 동안 계획 모드를 기본으로 사용합니다. 최신 버전에서는 계획을 거부할 때 피드백을 제공할 수 있어, 반복 작업이 더 빠릅니다."*

계획 모드를 사용해야 할 때:

- 처음 해보는 복잡한 작업
- 여러 파일에 걸친 대규모 리팩토링
- 아키텍처 변경이 필요한 작업
- 실수 시 복구 비용이 큰 작업 (예: 데이터베이스 마이그레이션)

욜로 모드 (YOLO Mode)

`claude --dangerously-skip-permissions` 플래그로 실행하면, Claude는 모든 작업에 대해 자동으로 승인합니다. 이름에 "dangerously"가 들어간 이유가 있습니다.

욜로 모드를 사용해야 할 때:

- 간단하고 명확한 작업 (예: "이 함수에 주석 추가해줘")
- 실험적인 프로토타입 작업
- 컨테이너 안에서 격리된 환경에서 작업할 때
- 반복적이고 시간이 많이 걸리는 작업 (예: 대량의 파일 처리)

⚠ **경고:** 욜로 모드는 호스트 시스템에서 직접 사용하지 마세요. ykdojo는 이를 **"보호되지 않은 성관계"**에 비유하며, 반드시 컨테이너 안에서 사용할 것을 권장합니다.

1.3. 컨텍스트: AI의 기억력을 지배하는 기술 (ykdojo Tip #5, #8, Ado Tip #15)

AI와의 협업에서 가장 중요한 자원은 **컨텍스트(Context)**입니다. Claude Code는 최대 200,000 토큰에 달하는 방대한 컨텍스트 윈도우를 가지고 있지만, 이는 무한하지 않습니다.

ykdojo의 비유: "AI 컨텍스트는 우유와 같다"

> *"신선하고, 압축된 상태를 유지하는 것이 핵심입니다. 오래된 우유는 상하듯이, 오래된 컨텍스트는 AI의 성능을 저하시킵니다."*

컨텍스트는 AI의 '작업 기억(Working Memory)'과 같습니다. 이 공간이 시스템 프롬프트, 코드 파일, 대화 기록 등으로 가득 차면, AI는 초기의 중요한 지시를 잊어버리는 **컨텍스트 드리프트(Context Drift)** 현상을 겪게 됩니다. 한 연구에 따르면, 여러 주제를 한 대화에서 섞으면 성능이 39%까지 저하될 수 있습니다.

컨텍스트 관리 전략 1: 단일 목적 대화 (ykdojo #14)

하나의 대화(터미널 세션)에서는 하나의 명확한 목표에만 집중하세요. 로그인 기능을 구현하다가 갑자기 "아, 그리고 대시보드 UI도 개선해줘"라고 하면 컨텍스트가 오염됩니다. 대신 새로운 터미널 탭을 열어 별도의 대화를 시작하세요.

컨텍스트 관리 전략 2: 선제적 압축 (ykdojo #8)

대화가 길어지면, 다음 에이전트를 위해 **HANDOFF.md** 파일을 작성하도록 지시하고 `/clear`로 새롭게 시작하세요.

ykdojo의 실제 프롬프트:

> *"Put the rest of the plan in the system-prompt-extraction folder as HANDOFF.md. Explain what you have tried, what worked, what didn't work, so that the next agent with fresh context is able to just load that file and nothing else to get started on this task and finish it up."*

Claude는 다음과 같은 내용을 포함한 HANDOFF.md를 생성합니다:

```
# System Prompt Slimming - Handoff Document

## Goal
Reduce Claude Code's system prompt by ~45% (currently at 11%, need ~34% more).

## Current Progress
### What's Been Done
- Removed verbose examples from tool descriptions
- Shortened permission explanations
- Consolidated redundant instructions

### What Worked
- Regex-based pattern matching for finding repetitive text
- Testing each change individually before committing

### What Didn't Work
- Removing safety warnings (caused unexpected behavior)
- Over-aggressive minification (broke JSON parsing)

## Next Steps
1. Target the MCP server descriptions (estimated 2k tokens)
2. Simplify the hooks documentation
3. Test thoroughly after each change
```

컨텍스트 관리 전략 3: /context로 X-Ray 비전 (Ado #15)

`/context` 명령어를 사용하면 무엇이 컨텍스트를 차지하는지 정확히 볼 수 있습니다:

- 시스템 프롬프트 크기

- MCP 서버 프롬프트
- 메모리 파일 (CLAUDE.md)
- 로드된 스킬과 에이전트
- 대화 기록

이 정보를 바탕으로 불필요한 MCP를 비활성화하거나, 사용하지 않는 스킬을 언로드할 수 있습니다.

컨텍스트 관리 전략 4: 자동 압축 비활성화 (ykdojo #8)

Claude Code는 컨텍스트가 가득 차면 자동으로 압축합니다. 하지만 ykdojo는 `/config`를 통해 자동 압축을 끄고 수동으로 관리하는 것을 선호합니다. 이유는:

1. 자동 압축은 45k 토큰을 예약하므로, 실제 사용 가능한 컨텍스트가 줄어듭니다.
2. 압축 시점을 직접 제어하여 중요한 정보가 손실되지 않도록 할 수 있습니다.
3. HANDOFF.md 방식이 더 명확하고 검증 가능합니다.

1.4. 올바른 추상화 수준 선택 (ykdojo Tip #32)

ykdojo는 Claude Code 사용을 **"거대한 빙산을 탐험하는 것"**에 비유합니다. 때로는 멀리서 전체를 조망하고(Vibe Coding), 때로는 깊이 잠수하여 세부 사항을 파악해야 합니다(Deep Dive).

Vibe Coding 레벨 (높은 추상화)

전체 구조와 흐름만 파악하고, 개별 코드 라인은 신경 쓰지 않습니다. 이 레벨은 다음 상황에 적합합니다:

- 일회성 프로젝트나 프로토타입
- 중요하지 않은 코드베이스 부분
- 빠른 실험이 필요할 때

Deep Dive 레벨 (낮은 추상화)

파일 구조, 함수, 개별 코드 라인, 심지어 의존성까지 꼼꼼히 검토합니다. 이 레벨은 다음 상황에 필요합니다:

- 프로덕션 코드
- 보안이 중요한 부분
- 복잡한 버그 디버깅
- 성능 최적화

ykdojo의 조언:

> *"어떤 사람들은 Vibe Coding이 나쁘다고 말하지만, 때로는 완전히 괜찮습니다. 핵심은 상황에 맞는 추상화 수준을 선택하는 것입니다. 이진법적 사고가 아닙니다."*

실전 예시:

새로운 기능을 추가할 때:

1. **높은 추상화**: "사용자 프로필 페이지를 만들어줘" → 전체 구조 파악
2. **중간 추상화**: "프로필 편집 폼의 유효성 검사 로직을 보여줘" → 특정 기능 검토
3. **낮은 추상화**: "이 정규식이 왜 이메일 유효성 검사에 실패하는지 설명해줘" → 세부 디버깅

1.5. 미지의 영역에서 더 용감하게 (ykdojo Tip #35)

ykdojo는 Claude Code를 사용하면서 **"미지의 영역에서 더 용감해졌다"**고 고백합니다. 이는 AI와의 협업이 단순히 생산성 향상을 넘어, 개발자의 자신감과 학습 능력까지 확장시킨다는 것을 보여줍니다.

실제 사례 1: React 전문가가 아니었지만

ykdojo는 Daft에서 일하면서 프론트엔드 코드에 문제를 발견했습니다. React 전문가가 아니었지만, Claude Code와 함께 문제를 반복적으로 탐구하며 결국 해결했습니다.

실제 사례 2: Rust를 처음 다뤘지만

Python과 Rust가 섞인 복잡한 문제를 해결해야 했습니다. Rust를 한 번도 다뤄본 적이 없었지만, Claude Code에게 질문하며 코드베이스를 탐색하고, 여러 해결책을 시도하며, 결국 우아한 솔루션을 찾아냈습니다.

반복적 문제 해결의 과정:

1. **초기 시도**: Claude가 제안한 솔루션을 시도 → 작동하지 않음
2. **속도 조절**: "천천히 가자. 이 줄이 정확히 무슨 의미인지 설명해줘"
3. **방향 전환**: 막다른 골목이면 다른 접근 방식 탐색
4. **깊이 조절**: 필요에 따라 추상화 수준을 높이거나 낮춤
5. **최종 해결**: 우아한 솔루션 발견

ykdojo의 교훈:

> *"미지의 세계에서도, Claude Code와 함께라면 여러분이 생각하는 것보다 훨씬 많은 것을 할 수 있습니다. 핵심은 속도를 제어하고, 올바른 질문을 하며, 반복적으로 접근하는 것입니다."*

Part 2: 기초부터 탄탄하게 - 환경 설정과 필수 명령어

이 장에서는 Claude Code의 세계로 들어서는 첫 단계를 넘어, 여러분과 AI가 최상의 시너지를 낼 수 있도록 작업 환경을 최적화하는 파워 유저들의 초기 설정법을 중심으로 기초를 다집니다.

2.1. 커스텀 상태 라인으로 모든 것을 한눈에 (ykdojo Tip #0)

ykdojo가 가장 먼저 소개하는 팁은 **커스텀 상태 라인(Custom Status Line)**입니다. 이는 터미널 하단에 실시간으로 중요한 정보를 표시하는 기능으로, 파워 유저들이 가장 먼저 설정하는 항목입니다.

기본 상태 라인의 문제점:

기본 상태 라인은 최소한의 정보만 표시합니다. 하지만 실전에서는 다음 정보들이 실시간으로 필요합니다:

- 현재 토큰 사용량 및 남은 한도
- Git 브랜치 및 변경 사항 상태
- 현재 작업 디렉토리
- 활성화된 MCP 서버 수
- 현재 사용 중인 모델

ykdojo의 커스텀 상태 라인 설정:

ykdojo는 자신의 GitHub 저장소에 `status-line.sh` 스크립트를 공개했습니다. 이 스크립트는 다음을 표시합니다:

```
~/project (main*) | Tokens: 45k/200k | Opus 4.5 | MCP: 3 active
```

설치 방법:

```bash
# 1. 저장소 클론
git clone https://github.com/ykdojo/claude-code-tips.git

# 2. 스크립트 심볼릭 링크 생성
ln -s $(pwd)/claude-code-tips/scripts/status-line.sh ~/.claude/scripts/status-line.sh

# 3. Claude Code 재시작
```

커스터마이징:

`~/.claude/settings.json`에서 상태 라인 형식을 직접 정의할 수도 있습니다:

```json
{
  "statusLine": {
    "format": "{{dir}} {{git}} | {{tokens}} | {{model}}",
    "updateInterval": 1000
  }
}
```

Ado의 `/statusline` 명령어 (Ado Tip #14):

Ado는 `/statusline` 명령어를 통해 대화형으로 상태 라인을 커스터마이징하는 방법을 소개합니다. 이 명령어를 실행하면 Claude가 사용 가능한 변수들을 보여주고, 원하는 형식을 대화로 설정할 수 있습니다.

2.2. 필수 슬래시 명령어 마스터 (ykdojo Tip #1, Ado Tip #4, #16, #17)

Claude Code의 진정한 힘은 슬래시 명령어에서 나옵니다. 이 명령어들은 AI와의 대화를 넘어, 시스템을 직접 제어하고 정보를 즉시 얻을 수 있게 해줍니다.

Tier 1: 생존 필수 명령어

명령어	설명	사용 시점
`/usage`	현재 토큰 사용량과 리셋 시간을 시각적으로 표시	매 세션 시작 시, 대화가 길어질 때
`/clear`	대화 내용을 깨끗이 지우고 새로운 컨텍스트로 시작	컨텍스트 오염 시, 새 작업 시작 시
`/stats`	GitHub 스타일 활동 그래프, 즐겨찾는 모델, 연속 사용일 등 분석	주간 회고 시, 사용 패턴 분석 시
`/context`	컨텍스트 윈도우 사용 현황 X-Ray	성능 저하 느낄 때, 최적화 필요 시

Tier 2: 생산성 향상 명령어

명령어	설명	사용 시점
`/chrome`	크롬 브라우저 통합 시작	웹 스크래핑, UI 테스트, 디버깅
`/mcp`	MCP 서버 목록 및 활성화/비활성화	MCP 관리, 컨텍스트 최적화
`/permissions`	승인된 명령어 목록 및 관리	보안 감사, 위험한 명령어 제거
`/export`	대화 내역을 마크다운으로 내보내기	문서화, 팀 공유, 학습 자료

! Prefix: 즉시 실행의 마법 (Ado Tip #4)

`!` 접두사를 붙이면 Claude의 처리 없이 **즉시 셸 명령을 실행**하고 결과를 컨텍스트에 주입합니다. 이는 토큰 낭비를 방지하고 속도를 높입니다.

