

# AI 코딩 셀프스터디 핸드북

부트캠프에서 손으로 한 번 해본 것을, 집에서 혼자 다시 세울 수 있게 만든 교재입니다.  
터미널을 여는 것부터 내 서비스에 진짜 주소가 붙기까지 — 순서대로 열한 장. 읽고,  
표시하고, 직접 쳐보세요.

이름 \_\_\_\_\_ 내가 만들 것 \_\_\_\_\_ 복습 시작일 \_\_\_\_\_

A4 · 인쇄 최적화 10개 장 + 부록 3 기준 도구 : Claude Code CLI

## 목차

|    |                         |                      |
|----|-------------------------|----------------------|
| 00 | 이 책을 쓰는 법               | 셀프스터디 규칙             |
| 01 | 멘탈 모델 — 누가 일하는가         | GUI와 CLI             |
| 02 | 터미널 — 컴퓨터에게 글자로 말 걸기    | 명령어 4개               |
| 03 | Claude Code 운전법         | 모델 · 모드 · 슬래시        |
| 04 | 토큰과 컨텍스트                | 연료와 책상               |
| 05 | 코드가 돌아가는 원리             | 비전공자용 지도             |
| 06 | Claude 세계의 구조           | API · MCP · Harness  |
| 07 | Git & GitHub — 코드의 타임머신 | commit · push · pull |
| 08 | localhost에서 진짜 주소까지     | 배포와 .env             |
| 09 | 집에서 이어서 작업하기            | 두 컴퓨터 동기화            |
| 10 | 더 잘하기                   | 디자인 · 도구 · 다음 단계     |
| 11 | 가장 크게 남길 것              | 마인드셋                 |
| A  | 명령어 · 슬래시 치트시트          | 부록                   |
| B  | 용어집                     | 부록                   |
| C  | 링크 모음                   | 부록                   |

## 이 책을 쓰는 법

읽기만 해서는 남지 않습니다. 이 책은 손을 움직이라고 만든 물건입니다.

### 세 가지 규칙

1. **한 장씩, 터미널을 켜 놓고.** 장마다 마지막에 「직접 해보기」가 하나씩 있습니다. 그걸 하지 않으면 그 장은 안 읽은 것으로 칩니다.
2. **모르는 단어가 나오면 즉시 부록 B(용어집).** 모르는 채로 다음 문단으로 넘어가지 마세요. 이 분야는 단어 하나 막히면 그 뒤가 전부 안 보입니다.
3. **막히면 AI에게 물어보세요.** 이 책의 어떤 문단이든 그대로 복사해서 Claude Code에 붙여넣고 "이거 내 상황에 맞게 다시 설명해줘"라고 하면 됩니다. 그게 이 도구를 쓰는 방식 그 자체입니다.

### 읽는 순서

01~06은 **개념**, 07~09는 **실전 절차**, 10~11은 **다음 단계**입니다. 급하면 06→07→08만 먼저 해도 서비스는 배포됩니다. 다만 04(토큰·컨텍스트)를 건너뛰면 어느 날 요금이나 성능에서 반드시 대가를 치릅니다.

이 문서에 없는 것

설치 절차(Node.js·Claude Code CLI 설치)는 화면을 따라가며 해야 하는 일이라 이 책에 넣지 않았습니다. 부록 C의 설치 가이드 링크를 보고 진행하세요.

# 멘탈 모델 — 누가 일하는가

도구 사용법을 배우기 전에, 머릿속 그림부터 바뀌어야 합니다. 이 장이 나머지 아홉 장의 전제입니다.

## GUI와 CLI

| 구분  | 뜻                       | 예시                                                                      |
|-----|-------------------------|-------------------------------------------------------------------------|
| GUI | 마우스로 클릭하는 화면 기반 방식      | 폴더 더블클릭, 버튼 클릭, 앱스토어 설치                                                 |
| CLI | 글자(명령어)로 지시하는 텍스트 기반 방식 | <code>cd</code> , <code>npm install</code> , <code>vercel deploy</code> |

GUI(Graphical User Interface)는 우리가 평소 쓰는 방식입니다. CLI(Command Line Interface)는 개발자들이 쓰는 방식이고요. 둘은 *같은 컴퓨터를 다루는 두 개의 문*입니다.

## 핵심 문장 하나

외울 것

내가 일하고 싶으면 GUI, AI한테 일을 시키고 싶으면 CLI.

예전(GUI 시대) = 내가 직접 운전 — 핸들 잡고, 클릭하고, 타이핑

지금(AI 시대) = 기사에게 목적지를 말함 — AI가 운전하고, 나는 방향만 정함

오늘 CLI를 쓰는 이유는 셋입니다. ① 클릭 10번 할 걸 명령어 1줄로 끝냅니다. ② 명령어는 AI가 자동으로 만들어주므로 외울 필요가 없습니다. ③ Vercel·Supabase 같은 배포 도구는 CLI로 할 때 가장 간편합니다.

## 무엇이 바뀌나

|        | 예전 멘탈 모델        | AI 시대 멘탈 모델    |
|--------|-----------------|----------------|
| 내 역할   | 실행자 — 내 손으로 다 함 | 지휘자 — AI에게 위임  |
| 중요한 능력 | 도구 사용법 암기       | 원하는 걸 명확히 요청하기 |
| 속도     | 내 손 속도가 한계      | AI 속도로 확장      |
| 실수했을 때 | 처음부터 다시         | "이렇게 고쳐줘" 한마디  |

## 마인드셋 세 가지 전환

1. "어떻게 하지?" → "누구에게 시키지?" — 방법을 몰라도, 잘 시키기만 하면 됩니다.
2. "내가 다 알아야 해" → "AI가 알고, 나는 검증한다" — 전문가가 아니어도 방향만 잡으면 됩니다.
3. "완벽하게 한 번에" → "빠르게 시키고 고쳐가며" — 대화하듯 반복해서 다듬는 게 정답입니다.

이제 경쟁력은 "내가 얼마나 잘 만드느냐"가 아니라 "AI에게 얼마나 잘 시키느냐"입니다.

## Harness Engineering

### 한줄 정의

AI 도구를 활용해 빠르게 소프트웨어를 만드는 능력. 말고빠(harness)로 말을 제어하듯, AI의 힘을 효과적으로 제어하고 활용하는 것.

- AI가 코드를 작성하고, 사람은 **방향을 제시하고 검증**합니다.
- 전통적인 코딩 지식보다 **문제 정의와 AI와의 대화** 능력이 중요해집니다.
- 명확한 지시 → 결과 검토 → 반복 개선. 이 루프가 Harness Engineering입니다.

## 오늘 만들 것의 크기 정하기 (MVP)

혼자 공부할 때도 똑같습니다. **하루 안에 만들고 배포까지 끝낼 수 있는 크기**로 범위를 자르세요. 여기서 그날의 결과가 결정됩니다. "무엇을 뺄 것인가"를 먼저 정하는 게 MVP입니다.

## 무엇까지 만들 수 있나 — 여덟 단계

"AI 코딩"이라고 하면 웹사이트부터 떠올리지만, 실제로 만들어지는 것의 대부분은 아래쪽 단계입니다. **위로 갈수록 필요한 도구가 하나씩 늘어날 뿐**, 전부 알아야 시작할 수 있는 게 아닙니다.

| 단계 | 무엇           | 예                                |
|----|--------------|----------------------------------|
| 1  | 글과 문서        | 기획서 초안, 리서치 요약, 번역               |
| 2  | 이미지와 발표자료    | 도식, 카드뉴스, 발표 자료                  |
| 3  | 파일 정리와 변환    | PDF·CSV에서 정보 추출, 이름 일괄 변경        |
| 4  | 데이터 분석과 리포트  | 지표 집계, 차트, 설문 분석                 |
| 5  | 웹페이지와 공유 화면  | 랜딩 페이지, 대시보드, 미니 도구              |
| 6  | 반복 작업 자동화    | 매주 나가는 보고서 만들어 두기, 자료 모아 메일로 보내기 |
| 7  | 데이터가 쌓이는 서비스 | 신청 접수, 게시판, 사내 어드민               |
| 8  | 본격적인 서비스     | 결제가 붙는 유료 서비스                    |

겉모습은 달라도 흐름은 같습니다 — **무언가를 입력받고, 규칙대로 처리하고, 결과를 저장하거나 보여준다.**

### 처음 고를 때

“내가 만들고 싶은 앱”보다 **“어제 내가 불편했던 일”** 을 고르는 편이 훨씬 쉽게 끝납니다.

## 처음엔 잘 되다가 무너지는 이유

말하면 화면이 나오고, 고쳐달라면 고쳐줍니다. 그런데 **기능이 하나씩 불기 시작하면** 어느 순간 한 번에 해내지 못하는 지점이 옵니다. 그때 코드를 한 번도 안 본 채 “다시 해줘”만 반복하면, 내 의도와 결과물이 조금씩 멀어집니다.

- 여기를 손보면 멀쩡하던 저기가 멈춥니다.
- 내 화면에서는 멀쩡한데 남에게 보낸 주소에서는 안 열립니다.

- 빨간 글씨가 떠도 어디부터 봐야 할지 몰라 같은 요청만 다시 넣게 됩니다.

원인은 결국 돌입니다.

| 무너지는 이유                | 이 책에서 채우는 곳                   |
|------------------------|-------------------------------|
| 전체가 어떻게 돌아가는지 모른 채 만든다 | 05~08장 — 코드가 도는 원리부터 배포까지의 지도 |
| 말기고 검증하는 루프가 없다        | 03·04장 — 계획·모드·되돌리기·컨텍스트 관리   |

스스로 점검

- ☐ GUI와 CLI 중, "AI에게 일을 시킬 때" 쓰는 쪽은 어디이고 그 이유는?
- ☐ AI 시대에 중요해지는 능력은 무엇으로 바뀌었나?
- ☐ Harness Engineering에서 사람이 맡는 두 가지 역할은?

**직접 해보기** — 내가 만들고 싶은 것을 한 문장으로 쓰고, 그중 하루 안에 배포 가능한 기능 하나만 남겨 표지의 「내가 만들 것」 칸에 적으세요.

## 터미널 — 컴퓨터에게 글자로 말 걸기

AI를 파견할 작업장. 검은 창이 정체와, 그 안에서 알아야 할 명령어 네 개.

### 터미널이란

#### 한줄 정의

컴퓨터에게 **글자로 명령을 내리는 창**. 설치 명령어들은 전부 여기에 붙여넣습니다.

