개발팀 가이드

2026

Claude Code 완벽 가이드

팀을 위한 AI 개발 도구 마스터하기

왜 Claude Code인가? 어떻게 사용하는가? 팀으로 어떻게 협업하는가?

대상
개발팀 전체

발표 시간
60-75분

슬라이드
71장

목차

기초부터 고급 기능까지 - Claude Code를 마스터하고 팀으로 효과적으로 협업하는 모든 것

오늘의 목표

이 발표가 끝나면...

01 Claude Code가 왜 다른 AI 도구와 다른지 이해

02 기본 사용법부터 고급 기능까지 숙지

03 팀으로서 일관되게 사용하는 방법 습득

04 내일부터 바로 적용 가능한 실전 스킬 확보

AI 코딩 도구의 현재

2025년 AI 코딩 도구 시장

GitHub Copilot

55% 생산성 향상 주장

Cursor

"Copilot보다 2배 빠르다" 주장

Claude Code

"Agentic Coding의 새로운 패러다임"

어떤 걸 선택해야 할까?

각 도구의 철학과 접근 방식을 이해하면 답이 보입니다

왜 Claude Code인가?

IDE-First vs Agent-First 접근의 차이

두 가지 철학

IDE-First vs Agent-First

	IDE-First (Copilot, Cursor)	Agent-First (Claude Code)
접근법	라인별 자동완성	전체 작업 수행
컨텍스트	현재 파일 중심	전체 코드베이스
역할	코드 제안	작업 완료
예시	"이 줄 다음에 뭐?"	"이 기능 전체 구현해줘"

패러다임의 차이: 보조 도구 vs 동료 개발자

기능 비교 테이블

Claude Code vs GitHub Copilot vs Cursor

기능	Claude Code	Copilot	Cursor
자동완성	-	✓	✓
다중 파일 수정	✓✓✓	△	✓✓
터미널 통합	✓✓✓	-	△
에이전트	✓✓✓	△	✓
MCP 에코시스템	✓✓✓	-	-
무료 티어	✓	✓	✓

MCP 에코시스템이 Claude Code만의 핵심 차별점

차별점 1: Agentic Approach

코드 생성기가 아닌 '동료 개발자'

계획 → 실행 → 검증 → 반복

스스로 파일 탐색

테스트 실행 및 버그 수정

다단계 작업을 자율적으로 완료

예시

"인증 기능 추가해줘"

→ 파일 생성

→ 테스트 작성

→ PR 생성까지 자동 완료

차별점 2: 전체 코드베이스 이해

"이 프로젝트가 어떻게 동작하는지 알아"

한 번에 전체 코드베이스 분석

수천 개 파일도 자동으로 탐색

아키텍처, 의존성, 패턴 파악

프로젝트 구조를 즉시 이해

기존 코드 스타일 학습 및 적용

팀의 컨벤션을 자동으로 따름

온보딩 시간 단축

몇 주 → 1-2일

"신입 개발자가 첫날부터 생산적으로 코딩할 수 있습니다"

차별점 3: MCP 에코시스템

AI의 USB-C 포트

MCP란?

외부 도구와 표준 프로토콜로 연결하는 개방형 시스템.
GitHub, Jira, Confluence, DB 등에 직접 접근 가능

GitHub	Jira	Confluence
PostgreSQL	Playwright	+수천 개

커뮤니티에서 수천 개 MCP 서버 제공

실제 사용 예시

"Jira ENG-4521 이슈 기능 구현하고 GitHub PR 만들어"

실제 생산성 수치

숫자로 보는 효과

4-8개월 → 2주
Augment Code 고객 사례

2-10x
Altana - 개발 속도 향상

7시간 자율 코딩
Rakuten - 기존 몇 주 작업

10-15분 → 5분
Anthropic 보안팀 - 스택 트레이스 분석

핵심: 반복적인 작업에서 가장 큰 효과

주의사항

알아야 할 것들

METR 연구 결과

숙련 개발자 작업 시간 19% 증가 사례

GitClear 보고서

코드 중복 8배 증가 (2024년)

Harness 설문

67% 개발자가 AI 코드 디버깅에 더 많은 시간

도구를 이해하고 올바르게 사용해야 함

Claude Code 시작하기

설치부터 첫 번째 대화까지

설치

5분 안에 시작하기

Terminal

```
# 설치 (방법 1)
npm install -g @anthropic-ai/claude-code

# 설치 (방법 2)
curl -fsSL https://code.claude.com/install.sh | sh

# 시작
claude
```