예시:

```
# 일반적인 방법 (느림, 토큰 낭비)
> "git status를 실행해줘"
Claude: "네, git status를 실행하겠습니다..."
[실행 후 결과 표시]

# ! Prefix 방법 (빠름, 효율적)
> !git status
[즉시 결과 표시, Claude는 결과만 읽음]
```

Ado의 조언:

> *"! Prefix는 제가 가장 자주 사용하는 기능입니다. `!git diff`, `!npm test`, `!docker ps` 등 간단한 상태 확인에 완벽합니다."*

2.3. CLAUDE.md: AI를 위한 프로젝트 설명서 (Ado Tip #1, #2, ykdojo Tip #30)

`CLAUDE.md` 파일은 **AI를 위한 프로젝트 설명서이자 행동 지침**입니다. Claude는 이 파일을 최우선으로 참고하여 프로젝트의 기술 스택, 코딩 스타일, 주요 라이브러리, 그리고 **해서는 안 될 일**들을 파악합니다.

/init: AI가 스스로 온보딩 (Ado Tip #1)

새 프로젝트에서 `/init` 명령어를 실행하면, Claude가 코드베이스를 분석하여 `CLAUDE.md`의 초안을 자동으로 생성해줍니다.

실제 예시:

```
> /init
```

Claude가 생성한 `CLAUDE.md`:

```
# Project: E-commerce Platform

## Tech Stack
- **Frontend**: Next.js 14, React 18, TailwindCSS
- **Backend**: Node.js, Express, PostgreSQL
- **ORM**: Drizzle
- **Authentication**: NextAuth.js
- **Payment**: Stripe

## Project Structure
- `/app`: Next.js app directory (routes, layouts)
- `/components`: Reusable React components
- `/lib`: Utility functions, database client
- `/api`: Backend API routes

## Coding Standards
- Use TypeScript strict mode
- Prefer server components over client components
- Use `async/await` instead of `.then()`
- Always validate user input with Zod

## DO NOT
- Never commit `.env` files
- Never use `any` type in TypeScript
- Never bypass authentication checks
- Never expose API keys in client code

## Common Commands
- `npm run dev`: Start development server
- `npm run build`: Build for production
- `npm run db:push`: Push schema changes to database
```

Memory Updates: 실시간 메모리 업데이트 (Ado Tip #2)

`CLAUDE.md`를 직접 편집하지 않고, 자연어로 지시하여 업데이트할 수 있습니다.

예시:

```
> Update Claude.md: always use bun instead of npm
```

Claude가 `CLAUDE.md`를 자동으로 수정합니다:

```
## Common Commands
- `bun dev`: Start development server (always use bun, not npm)
- `bun run build`: Build for production
- `bun db:push`: Push schema changes to database
```

간결함의 미학 (ykdojo Tip #30)

ykdojo는 `CLAUDE.md`를 **최대한 간결하고 명확하게** 유지하라고 조언합니다.

나쁜 예 (장황함):

```
## Authentication
Our authentication system is built using NextAuth.js, which is a complete authentication solution for
Next.js applications. It provides a flexible and secure way to add authentication to your app. We use the
Credentials provider to authenticate users with email and password. The session strategy is set to JWT,
which means that the session is stored in a JWT token on the client side. This is more scalable than
storing sessions in a database. When a user logs in, we verify their credentials against the database,
and if they are correct, we create a JWT token and send it to the client. The client then includes this
token in the Authorization header of subsequent requests...
```

좋은 예 (간결함):

```
## Authentication
- NextAuth.js with Credentials provider
- JWT session strategy
- **DO NOT**: Bypass auth checks, expose session secrets
```

ykdojo의 조언:

> *"처음에는 CLAUDE.md 없이 시작하세요. 같은 말을 반복하게 되면 그때 추가하세요. 과도한 정보는 컨텍스트를 낭비합니다."*

2.4. 터미널 별칭으로 빠른 접근 (ykdojo Tip #7)

파워 유저들은 타이핑을 최소화하기 위해 터미널 별칭을 적극 활용합니다. ykdojo가 사용하는 별칭들을 소개합니다.

~/.zshrc 또는 ~/.bashrc에 추가:

```
# Claude Code 기본 별칭
alias c='claude'
alias cc='claude --continue'  # 마지막 대화 이어가기
alias cr='claude --resume'    # 대화 목록에서 선택
alias ch='claude --chrome'    # 크롬 통합 모드

# Git 관련 별칭 (Claude와 함께 사용)
alias gb='git branch'
alias gco='git checkout'
alias gst='git status'
alias gd='git diff'

# 빠른 종료
alias q='exit'
```

사용 예시:

```
# 기존 방법
$ claude --continue
$ claude --chrome

# 별칭 사용
$ cc
$ ch
```

ykdojo의 조언:

> *"하루에 수십 번 실행하는 명령어라면, 별칭을 만드세요. 타이핑 시간이 쌓이면 엄청난 차이를 만듭니다."*

2.5. 세션 관리: 대화를 잃지 않는 법 (Ado Tip #9, #10, #11, #12)

Claude Code는 모든 대화를 자동으로 저장하지만, 효과적으로 관리하지 않으면 나중에 찾기 어렵습니다. Ado가 소개하는 세션 관리 기술을 익히면, 과거의 작업을 쉽게 이어갈 수 있습니다.

Continue Where You Left Off (Ado Tip #9)

```
$ claude --continue
# 또는
$ c -c
```

마지막으로 작업하던 대화를 즉시 이어갑니다. 컴퓨터를 재부팅하거나, 다른 작업을 하다가 돌아왔을 때 유용합니다.

Named Sessions: 세션에 이름 붙이기 (Ado Tip #10)

대화 중에 `/rename` 명령어를 사용하여 세션에 의미 있는 이름을 붙일 수 있습니다.

```
> /rename auth-system-refactor
```

이제 이 세션을 이름으로 재개할 수 있습니다:

```
$ claude --resume auth-system-refactor
```

Ado의 조언:

> *"중요한 작업은 항상 이름을 붙이세요. '2025-01-15 14:30'보다 'stripe-integration'이 훨씬 기억하기 쉽습니다."*

Claude Code Remote: 원격 세션 (Ado Tip #11)

웹 브라우저에서 시작한 Claude Code 세션을 로컬 터미널로 가져올 수 있습니다.

```
$ claude --teleport <session_id>
```

세션 ID는 웹 UI에서 확인할 수 있습니다. 이는 다음 상황에 유용합니다:

- 외출 중 웹에서 작업을 시작했는데, 집에 돌아와서 터미널에서 이어가고 싶을 때
- 팀원이 웹에서 시작한 세션을 로컬에서 디버깅할 때

/export: 대화 내역 내보내기 (Ado Tip #12)

```
> /export
```

현재 대화 내역을 마크다운 파일로 내보냅니다. 다음 용도로 활용할 수 있습니다:

- 팀원과 공유 (Slack, Notion 등)
- 문서화 (프로젝트 위키에 추가)

- 학습 자료 (나중에 복습)
- 블로그 포스트 작성

내보낸 파일 예시:

```
# Session: stripe-integration
**Date**: 2025-01-15
**Model**: Claude Opus 4.5

## User
Stripe 결제 통합을 시작하고 싶어. 먼저 Stripe 계정을 설정하는 방법을 알려줘.

## Claude
Stripe 결제 통합을 시작하겠습니다. 다음 단계를 따라주세요:

1. [Stripe 대시보드](https://dashboard.stripe.com)에 로그인...
```

Part 3: 생산성을 극대화하는 핵심 기술

이 장에서는 타이핑 시간을 줄이고, 정보 전달 속도를 높이며, 반복 작업을 자동화하는 파워 유저들의 핵심 기술을 다룹니다. 이 기술들을 익히면 Claude Code와의 협업 속도가 2배 이상 빨라집니다.

3.1. 음성으로 코딩하기 (ykdojo Tip #2)

ykdojo가 가장 혁신적이라고 평가하는 기술은 **음성 코딩**입니다. 타이핑보다 말이 훨씬 빠르기 때문입니다. 평균적으로 사람은 분당 40단어를 타이핑하지만, 분당 150단어를 말할 수 있습니다. 이는 **3.75배의 속도 차이**입니다.

ykdojo의 커스텀 음성 전사 시스템

ykdojo는 Claude Code를 사용하여 자신만의 음성 전사 시스템을 구축했습니다. 이 시스템은:

1. 키보드 단축키(예: `Cmd+Shift+Space`)를 누르면 녹음 시작
2. 다시 누르면 녹음 종료 및 전사 시작
3. 전사된 텍스트를 현재 커서 위치에 자동 삽입

사용 가능한 도구들:

도구	플랫폼	특징	가격
superwhisper	macOS	로컬 처리, 빠름, 정확함	$30 일회성
MacWhisper	macOS	OpenAI Whisper 기반	무료/프리미엄
ykdojo's Custom Tool	macOS	Claude Code로 제작, 완전 커스터마이징	오픈소스
Windows Speech Recognition	Windows	내장, 무료	무료

음성 코딩 워크플로우 예시:

```
[음성] "Claude, 사용자 인증 미들웨어를 만들어줘.
JWT 토큰을 검증하고, 유효하지 않으면 401 에러를 반환해.
Express.js를 사용하고, TypeScript로 작성해줘."

[Claude가 즉시 코드 생성]
```

ykdojo의 조언:

> *"약간의 발음 오류나 문법 실수는 Claude가 맥락으로 이해하므로 완벽할 필요 없습니다. 생각의 흐름을 끊지 않고 바로 말하세요."*

음성 코딩이 특히 유용한 상황:

- 복잡한 요구사항을 설명할 때
- 여러 단계의 작업을 한 번에 지시할 때
- 손이 피곤하거나 손목 터널 증후군이 있을 때
- 산책하거나 이동 중일 때

3.2. 터미널 출력 추출의 기술 (ykdojo Tip #6)

Claude가 생성한 긴 출력(예: 테스트 결과, 로그 분석, 코드 리뷰)을 효율적으로 추출하고 활용하는 방법입니다.

방법 1: pbcopy/pbpaste (macOS/Linux)

```
# Claude의 마지막 출력을 클립보드로 복사
> "마지막 출력을 pbcopy로 복사해줘"

# 또는 직접 명령어 실행
> !claude -r | tail -n 50 | pbcopy
```

방법 2: VS Code로 바로 열기

```
# Claude가 생성한 코드를 임시 파일로 저장하고 VS Code에서 열기
> "이 코드를 temp.ts 파일로 저장하고 VS Code로 열어줘"

# Claude 실행:
# cat > temp.ts << 'EOF'
# [코드 내용]
# EOF
# code temp.ts
```

방법 3: GitHub Desktop으로 변경 사항 검토

```
# Claude가 여러 파일을 수정한 후
> "GitHub Desktop을 열어줘"

# 또는
> !open -a "GitHub Desktop"
```

GitHub Desktop에서 변경된 파일들을 시각적으로 검토하고, 원하지 않는 변경 사항은 되돌릴 수 있습니다.

ykdojo의 워크플로우:

1. Claude에게 코드 작성 요청
2. GitHub Desktop으로 변경 사항 검토
3. 문제가 있으면 특정 파일만 되돌리기
4. Claude에게 “이 파일만 다시 수정해줘”라고 요청

3.3. Cmd+A/Ctrl+A: 전체 선택의 힘 (ykdojo Tip #10)

웹 페이지나 문서의 내용을 빠르게 컨텍스트에 넣고 싶을 때, `Cmd+A` (macOS) 또는 `Ctrl+A` (Windows/Linux)로 전체를 선택하고 복사하여 Claude Code에 붙여넣으세요.

실제 사례: 접근이 막힌 웹사이트

ykdojo는 Claude의 네이티브 브라우저로 접근이 막힌 웹사이트의 내용을 가져오기 위해 Gemini CLI를 사용했습니다.

워크플로우:

```
# 1. Gemini CLI로 웹사이트 접근
> "Gemini CLI를 사용하여 https://example.com의 내용을 가져와줘"

# 2. Gemini가 가져온 내용을 Cmd+A로 전체 선택
# 3. Claude Code에 붙여넣기
> "이 내용을 분석해서 핵심 정보를 요약해줘"
```

또 다른 활용: 긴 문서 요약

```
# PDF나 웹 페이지의 내용을 전체 복사한 후
> "이 논문의 핵심 내용을 3개의 불릿 포인트로 요약해줘"
```

3.4. 마크다운과 Notion 활용 (ykdojo Tip #19, #20)

Tip #19: 마크다운 활용

Claude Code는 마크다운을 완벽하게 지원합니다. 복잡한 구조의 정보를 전달할 때 마크다운을 사용하면 Claude가 더 정확하게 이해합니다.