- **Windows** — PowerShell. 시작 메뉴에서 "PowerShell" 검색 → Enter
- **Mac** — 터미널(Terminal). `⌘ + Space` → "터미널" 입력 → Enter

WINDOWS 사용자가 제일 자주 걸리는 것

PowerShell과 명령 프롬프트(CMD)는 다른 창입니다. 프롬프트가 `PS C:\`로 시작하면 PowerShell, `C:\`만 있으면 CMD입니다.

`'irm' is not recognized` 같은 말이 나오면 지금 CMD에 있는 것입니다. PowerShell을 새로 열고 다시 하세요.

**명령어 실행법:** 회색 박스의 텍스트를 복사(`⌘C` / `Ctrl+C`) → 터미널에 붙여넣기(`⌘V` / 마우스 우클릭) → `Enter`.

### 왜 이름이 '터미널'인가

라틴어 *terminus*(끝·경계·종착점)에서 왔습니다. 옛날에는 방 하나를 채우는 거대한 메인프레임 한 대뿐이었고, 사람들은 본체에서 멀리 떨어진 책상에서 **선의 맨 끝에 매달린 입출력 장치**로 명령을 넣고 결과를 받았습니다. 한국어 **단말기**(端末機) — '끝 단, 끝 말'의 기계 — 도 같은 뜻입니다.

| 분야  | '터미널'의 뜻     | 공통점         |
|-----|--------------|-------------|
| 교통  | 버스·공항 터미널    | 노선의 끝·종착점   |
| 전기  | 단자(terminal) | 전선의 끝·접점    |
| 컴퓨터 | 입출력 단말기      | 네트워크의 끝단 장치 |

60년 전 종이에 글자를 찍던 텔레타이프(Teletype, `tty`의 어원)가, 지금은 AI에게 일을 시키는 통로로 부활한 셈입니다.

# 알아야 할 명령어 4개 (+1)

| 하고 싶은 것   | WINDOWS                                 | MACOS               | 뜻                         |
|-----------|-----------------------------------------|---------------------|---------------------------|
| 현재 위치 보기  | <code>pwd</code>                        | <code>pwd</code>    | present working directory |
| 파일 목록 보기  | <code>ls</code> ( <code>dir</code> 도 됨) | <code>ls</code>     | list                      |
| 폴더 이동     | <code>cd</code> 폴더명                     | <code>cd</code> 폴더명 | change directory          |
| 한 단계 위로   | <code>cd ..</code>                      | <code>cd ..</code>  | 부모 폴더로                    |
| 작업 폴더 열기  | <code>start .</code>                    | <code>open .</code> | 탐색기/Finder 로 보기           |
| Claude 실행 | <code>claude</code>                     | <code>claude</code> | AI 에이전트 시작                |

거의 다 양쪽이 같습니다. 실제로는 `cd` 뒤에 **폴더 경로를 복사해서 붙여넣는 것**이 대부분이라, 외울 것도 사실상 없습니다.

작업 폴더는 여기에

**Windows** — `C:\ai` · **macOS** — `~/ai`

이름은 **영어와 숫자만**. 띄어쓰기와 한글은 나중에 반드시 문제를 만듭니다.

## 파일과 폴더 — GUI의 위치를 CLI로 번역하기

| GUI에서          | CLI에서                                            |
|----------------|--------------------------------------------------|
| "Downloads 폴더" | <code>Downloads</code> 디렉토리                      |
| 폴더 하나 들어가기     | <code>cd</code> 폴더이름                             |
| 현재 위치 확인       | <code>pwd</code>                                 |
| 폴더 안 파일 목록     | <code>ls</code> (Windows는 <code>dir</code> 도 가능) |

같은 위치를 GUI는 `내 Mac > 사용자 > 내이름 > Downloads`로, CLI는 **경로(path)**라는 한 줄 문자열로 씁니다 — `/Users/내이름/Downloads` (Mac) 또는 `C:\Users\내이름\Downloads` (Windows).

- `.` 은 현재 폴더, `..` 는 부모(한 단계 위) 폴더를 뜻합니다.
- 공백이 있는 폴더명은 따옴표로 감싸세요 — `cd "My Folder"`

## Local과 Global — 어디서 Claude를 켜느냐의 문제

한 줄 정의

같은 Claude Code라도 **어느 폴더에서 켜느냐**에 따라 AI가 볼 수 있는 범위가 달라집니다. 상위 폴더에서 켜면 그 아래 전부를, 개별 프로젝트 폴더에서 켜면 그 폴더 하나만 봅니다.

Global(상위 폴더에서 시작) = 건물 로비에 서서 모든 방으로 직원을 보낼 수 있는 본부

Local(프로젝트 폴더에서 시작) = 특정 방 하나에 들어가 그 방 일만 하는 것

그래서 **홈(바탕화면보다 위) 아래에 상위 폴더 하나**를 먼저 만들고 그 안에 프로젝트를 넣습니다. 이 폴더가 AI를 하위 어디로든 파견하는 본부가 됩니다.

폴더 구조 예시

ai/                      ← 여기서 claude 켜기 (Global 본부)  
├─ gimbab/            ← 프로젝트 1  
├─ my-portfolio/      ← 프로젝트 2  
└─ todo-app/          ← 프로젝트 3

| 구분        | GLOBAL (상위에서 켜기) | LOCAL (프로젝트에서 켜기) |
|-----------|------------------|-------------------|
| AI가 보는 범위 | 하위 모든 폴더         | 그 폴더 하나만          |
| 좋은 점      | 여러 프로젝트를 오가며 파견  | 그 작업에만 집중 · 안전    |
| 언제        | 여러 프로젝트를 관리할 때   | 한 프로젝트에 깊게 몰입할 때  |

폴더 만들 때 규칙  
폴더 이름은 **영어와 숫자만** 쓰세요. 한글·띄어쓰기는 쓰지 마세요. 예: `ai`  
**Windows** — `C:\ai` (C드라이브 바로 아래) · **macOS** — `~/ai` (터미널에서 `open ~` 하면 그 폴더가 열립니다)

나는 지금 어디에 있을까

- 터미널이면 프롬프트에 **현재 폴더 경로**가 보입니다.
- Claude Code 안이면 입력줄이 `>` 로 시작합니다. 이게 보이면 클로드 코드 안입니다.
- 추천 뷰 세팅: 왼쪽에 **작업 폴더의 터미널**, 오른쪽에 **같은 폴더의 Claude Code**.

스스로 점검

- ☐ `pwd` · `ls` · `cd` · `cd ..` 각각 무엇을 하는가?
- ☐ 프로젝트 폴더가 아니라 그 위 상위 폴더에서 `claude` 를 켜면 무엇이 달라지는가?
- ☐ 폴더 이름에 한글과 띄어쓰기를 쓰지 말라는 이유는?

**직접 해보기** — 홈 아래에 `ai` 폴더를 만들고, 터미널에서 `cd ai` → `pwd` → `ls` → `claude` 순서로 쳐서 본부를 세우세요.

# Claude Code 운전법

같은 모델이라도 어떤 껍데기에 태우고, 어떤 모드로 모느냐에 따라 결과가 달라집니다.

## 왜 데스크탑 앱이 아니라 터미널인가

- 데스크탑 앱 = 전화로 요리법을 물어보는 것 — 알리는 주지만 내 주방엔 못 들어옴
- 터미널 Claude Code = 셰프가 직접 내 주방에 와서 요리하는 것 — 재료를 만지고 불을 컴

데스크탑 앱(Claude·ChatGPT)의 다섯 가지 한계:

1. 내 파일·폴더에 직접 손을 못 댄다. 코드를 받으려면 복사 → 붙여넣기 → 저장을 내가 반복해야 합니다.
2. 명령어를 실행할 수 없다. `npm install`, 서버 켜기, `git push`, 배포를 대신 실행해주지 못합니다.
3. 에이전트 루프가 없다. 한 번 답하고 끝. CLI는 "읽기 → 수정 → 실행 → 에러 확인 → 재수정"을 스스로 반복합니다.
4. Harness(확장 도구)가 안 붙는다. MCP·Subagents·Hooks·슬래시 명령이 거의 안 돌아갑니다.
5. 컨텍스트 관리가 어렵다. `/compact` 나 프로젝트 규칙 파일 같은 정교한 관리가 안 됩니다.

| 기능                      | 데스크탑 앱    | 터미널 CLAUDE CODE |
|-------------------------|-----------|-----------------|
| 내 파일 직접 수정              | × (복붙 필요) | ○               |
| 명령어 실행 (설치·배포)          | ×         | ○               |
| Git · 서버 조작             | ×         | ○               |
| 에이전트 루프                 | ×         | ○               |
| MCP · Hooks · Subagents | △ 제한적     | ○               |
| HUD 계기판                 | ×         | ○               |

그럼 앱은 언제? 질문·브레인스토밍, 글쓰기·요약, 이미지 보고 물어보기. 즉 **생각과 대화**는 앱, **실제로 컴퓨터에서 일 시키기**는 터미널입니다.

## VSCode 플러그인을 쓰지 않는 이유

플러그인은 에디터 안에 갇힌 조수, 터미널 Claude Code는 컴퓨터 전체를 쓸 수 있는 AI 엔지니어입니다.

| 기능                | VSCode 플러그인 | 터미널 CLAUDE CODE |
|-------------------|-------------|-----------------|
| MCP 연결            | △ 제한적       | ○ 완전 지원         |
| Subagents · Hooks | ×           | ○               |
| 슬래시 명령            | ×           | ○               |
| 프로젝트 메모 파일        | △ 부분        | ○ 완전            |
| HUD 계기판           | ×           | ○               |

다만 **코드 한 블록만 고칠 때나 자동완성** 용도로는 플러그인도 괜찮습니다. "프로젝트 전체를 AI에게 맡기고 싶다"면 터미널입니다.

**/model** — 어떤 두뇌를 쓸지 고르기

한줄 정의

작업에 사용할 AI 모델(두뇌)을 바꾸는 명령어. 같은 운전자에게 **다른 엔진을 끼워주는 것과 같습니다.**

| 모델     | 언제 쓰나                 | 특징                          |
|--------|-----------------------|-----------------------------|
| Sonnet | 일상 코딩, 빠른 수정·질문       | 빠르고 저렴, 충분히 똑똑 — 기본 추천      |
| Opus   | 복잡한 문제, 큰 설계, 어려운 디버깅 | 가장 강력 · 긴 맥락 · 상대적으로 느리고 비쌈 |
| Haiku  | 직접 고를 일은 거의 없음        | 가볍고 빠름. 주로 서브에이전트용으로 자동 호출  |

**바꾸는 법:** `/model` 입력 → Enter → 화살표(↑↓)로 선택 → Enter. 이후 대화부터 적용됩니다.

모델 옆에는 **추론 강도(effort)** 설정도 있습니다. 깊게 생각할수록 시간과 토큰을 더 씁니다. 하루 종일 쓰는 학습·실습에는 **medium** 정도가 균형이 좋습니다.