Node.js 18+ 필요

또는 Homebrew: brew install claude

자동 업데이트

claude update 명령으로 최신 버전 유지

기본 CLI 명령어

핵심 CLI 명령어

명령어	설명
`claude`	대화 시작
`claude "질문"`	질문과 함께 시작
`claude -c`	이전 대화 재개
`claude -p "질문"`	답변 후 종료 (SDK 모드)
`claude update`	업데이트
`claude mcp list`	MCP 서버 목록

핵심 슬래시 커맨드

자주 쓰는 슬래시 커맨드

`/help`	도움말
`/cost`	비용 확인
`/model opus`	모델 변경
`/clear`	대화 초기화
`/compact`	대화 압축
`/context`	컨텍스트 사용량
`/config`	설정 열기
`/init`	CLAUDE.md 초기화

Tip: /help를 입력하면 모든 커맨드 목록을 볼 수 있습니다

키보드 단축키

생산성 10배 올리는 단축키

`Ctrl+C`
취소

`Ctrl+L`
화면 클리어

`Esc + Esc`
되돌리기

`Shift+Tab`
권한 모드 전환

`#` 시작
CLAUDE.md에 메모리 추가

`@`
파일 경로 자동완성

`!` 시작
Bash 모드

가장 많이 쓰는 단축키: Shift+Tab (권한 모드 전환)

권한 설정

안전하게 사용하기

권한 모드

`default`
모든 권한 확인

`acceptEdits`
파일 편집만 자동 승인

`plan`
미리보기 모드 (실행 안 함)

설정 예시

```
// settings.json
{
  "permissions": {
    "allow": ["Bash(npm:*)", "Bash(git:*)"],
    "deny": ["Bash(rm:*)", "Read(.env)"]
}}
```

IDE 통합

VS Code & JetBrains에서 사용

VS Code

마켓플레이스에서 "Claude Code" 설치

Cmd+Shift+E (Mac)

Ctrl+Shift+E (Win)

JetBrains

IDE Marketplace에서 설치

터미널 내 Claude Code 사용

Tip

터미널에서 직접 사용하는 것이 더 강력합니다 - IDE는 보조 도구로 활용

첫 번째 대화

실습: Hello Claude Code

Terminal

```
# 1. 시작
$ claude

# 2. 프로젝트 이해 요청
> "이 프로젝트 구조와 아키텍처 설명해줘"

# 3. 간단한 작업 요청
> "README.md에 설치 방법 추가해줘"

# 4. 비용 확인
> /cost
```

첫 대화에서는 프로젝트 이해부터 시작하세요

효과적인 프롬프트

좋은 프롬프트 vs 나쁜 프롬프트

나쁜 예 ❌

"왜 이 API가 이상해?"

"테스트 추가해"

좋은 예 ✓

"ExecutionFactory의 git 히스토리를 분석하고 API 설계 결정 과정 요약해줘"

"로그아웃 사용자 시나리오를 커버하는 테스트 케이스 작성해줘. 목(mock) 사용하지 말고"

프롬프트 작성 원칙

구체적인 맥락 + 원하는 결과 + 제약 조건 = 좋은 프롬프트

탐색-계획-코딩-커밋

권장 워크플로우

STEP 1
탐색
파일 읽기, 구조 파악

→

STEP 2
계획
"think hard" 사용

→

STEP 3
코딩
구현 실행

→

STEP 4
커밋
PR 생성

실제 예시

"먼저 이 모듈 구조를 분석해줘. 아직 코드는 수정하지 마"

→ 탐색 단계에서 코딩을 금지하여 정확한 이해 우선

핵심: 단계를 구분하여 Claude가 각 단계에 집중하도록

팀 협업 기능

CLAUDE.md와 설정 공유로 팀 전체가 같은 경험

CLAUDE.md - 팀의 두뇌

프로젝트 지식을 코드로 저장

프로젝트 개요, 기술 스택

코드 규칙, 테스트 요구사항

자주 쓰는 명령어

팀 워크플로우

Git에 커밋 → 팀 전체가 같은 컨텍스트

CLAUDE.md는 팀의 지식을 Claude에게 전달하는 가장 효과적인 방법입니다

CLAUDE.md 계층 구조

여러 레벨의 설정

```
~/.claude/CLAUDE.md          ← 개인 전역 설정
.claude/CLAUDE.md            ← 프로젝트 공유 설정 (Git)
.claude/CLAUDE.local.md      ← 개인 프로젝트 설정 (gitignored)
```