예시:

```
다음 API 엔드포인트를 구현해줘:

## POST /api/users
- **Request Body**: { name: string, email: string }
- **Response**: { id: number, name: string, email: string, createdAt: string }
- **Validation**:
  - name은 2-50자
  - email은 유효한 형식
- **Error Cases**:
  - 400: 유효성 검사 실패
  - 409: 이메일 중복
```

Tip #20: Notion으로 링크 보존

Slack이나 다른 곳에서 링크가 포함된 텍스트를 복사하여 Claude Code에 직접 붙여넣으면 링크가 사라집니다. 하지만 Notion에 먼저 붙여넣은 후, Notion에서 복사하면 마크다운 형식으로 변환되어 링크가 보존됩니다.

워크플로우:

1. Slack에서 링크가 포함된 메시지 복사
2. Notion 페이지에 붙여넣기
3. Notion에서 다시 복사
4. Claude Code에 붙여넣기 → 링크가 마크다운 형식으로 유지됨

3.5. 키보드 단축키 완전 정복 (Ado Tip #5, #6, #7, #8, ykdojo Tip #38)

파워 유저들은 마우스를 거의 사용하지 않습니다. 모든 작업을 키보드로 처리합니다.

필수 단축키 (Ado의 추천)

단축키	기능	사용 시점
`Esc Esc`	대화/코드 되감기 (Undo)	잘못된 변경 즉시 되돌리기
`Ctrl+R`	역방향 검색 (명령어 히스토리)	이전에 사용한 프롬프트 재사용
`Ctrl+S`	프롬프트 임시 저장 (Stash)	작성 중인 프롬프트를 나중을 위해 보관
`Tab` / `Enter`	프롬프트 제안 수락	Claude가 제안한 다음 작업 즉시 실행

Double Esc: 되감기의 마법 (Ado Tip #5)

Claude가 잘못된 코드를 작성했거나, 원하지 않는 파일을 삭제했을 때, `Esc`를 두 번 빠르게 누르면 마지막 작업을 되돌립니다. 이는 Git의 `git reset --hard HEAD~1`과 유사하지만, 훨씬 빠르고 안전합니다.

Ctrl+R: 역방향 검색 (Ado Tip #6)

터미널의 `Ctrl+R`처럼, Claude Code에서도 이전에 사용한 프롬프트를 검색할 수 있습니다.

```
# Ctrl+R 누르기
(reverse-i-search): auth

# "auth"가 포함된 이전 프롬프트들이 표시됨
> "사용자 인증 미들웨어를 만들어줘"
```

Prompt Stashing: 프롬프트 임시 저장 (Ado Tip #7)

긴 프롬프트를 작성하던 중, 갑자기 다른 작업을 해야 할 때 `Ctrl+S`로 임시 저장할 수 있습니다. 나중에 다시 `Ctrl+S`를 누르면 저장된 프롬프트가 복원됩니다.

Prompt Suggestions: 다음 작업 예측 (Ado Tip #8)

Claude는 현재 컨텍스트를 바탕으로 다음에 할 작업을 예측하여 제안합니다. 제안이 표시되면 `Tab` 또는 `Enter`를 눌러 즉시 수락할 수 있습니다.

예시:

```
# 코드 작성 후, Claude가 제안:
> "이 코드에 대한 테스트를 작성할까요?" [Tab to accept]

# Tab 누르면 즉시 테스트 작성 시작
```

3.6. Vim 모드로 프롬프트 편집 (Ado Tip #13)

Vim 사용자라면 `/vim` 명령어로 Vim 모드를 활성화할 수 있습니다. 이제 프롬프트 입력 박스에서 Vim 키 바인딩을 사용할 수 있습니다.

Vim 모드에서 사용 가능한 명령:

- `hjkl` : 커서 이동
- `w` , `b` : 단어 단위 이동
- `0` , `$` : 줄 시작/끝으로 이동
- `dd` : 줄 삭제
- `yy` : 줄 복사
- `p` : 붙여넣기
- `i` , `a` : 삽입 모드
- `Esc` : 일반 모드

Ado의 조언:

> *"Vim 사용자라면 이 기능 없이는 살 수 없을 겁니다. 긴 프롬프트를 편집할 때 특히 유용합니다."*

3.7. 입력 박스 탐색 및 편집 (ykdojo Tip #38)

Vim을 사용하지 않더라도, Claude Code의 입력 박스는 일반적인 터미널/readline 단축키를 지원합니다.

탐색 단축키:

단축키	기능
`Ctrl+A`	줄 시작으로 이동
`Ctrl+E`	줄 끝으로 이동
`Option+Left/Right` (Mac)	단어 단위 이동
`Alt+Left/Right` (Linux/Windows)	단어 단위 이동

편집 단축키:

단축키	기능
`Ctrl+W`	이전 단어 삭제
`Ctrl+U`	커서부터 줄 시작까지 삭제
`Ctrl+K`	커서부터 줄 끝까지 삭제
`Ctrl+C` / `Ctrl+L`	현재 입력 지우기
`Ctrl+G`	외부 에디터에서 프롬프트 편집

Ctrl+G: 외부 에디터로 편집

매우 긴 프롬프트를 작성해야 할 때, `Ctrl+G`를 누르면 `$EDITOR` 환경 변수에 설정된 에디터(Vim, VS Code 등)가 열립니다. 에디터에서 작성을 마치고 저장하면, 내용이 Claude Code 입력 박스에 자동으로 삽입됩니다.

설정 방법:

```
# ~/.zshrc 또는 ~/.bashrc에 추가
export EDITOR=vim  # 또는 code, nano, nvim 등
```

이미지 붙여넣기:

플랫폼	단축키
Mac	`Ctrl+V` (주의: Cmd+V가 아님)
Linux	`Ctrl+V`
Windows	`Alt+V`

여러 줄 입력:

- 가장 간단한 방법: `\` 입력 후 `Enter` → 새 줄 생성
- 또는 `/terminal-setup`으로 터미널별 설정 확인 (Mac Terminal.app에서는 `Option+Enter`)

Part 4: 컨텍스트 관리의 예술

컨텍스트는 AI의 작업 기억입니다. 이를 효과적으로 관리하는 것은 Claude Code 마스터리의 핵심입니다.

4.1. 선제적 컨텍스트 압축 (ykdojo Tip #8)

대화가 길어지면 컨텍스트가 가득 차서 성능이 저하됩니다. ykdojo는 **HANDOFF.md** 기법으로 이를 해결합니다.

HANDOFF.md 워크플로우:

```
# 1. 대화가 50k 토큰 이상 사용 중일 때
> "/context로 현재 사용량 확인"

# 2. HANDOFF.md 생성 요청
> "지금까지의 작업 내용을 HANDOFF.md 파일로 정리해줘.
   다음 에이전트가 이 파일만 읽고 작업을 이어갈 수 있도록
   시도한 것, 성공한 것, 실패한 것, 다음 단계를 명확히 작성해줘."

# 3. 새 세션 시작
> "/clear"

# 4. HANDOFF.md 로드
> "@HANDOFF.md 이 파일을 읽고 작업을 이어가줘"
```

HANDOFF.md 예시:

```
# 프로젝트: Stripe 결제 통합

## 목표
Stripe Checkout을 사용한 일회성 결제 시스템 구현

## 완료된 작업
✅ Stripe SDK 설치 및 API 키 설정
✅ `/api/create-checkout-session` 엔드포인트 구현
✅ 성공/취소 페이지 라우팅

## 시도했지만 실패한 것
❌ Webhook 서명 검증 - 로컬 환경에서 테스트 불가
  - 해결책: ngrok 사용 필요

## 현재 문제
- Webhook 이벤트 처리 로직 미완성
- 데이터베이스에 결제 기록 저장 필요

## 다음 단계
1. ngrok으로 로컬 서버 노출
2. Stripe 대시보드에서 Webhook URL 등록
3. `payment_intent.succeeded` 이벤트 핸들러 구현
4. 결제 기록을 `payments` 테이블에 저장
```

4.2. 터미널 탭으로 멀티태스킹 (ykdojo Tip #14)

하나의 대화에서 여러 작업을 섞으면 컨텍스트가 오염됩니다. ykdojo는 **3-4개의 터미널 탭**을 동시에 열어 각각 독립적인 작업을 수행합니다.

탭 분리 전략:

탭	용도	예시
탭 1	메인 개발 작업	새 기능 구현
탭 2	버그 수정	긴급 핫픽스
탭 3	리서치 및 실험	새로운 라이브러리 테스트
탭 4	DevOps 및 배포	CI/CD 디버깅

ykdojo의 조언:

> *"각 탭은 독립적인 '두뇌'입니다. 작업을 섞지 마세요. 한 탭에서 인증 시스템을 작업하다가 갑자기 UI 디자인을 요청하면, Claude는 혼란스러워합니다."*

4.3. 대화 복제 및 반복제 (ykdojo Tip #23)

현재 대화를 유지하면서 다른 접근 방식을 시도하고 싶을 때, **대화 복제** 기능을 사용합니다.

Clone: 전체 복제

```
> /clone
```

현재 대화의 모든 내용을 새로운 UUID로 복제합니다. 첫 메시지에 `[CLONED Jan 15 14:30]` 태그가 추가되어 복제본임을 알 수 있습니다.

사용 사례:

- A 방식과 B 방식 중 어느 것이 더 나은지 비교 실험
- 위험한 변경을 시도하기 전에 백업 생성
- 팀원과 공유하기 위한 스냅샷

Half-Clone: 반복제 (컨텍스트 절반 줄이기)

```
> /half-clone
```

대화의 후반부만 유지하고 전반부는 삭제합니다. 컨텍스트 사용량을 절반으로 줄이면서 최근 작업은 유지합니다.

사용 사례:

- 대화가 너무 길어져서 성능이 저하될 때
- 초기 실험 단계는 더 이상 필요 없고, 현재 구현만 중요할 때

설치 방법 (ykdojo의 dx 플러그인 사용):

```
claude plugin marketplace add ykdojo/claude-code-tips
claude plugin install dx@ykdojo
```

이제 `/dx:clone`과 `/dx:half-clone` 명령어를 사용할 수 있습니다.

4.4. /context로 X-Ray 비전 (Ado Tip #15)

`/context` 명령어는 컨텍스트 윈도우의 **X-Ray**입니다. 무엇이 공간을 차지하는지 정확히 보여줍니다.

출력 예시:

```
Context Usage: 87,432 / 200,000 tokens (43.7%)

Breakdown:
- System Prompt: 10,234 tokens (11.7%)
- MCP Servers: 15,678 tokens (18.0%)
  - supabase-mcp: 8,234 tokens
  - playwright-mcp: 4,123 tokens
  - firecrawl-mcp: 3,321 tokens
- Memory Files (CLAUDE.md): 2,345 tokens (2.7%)
- Skills: 1,234 tokens (1.4%)
- Conversation History: 57,941 tokens (66.3%)
```

최적화 전략:

1. **MCP 정리**: 사용하지 않는 MCP는 `/mcp`로 비활성화
2. **CLAUDE.md 간소화**: 불필요한 설명 제거
3. **대화 압축**: `/clear` 또는 HANDOFF.md 사용

Ado의 조언:

> *"일반적으로 10개 미만의 MCP와 80개 미만의 활성 도구를 유지하는 것이 최적입니다."*

4.5. realpath로 절대 경로 얻기 (ykdojo Tip #24)

다른 디렉토리의 파일을 Claude에게 알려줄 때, 상대 경로 대신 절대 경로를 사용하면 혼란을 방지할 수 있습니다.

```
# 잘못된 방법
> "../../config/database.ts 파일을 확인해줘"

# 올바른 방법
> !realpath ../../config/database.ts
/Users/ykdojo/projects/myapp/config/database.ts

> "@/Users/ykdojo/projects/myapp/config/database.ts 이 파일을 확인해줘"
```

Part 5: Git과 GitHub 워크플로우 완전 정복

Claude Code는 Git 및 GitHub와 완벽하게 통합됩니다. 파워 유저들은 이를 활용하여 코드 리뷰, 커밋, PR 생성까지 모두 자동화합니다.

5.1. Git과 GitHub CLI 프로 활용 (ykdojo Tip #4)

ykdojo는 Git과 GitHub CLI(`gh`)를 Claude Code와 결합하여 강력한 워크플로우를 만들었습니다.

