# Claude Code
# 핵심 가이드

설치부터 Agent Team까지, AI 코딩 도구의 모든 것

2026년 2월 | Opus 4.6 기준

프로젝트 브레인 • Plan Mode • Skill • MCP • Sub Agent • Agent Team

# 목차

PART 1

# Claude Code 시작하기

설치, 개발 환경 설정, 그리고 AI의 성능을 결정짓는 프로젝트 브레인까지

## Claude Code란?

Claude Code는 Anthropic이 만든 **에이전틱(Agentic) 코딩 도구**입니다. 터미널에서 동작하며 코드베이스를 읽고, 파일을 편집하고, 명령어를 실행하고, 개발 도구와 통합됩니다. 자연어로 기능 구현, 버그 수정, 테스트 작성을 요청할 수 있고, Git 커밋/브랜치 생성/PR 오픈까지 자동화할 수 있습니다.

터미널, IDE(VS Code, JetBrains), 데스크톱 앱, 웹 브라우저, 모바일(iOS) 등 다양한 환경에서 사용 가능합니다.

## 구독 플랜 및 설치

| 플랜 | 가격 | 주요 특징 |
|---|---|---|
| **Pro** | $20/월 | 모든 기능 사용 가능, 사용량 제한 있음 |
| **Max 5x** | $100/월 | Pro의 5배 사용량, Opus 4.6 접근, Agent Teams |
| **Max 20x** | $200/월 | Pro의 20배 사용량, 최대 우선순위 |
| **API** | 종량제 | Sonnet 4.5: 입력 $3/1M, 출력 $15/1M 토큰 |

### 설치 명령어

```
# macOS / Linux / WSL
curl -fsSL https://claude.ai/install.sh | bash

# Windows PowerShell
irm https://claude.ai/install.ps1 | iex

# Homebrew
brew install --cask claude-code
```

> **Tip:** 네이티브 설치는 Node.js가 필요 없으며, 자동 업데이트가 내장되어 있습니다. 설치 후 프로젝트 폴더에서 `claude` 명령어로 바로 시작할 수 있습니다.

## 개발 환경 (IDE) 설정

Claude Code는 터미널에서 직접 쓸 수도 있지만, **VS Code**에서 사용하는 것이 가장 편합니다. VS Code에 Anthropic 공식 Claude Code 확장 프로그램을 설치하면 됩니다.

> **주의:** 반드시 **Anthropic 공식** 확장 프로그램을 설치하세요. 비슷한 이름의 비공식 확장에는 악성 코드가 포함되어 있을 수 있습니다.

### IDE 화면 구성

| 영역 | 위치 | 역할 |
|---|---|---|
| 파일 탐색기 | 왼쪽 | 프로젝트의 모든 파일/폴더 탐색 |
| 코드 편집기 | 가운데 | 파일 내용 확인 및 직접 수정 |
| Claude Code 채팅 | 오른쪽/아래 | AI에게 명령 전달, 결과 확인 |

Claude Code는 현재 열려 있는 파일을 자동으로 인식합니다. 특정 파일에 대해 작업을 시키려면 해당 파일을 편집기에서 열어둔 상태로 대화하세요.

## 토큰과 컨텍스트 윈도우

### 토큰 (Token)

AI가 텍스트를 처리하는 기본 단위. 대략 한 단어가 한 토큰 정도입니다. 자동차에 비유하면 **연료**에 해당합니다.

### 컨텍스트 윈도우

AI가 한 번에 기억할 수 있는 대화의 양. Opus 4.6 기준 **200,000 토큰**(약 150,000 단어). 자동차의 **연료 탱크**에 해당합니다.

> **핵심:** 이 탱크를 얼마나 효율적으로 쓰느냐가 Claude Code 활용의 핵심입니다.

## 프로젝트 브레인 (CLAUDE.md)

`CLAUDE.md`는 프로젝트 폴더 최상위에 위치하는 파일로, Claude Code가 대화를 시작할 때 **가장 먼저 읽는 작업 지침서**입니다. 기술 스택, 코드 스타일, 프로젝트 구조 등의 정보를 담습니다.