우선순위: 프로젝트 > 개인

높은 우선순위

프로젝트 설정

중간 우선순위

개인 로컬 설정

낮은 우선순위

전역 설정

팀 설정은 프로젝트에, 개인 선호는 로컬에 분리

CLAUDE.md 예시

실전 CLAUDE.md 템플릿

CLAUDE.md

```
# 프로젝트: MyApp

## 기술 스택
React 18, TypeScript
Node.js, Express
PostgreSQL

## 코드 규칙
2칸 들여쓰기, Prettier + ESLint, 테스트 필수

## 명령어
npm run dev # 개발 서버
npm test # 테스트
```

/init 명령으로 자동 생성 가능

경로별 규칙

.claude/rules/ 디렉토리

폴더 구조

```
.claude/rules/
├── general.md # 전체 적용
├── typescript.md # *.ts, *.tsx에만
└── api.md # src/api/**에만
```

typescript.md

```
---
paths: src/**/*.{ts,tsx}
---

any 타입 금지
모든 함수에 반환 타입 명시
```

파일별 맞춤 규칙

특정 파일에만 적용되는 규칙 설정

팀 표준 강제

YAML frontmatter로 경로 패턴 지정

프론트엔드, 백엔드, API 등 영역별 다른 규칙 적용 가능

설정 파일 구조

.claude 폴더 완전 분석

.claude/ 디렉토리

```
.claude/
├── settings.json # 프로젝트 설정
├── settings.local.json # 개인 설정
├── CLAUDE.md # 팀 메모리
├── agents/ # 커스텀 에이전트
├── commands/ # 슬래시 커맨드
├── skills/ # 스킬
├── rules/ # 경로별 규칙
└── .mcp.json # MCP 서버 설정
```

Git에 커밋

settings.json, CLAUDE.md, rules/

Git에서 제외

settings.local.json, CLAUDE.local.md

확장 기능

agents/, commands/, skills/

팀 설정과 개인 설정을 명확히 분리

settings.json 핵심 설정

팀 전체 설정 관리

settings.json

```
{
  "model": "claude-sonnet-4-5-20250929",
  "permissions": {
    "allow": ["Bash(npm:*)", "Bash(git:*)"],
    "deny": ["Bash(curl:*)", "Read(.env*)"]
  },
  "hooks": { "PostToolUse": [...] },
  "enabledMcpjsonServers": ["github"]
}
```

팀원 모두가 같은 모델, 권한, MCP 서버를 사용

팀 온보딩 설정

신규 팀원을 위한 원클릭 설정

1. 저장소 클론 → 2. claude 실행 → 3. CLAUDE.md 자동 로드 → 완료!

MCP 서버 자동 설정

.mcp.json에서 자동으로 연결

권한 규칙 적용

settings.json에서 자동으로 적용

결과: 모든 팀원이 동일한 Claude 경험

신규 팀원도 첫날부터 팀의 베스트 프랙티스를 적용받습니다

Git 통합

Claude Code + Git 워크플로우

커밋 생성

> "변경사항 커밋해줘"

→ 자동으로 적절한 커밋 메시지 생성

PR 생성

> "GitHub에 PR 만들어줘"

→ 변경 내용 요약하여 PR 설명 작성

이슈 연동

> "JIRA ENG-4521 이슈 해결하고 PR 만들어"

→ 이슈 내용 파악 → 구현 → 테스트 → PR 생성까지 자동

컨벤션 자동 적용

브랜치 관리 자동화

리뷰어 자동 지정

팀 일관성 체크리스트

우리 팀 설정 체크리스트

- ☐ CLAUDE.md 작성 및 Git 커밋
- ☐ .claude/rules/ 규칙 정의
- ☐ settings.json 권한 설정
- ☐ MCP 서버 설정 공유
- ☐ 팀 온보딩 문서 작성
- ☐ 커스텀 커맨드 정의