한 방 정리

기본은 Sonnet. "이건 좀 어려운데?" 싶을 때만 Opus로 올리세요. **모델은 엔진, 모드는 운전 방식, 컨텍스트는 책상 크기** — 이 셋을 구분하면 됩니다.

**권한 모드 — 얼마나 물어보게 할지**

한줄 정의

같은 모델이라도 **어디까지 물어보고 어디까지 알아서 하게 할지**를 정하는 설정입니다. 자동차로 치면 엔진(모델)이 아니라 운전 방식입니다.

`Shift + Tab` 을 누를 때마다 다음 순서로 돌아갑니다.

| 모드          | 안 물어보고 하는 것                                                  | 화면 아래 표시                        |
|-------------|--------------------------------------------------------------|---------------------------------|
| Manual (기본) | 읽기만. 파일 수정·명령 실행은 매번 확인                                      | <code>⏏ manual mode on</code>   |
| acceptEdits | 작업 폴더 안 파일 수정과 <code>mkdir</code> · <code>mv</code> 같은 정리 명령 | <code>▶▶ accept edits on</code> |
| plan        | 읽고 계획만 세움. 소스는 안 건드림                                         | <code>⏏ plan mode on</code>     |

계정 조건이 맞으면 **auto** 가 `plan` 뒤에 하나 더 붙습니다. 안전하다고 판단한 작업은 묻지 않고 진행하는 모드입니다. 화면에 안 보인다면 그 계정에서는 아직 안 열린 것이니 신경 쓰지 않아도 됩니다.

설정값 이름은 `default` 이고 화면에는 **Manual** 로 표시됩니다. 같은 것입니다.

## plan 으로 시작해서 auto 로 넘기기

이게 실전에서 가장 많이 쓰는 흐름입니다. 계획을 먼저 보고, 납득한 다음에 자동으로 돌립니다.

1. `Shift + Tab` 으로 **plan** 진입 → 시킬 일을 말합니다.
2. Claude가 계획을 써서 보여주고 승인을 묻습니다.
3. 선택지가 세 개 뜹니다.

승인 화면

Claude has written up a plan and is ready to execute. Would you like to proceed?

- › 1. Yes, and use auto mode
- 2. Yes, manually approve edits
- 3. Tell Claude what to change  
shift+tab to approve with this feedback

- **1번을 권합니다.** 계획을 이미 읽었으니 단계마다 다시 물어볼 이유가 없습니다.
- 2번은 하나하나 확인하고 싶을 때. 느리지만 가장 안전합니다.
- 3번은 계획이 마음에 안 들 때. 고칠 점을 말하면 계획을 다시 씁니다.

AUTO 로 넘기기가 꺼려진다면

그건 모드가 위험해서가 아니라 **계획이 아직 못 미더워서**입니다. 3번을 고르고 이렇게 물어보세요 — “이 계획이 실패할 수 있는 이유를 확률 순으로 나열해줘.” “내가 놓친 전제가 뭐야?” “제일 위험한 단계 하나만 더 쪼개줘.” 계획이 단단해지면 auto 가 무섭지 않습니다.

그래도 지켜야 할 것

말하기 전에 **Git**으로 한 번 저장해두세요(07장). 되돌릴 지점이 있으면 대부분의 사고는 사고가 아니게 됩니다.

## 더 깊게 생각시키기 — `think`

이건 모드가 아니라 **프롬프트에 넣는 말**입니다. 답을 내기 전에 더 오래 따져보게 만듭니다.

- `think` → `think hard` → `think harder` → `ultrathink` 순으로 깊어집니다.
- 예: “`ultrathink` 해서 왜 이 에러가 나는지 분석해줘.”
- 생각도 출력 토큰을 씁니다. 어려운 문제에만 쓰세요.

## 되돌리기 — `/rewind`

왜 이걸 먼저 알아야 하나

“잘못되면 어찌지”가 비개발자를 가장 많이 멈춰 세웁니다. **되돌릴 수 있다는 걸 알면 과감해집니다.**

`/rewind` 는 대화와 파일 변경을 **이전 지점으로 되돌립니다**. 입력창이 비어 있을 때 `ESC` 를 두 번 눌러도 같은 메뉴가 열립니다.

고를 때 확인할 것

메뉴에서 어디까지 되돌릴지를 고르게 됩니다 — 대화만 되돌릴지, 파일 변경까지 되돌릴지. 파일까지 되돌리면 그 뒤에 손으로 고친 것도 같이 사라집니다. 한 번 읽고 고르세요.

## 자주 쓰는 슬래시 명령

| 명령                    | 하는 일                                          |
|-----------------------|-----------------------------------------------|
| <code>/status</code>  | 현재 모드, 진행 중인 작업, 컨텍스트 사용량, 로그인 방식을 한 번에 보여줌   |
| <code>/model</code>   | 모델 · 추론 강도 변경                                 |
| <code>/compact</code> | 길어진 대화를 요약해 공간 확보                             |
| <code>/clear</code>   | 대화를 완전히 비우고 새로 시작                             |
| <code>/rewind</code>  | 대화·파일 변경을 이전 지점으로 되돌리기                        |
| <code>/resume</code>  | 이전 세션을 불러와 이어서 작업                             |
| <code>/rename</code>  | 세션 이름 바꾸기                                     |
| <code>/btw</code>     | 본 작업을 흐트러뜨리지 않고 겹가지 질문하기                      |
| <code>/login</code>   | 계정 연결 상태 확인 · 다시 로그인                          |
| <code>/skills</code>  | 지금 쓸 수 있는 Skill 목록                            |
| <code>/config</code>  | 테마 · 기본 권한 모드 같은 환경 설정                        |
| <code>/exit</code>    | Claude Code 종료 (또는 <code>Ctrl + C</code> 두 번) |
| <code>/usage</code>   | 토큰·비용 상세 분해 보기                                |

`/STATUS`에서 반드시 확인할 것

로그인 방식이 **API 과금**으로 잡혀 있으면 즉시 `/exit` 후 다시 로그인하세요. 구독 요금제와 API 요금제는 비용 차이가 큼니다(04장 참고).

그리고 **지금 어느 폴더에 들어와 있는지도** 여기서 확인됩니다. Claude는 그 폴더의 파일을 **생성·읽기·수정·삭제**할 수 있습니다.

## 대화를 정리하는 두 가지 — `/clear` 와 `/compact`

비슷해 보이지만 판단 기준이 다릅니다.

| 명령                    | 언제                 | 무슨 일이 일어나나                   |
|-----------------------|--------------------|------------------------------|
| <code>/clear</code>   | 새 작업을 시작할 때        | 앞 대화를 끄습니다. 이전 맥락이 오히려 방해될 때 |
| <code>/compact</code> | 같은 작업인데 대화만 길어졌을 때 | 요약해서 책상 위 자리를 만듭니다           |

한 줄로: 새 기능을 시작하면 `/clear`, 같은 버그를 계속 쫓는 중이면 `/compact`.

# 입력창 기본기

이걸 모르면 긴 지시를 쓰다가 엔터를 눌러 중간에 날아갑니다.

| 하고 싶은 것     | WINDOWS      | MACOS          |
|-------------|--------------|----------------|
| 줄바꿈 (전송 말고) | Ctrl + Enter | Option + Enter |
| 스크린샷 붙여넣기   | Alt + V      | Cmd + V        |
| 답변 중단       | Esc          | Esc            |
| 입력창 비우기     | Esc 두 번      | Esc 두 번        |
| 단어 단위 커서 이동 | Alt + ↔      | Option + ↔     |
| 세션 종료       | Ctrl + C 두 번 | Ctrl + C 두 번   |

## 볼 대상을 좁혀주기 — @ 멘션

“알아서 찾아봐”보다 어디를 보라고 짚어주는 쪽이 훨씬 정확하고 훨씬 쉽니다.

- @파일이름 또는 @폴더이름 — 그것만 보게 합니다.
- 파일을 터미널 창으로 끌어다 놓아도 같은 효과입니다.
- 화면이 이상할 때는 스크린샷을 붙여넣으세요. 말로 열 줄 설명하는 것보다 빠릅니다.

예시

@견적서 폴더에 있는 PDF 세 개만 비교해줘.  
항목별 단가가 어떻게 다른지 표로 만들고,  
가장 싼 곳과 가장 비싼 곳의 차이가 어디서 나는지  
한 문단으로 설명해줘.

## 딴 얘기 하고 돌아오기 — /btw

작업 도중에 “근데 이건 뭐야?”가 떠오릅니다. 그냥 물어보면 본 대화가 옆길로 새니다.

- /btw 뒤에 물어보면 결가지 대화로 처리되고 본 작업 맥락은 그대로 남습니다.
- 다 물어봤으면 Esc 를 눌러 원래 세션으로 돌아오세요. 이걸 안 하면 계속 결가지에 머무릅니다.

## /resume — 하던 작업 이어서 하기

이전에 나눴던 대화를 그대로 불러와 끊긴 지점부터 이어갑니다. 어제 멈춘 프로젝트를 오늘 다시 열 때, 실수로 터미널을 꺾을 때 씁니다.

1. 작업하던 프로젝트 폴더로 이동 (cd 폴더명)
2. claude 입력
3. /resume 입력 → 최근 대화 목록에서 화살표로 선택 → Enter

/resume은 같은 폴더에서 했던 대화만 보여줍니다. 다른 프로젝트를 찾는다면 먼저 그 폴더로 cd 하세요. 너무 오래된 대화는 /compact로 한 번 정리하고 이어가면 쾌적합니다.

## HUD — AI가 일하는 걸 보는 계기판

터미널 하단에 항상 떠 있는 상태 표시줄입니다. 조종사가 고개를 숙이지 않고 전방 유리에서 계기판을 보듯, 시선을 떼지 않고 핵심 정보를 확인합니다.

- **컨텍스트 사용량** — 기억 공간을 얼마나 썼는지 게이지로
- **비용 추정** — 현재 세션에서 쓴 비용
- **활성 도구** — 지금 파일 읽기/쓰기/명령 실행 중 무엇을 하는지
- **Todo 진행률** — 시킨 작업 목록의 완료 상태
- **Git 브랜치** — 현재 작업 중인 코드 브랜치

## 계기판 직접 설정하기 — statusline

화면 아래에 늘 보이는 줄을 원하는 항목으로 바꿀 수 있습니다. 매번 `/status` 를 칠 필요가 없어집니다.

처음에는 이 세 가지면 충분합니다 — **지금 폴더**, **쓰고 있는 모델**, **컨텍스트 사용률**. 사용률이 보이면 `/compact` 할 시점을 감으로 알 수 있습니다.