> **비유:** CLAUDE.md는 **배의 방향타**입니다. 출발할 때 각도를 1도만 틀어도 수천 킬로미터를 항해하면 도착지가 완전히 달라집니다. AI와 대화를 시작하기 전에 이 파일이 먼저 주입되므로, 여기에 뭘 적느냐에 따라 전체 프로젝트의 방향이 결정됩니다.

### CLAUDE.md 생성 방법

1. `/init` 명령어로 자동 생성 (프로젝트 구조, 기술 스택, 아키텍처 자동 분석)
2. 직접 규칙 추가 (테스트 코드 작성, 함수 이름 규칙, 에러 처리 패턴 등)
3. **200~500줄** 사이로 유지 (너무 길면 AI가 중간 내용을 놓침)

## CLAUDE.md 작성 실전 팁

| # | 팁 | 이유 |
|---|---|---|
| 1 | 가장 중요한 규칙은 **파일 맨 위에** 배치 | AI 모델의 Primacy Bias - 시작/끝 부분을 가장 잘 기억 |
| 2 | API 문서 전체를 복사하지 말 것 | 토큰 낭비. 핵심 정보만 간결하게 |
| 3 | 같은 실수 2~3번 반복 시 규칙 추가 | 실제 문제가 생길 때 하나씩 추가하는 게 효과적 |
| 4 | 주기적으로 불필요한 규칙 정리 | CLAUDE.md도 기술 부채가 쌓일 수 있음 |

## CLAUDE.md 3가지 레이어

| 레이어 | 위치 | 적용 범위 |
|---|---|---|
| **글로벌** | `~/.claude/CLAUDE.md` | 모든 프로젝트 공통 (개인 스타일, 코딩 규칙) |
| **프로젝트** | 프로젝트 루트 `CLAUDE.md` | 해당 프로젝트만 적용 |
| **관리자** | 기업용 전용 | 조직 전체 정책 강제 |

세 레이어가 병합되어 최종 지침이 완성됩니다. 프로젝트별로 다른 설정이 필요하면 프로젝트 레이어에서 덮어씁니다.

## Rules 폴더와 Auto Memory

### Rules 폴더

`.claude/rules/` 폴더에 규칙을 주제별로 분리 저장할 수 있습니다. 설정 파일을 모듈화하는 것과 같은 개념입니다.

```
.claude/
  rules/
    workflow.md           # 워크플로우 규칙
    design-rules.md       # 디자인 규칙
    technical-defaults.md # 기술 기본 설정
```

### Auto Memory

`MEMORY.md` 파일에 중요한 정보를 자동 저장하는 기능입니다. 세션이 끝나도 다음 대화에서 이 정보를 기억합니다.

> **Part 1 핵심 요약:** Claude Code의 성능은 코드를 입력하기 전에 이미 결정됩니다. CLAUDE.md, Rules 폴더, 메모리를 제대로 설정하면 AI가 프로젝트를 정확히 이해한 상태에서 출발합니다. **설정에 5분 투자하면 이후 수십 시간을 아낄 수 있습니다.**

PART 2

# 실전 개발 워크플로우

검증 루프, Plan Mode, 컨텍스트 관리로 AI 코딩의 품질과 속도를 모두 잡는 법

## Task-Do-Verify 루프

AI 코딩의 핵심 워크플로우입니다. 대부분의 사람들이 AI 코딩에 실망하는 이유는 바로 **검증 단계를 빼먹기 때문**입니다.


### 디자인 작업

1. 원하는 디자인 스크린샷 전달
2. 결과물을 다시 스크린샷으로 찍기
3. 원본과 비교하여 수정 지시
4. 서너 번 반복으로 완성

### 코딩 작업

1. 테스트 코드 자동 작성/실행 지시
2. 테스트 통과 시 다음 단계
3. 실패 시 AI가 자동 수정
4. TDD(테스트 주도 개발) 방식

> **AI의 진짜 가치:** 완벽함이 아닌 **속도**에 있습니다. 80% 완성도에서 시작해서 빠르게 수렴하는 것이 핵심입니다.

## Context Rot 방지하기

AI 코딩에서 가장 흔한 실패 패턴: 처음에 잘 되다가 30분 후 갑자기 아무것도 동작하지 않는 현상.