6개 항목 완료 → 팀 Claude Code 환경 완성

Skills & Commands

반복 작업을 자동화하고 팀과 공유

Skills 개념

반복 작업을 자동화하라

Skill = 특정 작업에 특화된 지침 세트

마크다운 파일로 정의

한 번 작성, 모든 팀원이 사용

Git에 커밋하여 영구 보존

트리거 키워드로 자동 활성화

특정 키워드 입력 시 자동 적용

도구 접근 권한 제어

필요한 도구만 허용

Skills = 팀의 베스트 프랙티스를 코드화

Skill 파일 구조

SKILL.md 작성법

폴더 구조

```
~/.claude/skills/explaining-code/
├── SKILL.md # 필수
├── reference.md # 선택
└── examples.md # 선택
```

SKILL.md

```
---
name: explaining-code
description: 코드를 다이어그램으로 설명
allowed-tools: Read, Grep, Glob
---

# 스킬 지침 내용...
```

YAML Frontmatter

메타데이터와 권한 설정

Markdown Body

상세 지침 및 예시

Skill 예시 - 코드 리뷰

실전 Skill: 코드 리뷰

code-review/SKILL.md

```
---
name: code-review
description: 팀 표준 코드 리뷰
---

# 코드 리뷰 프로세스
1. git diff로 변경 확인
2. 각 파일 검토
3. 우선순위별 피드백:
   🔴 Critical: 반드시 수정
   🟡 Warning: 수정 권장
```

팀의 코드 리뷰 표준을 Skill로 정의하여 일관성 유지

커스텀 슬래시 커맨드

팀 전용 명령어 만들기

폴더 구조

```
.claude/commands/
├── deploy.md # /deploy
├── security-check.md
└── feature.md
```

deploy.md

```
---
description: 스테이징 배포
allowed-tools: Bash(npm:*), Bash(git:*)
---

npm run build &&
npm run test &&
npm run deploy:staging
```

/deploy 입력 → 빌드, 테스트, 배포 자동 실행

Hooks 시스템

이벤트 기반 자동화

이벤트	시기	용도
PreToolUse	도구 실행 전	검증, 차단
PostToolUse	도구 실행 후	포맷팅, 로깅
SessionStart	세션 시작	환경 설정
UserPromptSubmit	프롬프트 제출	검증, 강화

Git hooks처럼 Claude의 동작을 커스터마이즈

Hook 예시 - 자동 포맷팅

파일 수정 후 자동 Prettier

settings.json

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [{
          "type": "command",
          "command": "npx prettier --write \"$file_path\""
      }]}]}}}
```

결과: 모든 파일 수정 후 자동으로 코드 포맷팅 적용

Skills 공유 워크플로우

팀 Skill 관리 프로세스

피드백 반영 → 개선 → 재커밋 → 반복

Skills = 팀의 집단 지성을 코드로 축적

PART 6

Agent & Subagent

자율적으로 작업하는 AI 에이전트

AGENT & SUBAGENT

Agent 개념

자율적으로 작업하는 AI

Agent = 여러 Skills를 조합한 워크플로우

복잡한 다단계 작업을 자동화

- 복잡한 다단계 작업 자동화
- 독립된 컨텍스트에서 실행
- 결과만 메인 대화에 반환
- 메인 컨텍스트 보존

내장 Subagent 타입

기본 제공 에이전트

타입	용도	도구
General	다단계 작업	모두
Explore	코드베이스 탐색	Read, Grep, Glob
Plan	계획 수립	Read, Grep, Glob

사용 예시

```
> "Explore 에이전트로 인증 관련 코드 찾아줘"
> "Plan 에이전트로 리팩토링 계획 세워줘"
```

💡 언제 사용하나요?

General: 복잡한 멀티스텝 작업, 코드 생성 및 수정

Explore: 코드베이스 이해, 파일 검색, 패턴 찾기

Plan: 구현 전 계획, 아키텍처 설계

Task Tool 활용

병렬 작업 실행

```
> "3개 파일을 병렬로 분석해줘"

Claude:
Task tool로 3개 Subagent 생성
→ 각각 독립 컨텍스트에서 실행
→ 결과 종합하여 반환
```

장점

62%
시간 단축

100%
컨텍스트 보존

독립적
작업 수행

병렬 실행 다이어그램

커스텀 Agent 생성

.claude/agents/ 폴더