그대로 붙여넣으세요

```
/statusline 지금 폴더 이름이랑 쓰고 있는 모델, 컨텍스트 남은 양을 한 줄로 띄워줘
```

스스로 점검

- ☐ 데스크탑 앱이 못 하는 일 세 가지를 말할 수 있는가?
- ☐ `Shift + Tab` 을 눌렀을 때 돌아가는 모드 세 가지의 이름과 차이는?
- ☐ 계획 승인 화면의 1번과 2번은 무엇이 다른가?
- ☐ 새 작업을 시작할 때와 같은 버그를 계속 쫓을 때, 각각 무엇을 쓰는가?
- ☐ 잘못 갔을 때 되돌리는 명령과, 고를 때 확인해야 할 것은?

.....  
**직접 해보기** — `claude` 를 켜고 `/status` → `/model` → `Shift+Tab` 을 세 번 눌러 화면 아래 표시가 어떻게 바뀌는지 확인하고, 나온 모드 이름을 순서대로 여백에 적으세요. 그 다음 `/statusline` 프롬프트를 한 번 실행해 계기판을 커두세요.

## 토큰과 컨텍스트

AI의 연료와 책상. 이 장을 건너뛰면 어느 날 요금이나 성능에서 반드시 대가를 치릅니다.

### 토큰이란

#### 한줄 정의

AI가 글을 읽고 쓸 때 세는 **최소 단위**. 서비스의 속도·비용·기억 용량이 전부 이 토큰 수로 결정됩니다.

- 영어 단어 1개 ≈ **1.3토큰** ( `hello` = 1토큰, `unbelievable` = 3토큰 )
- 한글 1글자 ≈ **1~2토큰** — 한글은 영어보다 토큰을 더 먹습니다
- 공백, 줄바꿈, 이모지도 전부 토큰으로 계산됩니다

택시 미터기처럼, 글자 하나하나가 요금으로 찍힙니다.

### Input과 Output

**Input Token** — AI에게 읽으라고 주는 모든 것: 내 질문, 첨부한 파일·이미지·코드, 시스템 프롬프트, 프로젝트 메모 파일, 이전 도구 실행 결과, 그리고 **지금까지의 대화 전체**.

핵심은 여기입니다. 100번째 메시지를 보낼 때 AI는 1~99번 메시지를 **다시 다 읽고** 답합니다. 대화가 길어질수록 입력 토큰이 누적됩니다.

**Output Token** — AI가 만들어 돌려주는 것: 답변 텍스트, 작성한 코드, 리스트·표·요약, 도구 호출 인자.

|    | INPUT TOKEN | OUTPUT TOKEN    |
|----|-------------|-----------------|
| 가격 | 싸다 (1배)     | <b>3~5배 비쌈</b>  |
| 이유 | 그냥 읽기만 함    | 한 글자씩 "생각"하며 생성 |

그래서 **입력 5페이지 ≈ 출력 1페이지** 값입니다. 긴 답변을 요구하면 비용이 급격히 오릅니다.

### Context와 Context Window

#### 한줄 정의

**Context** = AI가 답을 만들 때 참고하는 정보 전체의 묶음. **Context Window** = 그걸 한 번에 담을 수 있는 최대 용량.

컨텍스트 = AI의 책상. 책상이 크면 자료를 많이 펴놓고 비교하며 일할 수 있고, 작으면 자꾸 치워야 합니다.

컨텍스트에 들어가는 것: ① 지금까지의 대화 ② 첨부한 파일·이미지 ③ 시스템 프롬프트 ④ 프로젝트 메모 파일 ⑤ 도구 실행 결과 ⑥ MCP에서 가져온 데이터.

**Context Window = Input + Output 합산 한도**입니다. Input이 한도의 대부분을 먹으면 Output이 잘립니다.

`/compact` 로 입력을 줄여 **출력할 공간**을 확보하는 이유가 이것입니다.

**꽤 차면 생기는 일:** 속도 저하(매번 전체를 다시 읽음), 비용 증가, 성능 저하(중요한 정보가 오래된 대화에 묻힘), 기억 상실(초반 규칙을 잊음).

## 책상이 크다고 좋은 게 아니다 — Lost in the middle

### 한줄 정의

아주 긴 맥락을 주면 AI는 맨 앞과 맨 뒤는 잘 보는데 가운데가 흐릿해집니다. 자료를 많이 줄수록 좋아진다는 직관이 여기서 깨집니다.

300쪽짜리 보고서를 받으면 사람도 앞부분과 결론만 제대로 읽습니다. 가운데 150쪽은 눈으로 훑고 지나갑니다. AI도 같습니다.

그래서 "파일 200개 다 줄 테니 알아서 해"는 오히려 결과가 나빠집니다. 컨텍스트 윈도우가 커졌다는 말은 *다 볼 수 있다*는 뜻이지 *다 똑같이 잘 본다*는 뜻이 아닙니다.

### 대응 세 가지

1. 필요한 것만 준다. @ 멘션으로 볼 파일을 짚어주세요(03장).
2. 중요한 지시는 맨 끝에 한 번 더. 긴 자료를 붙인 뒤 마지막 줄에 핵심 요구를 다시 씁니다.
3. 길어지면 정리한다. /compact 로 가운데를 요약해 없앱니다.

### 한 방 정리

"많이 주기"가 아니라 "맞는 걸 주기"입니다. 이 감각이 잡히면 같은 일을 훨씬 적은 비용으로 더 정확하게 끝냅니다.

## 토큰을 많이 먹는 일곱 가지 상황

1. 긴 대화가 계속 이어질 때 (가장 흔함) — 매번 대화 전체를 다시 읽습니다. → /compact 또는 /clear
2. 큰 파일을 통째로 첨부할 때 — PDF 50페이지 ≈ 3만 토큰. → 관련 부분만 잘라서
3. 이미지·스크린샷이 많을 때 — 1장 ≈ 800~1,600토큰, 고해상도는 3,000토큰까지. → 꼭 필요한 것만, 리사이즈해서
4. 웹 검색·URL 읽기 — 페이지 전체를 가져옵니다. → "핵심 3줄로 요약"처럼 출력 범위 제한
5. 코딩 작업 — 전체 폴더 탐색 1만+, 긴 에러 로그 5,000~2만. → .gitignore 설정을 잘 해두기
5. MCP를 많이 붙였을 때 — 연결된 MCP 설명서가 매 요청마다 자동 포함. 5개 = 항상 5,000~1만 토큰 고정 비용. → 필요한 것만 켜두기
7. 출력을 길게 시킬 때 (가장 비쌈) — "전체 코드 다 다시 써줘" 같은 요청. → "수정된 부분만", "간결하게", "표로만"

### 토큰 폭탄 TOP 3

1. node\_modules 폴더를 읽게 시킴 — 사실상 무한대
2. .env 파일이 들어간 채로 "전체 코드 리뷰" — 보안 + 토큰 폭탄 동시
3. 에러가 안 풀려 같은 질문 20번 반복 — 매번 누적되며 컨텍스트가 꽉 참

# 비용 감각

| 상황                     | 대략 토큰     | 비용 감각  |
|------------------------|-----------|--------|
| "안녕" 한 마디 답변           | ~50       | 거의 0   |
| 코드 100줄 작성             | ~2,000    | 수십 원   |
| 긴 보고서 작성               | ~10,000   | 수백 원   |
| 1시간 코딩 세션 (compact 없이) | ~100,000+ | 수천 원   |
| 폴더 통째로 읽기 사고           | ~500,000+ | 만 원 단위 |

## 구독 VS API

**구독 요금제**는 정해진 할당량 안에서만 쓰이므로 예상 밖 청구가 없습니다. 학습·실습은 구독으로 하세요.

**API 요금제**는 쓴 만큼 청구되며 구독 대비 훨씬 비쌉니다. API는 *내가 만든 서비스에 AI 기능을 넣어 사용자가 쓰게 할 때* 붙이는 것입니다. 실습용으로 켜두지 마세요.

## Compacting — 요약해서 공간 만들기

### 한줄 정의

긴 대화를 **핵심만 남기고 요약**해서 기억 공간을 다시 확보하는 작업. 100페이지 회의록을 "핵심 결정 3줄 + 할 일 5개"로 정리하는 것과 같습니다.

`/compact` 를 치면 지금까지의 대화를 AI가 스스로 요약하고, 중요한 맥락(프로젝트 목표, 파일 구조, 진행 상황, 결정 사항)만 추린 뒤 세부 대화는 버립니다.

**언제 쓰나:** HUD에서 컨텍스트가 **70~80%**를 넘었을 때 / 큰 작업 단계가 끝났을 때 / AI가 초반에 말한 걸 잊기 시작할 때 / 응답이 눈에 띄게 느려졌을 때.

## Context Engineering

### 한줄 정의

AI가 좋은 답을 내도록 **무엇을 / 어떤 순서로 / 얼마나** 컨텍스트에 넣을지 설계하는 작업.

**프롬프트 엔지니어링**이 "질문을 잘 하는 기술"이라면, **컨텍스트 엔지니어링**은 "작업실 전체를 세팅하는 기술"입니다.

실전 기법 네 가지:

- **스코핑** — 한 대화에서 너무 많은 것을 시키지 않기. 작업을 영역별로 나눠 새 세션으로
- **기록** — 프로젝트 규칙·코딩 스타일·자주 쓰는 명령어를 프로젝트 메모 파일에 미리 적어두기
- **압축** — `/compact` 로 길어진 대화를 요약
- **선별적 첨부** — 관련 파일만 골라 보여주기. "전체 코드를 다 읽어"는 오히려 독

기억할 것

좋은 질문 + 나쁜 컨텍스트 = 나쁜 답

평범한 질문 + 좋은 컨텍스트 = 좋은 답

## 절약 팁 다섯

1. 긴 파일은 통째로 첨부하지 말고 **관련 부분만** (Input 절약)
2. **"간결하게 답해"**라고 명시 (Output 절약)
3. `/compact` 자주 사용
4. 새 작업은 새 세션(`/clear`)에서 — 이전 대화가 입력에 안 끼어듭니다
5. 반복 규칙은 **프로젝트 메모 파일**에 — 매번 설명하는 토큰 낭비 방지

---

스스로 점검

- ☐ Output 토큰이 Input보다 비싼 이유는?
- ☐ 대화가 길어질수록 비용이 왜 **가속** 되는가?
- ☐ HUD의 컨텍스트 게이지가 몇 %를 넘으면 `/compact` 를 돌려야 하는가?