> **Context Rot이란?** 오류가 누적되면서 AI가 자기가 이전에 뭘 했는지 혼란스러워하는 현상입니다. 매 단계마다 검증하는 습관이 이 문제를 근본적으로 방지합니다. **귀찮아도 매번 검증하는 습관이 결국 가장 빠른 길입니다.**

## 병렬 작업과 Hook 시스템

### 병렬 작업

IDE에서 탭을 여러 개 열어 각각 다른 작업을 시킬 수 있습니다. 추천 동시 실행 수는 **3~4개**입니다.

### Hook 시스템

Claude Code가 특정 동작을 할 때 자동으로 실행되는 사용자 정의 스크립트입니다.

| Hook 이벤트 | 발생 시점 | 활용 예시 |
| --- | --- | --- |
| `PreToolUse` | 도구 실행 전 | 보호 파일 편집 차단 |

| | | |
|---|---|---|
| `PostToolUse` | 도구 실행 후 | 자동 코드 포맷팅 |
| `Stop` | 응답 완료 시 | 완료 알림 소리 |
| `Notification` | 알림 전송 시 | macOS 알림 |
| `SessionStart` | 세션 시작 시 | 환경 초기화 |
| `UserPromptSubmit` | 프롬프트 제출 시 | 입력 검증 |

### Hook 종료 코드

| 코드 | 동작 |
|---|---|
| **Exit 0** | 작업 진행 허용 |
| **Exit 2** | 작업 차단 (stderr 메시지가 Claude에 피드백) |
| **기타** | 작업 진행, stderr는 로그에만 기록 |

## 4가지 권한 모드

| 모드 | 설명 | 추천 대상 |
|---|---|---|
| **기본 모드** | 파일 변경마다 허락 구함 | 안전 최우선 |
| **자동 수정 모드** | 기존 파일 수정 자동, 새 파일만 확인 | 초보자 추천 |
| **Bypass Permissions** | 모든 권한 AI에게 위임 | 격리 환경만 |
| **Plan Mode** | 설계를 먼저, 코드 수정 불가 | 복잡한 작업 |

> **Bypass Permissions 주의:** AI가 시스템 파일을 통째로 날려버린 사례가 보고된 적 있습니다. 격리된 환경에서만 사용하세요.

## Plan Mode - 가장 강력한 기능

**짓기 전에 설계도를 먼저 그리는 방식.** Plan Mode에서 Claude Code는 코드를 직접 수정하지 않습니다.

1. 프로젝트 구조를 읽고 분석합니다
2. 질문이 있으면 사용자에게 물어봅니다
3. 어떻게 구현할지 상세한 계획을 세웁니다
4. 사용자가 검토하고 승인하면 코드를 작성합니다

> **효과:** 35분 걸리는 작업이 15분으로 단축. 계획 1분 투자 = 빌드 10분 절약. 특히 데이터베이스, 결제, 인증 같은 복잡한 기능에서 효과 극대화.

> **비유:** 플랜 없이 코딩하는 건 네비게이션 없이 처음 가는 곳을 운전하는 것과 같습니다. 도착은 하겠지만, 기름값이 3배로 나옵니다.

## 컨텍스트 관리 전략

첫 메시지를 입력하기도 전에 이미 **25% 이상**이 소비되어 있습니다 (시스템 프롬프트, 도구 설명, MCP, 메모리 등).

| # | 전략 | 설명 |
|---|---|---|
| 1 | `/context` | 현재 컨텍스트 사용량 확인 |
| 2 | `/compact` | 대화를 고밀도 요약으로 압축 |
| 3 | 짧고 구체적인 프롬프트 | 핵심만 전달하여 토큰 절약 |
| 4 | Sub Agent 활용 | 리서치를 별도 에이전트에 위임, 메인 컨텍스트 보호 |
| 5 | Extended Thinking | 사고 과정 토큰이 컨텍스트에 누적되지 않음 |
| 6 | 모델 적재적소 | 단순 작업 Sonnet, 복잡한 작업 Opus |