자동 커밋 메시지 생성

```
> "변경 사항을 분석하고 적절한 커밋 메시지를 작성한 후 커밋해줘"

# Claude 실행:
# git diff를 분석
# 커밋 메시지 생성: "feat: Add user authentication middleware with JWT validation"
# git add . && git commit -m "..."
```

Draft PR 자동 생성

```
> "현재 브랜치의 변경 사항으로 draft PR을 만들어줘.
   제목은 변경 내용을 요약하고, 본문에는 주요 변경 사항을 리스트로 작성해줘."

# Claude 실행:
# gh pr create --draft --title "..." --body "..."
```

PR 템플릿 활용:

`.github/pull_request_template.md` 파일을 만들어두면, Claude가 이를 참고하여 PR 본문을 작성합니다.

```
## 변경 사항
-

## 테스트 방법
-

## 체크리스트
- [ ] 테스트 추가됨
- [ ] 문서 업데이트됨
- [ ] Breaking change 없음
```

5.2. Git worktrees로 병렬 브랜치 작업 (ykdojo Tip #16)

여러 브랜치에서 동시에 작업해야 할 때, `git worktree`를 사용하면 브랜치를 전환하지 않고 병렬로 작업할 수 있습니다.

Git Worktree란?

하나의 저장소에서 여러 작업 디렉토리를 만들어, 각각 다른 브랜치를 체크아웃할 수 있는 기능입니다.

사용 방법:

```
# 메인 프로젝트: ~/projects/myapp (main 브랜치)
# Worktree 1: ~/projects/myapp-feature-auth (feature/auth 브랜치)
# Worktree 2: ~/projects/myapp-hotfix (hotfix/critical-bug 브랜치)

# Worktree 생성
> "git worktree를 사용하여 feature/auth 브랜치를 ../myapp-feature-auth에 체크아웃해줘"

# Claude 실행:
# git worktree add ../myapp-feature-auth feature/auth
```

워크플로우:

1. 터미널 탭 1: `~/projects/myapp-feature-auth`에서 새 기능 개발

2. 터미널 탭 2: `~/projects/myapp-hotfix`에서 긴급 버그 수정
3. 터미널 탭 3: `~/projects/myapp`에서 메인 브랜치 유지

ykdojo의 조언:

> *"Worktree는 컨텍스트 스위칭 비용을 제거합니다. 브랜치를 전환할 때마다 node_modules를 다시 설치하거나, 빌드를 다시 실행할 필요가 없습니다."*

5.3. 대화형 PR 리뷰 (ykdojo Tip #26)

Claude Code는 단순한 코드 생성기를 넘어, 훌륭한 코드 리뷰어입니다.

PR 리뷰 워크플로우:

```
# 1. PR 체크아웃
> "gh pr checkout 123을 실행하고 이 PR의 변경 사항을 요약해줘"

# 2. 전체 개요 파악
> "이 PR이 어떤 문제를 해결하는지, 주요 변경 파일은 무엇인지 설명해줘"

# 3. 파일별 심층 리뷰
> "src/auth/middleware.ts 파일의 변경점을 분석해줘.
   보안 이슈나 성능 문제가 있는지 확인해줘"

# 4. 테스트 커버리지 확인
> "이 변경에 대한 테스트가 충분한지 평가해줘"

# 5. 개선 제안
> "이 로직을 더 효율적으로 바꿀 방법이 있을까?"

# 6. 자동 수정
> "네가 제안한 개선 사항을 적용하고 테스트를 실행해줘"
```

ykdojo의 실제 사례:

한 PR에서 복잡한 인증 로직을 리뷰하던 중, Claude가 다음을 발견했습니다:

- 토큰 만료 시간 검증 로직의 타임존 버그
- 불필요한 데이터베이스 쿼리 (N+1 문제)
- 에러 핸들링 누락

Claude는 이를 모두 수정하고, 테스트 코드까지 추가했습니다.

5.4. 승인된 명령어 감사 (ykdojo Tip #33)

Claude Code는 반복적인 명령어를 자동 승인하도록 설정할 수 있지만, 이는 위험할 수 있습니다. 실제로 한 사용자는 `rm -rf tests/ patches/ plan/ ~/`를 승인하여 홈 디렉토리를 삭제한 사례가 있습니다.

cc-safe: 위험한 명령어 감사 도구

ykdojo가 만든 `cc-safe`는 `.claude/settings.json` 파일을 스캔하여 위험한 승인 명령어를 감지합니다.

설치 및 사용:

```
# 설치
npm install -g cc-safe

# 현재 디렉토리 스캔
npx cc-safe .

# 전체 프로젝트 폴더 스캔
npx cc-safe ~/projects
```

감지하는 위험 패턴:

- `sudo`, `rm -rf`, `chmod 777`
- `curl | sh`, `wget | bash`
- `git reset --hard`, `git push --force`
- `npm publish`, `docker run --privileged`

출력 예시:

```
⚠️  Found 3 risky approved commands:

1. /Users/ykdojo/projects/myapp/.claude/settings.json
   - "Bash(rm -rf dist/)"  ← Dangerous: rm -rf

2. /Users/ykdojo/projects/myapp/.claude/settings.json
   - "Bash(sudo npm install -g)"  ← Dangerous: sudo

3. /Users/ykdojo/projects/experiment/.claude/settings.json
   - "Bash(curl https://install.sh | sh)"  ← Dangerous: curl | sh
```

ykdojo의 조언:

> *"한 달에 한 번은 `cc-safe`를 실행하여 승인된 명령어를 감사하세요. 특히 `--dangerously-skip-permissions` 모드를 자주 사용한다면 필수입니다."*

Part 6: 고급 기능 - MCP, Hooks, Agents

이제 Claude Code를 단순한 코딩 도우미에서 벗어나, 여러분의 명령을 수행하는 강력한 AI 에이전트 팀으로 만들어주는 고급 기능들을 탐험할 차례입니다.

6.1. MCP: 외부 세계와의 연결 (Ado Tip #22, #23, #24, #25)

MCP(Model Context Protocol)는 Claude가 외부 서비스 및 API와 직접 통신할 수 있도록 만드는 프로토콜입니다.

MCP 서버 설치:

```
# Playwright MCP (브라우저 자동화)
claude mcp add -s user playwright npx @playwright/mcp@latest

# Supabase MCP (데이터베이스 직접 쿼리)
claude mcp add -s user supabase npx @supabase/mcp@latest

# Firecrawl MCP (웹 크롤링)
claude mcp add -s user firecrawl npx @firecrawl/mcp@latest
```

Supabase MCP 활용 사례 (Ado Tip #24):

```
> "Supabase 데이터베이스의 users 테이블에서
   지난 7일간 가입한 사용자 수를 조회해줘"

# Claude가 직접 SQL 쿼리 실행:
# SELECT COUNT(*) FROM users WHERE created_at >= NOW() - INTERVAL '7 days'

# 결과: 1,234명
```

Firecrawl MCP 활용 사례 (Ado Tip #25):

```
> "https://example.com의 모든 제품 페이지를 크롤링하고,
   제품명과 가격을 CSV 파일로 저장해줘"

# Claude가 Firecrawl MCP를 사용하여:
# 1. 웹사이트 크롤링
# 2. 제품 정보 추출
# 3. products.csv 파일 생성
```

Playwright MCP 활용 사례 (Ado Tip #22):

```
> "Playwright를 사용하여 로그인 폼을 테스트해줘.
   잘못된 비밀번호 입력 시 에러 메시지가 표시되는지 확인해"

# Claude가 Playwright로:
# 1. 브라우저 실행
# 2. 로그인 페이지 접속
# 3. 잘못된 비밀번호 입력
# 4. 에러 메시지 확인
# 5. 스크린샷 저장
```

⚠ MCP 성능 최적화 (ykdojo의 경고):

MCP는 강력하지만, 활성화된 MCP가 많아질수록 컨텍스트 윈도우를 많이 차지합니다. ykdojo는 다음을 권장합니다:

- **10개 미만의 MCP 서버**
- **80개 미만의 활성 도구**
- 사용하지 않는 MCP는 `/mcp`로 비활성화

6.2. Hooks: 규칙의 강제 (Ado Tip #26)

Hooks는 특정 이벤트 발생 시 자동으로 실행되는 셸 명령어입니다. 이는 AI의 확률적 행동에 **결정론적 제어**를 추가합니다.

Hook 종류:

Hook	실행 시점	사용 사례
`PreToolUse`	도구 실행 전	위험한 명령어 차단
`PostToolUse`	도구 실행 후	로그 기록, 알림 전송
`PermissionRequest`	권한 요청 시	자동 승인/거부
`Notification`	Claude의 알림 시	외부 시스템 통합
`SubagentStart`	서브에이전트 시작 시	모니터링
`SubagentStop`	서브에이전트 종료 시	결과 수집

실전 예시: tmux 없이 장시간 명령어 경고

```
{
  "hooks": {
    "PreToolUse": {
      "command": "bash",
      "args": ["-c", "if [[ $TOOL_NAME == 'Bash' ]] && [[ ! $TMUX ]]; then echo '⚠️  Warning: Long-running command without tmux'; fi"]
    }
  }
}
```

실전 예시: 위험한 명령어 차단

```
{
  "hooks": {
    "PreToolUse": {
      "command": "bash",
      "args": ["-c", "if echo $TOOL_INPUT | grep -q 'rm -rf /'; then echo 'BLOCKED: Dangerous command'; exit 1; fi"]
    }
  }
}
```

Ado의 조언:

> *"Hooks는 AI에게 '가드레일'을 제공합니다. 확률적으로 실수할 수 있는 AI에게 절대적인 규칙을 강제할 수 있습니다."*

6.3. Skills: 재사용 가능한 지식 (Ado Tip #27)

Skills는 재사용 가능한 지식 조각입니다. Claude가 필요할 때 자동으로 호출하거나, 사용자가 수동으로 호출할 수 있습니다.

Skill 생성 예시: Google 번역 링크

`~/.claude/skills/google-translate/skill.md`:

```
# Google Translate Skill

When the user asks how to pronounce a word in a specific language, generate a Google Translate link.

## Format
https://translate.google.com/?sl=auto&tl={target_language}&text={word}

## Example
User: "How do you pronounce 'hello' in Korean?"
Response: "Here's the pronunciation: https://translate.google.com/?sl=auto&tl=ko&text=hello"
```

이제 Claude는 발음 질문을 받으면 자동으로 이 스킬을 사용합니다.

ykdojo의 Reddit Fetch Skill:

ykdojo는 Reddit 콘텐츠를 가져오는 스킬을 만들어, Gemini CLI를 통해 Reddit을 크롤링하도록 했습니다. 이 스킬은 `/dx` 플러그인에 포함되어 있습니다.

6.4. Agents: 전문화된 서브에이전트 (Ado Tip #28)

Agents(서브에이전트)는 특정 작업에 전문화된 독립적인 AI 프로세스입니다. 메인 Claude가 작업을 위임할 수 있습니다.

서브에이전트의 특징:

- 각자 **독립적인 200k 컨텍스트 윈도우**
- **병렬 실행** 가능
- 전문화된 시스템 프롬프트
- 작업 완료 후 결과를 메인 에이전트에게 반환

실전 예시: TDD 가이드 에이전트

```
> "tdd-guide 에이전트를 실행하여 이 함수에 대한 테스트를 작성해줘"

# Claude가 tdd-guide 서브에이전트 실행:
# - 함수 분석
# - 테스트 케이스 설계
# - 테스트 코드 작성
# - 결과를 메인 에이전트에게 반환
```

실전 예시: 보안 감사 에이전트

```
> "security-auditor 에이전트를 실행하여
   이 코드베이스의 보안 취약점을 찾아줘"

# security-auditor 에이전트:
# - SQL 인젝션 취약점 검사
# - XSS 취약점 검사
# - 하드코딩된 비밀번호 검사
# - 보고서 생성
```

Ado의 비유:

> *"산타는 모든 선물을 혼자 포장하지 않습니다. 엘프들이 있죠. 서브에이전트는 Claude의 엘프입니다."*

6.5. Plugins: 기능 번들 (Ado Tip #29)

Plugins는 Hooks, Skills, Agents, MCP 서버를 하나의 패키지로 묶어 배포하고 설치하는 방법입니다.

플러그인 설치:

```
# Marketplace에서 검색
/plugin search frontend

# 설치
/plugin install frontend-design@anthropic
```

ykdojo의 dx 플러그인:

```
claude plugin marketplace add ykdojo/claude-code-tips
claude plugin install dx@ykdojo
```

dx 플러그인에 포함된 것:

- `/dx:gha <url>` : GitHub Actions 실패 분석
- `/dx:handoff` : HANDOFF.md 생성
- `/dx:clone` : 대화 복제
- `/dx:half-clone` : 대화 반복제
- `reddit-fetch` 스킬: Reddit 콘텐츠 자동 가져오기

6.6. CLAUDE.md vs Skills vs Slash Commands vs Plugins (ykdojo Tip #25)

이 기능들은 비슷해 보이지만, 각각의 목적과 사용 시점이 다릅니다.