---

**직접 해보기** — 실습 세션에서 `/usage` 를 쳐서 Input·Output 토큰과 비용을 확인하고, `/compact` 전후의 컨텍스트 게이지를 비교해 적으세요.

---

## 코드가 돌아가는 원리

비전공자가 제일 자주 막히는 지점은 코딩이 아니라 *용어*입니다. 전체 그림을 먼저 그려두면 그 뒤가 보입니다.

### 왜 이 장이 필요한가

AI가 코드를 다 써줘도, **결과를 읽고 판단하는 건 나**입니다. “404가 났어요”와 “서버는 살아 있는데 그 주소에 아무것도 없대요”는 같은 말인데, 앞의 문장만 알면 다음에 뭘 해야 할지 모릅니다.

이 장은 문법을 배우는 장이 아닙니다. **단어를 아는 장**입니다.

### 요청과 응답 — 웹에서 벌어지는 일 전부

식당에서 주문하는 것과 같습니다. 손님이 **요청(request)** 하고, 주방이 만들어서 **응답(response)** 을 돌려줍니다.

브라우저에 주소를 치면 이 일이 일어납니다.

1. 내 브라우저가 “**이 주소의 화면을 주세요**” 하고 요청을 보냅니다.
2. 그 주소의 컴퓨터(**서버**)가 받아서 처리합니다.
3. 결과를 돌려줍니다. 화면일 수도 있고, 데이터일 수도 있고, 에러일 수도 있습니다.

내 컴퓨터에서 열리는 `localhost` 도 똑같습니다. 다만 주방이 내 집 안에 있을 뿐입니다(08장).

### 상태 코드 — 응답에 붙는 세 자리 숫자

응답에는 항상 **어떻게 됐는지 알려주는 숫자**가 붙습니다. 이 셋만 알면 대부분의 상황이 구분됩니다.

| 숫자         | 뜻                    | 내가 할 일                             |
|------------|----------------------|------------------------------------|
| <b>200</b> | 정상. 잘 됐습니다           | 없음                                 |
| <b>404</b> | 그 주소에 그런 게 없다        | 주소가 맞는지, 파일 이름이 맞는지 확인             |
| <b>500</b> | 주소는 맞는데 서버가 처리하다 터졌다 | 서버 로그를 보고 원인 찾기 — AI에게 로그를 그대로 주세요 |

#### 구분이 왜 중요한가

**404는 내 잘못(주소), 500은 코드 잘못**입니다. 이 둘을 구분하면 AI에게 다른 걸 시키게 됩니다. 400번대는 요청한 쪽, 500번대는 서버 쪽 문제라고 외워두세요.

프론트엔드와 백엔드

|        | 프론트엔드          | 백엔드                    |
|--------|----------------|------------------------|
| 무엇     | 사람이 보는 화면      | 뒤에서 처리하는 부분            |
| 어디서 도나 | 사용자의 브라우저      | 서버                     |
| 예      | 버튼, 입력창, 목록 화면 | 로그인 확인, 저장, 계산, 발송     |
| 이 책에서는 | Vercel 이 말합니다  | Supabase 가 상당 부분 대신합니다 |

“화면은 나왔는데 저장이 안 돼요”는 프론트는 되고 백엔드가 안 되는 상황입니다. 어느 쪽 문제인지 말할 수 있으면 AI가 훨씬 빨리 고칩니다.

데이터베이스와 CRUD

한줄 정의

데이터베이스 = 기억해야 하는 정보를 담아두는 창고. 회원, 글, 주문, 기록처럼 다음에도 남아 있어야 하는 것들이 들어갑니다.

거기서 하는 일은 결국 네 가지뿐이고, 앞글자를 따서 **CRUD** 라고 부릅니다.

|        | 뜻   | 예            |
|--------|-----|--------------|
| Create | 만들기 | 회원가입, 글쓰기    |
| Read   | 읽기  | 목록 보기, 상세 보기 |
| Update | 고치기 | 프로필 수정       |
| Delete | 지우기 | 글 삭제, 탈퇴     |

Claude Code 가 내 폴더에서 하는 일도 정확히 이 네 가지입니다 — 파일을 만들고, 읽고, 고치고, 지웁니다. 그래서 어느 폴더에서 커느냐가 곧 권한 설정입니다(02장).

데이터베이스에 명령을 내리는 언어를 SQL 이라고 합니다. 외울 필요 없습니다 — AI가 씁니다. 다만 결과를 보고 “이건 읽기인데 왜 지워졌지?” 를 물을 수 있으면 됩니다.

라이브러리와 프레임워크

라이브러리 = 필요할 때 꺼내 쓰는 도구. 내가 주도하고, 필요할 때 부릅니다.

프레임워크 = 이미 지어진 건물 골조. 정해진 자리에 내 내용을 채워 넣습니다.

왜 알아야 하나면, AI가 “Next.js 로 만들까요?” 라고 물을 때 그게 골조를 하나 고르는 결정이라는 걸 알아야 하기 때문입니다. 골조는 도중에 바꾸기 어렵습니다. 공구는 아무 때나 바뀌도 됩니다.

초보자에게 권하는 판단: **AI가 추천하는 기본값을 그대로 쓰세요.** 대신 “왜 이걸 골랐어?” 는 한 번 물어보세요. 그 답이 나중에 바꿀지 말지를 정할 때 근거가 됩니다.

## 에러와 로그 — 겁내지 않고 읽는 법

빨간 글자가 쏟아지면 대부분 창을 닫아버립니다. 그럴 필요 없습니다.

1. 맨 아래부터 보세요. 보통 마지막 줄에 진짜 원인이 있습니다.
2. 파일 이름과 줄 번호가 적혀 있습니다. 거기가 사고 지점입니다.
3. 그대로 복사해서 AI에게 주세요. 요약하지 마세요 — 요약하면서 원인이 날아갑니다.

### 같이 줘야 하는 것

에러 문구만 주지 말고 **재현 조건**을 붙이세요 — “이 기기에서, 이 순서로, 이 경로로 들어왔을 때만. 다른 데선 안 뚫.”  
이 한 줄이 해결 속도를 몇 배로 바꿉니다.

### 스스로 점검

- ☐ 404와 500은 각각 누구 쪽 문제이고, 나는 다르게 무엇을 하는가?
- ☐ “화면은 나오는데 저장이 안 된다”는 프론트와 백엔드 중 어느 쪽 이야기인가?
- ☐ CRUD 네 글자를 풀어 쓰고, Claude Code 가 내 폴더에서 하는 일과 어떻게 겹치는지 말할 수 있는가?
- ☐ 에러 로그를 AI에게 줄 때 반드시 같이 줘야 하는 것은?

**직접 해보기** — 브라우저에서 아무 사이트나 열고 주소 끝에 `/이건없는주소` 를 붙여 Enter 하세요. 어떤 숫자가 나오는지 확인하고, 그 화면을 캡처해 Claude Code 에 붙여넣어 “이게 무슨 뜻이고 누구 잘못이야?” 라고 물어보세요.

## Claude 세계의 구조

모델 하나만 쓰는 게 아닙니다. 그 주변을 감싸는 겹이 생산성의 차이를 만듭니다.

### 여섯 겹

#### 1. Core — Claude 모델

실제로 사고하고 코드를 생성하는 LLM. 모든 것의 중심이지만, 혼자서는 파일을 읽거나 명령을 실행할 수 없습니다.

#### 2. Runtime — Claude Code (CLI)

모델에게 *손과 발*을 달아주는 실행 환경. 터미널에서 `claude`를 치면 열리는 그것. 파일 읽기·쓰기, 명령 실행, Git 조작을 담당합니다.

#### 3. Context Layer — 프로젝트 메모 파일 & 설정

"우리 프로젝트는 이렇게 생겼어"를 알려주는 메모리. 규칙, 코딩 스타일, 자주 쓰는 명령어를 적어두면 매번 설명할 필요가 없습니다.

#### 4. Extension Layer — 확장 도구

MCP(외부 서비스 연결), Subagents(특정 작업 전담 에이전트), Hooks(이벤트 발생 시 자동 실행), Slash Commands(자주 쓰는 동작의 단축어).

#### 5. Harness Preset

위 확장들을 한 번에 셋업해주는 플러그인 팩. 검증된 프롬프트·명령어·훅이 한꺼번에 설치됩니다.

#### 6. Interface — HUD & 터미널

사용자가 실제로 보는 계기판.

#### 핵심

모델이 좋아지는 건 만드는 회사가 하는 일. **Harness를 잘 두르는 건 내가 하는 일**입니다. 같은 Claude를 써도 여기서 생산성이 갑니다.

## API란 무엇인가

#### 한줄 정의

서로 다른 프로그램/서비스가 **정해진 규격**으로 요청하고 응답받는 소통 창구. 식당 메뉴판처럼 "이런 걸 주문할 수 있고 이런 형태로 나온다"가 정의돼 있습니다.

손님(내 앱) → 메뉴판(API 문서) → 주문서(요청) → 주방(서버) → 음식(응답)

손님은 주방이 어떻게 돌아가는지 몰라도 메뉴판대로 주문하면 됩니다. 내부 구현을 몰라도 **약속된 방식**으로 쓸 수 있는 게 API의 핵심입니다.

**일상 속 예시** — 날씨 앱이 기상청 API에 "서울 날씨 알려줘", 쇼핑몰이 결제 API로 결제 요청, 배달앱이 지도 API로 위치 불러오기.

### 최소한만 알면 되는 구성 요소

| 요소       | 뜻                                                                                                 |
|----------|---------------------------------------------------------------------------------------------------|
| Endpoint | 요청을 보낼 주소(URL)                                                                                    |
| Method   | <code>GET</code> (읽기) / <code>POST</code> (생성) / <code>PUT</code> (수정) / <code>DELETE</code> (삭제) |
| Request  | 내가 보내는 데이터 (보통 JSON)                                                                              |
| Response | 서버가 돌려주는 데이터 (보통 JSON)                                                                            |
| API Key  | "쓸 권한 있어요"를 증명하는 비밀 토큰                                                                            |

오늘 API를 직접 다룰 일은 거의 없습니다. Claude Code가 코드를 알아서 작성해주시니까요. 다만 "이 서비스에 API가 있나?"를 확인하는 습관만 있으면, "이 API 붙여줘" 한마디로 기능이 생깁니다.

## MCP란 무엇인가

### 한 줄 정의

AI가 외부 서비스(Notion, GitHub, Slack, Figma, Stripe...)와 대화할 수 있게 해주는 **표준 플러그인 규격**. USB-C 포트가 모든 기기를 연결하듯, MCP는 AI와 외부 도구를 잇는 공용 단자입니다.