```
# .claude/agents/code-reviewer.md
---
name: code-reviewer
description: 코드 리뷰 전문가
tools: Read, Grep, Glob, Bash
model: sonnet
---
당신은 시니어 코드 리뷰어입니다.
코드 품질, 보안, 성능을 검토합니다.
...
```

📝 YAML Frontmatter

name: 에이전트 이름

description: 설명

tools: 사용 가능 도구

model: 사용할 모델

💡 활용 팁

팀의 코딩 컨벤션, 리뷰 기준을 프롬프트에 포함하세요.

Git에 커밋하여 팀원과 공유!

AGENT & SUBAGENT

Agent 체이닝

에이전트 연결 사용

```
> "먼저 코드 분석 에이전트로 문제점 찾고,
그 다음 최적화 에이전트로 수정해줘"
```

🔗 체이닝 장점

- ✓ 복잡한 워크플로우 자동화
- ✓ 각 단계별 전문 에이전트 활용

Claude 실행 흐름

Step 1
code-analyzer 실행

→

Step 2
결과 → optimizer 전달

→

Step 3
최종 결과 반환

데이터가 에이전트 간 자동 전달

실전 예시 - PPT Agent

9개 Skill 조합 Agent

PPT Agent Pipeline

PART 7

MCP 통합

Model Context Protocol로 무한 확장

MCP 개념

Model Context Protocol

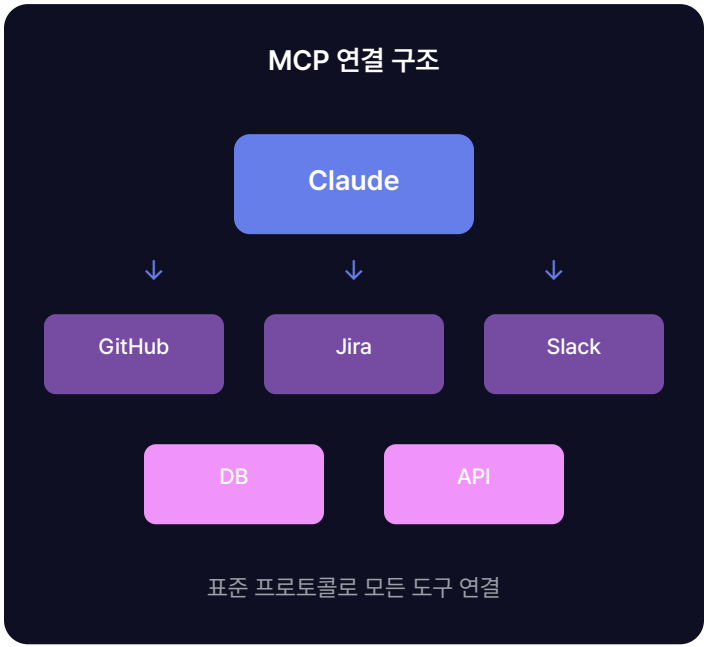
MCP = AI가 외부 도구와 소통하는 표준 프로토콜

USB-C처럼 한 번 연결하면 모든 기능 사용

2024.11
출시

수천 개
MCP 서버 구축

- ✓ 업계 표준으로 자리잡음
- ✓ 다양한 도구와 즉시 연동 가능



MCP 서버 설정

외부 도구 연결하기

```
# HTTP 서버 추가
claude mcp add --transport http github \
https://api.githubcopilot.com/mcp/

# 환경 변수와 함께
claude mcp add --transport http stripe \
--header "Authorization: Bearer $STRIPE_KEY" \
https://mcp.stripe.com/mcp

# 목록 확인
claude mcp list

# 서버 제거
claude mcp remove github
```

📋 주요 명령어

mcp add 서버 추가

mcp list 목록 확인

mcp remove 서버 제거

--transport 연결 방식

💡 Transport 타입

http: REST API 서버

stdio: 로컬 프로세스

sse: 스트리밍 연결

주요 MCP 서버

추천 MCP 서버

서버	용도
GitHub	이슈, PR, 코드 리뷰 자동화
Atlassian	Jira, Confluence 연동
Playwright	브라우저 자동화, E2E 테스트
Context7	최신 라이브러리 문서 조회
PostgreSQL / Stripe	DB 쿼리, 결제 API 연동

MCP 통합

Context7 활용

공식 문서 조회

```
> "React useEffect 공식 문서 찾아줘"