기능	로딩 시점	주요 사용자	토큰 효율성	사용 사례
CLAUDE.md	모든 대화 시작 시	개발자	낮음 (항상 로드)	프로젝트 설명, 코딩 스타일, 금지 사항
Skills	필요 시 자동	Claude	높음 (필요 시만)	특정 작업 자동화 (번역, 크롤링 등)
Slash Commands	수동 호출 시	개발자	높음 (호출 시만)	반복 작업 (커밋, PR 생성 등)
Plugins	설치 시	개발자	-	여러 기능을 하나로 묶어 배포

ykdojo의 조언:

> *"Skills와 Slash Commands는 원래 다른 의도로 설계되었지만, 결국 통합되었습니다. Skills는 Claude가 사용하도록, Slash Commands는 개발자가 사용하도록 만들어졌습니다."*

Part 7: 시스템 최적화와 자동화

파워 유저들은 Claude Code 자체를 최적화하고, 반복 작업을 자동화하여 생산성을 극한까지 끌어올립니다.

7.1. 시스템 프롬프트 슬림화 (ykdojo Tip #15)

Claude Code의 모든 행동은 수만 토큰에 달하는 거대한 **시스템 프롬프트**에 의해 결정됩니다. ykdojo는 이 시스템 프롬프트를 직접 수정하여 **19k 토큰에서 10k 이하로 절반 가까이 줄이는 데 성공**했습니다.

왜 시스템 프롬프트를 줄여야 하는가?

- 더 많은 코드 파일과 대화 기록을 컨텍스트에 담을 수 있음
- 응답 속도 향상 (처리할 토큰이 적음)
- 비용 절감 (토큰 사용량 감소)

ykdojo의 58가지 패치:

ykdojo는 58개의 패치를 통해 다음을 제거했습니다:

1. **장황한 예시**: 도구 설명에 포함된 불필요하게 긴 예시
2. **반복적인 지시**: 여러 곳에서 반복되는 동일한 지시사항
3. **과도한 안전 경고**: 너무 자주 반복되는 경고 메시지
4. **사용하지 않는 도구 설명**: 거의 사용되지 않는 도구의 긴 설명

패치 적용 방법:

```
# ykdojo의 저장소 클론
git clone https://github.com/ykdojo/claude-code-tips.git
cd claude-code-tips

# 패치 적용 스크립트 실행
./scripts/apply-patches.sh
```

⚠ 경고: 전문가의 영역

시스템 프롬프트를 잘못 수정하면 Claude의 성능이 심각하게 저하될 수 있습니다. 충분한 이해 없이 시도하지 마세요.

ykdojo의 조언:

> *"처음에는 제가 공유한 패치를 그대로 적용해보세요. 효과를 확인한 후, 자신만의 패치를 만들어보세요."*

7.2. 장시간 작업을 위한 수동 지수 백오프 (ykdojo Tip #17)

장시간 실행되는 작업(예: 대용량 데이터 처리, 복잡한 빌드)을 Claude에게 맡길 때, 매 초마다 상태를 확인하면 비효율적입니다. **지수 백오프(Exponential Backoff)** 전략을 사용하세요.

지수 백오프란?

처음에는 짧은 간격으로 확인하고, 점차 간격을 늘려가는 방식입니다.

- 1분 후 확인
- 2분 후 확인
- 4분 후 확인
- 8분 후 확인
- …

실전 프롬프트:

```
> "대용량 CSV 파일(10GB)을 처리하는 스크립트를 실행해줘.
   지수 백오프 방식으로 진행 상황을 확인해줘.
   처음 1분 후, 그 다음 2분 후, 4분 후, 8분 후 확인해."

# Claude가 실행:
# 1. 스크립트 실행
# 2. 1분 대기 후 상태 확인
# 3. 2분 대기 후 상태 확인
# 4. 작업 완료 시까지 반복
```

ykdojo의 조언:

> *"이 방법은 토큰을 절약하고, Claude가 다른 작업을 병렬로 수행할 수 있게 해줍니다."*

7.3. 백그라운드에서 bash 명령 및 에이전트 실행 (ykdojo Tip #36)

장시간 실행되는 명령어나 에이전트를 백그라운드로 보내고, Claude가 다른 작업을 계속할 수 있게 합니다.

Ctrl+B: 백그라운드로 보내기

명령어가 실행 중일 때 `Ctrl+B`를 누르면 백그라운드로 이동합니다. Claude는 `BashOutput` 도구를 사용하여 나중에 상태를 확인할 수 있습니다.

서브에이전트 백그라운드 실행:

```
> "security-auditor 에이전트를 백그라운드에서 실행하여
   전체 코드베이스를 스캔해줘. 완료되면 알려줘."

# Claude가:
# 1. 서브에이전트 백그라운드 실행
# 2. 메인 작업 계속 진행
# 3. 주기적으로 서브에이전트 상태 확인
# 4. 완료 시 결과 수집
```

병렬 서브에이전트 활용:

```
> "대규모 코드베이스를 분석해야 해.
   3개의 서브에이전트를 백그라운드에서 실행하여
   각각 다른 모듈을 분석하도록 해줘."

# Claude가:
# - Agent 1: /src/auth 모듈 분석
# - Agent 2: /src/api 모듈 분석
# - Agent 3: /src/database 모듈 분석
# 모두 병렬로 실행
```

7.4. 자동화의 자동화 (ykdojo Tip #41)

ykdojo는 이를 **"Automation of automation"**이라고 부릅니다. 이는 워크플로우 자체를 자동화하여 개발자가 더 높은 수준의 추상화에 집중할 수 있게 만드는 에이전틱 개발의 핵심 철학입니다.

ykdojo의 자동화 여정:

1. **Level 0**: ChatGPT에서 코드를 복사하여 터미널에 붙여넣기

2. **Level 1**: ChatGPT 플러그인(Kaguya)을 만들어 자동 실행
3. **Level 2**: Claude Code 사용으로 터미널 통합
4. **Level 3**: 음성 전사 시스템 구축으로 타이핑 자동화
5. **Level 4**: CLAUDE.md로 반복 지시 자동화
6. **Level 5**: 커스텀 슬래시 명령어로 워크플로우 자동화
7. **Level 6**: Skills로 Claude의 자동 판단 자동화
8. **Level 7**: Hooks로 규칙 강제 자동화

ykdojo의 철학:

> *"같은 작업을 2-3번 반복하는 것은 괜찮습니다. 하지만 그 이상 반복한다면, 자동화할 방법을 찾으세요. 그리고 그 자동화 과정 자체도 자동화하세요."*

7.5. Headless 모드로 CI/CD 통합 (Ado Tip #30)

Claude Code는 대화형 도구를 넘어, **CI/CD 파이프라인**에 통합할 수 있는 강력한 CLI 도구입니다.

Headless 모드 사용법:

```
# 기본 사용
claude -p "Fix the lint errors"

# 파이프라인 통합
git diff | claude -p "Explain these changes"

# JSON 출력
echo "Review this PR" | claude -p --json
```

CI/CD 통합 예시: GitHub Actions

`.github/workflows/claude-review.yml`:

```yaml
name: Claude Code Review

on:
  pull_request:
    types: [opened, synchronize]

jobs:
  review:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Install Claude Code
        run: curl -fsSL https://claude.ai/install.sh | sh

      - name: Review PR
        env:
          ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
        run: |
          git diff origin/main...HEAD | \
          claude -p "Review this PR and identify potential issues" \
          > review.md

      - name: Comment on PR
        uses: actions/github-script@v6
        with:
          script: |
            const fs = require('fs');
            const review = fs.readFileSync('review.md', 'utf8');
            github.rest.issues.createComment({
              issue_number: context.issue.number,
              owner: context.repo.owner,
              repo: context.repo.repo,
              body: review
            });
```

Ado의 조언:

> *"Headless 모드는 AI를 파이프라인에 통합합니다. `-p` 플래그는 비대화형으로 실행하고 stdout으로 직접 출력합니다."*

Part 8: 컨테이너와 샌드박스

안전한 실험과 위험한 작업을 위한 격리 환경 구축 방법을 다룹니다.

8.1. 컨테이너로 위험한 작업 격리 (ykdojo Tip #21)

`--dangerously-skip-permissions` 플래그는 매우 편리하지만 위험합니다. ykdojo는 이를 **"보호되지 않은 성관계"**에 비유하며, **컨테이너**라는 안전장치를 사용할 것을 강력히 권장합니다.

Docker 컨테이너 설정:

ykdojo는 자신의 저장소에 `Dockerfile`과 설정 스크립트를 공개했습니다.

```
FROM ubuntu:22.04

# 기본 도구 설치
RUN apt-get update && apt-get install -y \
    curl git tmux vim \
    nodejs npm python3 python3-pip

# Claude Code 설치
RUN curl -fsSL https://claude.ai/install.sh | sh

# Gemini CLI 설치 (선택사항)
RUN npm install -g @google/generative-ai-cli

# 작업 디렉토리
WORKDIR /workspace

CMD ["/bin/bash"]
```

컨테이너 실행:

```
# 이미지 빌드
docker build -t claude-sandbox .

# 컨테이너 실행 (호스트 디렉토리 마운트)
docker run -it --rm \
  -v $(pwd):/workspace \
  -e ANTHROPIC_API_KEY=$ANTHROPIC_API_KEY \
  claude-sandbox

# 컨테이너 안에서 Claude Code 실행
claude --dangerously-skip-permissions
```

고급: 워커 Claude 오케스트레이션

로컬의 메인 Claude가 `tmux`를 통해 컨테이너 안의 "워커" Claude에게 작업을 지시하고 결과를 받아오는 워크플로우:

```
# 메인 Claude에게 지시:
> "Docker 컨테이너를 실행하고, 그 안에서 Claude Code를
   --dangerously-skip-permissions 모드로 실행해줘.
   tmux를 사용하여 워커 Claude에게 '전체 코드베이스를 분석하고
   보안 취약점을 찾아줘'라고 지시해줘."

# 메인 Claude가:
# 1. docker run으로 컨테이너 시작
# 2. tmux 세션 생성
# 3. 워커 Claude 실행
# 4. tmux send-keys로 프롬프트 전송
# 5. tmux capture-pane으로 출력 읽기
# 6. 결과를 메인 세션에 반환
```

ykdojo의 실제 사례:

ykdojo는 새 버전의 Claude Code가 출시될 때마다, 시스템 프롬프트 패치를 업데이트하는 작업을 컨테이너 안의 Claude에게 맡깁니다. 워커 Claude는:

1. 새 버전의 minified JavaScript 분석

2. 변수 매핑 찾기
3. 패치 파일 생성
4. 패치 적용 테스트
5. 실패한 패치 수정
6. 문서 업데이트

이 모든 과정이 자율적으로 진행되며, 메인 Claude는 최종 결과만 검토합니다.

8.2. Sandbox 모드와 권한 관리 (Ado Tip #20)

`/sandbox` 명령어를 사용하면 **한 번 경계를 정의**하고, Claude가 그 안에서 자유롭게 작업하도록 할 수 있습니다.

Sandbox 설정 예시:

```
> /sandbox

# Claude가 묻습니다:
"어떤 명령어를 자동 승인할까요?"

> "npm install, npm test, git status, git diff를 자동 승인해줘"

# 이제 이 명령어들은 매번 물어보지 않고 자동 실행됩니다.
```

와일드카드 지원:

```
> /sandbox

> "mcp__server__* 패턴의 모든 MCP 서버 도구를 자동 승인해줘"

# 모든 MCP 서버의 도구들이 자동 승인됩니다.
```

Ado의 조언:

> *"Sandbox는 속도와 보안을 동시에 제공합니다. 신뢰할 수 있는 작업에는 Sandbox를, 실험적인 작업에는 컨테이너를 사용하세요."*

8.3. YOLO 모드: 위험을 감수할 때 (Ado Tip #20)

```
claude --dangerously-skip-permissions
```

이 플래그는 모든 권한 요청에 자동으로 "예"라고 답합니다. 이름에 "danger"가 들어간 이유가 있습니다.

YOLO 모드를 사용해야 할 때:

- 컨테이너 안에서 실험할 때
- 장시간 자율 작업이 필요할 때 (예: Reddit 리서치)
- 신뢰할 수 있는 반복 작업

절대 사용하지 말아야 할 때:

- 호스트 시스템에서 직접 실행
- 중요한 데이터가 있는 디렉토리
- 프로덕션 환경

Part 9: 브라우저 통합과 웹 자동화

Claude Code는 브라우저를 직접 제어하여 웹 스크래핑, UI 테스트, 디버깅을 수행할 수 있습니다.

9.1. 네이티브 브라우저 통합 (Ado Tip #21)

Claude Code는 Chrome과 직접 통합되어, 브라우저를 제어하고 상호작용할 수 있습니다.