MCP 없는 Claude = 요리는 잘하는데 냉장고 문을 못 여는 셰프

MCP 연결된 Claude = 냉장고, 찬장, 시장, 배달앱까지 다 쓸 수 있는 셰프

같은 셰프인데 **닿을 수 있는 재료의 범위**가 완전히 달라집니다.

|        | MCP 없으면      | MCP 있으면                  |
|--------|--------------|--------------------------|
| 매출 조회  | 못 함          | "이번 달 결제 건수 보여줘" → 즉시 조회 |
| 회의록 정리 | 복붙 필요        | AI가 직접 페이지 생성            |
| 이슈 생성  | 사이트에 직접 가야 함 | AI가 자동 생성                |

**대표적인 MCP** — Notion(페이지·DB 읽고 쓰기), GitHub(코드·이슈·PR), Figma(디자인 파일), Linear·Jira(티켓), Stripe(결제·금융), HubSpot·Salesforce(CRM), Sentry(에러 모니터링), Intercom(고객 대화).

단, MCP를 많이 연결할수록 **매 요청마다 고정 토큰 비용**이 붙습니다(04장). 필요한 것만 켜주세요.

## API와 MCP의 차이

### 한 줄 요약

둘 다 외부 서비스와 대화하는 창구인 건 같습니다. **누가 읽고 쓰느냐**가 다릅니다. API는 사람(개발자)이 읽고 코딩, MCP는 AI가 읽고 자동 실행.

| 기준    | API               | MCP                |
|-------|-------------------|--------------------|
| 누가 쓰나 | 사람이 코드로 호출        | AI가 자동으로 호출        |
| 규격    | 서비스마다 제각각         | 표준 규격 하나           |
| 연결 방식 | 문서 읽고 코드 작성       | 설치 한 번이면 AI가 자동 인식 |
| 대상    | 모든 개발자·프로그램       | AI 에이전트 전용         |
| 생긴 시기 | 2000년대 초 (오래된 표준) | 2024년 발표 (신규)      |

**같은 일을 시킬 때** — API 방식은 문서 읽기 → 키 발급 → 코드 작성 → 응답 파싱 → 에러 처리(수 시간). MCP 방식은 설치 한 번(5분) → AI에게 그냥 말하기.

그럼 **API는 쓸모없어졌나?** 아닙니다. MCP는 **내부적으로 API를 호출**합니다.

호출 경로

[사용자] → [Claude] → [MCP 서버] → [실제 API] → [상대 서버]

**언제 뭘 쓰나** — MCP가 있으면 무조건 MCP. 없으면 "이 API 문서 보고 코드 써줘"라고 AI에게 부탁. 내 앱에 기능을 넣을 때는 API 직접 호출(보통 AI가 코드를 씁니다).

**CLAUDE.md — 같은 말을 두 번 하지 않기**

한 줄 정의

Claude Code 가 대화를 시작할 때마다 자동으로 읽는 메모 파일입니다. 매번 설명하던 것을 여기 한 번 적어두면 그 뒤로는 알아서 지킵니다.

두 군데에 둘 수 있고, 성격이 다릅니다.

|       | 어디에                              | 무엇을 적나                                        |
|-------|----------------------------------|-----------------------------------------------|
| 프로젝트용 | 프로젝트 폴더 안 <code>CLAUDE.md</code> | "이 사이트 색은 이거", "배포는 Vercel 로", "이 폴더는 건드리지 마" |
| 내 전체용 | <code>~/claude/CLAUDE.md</code>  | "답변은 한국어로", "코드 고치기 전에 계획부터 보여줘"              |

둘 다 읽히고, 겹치면 **프로젝트용이 이깁니다**. 내 취향은 전체용에, 이 프로젝트만의 규칙은 프로젝트용에 두면 됩니다.

**무엇을 적을까** — 오늘 AI가 틀렸던 것, 내가 내린 결정과 그 이유, 쓰지 말라고 한 것. 즉 **"아까 말했잖아"**가 나온 순간이 곧 여기 적을 거리입니다.

길게 쓰지 마세요

200줄이 넘어가면 오히려 손해입니다. 매 대화마다 통째로 읽히기 때문에 토큰을 계속 먹고, 길수록 안 지켜집니다 (04장의 lost in the middle). **짧고 구체적으로**. "포맷 예쁘게"보다 "들여쓰기는 공백 2칸"이 훨씬 잘 지켜집니다.

여기에 하나씩 쌓이는 게 **복리**입니다. 같은 피드백을 딱 한 번만 하게 만드는 것 — 이게 1주차에 오래 걸리던 일이 4주차에 한마디로 끝나는 이유입니다.

---

스스로 점검

- ☐ 모델(Core)과 런타임(Claude Code)의 역할 차이를 한 문장으로 설명할 수 있는가?
- ☐ API와 MCP는 결국 무슨 관계인가? (누가 누구를 호출하는가)
- ☐ MCP를 많이 붙일 때 생기는 비용상의 부작용은?

---

**직접 해보기** — 내 업무에서 매일 들어가는 서비스 하나를 고르고, 그것의 MCP가 있는지 검색한 뒤 붙였을 때 AI에게 시킬 일 세 가지를 적으세요.

---

## Git & GitHub — 코드의 타임머신

"되돌릴 수 있다"는 안정감이 있어야 마음껏 시도할 수 있습니다.

### Git이란

#### 한줄 정의

파일이 **언제·무엇이·어떻게 바뀌었는지**를 기록해두는 버전 관리 시스템(VCS). 게임의 세이브 포인트처럼, 원하는 시점으로 언제든 되돌아갈 수 있습니다.

게임 세이브 = 보스전 직전에 저장 → 죽어도 그 지점부터 다시

Git = 코드가 잘 돌아갈 때 저장(commit) → 망가져도 그 순간으로 복귀

### 왜 필요한가

- "어제는 됐는데 오늘 안 돼요" → 어제 잘 되던 버전으로 즉시 복귀
- "이것저것 고치다 다 꼬였어요" → 마지막 저장 지점으로 되돌리기
- 여러 명이 같은 코드를 동시에 고쳐도 충돌 없이 합칠 수 있음
- 파일 이름을 `최종_진짜최종_v2_real.zip` 으로 안 만들어도 됨

### Git과 GitHub는 다릅니다

| 구분  | GIT              | GITHUB                |
|-----|------------------|-----------------------|
| 정체  | 내 컴퓨터에서 도는 기록 도구 | Git 기록을 인터넷에 올려두는 사이트 |
| 비유  | 사진 찍는 카메라        | 사진 올리는 클라우드           |
| 없으면 | 되돌리기·변경 기록이 안 됨  | 백업·공유·협업이 안 됨         |

즉 **Git으로 저장(기록)하고 → GitHub에 올려서(백업·공유)** 합니다.

### 핵심 세 단어

| 단어     | 하는 일                        |
|--------|-----------------------------|
| commit | "지금 이 상태를 저장!" — 세이브 포인트 찍기 |
| push   | 내 저장 기록을 GitHub로 올리기 (백업)   |
| pull   | GitHub의 최신 기록을 내 컴퓨터로 내려받기  |

명령어를 외울 필요는 없습니다. Claude Code에게 그냥 말하면 됩니다 — "방금 작업한 거 commit하고 GitHub에 push해줘", "아까 잘 되던 버전으로 되돌려줘". AI가 알아서 `git` 명령어를 실행하니 우리는 **개념만** 이해하면 충분합니다.

---

스스로 점검

- ☐ Git과 GitHub의 차이를 카메라와 클라우드에 빔대어 설명할 수 있는가?
- ☐ commit · push · pull 중 "내 컴퓨터 안에서만" 일어나는 것은?
- ☐ 계획을 승인하고 auto 로 넘기기 전에 commit 을 해두라고 하는 이유는? (03장과 연결)

---

**직접 해보기** — 프로젝트 폴더에서 Claude Code에게 "지금 상태를 commit해줘"라고 시키고, 파일 하나를 일부러 망가뜨린 뒤 "아까 commit한 상태로 되돌려줘"까지 해보세요.

---

## localhost에서 진짜 주소까지

내 컴퓨터에서만 돌던 화면에 인터넷 주소를 붙이는 과정. 그리고 그 전에 열쇠를 숨기는 법.

### localhost — 내 컴퓨터 안의 미리보기

#### 한줄 정의

`http://localhost:3000` 같은 주소로, 내 컴퓨터 안에서만 열리는 웹페이지. 내 컴퓨터가 잠깐 서버 역할을 하는 것입니다.

- 뒤에 붙는 숫자는 **포트(port)** — 내 컴퓨터 안에서 어떤 서버인지 구분하는 번호입니다.
- 장점: 수정하면 즉시 반영(새로고침만) → 만들면서 확인하기 좋음
- 단점: 내 컴퓨터를 끄면 사라지고, 다른 사람은 접속할 수 없음

AI에게 "네가 만드는 거 보고 싶으니까 localhost로 띄워서 보여줘"라고 하면 실시간으로 확인할 수 있습니다.

#### 가장 흔한 오해 두 가지

"localhost 링크를 친구에게 보내면 보겠지?" → 아닙니다. 친구 컴퓨터의 localhost는 내 서비스가 아닙니다.

"HTML 파일만 있으면 서비스가 다 되겠지?" → 간단한 소개 페이지는 됩니다. 로그인·DB·결제 같은 건 서버와 배포가 필요합니다.

### HTML 파일 — 파일 하나짜리 웹페이지

`index.html` 같은 파일 하나를 더블클릭하면 브라우저에 화면이 뜹니다. 서버 없이 열리고 전달·보관이 쉽지만, 버튼 클릭·로그인·DB 저장 같은 앱 기능에는 한계가 있어 **한 장짜리 소개·랜딩 페이지**에 적합합니다.

| 상황             | 쓸 것                                  |
|----------------|--------------------------------------|
| 만드는 중 (연습·테스트) | localhost로 미리보기                      |
| 한 장짜리 소개·포스터   | HTML 파일 1개                           |
| 링크로 공유·제출      | 배포해서 <code>https://...</code> 주소 만들기 |

### 세 가지 도구 — GitHub · Vercel · Supabase

#### 한 줄로

**GitHub** = 내 코드를 보관하는 인터넷 창고

**Vercel** = 만든 걸 인터넷에 띄워주는 집 주소(URL) 발급소

**Supabase** = 데이터를 저장·관리해주는 창고 겸 관리실(데이터베이스)

결과물을 ① 안전하게 보관하고(GitHub) → ② 공유할 URL로 띄우고(Vercel) → ③ 회원가입·글 저장 같은 기능을 붙이려면(Supabase) 이 셋이 필요합니다. 대부분은 AI가 알아서 해주니 **가입만** 해주면 됩니다.