Claude: [Context7 MCP 사용]
공식 React 문서에서 useEffect 패턴 조회
최신 버전 기준 정확한 정보 제공
```

장점

📚 최신 문서

오래된 학습 데이터 대신 최신 공식 문서 사용

🎯 버전 정확성

버전별 정확한 API 정보 제공

💡 활용 예시

"Next.js 14 App Router 패턴"

"TypeScript 5.0 새로운 기능"

"Tailwind CSS 최신 유틸리티"

"Prisma ORM 마이그레이션"

Claude의 학습 데이터가 오래되었을 때 특히 유용합니다. 최신 라이브러리 버전의 정확한 API를 확인할 수 있습니다.

Playwright 활용

브라우저 자동화

```
> "로그인 플로우 E2E 테스트 만들어줘"

Claude: [Playwright MCP 사용]
1. 브라우저 열기
2. 로그인 페이지 접속
3. 폼 입력
4. 검증
5. 스크린샷 캡처
```

🎭 Playwright 강점

- ✓ 실제 브라우저 렌더링
- ✓ 크로스 브라우저 지원
- ✓ 모바일 에뮬레이션
- ✓ 네트워크 모킹
- ✓ 접근성 테스트

활용 사례

- ✓ E2E 테스트 자동 생성
- ✓ UI 버그 재현 및 수정
- ✓ 스크린샷 비교 테스트
- ✓ 웹 스크래핑

MCP 통합

엔터프라이즈 MCP 관리

조직 전체 MCP 제어

managed-mcp.json

```
{
"mcpServers": {
"github": { "url": "..." },
"jira": { "url": "..." }
}
}
```

🔒 사용자 수정 불가

🛡️ 중앙 관리 보안 정책

✅ 승인된 서버만 허용

🏢 엔터프라이즈 장점

일관된 환경

모든 팀원이 동일한 MCP 서버 설정 사용

보안 강화

승인되지 않은 외부 서버 접근 차단

중앙 관리

IT 팀에서 전사 정책 일괄 적용

PART 8

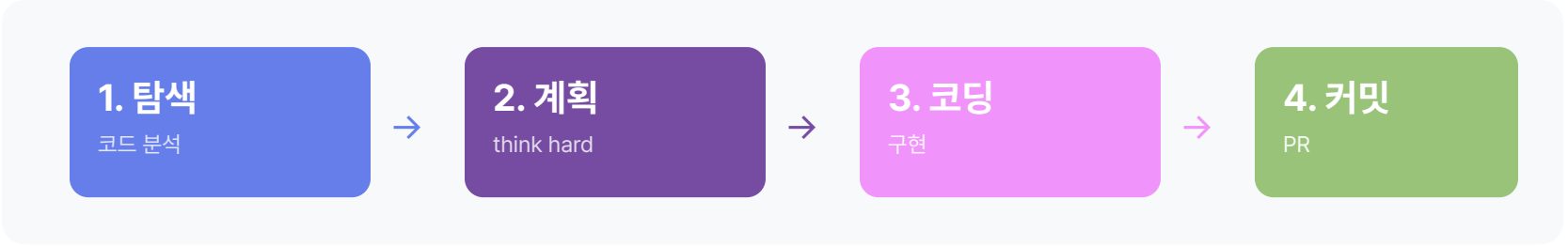
워크플로우 & Best Practices

효율적인 Claude Code 활용법

BEST PRACTICES

탐색-계획-코딩-커밋

권장 워크플로우 #1




```
1. 탐색
> "이 모듈 구조 분석해줘 (코딩 금지)"

2. 계획
> "think hard로 구현 계획 세워줘"

3-4. 코딩 & 커밋
> "계획대로 구현하고 PR 만들어줘"
```

💡 왜 이 순서인가?

탐색 없이 바로 코딩하면 Claude가 잘못된 가정으로 코드를 작성할 수 있습니다.

계획 단계에서 "think hard"를 사용하면 더 깊은 분석이 가능합니다.

BEST PRACTICES

TDD 워크플로우

권장 워크플로우 #2

1. 테스트 작성

> "이 기능의 테스트 먼저 작성해줘"

2. 실패 확인

> "테스트 실행해봐"

3. 구현

> "테스트 통과하도록 구현해줘"

4. 리팩토링

> "코드 정리하고 테스트 다시 확인해줘"

Claude가 테스트 주도로 안전하게 코드를 작성합니다

BEST PRACTICES

다중 Claude 인스턴스

권장 워크플로우 #3

Terminal 1

코드 작성

기능 구현에 집중

Terminal 2

코드 검증

린트, 타입 체크

Terminal 3

테스트 실행

유닛/통합 테스트

Git Worktree 활용