Chrome 확장 설치:

1. https://claude.ai/chrome 방문
2. Chrome 확장 프로그램 설치
3. Claude Code 실행: `claude --chrome`

가능한 작업:

- 페이지 탐색
- 버튼 클릭 및 폼 작성
- 콘솔 에러 읽기
- DOM 검사
- 스크린샷 촬영

실전 예시: 버그 재현 및 수정

```
> "localhost:3000으로 이동하여 로그인 버튼을 클릭해줘.
   콘솔에 에러가 있는지 확인하고, 있다면 코드를 수정해줘."

# Claude가:
# 1. 브라우저에서 localhost:3000 열기
# 2. 로그인 버튼 찾기 및 클릭
# 3. 콘솔 에러 확인: "Uncaught TypeError: Cannot read property 'email' of undefined"
# 4. 코드에서 해당 부분 찾기
# 5. 수정: user?.email로 옵셔널 체이닝 추가
# 6. 브라우저 새로고침하여 수정 확인
```

Ado의 조언:

> *"'버그를 수정하고 작동하는지 확인해줘'가 이제 하나의 프롬프트로 가능합니다."*

9.2. Playwright MCP (Ado Tip #22)

네이티브 브라우저 통합으로 접근이 어려운 동적 웹사이트(JavaScript 기반)와 상호작용이 필요할 때 Playwright MCP를 사용합니다.

설치:

```
claude mcp add -s user playwright npx @playwright/mcp@latest
```

Playwright의 장점:

- 헤드리스 브라우저 지원 (백그라운드 실행)
- 여러 브라우저 지원 (Chrome, Firefox, Safari)
- 네트워크 요청 가로채기
- 파일 다운로드 자동화

실전 예시: E2E 테스트 자동화

```
> "Playwright를 사용하여 전체 사용자 여정을 테스트해줘:
   1. 회원가입
   2. 로그인
   3. 제품 검색
   4. 장바구니 추가
   5. 결제
   각 단계마다 스크린샷을 저장해줘."

# Claude가 Playwright로 전체 시나리오 실행
```

9.3. Gemini CLI를 대체 수단으로 (ykdojo Tip #11)

Claude의 네이티브 브라우저나 Playwright로도 접근이 막힌 웹사이트가 있을 때, **Gemini CLI**를 대체 수단으로 사용할 수 있습니다.

Gemini CLI 설치:

```
npm install -g @google/generative-ai-cli
export GOOGLE_API_KEY=your_api_key
```

ykdojo의 Reddit 리서치 워크플로우:

```
# 1. Claude에게 지시
> "Gemini CLI를 사용하여 Reddit의 r/programming에서
   'Claude Code' 관련 게시물을 찾고 요약해줘"

# 2. Claude가 tmux 세션에서 Gemini CLI 실행
# 3. Gemini가 Reddit 크롤링 및 분석
# 4. 결과를 Claude에게 반환
# 5. Claude가 요약 및 정리
```

멀티 모델 오케스트레이션:

```
> "이 코드를 Claude Opus로 작성하고,
   Gemini로 코드 리뷰를 받은 후,
   GPT-4로 최종 최적화를 해줘"

# Claude Code가 여러 AI 모델을 조율하여 작업 수행
```

ykdojo의 조언:

> *"핵심은 Claude Code의 부드러운 UI/UX입니다. 여러 CLI를 수동으로 전환하고 복사-붙여넣기하는 대신, Claude Code가 모든 것을 조율하는 중앙 인터페이스가 됩니다."*

Part 10: 실전 활용 사례

이론을 넘어, 실제 개발 현장에서 Claude Code가 어떻게 문제를 해결하고 생산성을 향상시키는지 구체적인 사례를 통해 살펴봅니다.

10.1. 작성-테스트 사이클 완성 (ykdojo Tip #9)

코드를 작성하는 것만으로는 부족합니다. 테스트를 작성하고, 실행하고, 수정하는 전체 사이클을 자동화해야 합니다.

완전한 워크플로우:

```
> "사용자 인증 미들웨어를 작성하고, 테스트 코드도 함께 작성해줘.
   테스트를 실행하여 모두 통과하는지 확인해줘."

# Claude가:
# 1. 미들웨어 코드 작성 (src/middleware/auth.ts)
# 2. 테스트 코드 작성 (src/middleware/auth.test.ts)
# 3. npm test 실행
# 4. 테스트 실패 시 코드 수정
# 5. 모든 테스트 통과할 때까지 반복
```

TDD (Test-Driven Development) 워크플로우:

```
> "TDD 방식으로 작업해줘. 먼저 실패하는 테스트를 작성하고,
   그 테스트를 통과시키는 코드를 작성해줘."

# Claude가:
# 1. 실패하는 테스트 작성
# 2. git commit -m "Add failing test for user auth"
# 3. 테스트를 통과시키는 최소한의 코드 작성
# 4. 테스트 실행 → 통과 확인
# 5. git commit -m "Implement user auth to pass test"
```

10.2. 자신만의 워크플로우에 투자 (ykdojo Tip #12)

ykdojo는 자신만의 워크플로우를 구축하는 데 많은 시간을 투자했으며, 이것이 장기적으로 가장 큰 생산성 향상을 가져왔다고 말합니다.

ykdojo의 커스텀 워크플로우:

1. **음성 전사 시스템**: 키보드 단축키로 음성을 텍스트로 변환
2. **커스텀 상태 라인**: 토큰, Git 상태, MCP 수 실시간 표시
3. **자동 커밋 명령어**: `/commit`으로 변경 사항 분석 및 커밋
4. **GitHub Actions 디버거**: `/gha <url>`로 실패 원인 자동 분석
5. **HANDOFF.md 생성기**: `/handoff`로 컨텍스트 압축

여러분만의 워크플로우 만들기:

```
# 1. 반복하는 작업 식별
"나는 매번 PR을 만들 때 같은 형식의 설명을 작성한다"

# 2. 자동화 방법 설계
"PR 템플릿을 만들고, /pr 명령어로 자동 생성하자"

# 3. Claude에게 구현 요청
> "~/.claude/commands/pr.md 파일을 만들어줘.
   이 명령어는 현재 브랜치의 변경 사항을 분석하고,
   PR 템플릿에 맞춰 draft PR을 생성해야 해."

# 4. 테스트 및 개선
# 5. 팀과 공유
```

10.3. 대화 기록 검색 (ykdojo Tip #13)

과거의 대화에서 유용한 정보를 찾아야 할 때가 있습니다.

대화 목록 확인:

```
$ claude -r
# 또는
$ c -r

# 출력:
1. [2025-01-15 14:30] stripe-integration
2. [2025-01-14 09:15] auth-system-refactor
3. [2025-01-13 16:45] database-migration
...
```

특정 대화 재개:

```
$ claude --resume stripe-integration
```

대화 내용 검색 (grep 활용):

```
# 모든 대화 파일에서 "Stripe" 검색
$ grep -r "Stripe" ~/.claude/conversations/

# 결과:
~/.claude/conversations/abc123.json: "Stripe API 키를 환경 변수로 설정..."
~/.claude/conversations/def456.json: "Stripe Webhook 서명 검증..."
```

10.4. 글쓰기 도우미로 활용 (ykdojo Tip #18)

Claude Code는 코딩뿐만 아니라 기술 문서, 블로그 포스트, 튜토리얼 작성에도 탁월합니다.

기술 문서 작성:

```
> "이 API 엔드포인트들에 대한 OpenAPI 스펙을 작성해줘.
   각 엔드포인트의 요청/응답 예시도 포함해줘."

# Claude가 완전한 OpenAPI YAML 파일 생성
```

블로그 포스트 작성:

```
> "내가 방금 구현한 인증 시스템에 대한 블로그 포스트를 작성해줘.
   대상 독자는 주니어 개발자이고, 단계별로 설명해야 해."

# Claude가:
# 1. 코드 분석
# 2. 핵심 개념 추출
# 3. 초보자 친화적인 설명 작성
# 4. 코드 예시 포함
```

ykdojo의 조언:

> *"Claude는 때때로 반복적인 요약을 작성합니다. '마지막 문단에서 이전 내용을 반복하지 마'라고 명시적으로 지시하세요."*

10.5. 연구 도구로 활용 (ykdojo Tip #27)

Claude Code는 Google을 대체할 수 있는 강력한 연구 도구입니다.

GitHub Actions 디버깅:

```
> "이 GitHub Actions 워크플로우가 실패했어.
   로그를 분석하고 원인을 찾아줘."

[로그 전체 붙여넣기 - 수천 줄]

# Claude가:
# 1. 로그 파싱
# 2. 에러 메시지 식별
# 3. 관련 코드 찾기
# 4. 원인 분석: "Node.js 버전 불일치"
# 5. 해결책 제시: "package.json의 engines 필드 업데이트"
```

Reddit 감정 분석:

```
> "Reddit의 r/programming에서 'Claude Code'에 대한
   최근 1주일 게시물을 분석하고, 전반적인 감정과
   주요 피드백을 요약해줘."

# Claude가 Gemini CLI 또는 MCP를 통해:
# 1. Reddit 크롤링
# 2. 게시물 및 댓글 수집
# 3. 감정 분석
# 4. 주요 주제 추출
# 5. 요약 리포트 생성
```

ykdojo의 실제 사례: $10,000 절약:

ykdojo는 Claude Code를 사용한 연구로 $10,000를 절약했습니다. 어떤 서비스를 구독하려고 했는데, Claude Code로 비슷한 기능을 직접 구현할 수 있다는 것을 발견했기 때문입니다.

10.6. 출력 검증 방법 마스터 (ykdojo Tip #28)

AI가 생성한 코드나 정보를 맹목적으로 신뢰해서는 안 됩니다. 여러 검증 방법을 사용하세요.

방법 1: 테스트 코드 작성

```
> "이 함수에 대한 테스트를 작성해줘.
   엣지 케이스도 포함해야 해."
```

방법 2: GitHub Desktop으로 시각적 검토

변경된 파일들을 GitHub Desktop에서 열어 diff를 시각적으로 확인합니다.

방법 3: Draft PR 생성

```
> "draft PR을 만들어줘"
```

PR 화면에서 변경 사항을 검토하고, 문제가 있으면 수정 요청합니다.

방법 4: Claude에게 자기 검증 요청

```
> "방금 생성한 코드를 다시 검토해줘.
   모든 주장을 검증하고, 끝에 검증 결과를 표로 정리해줘."

# Claude가 자기 검증:
| 주장 | 검증 방법 | 결과 |
|------|-----------|------|
| "이 함수는 O(n) 시간 복잡도" | 코드 분석 | ✅ 확인됨 |
| "모든 엣지 케이스 처리" | 테스트 코드 검토 | ❌ null 케이스 누락 |
```

ykdojo의 조언:

> *"제가 가장 좋아하는 프롬프트는 '모든 것을 다시 확인하고, 검증 가능한 것과 불가능한 것을 표로 만들어줘'입니다."*

10.7. DevOps 엔지니어로 활용 (ykdojo Tip #29)

Claude Code는 복잡한 DevOps 작업을 자동화하는 데 탁월합니다.

GitHub Actions 디버깅 자동화:

ykdojo는 `/gha` 명령어를 만들어 GitHub Actions 실패를 자동으로 조사합니다.

```
> /dx:gha https://github.com/user/repo/actions/runs/12345

# Claude가:
# 1. gh CLI로 워크플로우 로그 가져오기
# 2. 실패한 단계 식별
# 3. 에러 메시지 분석
# 4. 최근 커밋과 연관성 확인
# 5. Flaky test인지 판단
# 6. 수정 방법 제안
# 7. Draft PR 생성
```

Docker 이미지 최적화:

```
> "이 Dockerfile을 분석하고 이미지 크기를 줄일 방법을 제안해줘"

# Claude가:
# - 멀티 스테이지 빌드 제안
# - 불필요한 패키지 제거
# - 레이어 캐싱 최적화
# - .dockerignore 파일 생성
```

10.8. 범용 인터페이스로 활용 (ykdojo Tip #31)

ykdojo는 Claude Code를 **"컴퓨터에 대한 범용 인터페이스"**라고 부릅니다.