## GitHub — 코드의 클라우드

- 내가 만든 코드를 인터넷에 **백업**. 노트북이 망가져도 코드는 안전합니다.
- **버전 기록**이 남아 잘못 고쳐도 예전 상태로 되돌릴 수 있습니다.
- 다른 컴퓨터에서 **이어서 작업**할 때 여기서 내려받습니다(09장).

## Vercel — 내 사이트를 세상에 띄워주는 곳

- localhost에서만 돌던 화면을 누구나 **접속 가능한 URL**로 만들어 줍니다.
- GitHub에 올린 코드를 자동으로 가져와 **몇 초 만에 배포**합니다.
- 글로벌 CDN, 수정할 때마다 **프리뷰 URL** 자동 생성, 커스텀 도메인 연결도 간단합니다.

## Supabase — 데이터 창고 + 로그인 담당

- 회원가입, 글·사진 저장, 좋아요 수 같은 **기억해야 하는 정보**를 보관하는 데이터베이스(PostgreSQL 기반)
- **인증(Auth)** — Google, GitHub 등 소셜 로그인을 간단히 구현
- **스토리지** — 이미지·파일 업로드/다운로드
- **Realtime** — 데이터 변경을 실시간 구독 (채팅, 알림 등)

흐름은 **GitHub(보관) → Vercel(배포) → Supabase(데이터)**입니다.

## 배포하기

프론트엔드만 필요하면 Vercel만. 데이터베이스·백엔드도 필요하면 Supabase까지. 이것들을 **전부 CLI**로 진행해서 AI가 나 대신 하게 시킵니다.

복사해서 붙여넣을 프롬프트

supabase.com/docs, vercel.com/docs 의 CLI를 설치해서 CLI를 활용해서 배포를 진행하자. github, supabase와 vercel은 다 가입했고 비개발자임을 고려해서 진행해줘.

막힐 때 쓰는 문장

"네가 직접 CLI로 진행해. 로그인은 내가 해줄게."  
더 일반화하면 → "네가 직접 하려면 내가 뭘 해줘야 해?"

이 두 번째 문장이 이 부트캠프에서 가장 많이 쓰게 되는 프롬프트입니다. AI가 못 하는 일이 있을 때 "그럼 내가 뭘 해주면 되냐"로 되묻는 습관을 들이세요.

## 환경변수 `.env` — 열쇠를 코드 밖에 숨기기

한줄 정의

프로그램이 실행될 때 필요한 **비밀 정보(API 키, DB 비밀번호 등)**를 코드 밖에 따로 보관해두는 파일.

`.env` = 집 현관의 열쇠 보관함. 집(코드)은 누구나 볼 수 있지만, 열쇠(API 키)는 보관함 안에 숨겨둡니다.

이렇게 쓰면 안 됨

이렇게 써야 함

```
const API_KEY = "sk-abc123..."
```

```
const API_KEY = process.env.OPENAI_API_KEY
```

## GitHub에 올리면 전 세계가 내 키를 봄

`.env` 파일은 Git에 올리지 않음

누가 훔쳐 쓰면 내 돈이 빠져나감

나만 볼 수 있게 로컬에 보관

프로젝트 폴더 **최상단(루트)**에 `.env` (또는 `.env.local`) 파일을 만들고 `KEY=VALUE` 형태로 한 줄에 하나씩, 따옴표 없이 씁니다. 키 이름은 보통 대문자와 언더스코어입니다.

## · ENV 예시

```
OPENAI_API_KEY=sk-proj-...  
SUPABASE_URL=https://xxxx.supabase.co  
SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZWQifQ%3D%3D  
DATABASE_URL=postgresql://user:pass@host:5432/db
```

## 가장 중요한 규칙 - .GITIGNORE

`.env` 는 절대로 GitHub에 올라가면 안 됩니다. 반드시 `.gitignore` 파일에 아래를 추가하세요.

```
.env
.env.local
.env.*.local
```

실수로 올라간 키는 이미 유출된 것입니다. 지우기만 해서는 안전해지지 않습니다. 해당 서비스에서 키를 **폐기 (revoke)**하고 새로 발급받으세요.

## 비개발자가 가장 헤매는 네 가지

1. 파일 이름이 점(.)으로 시작해 Mac/Windows에서 숨김 파일로 취급됩니다.
2. 반드시 프로젝트 루트 폴더에 있어야 합니다. 하위 폴더는 안 됩니다.
3. `.env`를 수정하면 개발 서버를 꺾다 켜야 반영됩니다.
1. 배포할 때는 파일이 아니라 Vercel 대시보드에서 환경변수를 따로 입력해야 합니다.

## AI에게 시킬 문장

OpenAI API 키를 받았는데 .env에 추가하고, .gitignore에도 넣어줘.  
그리고 이 키를 src/api.js에서 사용하도록 수정해줘.

스스로 점검

- ☐ localhost 주소를 친구에게 보내면 왜 안 열리는가?
- ☐ GitHub · Vercel · Supabase가 각각 맡는 역할 한 단어씩은?
- ☐ `.env` 가 실수로 GitHub에 올라갔다면 무엇을 해야 하는가? (지우는 것으로 충분한가?)

**직접 해보기** — 배포 프롬프트를 그대로 붙여넣어 내 프로젝트에 실제 URL을 붙이고, 그 주소를 표지 여백에 적으세요.

## 집에서 이어서 작업하기

부트캠프에서 만든 걸 집 컴퓨터로 옮겨 계속 만드는 절차. 이 책에서 가장 실용적인 장입니다.

### 한줄 요약

개념

**GitHub = 코드를 올려두는 인터넷 창고.** 지금 노트북에서 올리고(push) → 집 컴퓨터에서 내려받으면(clone) 똑같은 프로젝트를 이어서 작업할 수 있습니다. USB로 파일 옮기는 것과 같은데, 인터넷으로 한다고 보면 됩니다.

GitHub = 인터넷 사물함

지금 노트북 = 작업물을 사물함에 넣기 (push)

집 컴퓨터 = 사물함에서 꺼내서 이어서 작업 (clone → pull)

### 1 — 지금 노트북에서 GitHub에 올리기 (push)

가장 쉬운 방법은 Claude Code에게 시키는 것입니다.

복사해서 붙여넣을 프롬프트

내 프로젝트를 GitHub에 새 repo로 올리고 싶어. 비개발자니까  
git 초기화 → commit → GitHub repo 생성 → push 까지 전부 CLI로 진행해줘.  
GitHub은 가입돼 있어. 로그인이 필요하면 내가 할 수 있게 알려줘.

직접 명령어로 하고 싶다면

```
cd 내프로젝트폴더
git init
git add .
git commit -m "first commit"
gh repo create my-project --private --source=. --push
```

또는 GITHUB 웹사이트에서 빈 REPO를 먼저 만든 뒤

```
git remote add origin https://github.com/내아이디/my-project.git
git branch -M main
git push -u origin main
```

### 2 — 집 컴퓨터에서 내려받기 (clone)

먼저 집 컴퓨터에도 **Claude Code · git · Node.js**를 설치하세요(부록 C의 설치 가이드). 그다음 터미널에 복사 → 붙여넣기 → Enter:

```
cd ~
git clone https://github.com/내아이디/my-project.git
cd my-project
npm install
claude
```

.ENV 파일은 GITHUB에 안 올라갑니다

API 키·비밀번호가 든 `.env`는 보안을 위해 `.gitignore`로 제외돼 있습니다. 그래서 집 컴퓨터에서는 `.env` 파일을 따로 복사해서 프로젝트 최상단에 직접 만들어야 합니다. (08장 참고)

### 3 — 두 컴퓨터를 계속 동기화하는 습관

작업 시작 전 — 최신 버전 받기

```
git pull
```

작업 끝난 후 — 내 변경사항 올리기

```
git add .
git commit -m "오늘 작업한 내용 메모"
git push
```

이것만 지키면 됩니다

한 컴퓨터에서 작업이 끝나면 항상 **push**, 다른 컴퓨터에서 시작 전 항상 **pull**. 이 두 가지면 두 컴퓨터가 늘 같은 상태로 유지됩니다.

헛갈리면 Claude Code에게 "방금 작업한 거 commit하고 push해줘" / "GitHub에서 최신으로 pull해줘"라고 말하면 알아서 해줍니다.

스스로 점검

- ☐ 집 컴퓨터에서 clone한 뒤에도 앱이 안 돌아간다면, 가장 먼저 의심할 파일은?
- ☐ 작업 시작 전과 끝난 후에 각각 무엇을 해야 하는가?
- ☐ `git clone` 과 `git pull` 은 언제 각각 쓰는가?

**직접 해보기** — 오늘 만든 프로젝트를 GitHub에 올리고, 다른 폴더에 clone해서 `npm install` → 실행까지 되는지 확인하세요. 여기까지 되면 어디서든 이어서 작업할 수 있습니다.

# 더 잘하기

돌아가는 것을 만든 다음에 오는 것들 — 보기 좋게, 빠르게, 그리고 자동으로.

## 디자인을 이쁘게 하고 싶다면

아래 사이트에서 마음에 드는 디자인을 찾아 "이렇게 만들고 싶어"라고 AI에게 보여주세요. 참고 대상이 있을 때와 없을 때의 결과 차이가 큼니다. (주소는 부록 C)

| 분류    | 사이트                  | 무엇이 있나                      |
|-------|----------------------|-----------------------------|
| 앱 UI  | Mobbin               | iOS/Android/Web UI 패턴 라이브러리 |
| 앱 UI  | Screensdesign        | 인기 iOS 앱 전체 플로우 영상          |
| 앱 UI  | Wwit.design          | 한국 앱 위주 UI 모음               |
| AI 패턴 | The Shape of AI      | AI UX 패턴 전문 라이브러리           |
| AI 패턴 | KoruUX – AI Patterns | AI 인터페이스 14가지 핵심 UX 패턴      |
| 웹     | Awwwards             | 최고 수준 UI/UX 수상작             |
| 웹     | Dribbble / Behance   | AI 앱·제품 디자인 시안과 포트폴리오       |

## 작업 속도를 올리는 것들

- 터미널 앱 — Mac은 Ghostty, Windows는 Wave Terminal을 권합니다(기본 터미널보다 쾌적합니다).
- Windows에서 스크린샷 붙여넣기 — `Ctrl+V`가 아니라 `Alt+V`입니다. Mac은 `⌘C` → `⌘V` 또는 드래그 앤 드롭이 그대로 됩니다.
- 타자가 느리다면 — 음성 입력 도구(Wispr Flow 등)를 쓰면 AI에게 시키는 속도 자체가 달라집니다.
- 세션 복습 — `/insights 한국어로 작성해줘` 로 그날 세션을 분석·정리받을 수 있습니다.