## 주요 슬래시 명령어

| 명령어 | 설명 |
|---|---|
| `/init` | CLAUDE.md 자동 생성 |
| `/context` | 컨텍스트 사용량 확인 |
| `/compact` | 대화 압축 |
| `/clear` | 대화 완전 초기화 |
| `/cost` | 토큰 비용 확인 |
| `/model` | 모델 변경 (Sonnet / Opus / Haiku) |
| `/review` | 최근 변경사항 코드 리뷰 |
| `/config` | 설정 인터페이스 |
| `/permissions` | 권한 규칙 관리 |
| `/hooks` | Hook 설정 |
| `/agents` | 서브에이전트 관리 |
| `/add-dir` | 추가 디렉토리 포함 |

PART 3

# 자동화와 확장

Skill, MCP, Sub Agent, Agent Team으로 개인이 팀 수준의 생산성을 내는 법

## Skill 시스템

Skill은 **AI를 위한 업무 매뉴얼**입니다. 반복 작업을 단계별 체크리스트로 파일에 저장합니다. `/스킬이름`으로 바로 실행할 수 있습니다.

### Skill 구조

```
# 저장 위치
.claude/skills/my-skill.skill.md

# 파일 형식 (Markdown)
---
name: my-skill
description: 이 스킬의 간단한 설명
---
실제 작업 단계가 여기에 적힙니다...
```

> **영리한 설계:** Front Matter(이름/설명)만 먼저 로드하므로 약 60~70 토큰만 소비. 실제 내용은 호출 시에만 로드. **Skill을 100개 만들어도 컨텍스트를 거의 차지하지 않습니다.**

### Skill 활용 사례

| Skill | 수동 작업 | Skill 적용 |
|---|---|---|
| 리드 스크래핑 | 30분+ | 90초 |
| 이메일 분류/답장 | 1시간+ | 수 분 |
| 주제별 리서치 | 수 시간 | 수 분 |
| 맞춤 웹사이트 생성 | 수 일 | 30초 |

### Skill 만드는 방법

1. 작업을 음성이나 텍스트로 Claude에게 설명
2. Claude가 Skill 파일 형식으로 자동 정리
3. 70% 정확도에서 시작 (괜찮습니다!)
4. 테스트하며 실패 부분 수정 (2~3회 반복)
5. 98~99% 정확도 달성

> **Tip:** 새로 만든 Skill은 반드시 **새로운 세션**에서 테스트하세요. 기존 대화 컨텍스트가 결과에 영향을 줄 수 있습니다.

## MCP (Model Context Protocol)

MCP는 **다른 사람이 만들어 놓은 Skill**과 비슷합니다. Gmail, Slack, Notion, Google Sheets 등 외부 서비스와 Claude Code를 연결하는 플러그인입니다.

### MCP vs Skill 비교

| 항목 | MCP | Skill |
|---|---|---|
| 토큰 사용 | 수천 토큰 | 60 토큰 |
| 처리 속도 | 이메일당 수 초 | 0.3초 |
| 용도 | **프로토타입용** | **운영용** |

> **추천 워크플로우:** MCP로 빠르게 기능 테스트 → 잘 작동하면 API 직접 호출하는 Skill로 변환 → 토큰 사용량 극적 감소

> **최추천 MCP:** Anthropic Chrome DevTools MCP. Chrome 브라우저를 직접 제어하여 웹 페이지 열기, 버튼 클릭, 데이터 추출이 가능합니다. API가 없는 서비스에서 데이터를 가져올 때 필수입니다.

## Sub Agent (서브 에이전트)

메인 Claude Code가 특정 업무를 **별도 에이전트에게 위임**하는 개념입니다. Sub Agent는 자기만의 별도 컨텍스트에서 작업하고, 결과 요약만 메인에게 돌려줍니다.