비디오 편집:

```
> "이 비디오 파일의 처음 30초를 잘라내고,
   1080p로 리사이즈한 후 output.mp4로 저장해줘"

# Claude가 ffmpeg 사용하여 처리
```

오디오 전사:

```
> "이 폴더의 모든 mp3 파일을 전사하고,
   각각 텍스트 파일로 저장해줘"

# Claude가 Whisper 사용하여 일괄 처리
```

디스크 공간 정리:

```
> "디스크 공간이 부족해. 어떤 파일들이 많은 공간을 차지하는지
   분석하고 정리 방법을 제안해줘"

# Claude가:
# 1. du -sh * | sort -h 실행
# 2. 큰 파일/폴더 식별
# 3. 제안: "Final Cut Pro 캐시 파일 10GB 삭제 가능"
# 4. 제안: "Docker 이미지 5GB 정리 가능 (docker system prune)"
```

ykdojo의 철학:

> *"컴퓨터에서 하고 싶은 일이 있으면, 일단 Claude Code에게 물어보세요. 놀랍도록 많은 것을 할 수 있습니다."*

10.9. 테스트 많이 작성 (TDD) (ykdojo Tip #34)

AI가 생성한 코드는 빠르지만, 실수도 빠르게 만들어집니다. 테스트는 안전망입니다.

TDD 워크플로우:

```
> "TDD 방식으로 사용자 등록 기능을 구현해줘"

# Claude가:
# 1. 실패하는 테스트 작성:
#    - 유효한 입력으로 등록 성공
#    - 중복 이메일 거부
#    - 약한 비밀번호 거부
# 2. 테스트 실행 → 모두 실패 확인
# 3. git commit -m "Add failing tests for user registration"
# 4. 최소한의 구현 코드 작성
# 5. 테스트 실행 → 통과 확인
# 6. git commit -m "Implement user registration"
```

ykdojo의 조언:

> *"어떤 사람들은 AI가 자신의 작업을 테스트할 수 없다고 말하지만, 실제로는 가능합니다. 테스트를 작성할 때는 다른 방식으로 문제를 생각하기 때문입니다."*

10.10. 복잡한 코드 단순화 (ykdojo Tip #40)

Claude는 때때로 과도하게 복잡한 코드를 작성합니다.

단순화 요청:

```
> "이 코드가 너무 복잡해. 왜 이렇게 작성했는지 설명하고,
   더 간단하게 바꿀 수 있는지 확인해줘"

# Claude가:
# 1. 각 부분의 이유 설명
# 2. 불필요한 추상화 식별
# 3. 단순화된 버전 제시
# 4. 두 버전의 장단점 비교
```

ykdojo의 조언:

> *"AI를 통해서만 코드를 작성하면 이해할 수 없다는 말은 틀렸습니다. 충분히 질문하면 오히려 더 빠르게 이해할 수 있습니다."*

Part 11: 고급 패턴과 철학

기술을 넘어, 에이전틱 개발자로서의 사고방식과 철학을 다룹니다.

11.1. 계획과 빠른 프로토타이핑의 균형 (ykdojo Tip #39)

계획의 중요성:

```
> "먼저 전체 계획을 세워줘. 어떤 기술을 사용할지,
   파일 구조는 어떻게 할지, 각 기능은 어디에 위치할지 결정해줘."
```

프로토타이핑의 가치:

```
> "두 가지 접근 방식을 빠르게 프로토타입으로 만들어서
   어느 것이 더 나은지 비교해줘"
```

ykdojo의 실제 경험:

diff viewer를 만들 때, 여러 접근 방식을 시도했습니다:

1. bash + tmux + lazygit
2. Ink + Node.js 커스텀 뷰어

결과적으로 어느 것도 발표하지 않았지만, 이 과정에서 **계획의 중요성**을 다시 깨달았습니다.

11.2. 개인화된 소프트웨어 시대 (ykdojo Tip #37)

우리는 **개인화된 맞춤형 소프트웨어의 시대**에 살고 있습니다.

ykdojo의 사례:

- 음성 전사 시스템: 자신의 워크플로우에 완벽히 맞춤
- Claude Code 커스터마이징: 시스템 프롬프트 패치
- Slack MCP CLI 버전: Docker-in-Docker 문제 해결을 위해 직접 제작

여러분도 할 수 있습니다:

```
> "내가 매일 사용하는 워크플로우를 자동화하는 CLI 도구를 만들어줘"
```

ykdojo의 조언:

> *"원하는 것이 있으면 Claude Code에게 만들어달라고 하세요. 충분히 작은 프로젝트라면 1-2시간 안에 완성할 수 있습니다."*

11.3. 사용이 최고의 학습 (ykdojo Tip #22)

10억 토큰 규칙:

ykdojo는 "10,000시간 규칙" 대신 **"10억 토큰 규칙"**을 제안합니다.

> *"AI를 진정으로 이해하고 직관을 얻고 싶다면, 많은 토큰을 소비하세요. 요즘은 가능합니다. Opus 4.5는 강력하면서도 저렴하여, 여러 세션을 동시에 실행할 수 있습니다."*

세계적인 암벽 등반가의 조언:

> *"암벽 등반을 잘하려면 어떻게 해야 하나요?" "암벽 등반을 하세요."*

이것이 Claude Code에도 적용됩니다. 사용하는 것이 최고의 학습입니다.

11.4. 지식 공유 및 기여 (ykdojo Tip #42)

지식 공유의 선순환:

ykdojo는 자신의 팁을 공유하면서 더 많은 것을 배웠습니다.

- 시스템 프롬프트 슬림화 팁 공유 → `--system-prompt` 플래그 존재를 알게 됨
- Skills vs Slash Commands 설명 공유 → Reddit 댓글에서 새로운 활용법 배움

Anthropic에 기여:

ykdojo는 여러 기능 요청과 버그 리포트를 제출했고, 많은 것이 실제로 반영되었습니다:

- `/permissions` 명령어에 검색 기능 추가
- 스크롤 위치 리셋 버그 수정

ykdojo의 조언:

> *"지식 공유는 일방통행이 아닙니다. 브랜드 구축이나 학습 고착화를 넘어, 공유 과정에서 새로운 것을 배웁니다."*

11.5. 계속 학습하기 (ykdojo Tip #43)

학습 방법:

1. **Claude Code에게 직접 물어보기**: Claude는 자신의 기능에 대한 전문 서브에이전트를 가지고 있습니다.

   ```
   > "Claude Code의 Hooks 기능에 대해 자세히 설명해줘"
   ```

2. **릴리스 노트 확인**: `/release-notes` 로 최신 기능 확인
3. **커뮤니티 학습**: r/ClaudeAI 서브레딧에서 다른 사용자의 워크플로우 배우기
4. **Ado 팔로우**: Ado(@adocomplete)의 "Advent of Claude" 시리즈 구독
   - Twitter/X: 매일 팁 공유
   - LinkedIn: 심화 내용

Part 12: 고급 기능과 SDK

최신 기능과 개발자를 위한 SDK를 다룹니다.

12.1. Extended Thinking (Ado Tip #19)

Ultrathink 키워드:

```
> "ultrathink: 이 아키텍처 결정의 장단점을 깊이 분석해줘"
```

`ultrathink` 키워드를 포함하면, Claude는 응답하기 전에 최대 32k 토큰을 내부 추론에 할당합니다. 복잡한 아키텍처 결정이나 까다로운 디버깅에 유용합니다.

MAX_THINKING_TOKENS 설정:

```
{
  "thinking": {
    "maxTokens": 5000
  }
}
```

이 설정이 있으면 `ultrathink` 키워드보다 우선합니다.

API에서 Extended Thinking 사용:

```
thinking: { type: "enabled", budget_tokens: 5000 }
```

Claude는 `thinking` 블록에 단계별 추론을 표시합니다.

12.2. LSP 통합 (Ado Tip #31)

Language Server Protocol (LSP)는 Claude에게 IDE 수준의 코드 인텔리전스를 제공합니다.

LSP가 제공하는 것:

- **즉시 진단**: 각 편집 후 에러와 경고를 즉시 확인
- **코드 탐색**: 정의로 이동, 참조 찾기, 호버 정보
- **언어 인식**: 타입 정보 및 문서

실전 효과:

```
> "이 함수를 리팩토링해줘"

# LSP 없이: Claude가 코드를 수정하고, 실행해봐야 에러 발견
# LSP 있음: Claude가 수정하는 즉시 타입 에러 감지 및 수정
```

12.3. Claude Agent SDK (Ado Tip #31)

Claude Code의 에이전트 루프, 도구, 컨텍스트 관리를 **SDK로 사용**할 수 있습니다.

10줄로 에이전트 만들기:

```
import { query } from '@anthropic-ai/claude-agent-sdk';

for await (const msg of query({
  prompt: "Generate markdown API docs for all public functions in src/",
  options: {
    allowedTools: ["Read", "Write", "Glob"],
    permissionMode: "acceptEdits"
  }
})) {
  if (msg.type === 'result') console.log("Docs generated:", msg.result);
}
```

활용 사례:

- 커스텀 CI/CD 파이프라인에 AI 에이전트 통합
- 사내 도구에 Claude Code 기능 임베딩
- 특수한 워크플로우를 위한 전용 에이전트 개발

12.4. 팀 설정과 공유 워크플로우 (Ado Tip #30)

팀 설정 공유:

`.claude/team-settings.json`:

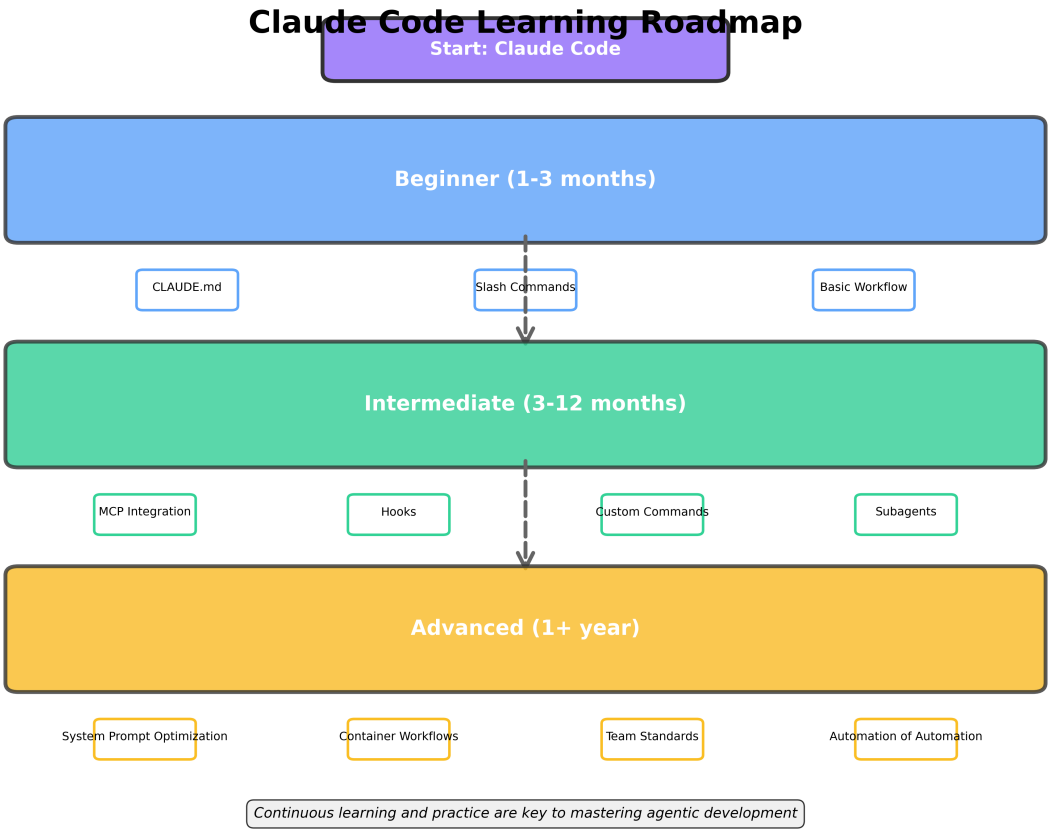
```
{
  "permissions": {
    "allow": ["Read(src/)", "Write(src/)", "Bash(npm test)"]
  },
  "hooks": {
    "PreToolUse": {
      "command": "bash",
      "args": ["-c", "echo 'Team hook: validating...'"]
    }
  },
  "mcpServers": {
    "company-db": {
      "command": "npx",
      "args": ["@company/db-mcp"]
    }
  }
}
```

공유 워크플로우:

팀 저장소에 `.claude/` 폴더를 커밋하여 모든 팀원이 동일한 설정을 사용하도록 합니다.

Part 13: 학습 로드맵과 다음 단계

Claude Code 마스터리는 하루아침에 이루어지지 않습니다. 체계적인 학습 로드맵을 따라 단계적으로 성장하세요.