```
# 별도 작업 디렉토리 생성
git worktree add ../feature-auth feature/auth
cd ../feature-auth
claude
```

💡 장점

- ✓ 각 터미널이 독립적 컨텍스트
- ✓ 병렬 작업으로 생산성 향상
- ✓ 브랜치별 격리된 작업 환경

BEST PRACTICES

병렬 Subagent 전략

시간 단축의 비밀

🎯 Sweet Spot: 2-4개 병렬

최적의 성능과 비용 균형

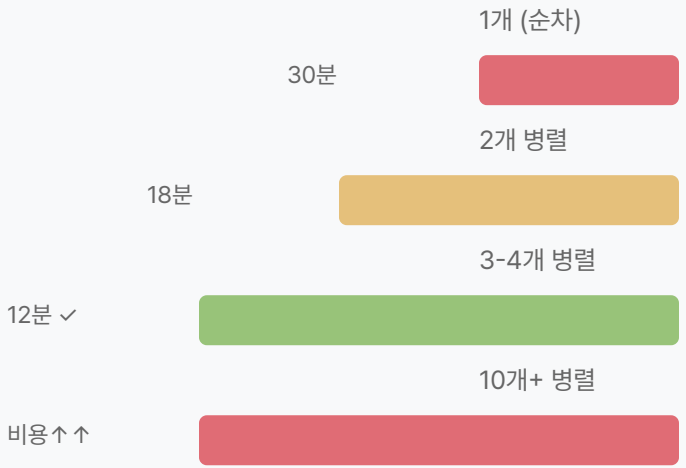
```
> "이 3개 파일을 병렬로 분석해줘"

시간 비교:
순차: 30분
병렬: 12분 (60% 단축) ✓
```

⚠️ 주의: 49개 병렬 → 비용 폭발 사례

너무 많은 병렬 처리는 비용 급증을 유발합니다

📊 병렬 처리 효율



BEST PRACTICES

Context 관리

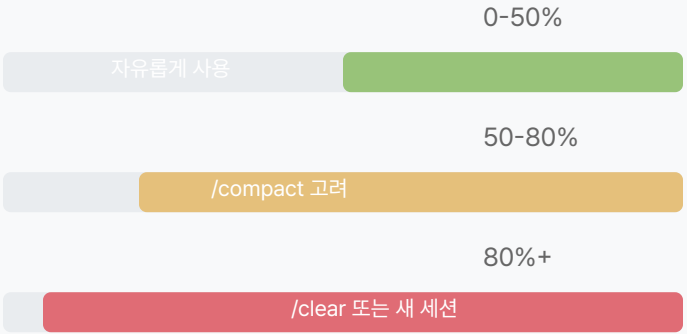
200K 토큰 활용법

```
/context        # 사용량 확인
/compact        # 컨텍스트 압축
/clear          # 컨텍스트 초기화
```

관리 전략

1. 정기적으로 /compact
2. 완료된 작업 후 /clear
3. 중요 정보는 CLAUDE.md에 저장
4. 대규모 작업은 Subagent 활용

📊 컨텍스트 사용량 예시



💡 Subagent는 별도 컨텍스트를 사용하므로 대용량 작업에 효과적입니다.

BEST PRACTICES

비용 최적화

효율적인 토큰 사용

🔀 모델 믹싱

Haiku
탐색 (저렴)

Sonnet
코딩 (균형)

Opus
복잡한 설계 (고성능)

📁 파일 통신

긴 데이터는 파일로 전달 → 컨텍스트 낭비 방지

💡 비용 절약 팁

✓ 탐색은 Haiku로

파일 탐색, 단순 질문 시 저렴한 모델 사용

✓ 정기적 /compact

컨텍스트가 커지면 비용도 증가

✓ Subagent 활용

대용량 작업은 분리하여 처리

✓ /cost로 모니터링

BEST PRACTICES

헤드리스 모드

CI/CD 통합