### 내장 Sub Agent

| 에이전트 | 모델 | 용도 |
|---|---|---|
| **Explore** | Haiku (빠름/저렴) | 코드베이스 탐색, 파일 검색 |
| **Plan** | 상속 | Plan Mode에서 리서치 |
| **General-purpose** | 상속 | 복잡한 멀티 스텝 작업 |
| **Bash** | 상속 | 별도 컨텍스트에서 명령어 실행 |

### 유용한 커스텀 Sub Agent

| 에이전트 | 역할 |
|---|---|
| 리서치 에이전트 | 저렴한 Sonnet으로 방대한 자료 검색/요약 |
| 코드 리뷰 에이전트 | 새로운 시각에서 코드 품질 검토 |
| 품질 보증 에이전트 | 테스트 코드 자동 작성/실행 |

> **확률의 수학:** 각 Sub Agent 성공률 95%일 때, 10개를 동시에 돌리면 전체 성공률은 **59%**입니다 (0.95의 10제곱). 작업을 잘게 쪼개서 단

> 순한 일을 시키는 것이 핵심입니다.

## Agent Team

Sub Agent의 진화형. 팀원들이 **서로 소통**하며 자율적으로 협업합니다.

### Agent Team 특징

| 특징 | 설명 |
|---|---|
| 독립 인스턴스 | 각 팀원이 자기만의 컨텍스트, CLAUDE.md, MCP 접근 권한 보유 |
| 자율 협업 | 공유 태스크 보드에서 진행 상황 확인하며 자율 작업 |
| 적대적 토론 | 두 에이전트가 반대 입장에서 토론하여 품질 향상 (GAN 원리) |
| 활성화 | `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` 설정 필요 |

> **비용 주의:** 일반 세션 대비 **7배** 토큰 소비. 에이전트 10명 x 15분 = 약 8만원. 대규모 작업(보안 감사, 코드 리팩토링)에만 사용 권장.

## Git Worktree

Agent Team과 함께 사용할 때 효과적입니다. 각 Branch가 별도 작업 디렉토리를 가져서 **파일 충돌 없이 병렬 개발**이 가능합니다. 한 에이전트는 소개 페이지, 다른 에이전트는 연락처 페이지, 또 다른 에이전트는 서비스 페이지를 동시에 만들고 나중에 합칩니다.

## 전체 워크플로우 조합


반복 작업은 **Skill**로, 외부 연동은 **MCP/API Skill**로 자동화합니다. 이 워크플로우를 한번 구축해 놓으면 어떤 프로젝트든 동일한 품질 수준으로 빠르게 진행할 수 있습니다.

---

## 부록: 주요 키보드 단축키

| 단축키 | 설명 |
|---|---|
| `Escape` | Claude 중지 (Ctrl+C가 아님!) |
| `Escape` x 2 | 이전 메시지 목록으로 점프 |
| `Shift+Tab` | Plan Mode 전환 |
| `Cmd+T` | Extended Thinking 토글 |
| `Cmd+P` | 모델 선택기 |
| `Ctrl+G` | 외부 에디터에서 편집 |

| | |
|---|---|
| `Ctrl+O` | 상세 모드(verbose) 토글 |
| `Ctrl+B` | 현재 작업을 백그라운드로 전환 |

## 부록: 오늘 당장 실행할 액션 아이템

1. **CLAUDE.md 만들기:** `/init`으로 시작. 자주 하는 실수나 반드시 따라야 할 규칙을 3줄만 추가해 보세요.
2. **Plan Mode 사용하기:** 다음 작업에서 바로 빌드하고 싶은 충동을 참고, 계획을 먼저 세워보세요.
3. **Skill 하나 만들기:** 매일 반복하는 작업 하나를 Skill로 만들어 보세요. 70% 정확도로 시작해서 점점 개선하면 됩니다.

## 부록: 보안 주의사항

> **반드시 기억하세요:**
> AI가 작성한 코드에는 보안 취약점이 숨어 있을 수 있습니다. 돈을 받고 서비스를 운영하기 전에는 반드시 보안 검토를 하셔야 합니다.
> 특히 **결제 기능**이나 **개인정보**를 다루는 서비스라면 전문가의 보안 리뷰를 꼭 받으세요.
> Claude Code는 엄청난 도구이지만, 모든 도구가 그렇듯 **제대로 알고 쓸 때 진가를 발휘합니다.**

**Claude Code 핵심 가이드** | 2026년 2월 기준 | Opus 4.6

AI 도구는 빠르게 발전하고 있습니다. 최신 정보는 공식 문서를 참고하세요.