초급부터 고급까지, 단계별로 Claude Code 전문가로 성장하는 로드맵

13.1. 초급자를 위한 로드맵 (1-3개월)

목표: Claude Code를 기본적인 개발 워크플로우에 통합하고, AI와 협업하는 습관 들이기

필수 학습 항목:

주차	학습 내용	실습 과제
1주	설치 및 기본 명령어 (`/usage` , `/clear` , `/stats`)	간단한 함수 작성 및 테스트
2주	CLAUDE.md 작성법, `/init` 활용	현재 프로젝트에 CLAUDE.md 추가
3-4주	컨텍스트 관리 기초, 단일 목적 대화	3개의 독립적인 작업을 별도 세션에서 수행
5-6주	Git 통합 (자동 커밋, PR 생성)	Draft PR 5개 이상 생성
7-8주	터미널 별칭 설정, 키보드 단축키	자신만의 별칭 3개 이상 만들기
9-12주	음성 코딩 시도, 커스텀 명령어 1개 만들기	반복 작업을 자동화하는 명령어 제작

성공 지표:

- [ ] 모든 새 프로젝트에 `/init` 으로 CLAUDE.md 생성
- [ ] 일주일에 최소 3회 Claude Code 사용
- [ ] 간단한 버그 수정을 Claude에게 맡길 수 있음
- [ ] 컨텍스트 오염을 인식하고 `/clear` 사용

추천 자료:

- Anthropic 공식 문서: https://code.claude.com/docs
- Ado의 Advent of Claude 시리즈 (Day 1-10)

13.2. 중급자를 위한 로드맵 (3-12개월)

목표: 생산성을 눈에 띄게 향상시키고, 복잡한 문제를 해결하는 데 Claude를 주도적으로 활용

필수 학습 항목:

월	학습 내용	실습 과제
3-4월	MCP 서버 1개 이상 연동 (Playwright 또는 Supabase)	MCP를 활용한 자동화 작업 1건
5-6월	Hooks 설정, 위험한 명령어 차단	팀 규칙을 강제하는 Hook 3개 작성
7-8월	Skills 및 Slash Commands 제작	자주 사용하는 워크플로우를 명령어로
9-10월	컨테이너 워크플로우, YOLO 모드 안전 사용	Docker 컨테이너에서 실험 프로젝트 진행
11-12월	Subagents 활용, 병렬 작업 위임	복잡한 작업을 서브에이전트에게 분할

성공 지표:

- [ ] 생산성이 50% 이상 향상됨 (자가 평가)
- [ ] MCP 서버를 실무에 활용 중
- [ ] 자신만의 커스텀 명령어 5개 이상 보유
- [ ] 팀원에게 Claude Code 팁을 공유할 수 있음
- [ ] 대화 복제 및 HANDOFF.md를 자연스럽게 사용

추천 자료:

- ykdojo GitHub 저장소: https://github.com/ykdojo/claude-code-tips
- Ado의 Advent of Claude 시리즈 (Day 11-25)
- r/ClaudeAI 서브레딧의 고급 팁

13.3. 고급자를 위한 로드맵 (1년 이상)

목표: Claude Code를 확장하고, 팀과 조직의 개발 문화를 선도하는 시스템 지휘자가 되기

필수 학습 항목:

분기	학습 내용	실습 과제
Q1	시스템 프롬프트 분석 및 패치 적용	ykdojo 패치 적용 및 효과 측정
Q2	맞춤형 MCP 서버 개발	회사 내부 API를 위한 MCP 서버 제작
Q3	멀티 에이전트 오케스트레이션	워커 Claude를 컨테이너에서 조율
Q4	Claude Agent SDK 활용	CI/CD 파이프라인에 AI 에이전트 통합

성공 지표:

- [ ] 시스템 프롬프트를 안전하게 수정할 수 있음

- [ ] 맞춤형 MCP 서버 또는 플러그인 개발 경험
- [ ] 팀 전체의 Claude Code 도입을 주도
- [ ] 컨퍼런스 또는 블로그에서 경험 공유
- [ ] Anthropic에 기능 요청 또는 버그 리포트 기여

추천 자료:

- Claude Agent SDK 문서
- Anthropic Engineering 블로그
- ykdojo의 고급 팁 (Tip #15, #21, #41)
- MCP 서버 개발 가이드

13.4. 필독 참고 자료

공식 문서 및 가이드:

1. **Claude Code 공식 문서**: https://code.claude.com/docs/en/overview
   - 설치, 기본 사용법, 모든 명령어 레퍼런스
2. **Anthropic Engineering 블로그**: https://www.anthropic.com/engineering/claude-code-best-practices
   - 베스트 프랙티스, 아키텍처 설명, 최신 기능 소개
3. **Claude Code GitHub**: https://github.com/anthropics/claude-code
   - 이슈 트래커, 기능 요청, 커뮤니티 논의

커뮤니티 자료:

1. **ykdojo의 claude-code-tips**: https://github.com/ykdojo/claude-code-tips
   - 43가지 파워 팁, 커스텀 스크립트, dx 플러그인
2. **Ado의 Advent of Claude 2025**: https://adocomplete.com/advent-of-claude-2025/
   - 31일간의 팁 시리즈, 초급부터 고급까지
3. **r/ClaudeAI 서브레딧**: https://www.reddit.com/r/ClaudeAI/
   - 커뮤니티 질문, 워크플로우 공유, 최신 소식

실전 사례 및 경험담:

1. **Jacob's Tech Tavern**: https://blog.jacobstechtavern.com/p/claude-code-productivity
   - "Claude Code가 나를 50-100% 더 생산적으로 만들었다"
2. **The Pragmatic Engineer**: https://newsletter.pragmaticengineer.com/p/how-claude-code-is-built
   - Claude Code가 어떻게 만들어졌는지 내부 구조 분석
3. **Lenny's Newsletter**: https://www.lennysnewsletter.com/p/everyone-should-be-using-claude-code
   - 비개발자도 Claude Code를 사용해야 하는 이유

비디오 및 강의:

1. **ykdojo의 Claude Code Masterclass**: 31개월간의 에이전틱 코딩 경험 공유

맺음말: AI는 부조종사, 주인공은 당신입니다

우리는 이 책을 통해 Claude Code라는 강력한 AI 에이전트와 함께 일하는 **70가지가 넘는 방법**을 살펴보았습니다. 해커톤 우승자 ykdojo의 생생한 경험과, Anthropic DevRel Ado Kukic의 체계적인 가이드를 통해, 여러분은 이제 에이전틱 개발자로서의 첫걸음을 뗄 준비가 되었습니다.

핵심을 되새기며:

1. **큰 문제를 작은 단위로 분해하라**: AI는 명확한 지시를 받았을 때 가장 뛰어난 성능을 발휘합니다.
2. **컨텍스트는 우유와 같다**: 신선하고 압축된 상태를 유지하세요. 오래된 컨텍스트는 성능을 저하시킵니다.
3. **올바른 추상화 수준을 선택하라**: 때로는 Vibe Coding으로, 때로는 Deep Dive로 상황에 맞게 조절하세요.
4. **자동화의 자동화**: 같은 작업을 3번 이상 반복한다면, 자동화할 방법을 찾으세요.
5. **사용이 최고의 학습**: 10억 토큰을 소비하며 직관을 키우세요.

ykdojo의 마지막 조언:

> *"AI 사용법을 배우는 가장 좋은 방법은 AI를 사용하는 것입니다. 세계적인 암벽 등반가가 '암벽 등반을 잘하려면 암벽 등반을 하세요'라고 말했듯이, Claude Code를 마스터하려면 Claude Code를 사용하세요. 실수를 두려워하지 마세요. 컨테이너 안에서 실험하고, 실패하고, 배우세요."*

Ado의 마지막 메시지:

> *"이 31일간의 여정을 돌아보니, 단순한 팁 공유를 넘어 인간-AI 협업의 철학을 나눴다는 것을 깨달았습니다. Claude Code의 최고 기능들은 여러분에게 제어권을 줍니다. Plan 모드, Agent Skills, Hooks, Sandbox 경계, 세션 관리… 이것들은 AI와 함께 일하는 도구이지, AI에게 항복하는 것이 아닙니다."*

개발의 새로운 시대:

우리는 코드를 '작성'하는 시대를 넘어, 시스템을 '설계'하고 AI를 '지휘'하는 시대로 나아가고 있습니다. 이제 개발자는 단순히 키보드 앞에 앉아 코드를 타이핑하는 사람이 아닙니다. 우리는 **에이전틱 개발자**입니다. 여러 AI 에이전트를 조율하고, 복잡한 시스템을 설계하며, 높은 수준의 추상화에서 문제를 해결하는 **시스템 지휘자**입니다.

하지만 잊지 마세요. **AI는 부조종사이고, 주인공은 여러분입니다.** Claude Code는 여러분의 생각을 증폭시키고, 반복 작업을 자동화하며, 미지의 영역을 탐험하는 데 도움을 주는 강력한 도구입니다. 하지만 최종 결정, 창의적인 방향, 그리고 책임은 여전히 여러분의 몫입니다.

이 책이 그 위대한 여정의 든든한 첫걸음이 되었기를 바랍니다. 이제 터미널을 열고, `claude`를 입력하고, 여러분만의 에이전틱 개발 이야기를 시작하세요.

Happy Agentic Coding! 🚀

참고 자료 (References)

[1] Anthropic. (2025). *Claude Code: Best practices for agentic coding*. https://www.anthropic.com/engineering/claude-code-best-practices

[2] @affaanmustafa. (2025). *The Shorthand Guide to Everything Claude Code*.

[3] Bartlett, J. (2025). *Claude Code has made me 50-100% more productive*. Jacob's Tech Tavern. https://blog.jacobstechtavern.com/p/claude-code-productivity

[4] Claude Code Docs. (2025). *Installation*. https://code.claude.com/docs/en/getting-started/installation

[5] Anthropic. (2025). *Claude Code: A new way to build software*. https://www.anthropic.com/news/claude-code

[6] @daxesh_iroid. (2025). *Tweet on building a Cal.ai app in 30 minutes*. X.

[7] Orosz, G. (2025). *How Claude Code is built*. The Pragmatic Engineer. https://newsletter.pragmaticengineer.com/p/how-claude-code-is-built

[8] Lenny's Newsletter. (2025). *Everyone should be using Claude Code more*. https://www.lennysnewsletter.com/p/everyone-should-be-using-claude-code

[9] ykdojo. (2025). *claude-code-tips: 40+ tips for getting the most out of Claude Code*. GitHub. https://github.com/ykdojo/claude-code-tips

[10] Kukic, A. (2026). *Advent of Claude: 31 Days of Claude Code*. adocomplete.com. https://adocomplete.com/advent-of-claude-2025/

[11] Medium. (2025). *20 Real Use Cases That Prove Claude Code Is a Game Changer*. https://medium.com/@agencyai/20-real-use-cases-that-prove-claude-code-is-a-game-changer-46ceefaf19ed

[12] Reddit. (2025). *The Complete Guide to Claude Code v2: CLAUDE.md, MCP, and Advanced Features*. r/ClaudeAI. https://www.reddit.com/r/ClaudeAI/comments/1qcwckg/the_complete_guide_to_claude_code_v2_claudemd_mcp/

부록: 빠른 참조 가이드

필수 명령어 치트시트

명령어	설명
`/init`	프로젝트 CLAUDE.md 자동 생성
`/usage`	토큰 사용량 및 한도 확인
`/context`	컨텍스트 윈도우 사용 현황
`/clear`	대화 내용 지우기
`/stats`	사용 통계 및 활동 그래프
`/clone`	대화 복제
`/half-clone`	대화 반복제 (컨텍스트 절반 줄이기)
`/export`	대화 내역 마크다운으로 내보내기
`/sandbox`	권한 경계 설정
`/mcp`	MCP 서버 관리
`/permissions`	승인된 명령어 관리
`/vim`	Vim 모드 활성화
`/release-notes`	최신 릴리스 노트 확인

키보드 단축키 치트시트

단축키	기능
`!command`	Bash 명령어 즉시 실행
`Esc Esc`	대화/코드 되감기
`Ctrl+R`	역방향 검색 (명령어 히스토리)
`Ctrl+S`	프롬프트 임시 저장
`Shift+Tab (×2)`	Plan 모드 토글
`Alt+P` / `Option+P`	모델 전환
`Ctrl+O`	Verbose 모드 토글
`Tab` / `Enter`	프롬프트 제안 수락
`Ctrl+B`	백그라운드로 보내기
`Ctrl+G`	외부 에디터로 편집

CLI 플래그 치트시트

플래그	설명
`-p "prompt"`	Headless 모드 (비대화형)
`--continue`	마지막 세션 이어가기
`--resume`	세션 목록에서 선택
`--resume name`	이름으로 세션 재개
`--teleport id`	웹 세션 로컬로 가져오기
`--chrome`	Chrome 통합 모드
`--dangerously-skip-permissions`	YOLO 모드 (위험)

이 책의 모든 내용은 2025년 1월 기준이며, Claude Code는 지속적으로 업데이트됩니다. 최신 정보는 공식 문서를 참조하세요.