```
# 자동화 스크립트
claude -p "lint 오류 수정해줘" \
--output-format stream-json

# Pre-commit hook
claude -p "변경사항 검토해줘"
```

주요 옵션

`-p` 프롬프트 직접 전달

`--output-format` 출력 형식 (json, stream-json)

`--max-turns` 최대 대화 턴 수

`--allowedTools` 허용 도구 제한

🚀 활용 사례

CI/CD 파이프라인

PR 머지 전 자동 코드 리뷰

Pre-commit Hook

커밋 전 코드 검증

정기 작업

의존성 업데이트, 문서 생성

스크립트 자동화

BEST PRACTICES

피해야 할 것들

흔한 실수

❌ 탐색 없이 바로 코딩 요청

Claude가 잘못된 가정으로 코드를 작성할 수 있습니다

❌ 권한 스킵 남용

--dangerously-skip-permissions 무분별 사용

❌ 모호한 프롬프트

"테스트 추가해" → 어떤 테스트? 어디에?

❌ 컨텍스트 관리 미흡

메모리 넘칠 때까지 /compact 안함

❌ 과도한 CLAUDE.md

검증 없이 무분별하게 규칙을 추가

✓ 대신 이렇게!

탐색 → 계획 → 구현 워크플로우 준수

PART 9

팀 도입 가이드

4주 마스터 플랜

도입 로드맵

4주 마스터 플랜

Week 1

개인 숙달

- ✓ 설치 및 기본 사용
- ✓ 핵심 커맨드 습득
- ✓ 첫 프로젝트 사용

→

Week 2

팀 설정

- ✓ CLAUDE.md 공동 작성
- ✓ 권한 설정 표준화
- ✓ 팀 컨벤션 정립

→

Week 3

고급 기능

- ✓ Skills, Agents 구축
- ✓ MCP 연동
- ✓ 자동화 구축

→

Week 4

최적화

- ✓ 워크플로우 정립
- ✓ 비용 최적화
- ✓ 효과 측정

💡 **핵심 포인트**

- ✓ 점진적 도입: 개인 → 팀 → 조직 순서로 확장
- ✓ 정기 회고: 매주 효과적인 사용법 공유 및 개선점 논의

Day 1-3 체크리스트

첫 3일 완료 목표

Day 1

- ☐ Claude Code 설치
- ☐ 첫 대화 시작
- ☐ /help, /cost 사용해보기

목표: 기본 환경 구축

Day 2

- ☐ 프로젝트에서 사용
- ☐ 간단한 작업 요청
- ☐ 권한 설정 이해

목표: 실무 적용 경험

Day 3

- ☐ CLAUDE.md 작성
- ☐ 슬래시 커맨드 활용
- ☐ 팀원과 설정 공유

목표: 팀 협업 시작

💡 첫 프롬프트 추천

Day 1: "이 프로젝트 구조와 아키텍처 설명해줘"

Day 2: "README.md에 설치 방법 추가해줘"

성공 지표

무엇을 측정할 것인가

지표	측정 방법
⏱️ 작업 완료 시간	티켓 사이클 타임 비교 (Before/After)
✅ 코드 품질	PR 리뷰 코멘트 수, 버그 발생률
🆕 온보딩 시간	신규 팀원 첫 PR까지 소요 시간
😊 팀원 만족도	정기 설문조사 (NPS)
💰 비용 효율	월간 토큰 사용량 대비 생산성

PART 10

마무리

핵심 요약 & 다음 단계

마무리

5가지 핵심 테이크어웨이

오늘 기억할 것

1 **Agentic Coding**
자동완성이 아닌 자율 에이전트

2 **CLAUDE.md**
팀 지식을 코드로 자산화

3 **Skills & Agents**
반복 작업 자동화

4 **MCP**
외부 도구와 무한 연결

5 **탐색 → 계획 → 코딩 → 커밋**
워크플로우의 기본

Claude Code는 도구가 아니라 팀원입니다 🤝

마무리

내일 할 수 있는 3가지

바로 시작하기

1

오늘

Claude Code 설치하고 "이 프로젝트 설명해줘"

2

내일

CLAUDE.md 작성하고 Git 커밋

3

이번 주

첫 번째 커스텀 커맨드 만들기

Q&A

질문 & 답변

궁금한 점이 있으신가요?

📚 docs.claude.com | 🌐 mcpcat.io

💬 github.com/anthropics/claude-code

📝 anthropic.com/engineering/claude-code-best-practices