## 다음 단계 — 상시 가동 에이전트

Claude Code 같은 세션 단위 도구 다음에 오는 것이 상시 가동 에이전트입니다. 지금 당장 필요하진 않지만, 방향을 알고 있으면 좋습니다.

| 구분     | 세션형 (CLAUDE CODE · OPENCLAW) | 상시형 (HERMES 계열)                      |
|--------|------------------------------|--------------------------------------|
| 사는 곳   | 내 노트북 터미널                    | 서버 · VPS · 클라우드                      |
| 가동 시간  | 내가 켜 있을 때만                   | 24시간 상시                              |
| 메모리    | 세션 단위 (종료하면 휘발)              | 영속 — 세션이 끝나도 유지                      |
| 학습     | 메모 파일에 수동 저장                 | 스킬을 스스로 만들어 재사용                      |
| 말 거는 법 | 터미널 한 곳                      | Telegram · Slack · Discord · Email 등 |
| 역할     | 코딩 페어 프로그래머                  | 자율 운영 에이전트                           |
| 비유     | 부르면 오는 조수                    | 내 서버에 사는 룸메이트                        |

**언제 무엇을** — 코드를 직접 짜는 작업, 내 옆에서 같이 일하는 페어 프로그래머가 필요할 때는 세션형. 자리를 비워도 일이 돌아가야 할 때(밤새 리서치, 24시간 모니터링), 여러 채널로 알림이 필요한 자동화, 매일 반복되는 업무에는 상시형.

업계는 "세션 단위 도구"에서 "상시 가동 페르소나"로 무게중심이 옮겨가는 중입니다. 에이전트가 충분히 똑똑해졌고, 컴퓨트 단가가 내려갔고, 사람이 노트북 앞에 24시간 있을 수 없기 때문입니다. **"Claude Code로 만들고, 상시 에이전트로 운영한다"**가 점점 표준이 되고 있습니다.

### Harness — 에이전트에게 붙이는 장비

같은 모델이라도 **뭘 붙이느냐**에 따라 결과가 달라집니다. 네 가지만 구분하면 됩니다.

|                 | 무엇                              | 언제 쓰나                      |
|-----------------|---------------------------------|----------------------------|
| <b>Skill</b>    | 자주 하는 작업 방식을 저장해두고 이름으로 불러 쓰는 것 | 같은 절차를 반복하고 있을 때           |
| <b>Plugin</b>   | 스킬·명령어·설정을 묶어서 통째로 설치하는 것       | 남이 만든 검증된 세트를 그대로 쓰고 싶을 때  |
| <b>Hook</b>     | 특정 순간에 자동으로 실행되는 규칙             | "저장할 때마다 검사" 처럼 잊으면 안 되는 것 |
| <b>Subagent</b> | 큰 일을 나눠서 여러 개가 동시에 처리하는 것       | 혼자 하면 오래 걸리는 조사·검토         |

지금 당장 할 일은 아닙니다

이건 **"이런 게 있다"**만 알아두는 장입니다. 필요해지는 순간이 오면 그때 찾으세요. 대부분은 `CLAUDE.md` (06장) 하나만 잘 써도 충분히 멀리 갑니다.

그리고 이 전부를 **Claude Code 하나가 부립니다**. 이미지가 필요하면 이미지 모델을 부르고, 일이 크면 자기 분신을 여러 개 띄웁니다. 내가 배울 건 *창구 하나 다루는 법*이지 도구 목록이 아닙니다.

오늘 배운 걸 스스로 정리해주는 학습 스킬도 있습니다. 구체적인 저장소 주소는 부록 C에 모아두었습니다.

---

스스로 점검

- ☐ 디자인 결과를 끌어올리는 가장 간단한 방법은 무엇인가?
- ☐ 세션형 에이전트와 상시형 에이전트를 가르는 결정적 차이 두 가지는?
- ☐ 내 업무 중 "자리를 비워도 돌아가야 하는" 일이 있는가?

---

**직접 해보기** — 디자인 갤러리에서 마음에 드는 화면을 하나 캡처해 Claude Code에 붙여넣고 "이 느낌으로 바꿔줘"를 시켜보세요.

---

## 가장 크게 남길 것

기술은 6개월이면 바뀝니다. 바뀌지 않는 것만 적었습니다.

### 다섯 가지

1. **건강관리.** 아직 마라톤 초입부입니다. 잘 돼서 아프면 안 되고, 오래 해야 하니까요.
2. **Unlearn, then learn.** 배우는 것보다 *지금까지의 방식을 내려놓는* 것이 먼저입니다.
3. **앞으로가 중요합니다.** 결국 자동화를 해내는 사람은 의지·집념·분노가 있는 사람입니다. 똑똑하게 게으르게 일하세요.
4. **왜 이걸 하는가.** 잠재력을 여는 것. Unlock your full potential.
5. **제일 무서운 건 가속도입니다.** 지금 속도가 아니라, 속도가 붙는 속도를 보세요.

### 결국 남는 문장 하나

#### 기억할 것

이제 경쟁력은 **"내가 얼마나 잘 만드느냐"**가 아니라 **"AI에게 얼마나 잘 시키느냐"**입니다. 그리고 그건 오늘 하루의 연습이 아니라, 앞으로의 습관에서 갑니다.

#### 마지막 점검

- ☐ 이 열한 장 중 지금 당장 다시 읽어야 할 장은 어디인가?
- ☐ 이번 주에 AI에게 넘길 반복 업무 하나를 정했는가?
- ☐ 다음에 만들 것을 한 문장으로 쓸 수 있는가?

**직접 해보기** — 아래 여백에 "다음에 만들 것"과 "이번 주에 AI에게 넘길 업무"를 각각 한 줄로 쓰세요. 이 책의 마지막 과제입니다.

다음에 만들 것

AI에게 넘길 업무

# 명령어 · 슬래시 치트시트

## 터미널 명령어

| 하는 일                   | WINDOWS                            | MACOS                       |
|------------------------|------------------------------------|-----------------------------|
| 현재 위치(폴더 경로) 보기        | <code>pwd</code>                   | <code>pwd</code>            |
| 현재 폴더 안 파일 목록          | <code>ls</code> / <code>dir</code> | <code>ls</code>             |
| 그 폴더로 이동               | <code>cd</code> 폴더명                | <code>cd</code> 폴더명         |
| 한 단계 위(부모)로            | <code>cd ..</code>                 | <code>cd ..</code>          |
| 공백 있는 폴더명              | <code>cd "My Folder"</code>        | <code>cd "My Folder"</code> |
| 홈 폴더를 탐색기/Finder로      | <code>start .</code>               | <code>open ~</code>         |
| 현재 폴더에서 Claude Code 실행 | <code>claude</code>                | <code>claude</code>         |

## Claude Code 슬래시 명령

| 명령                       | 하는 일                                                 |
|--------------------------|------------------------------------------------------|
| <code>/model</code>      | 모델 · 추론 강도 변경                                        |
| <code>/status</code>     | 모드 · 폴더 · 로그인 방식 · 컨텍스트 확인                           |
| <code>/plan</code>       | plan 모드 진입 (또는 <code>Shift+Tab</code> )              |
| <code>/compact</code>    | 대화를 요약해 컨텍스트 공간 확보 — <b>같은 작업</b> 을 이어갈 때            |
| <code>/clear</code>      | 대화를 비우고 새로 시작 — <b>새 작업</b> 을 시작할 때                  |
| <code>/rewind</code>     | 대화·파일 변경을 이전 지점으로 되돌리기 ( <code>Esc</code> 두 번으로도 열림) |
| <code>/resume</code>     | 이전 세션 이어서 작업                                         |
| <code>/rename</code>     | 세션 이름 바꾸기                                            |
| <code>/btw</code>        | 결가지 질문 — 끝나면 <code>Esc</code> 로 본 세션 복귀              |
| <code>/login</code>      | 계정 연결 확인 · 다시 로그인                                    |
| <code>/skills</code>     | 쓸 수 있는 Skill 목록                                      |
| <code>/config</code>     | 테마 · 기본 권한 모드 설정                                     |
| <code>/statusline</code> | 화면 아래 상태줄 설정                                         |
| <code>/usage</code>      | 토큰 · 비용 상세 보기                                        |
| <code>/exit</code>       | 종료 (또는 <code>Ctrl + C</code> 두 번)                    |

권한 모드 — `Shift+Tab` 을 누를 때마다 `Manual` → `acceptEdits` → `plan` 순으로 돕니다. 계정에 따라 `auto` 가 뒤에 하나 더 붙습니다.

| 입력창에서             | WINDOWS                   | MACOS                       |
|-------------------|---------------------------|-----------------------------|
| 줄바꿈 (전송 말고)       | <code>Ctrl + Enter</code> | <code>Option + Enter</code> |
| 스크린샷 붙여넣기         | <code>Alt + V</code>      | <code>Cmd + V</code>        |
| 답변 중단             | <code>Esc</code>          | <code>Esc</code>            |
| 입력창 비우기 · 되돌리기 메뉴 | <code>Esc</code> 두 번      | <code>Esc</code> 두 번        |
| 세션 종료             | <code>Ctrl + C</code> 두 번 | <code>Ctrl + C</code> 두 번   |

더 깊게 생각시키려면 프롬프트에 `think` · `think hard` · `think harder` · `ultrathink` 를 넣습니다. 볼 파일을 짚어줄 땐 `@파일이름` .

## Git 명령어

| 명령어                             | 하는 일                     |
|---------------------------------|--------------------------|
| <code>git init</code>           | 이 폴더를 Git으로 관리 시작        |
| <code>git add .</code>          | 변경된 파일 전부를 저장 대상으로       |
| <code>git commit -m "메모"</code> | 세이프 포인트 찍기               |
| <code>git push</code>           | GitHub에 올리기              |
| <code>git pull</code>           | GitHub에서 최신으로 내려받기       |
| <code>git clone 주소</code>       | GitHub 저장소를 내 컴퓨터로 처음 복제 |

## 자주 쓰는 프롬프트

- "네가 직접 하려면 내가 뭘 해줘야 해?"
- "네가 만드는 거 보고 싶으니까 localhost로 띄워서 보여줘"
- "방금 작업한 거 commit하고 push해줘"
- "아까 잘 되던 버전으로 되돌려줘"
- "간결하게, 수정된 부분만 보여줘" (토큰 절약)
- "ultrathink해서 왜 이 버그가 나는지 분석해줘"

# 용어집

모르는 말이 나오면 여기서 찾으세요. 여기에도 없으면 그 단어 그대로 AI에게 물어보면 됩니다.

## 도구 · 환경

| 용어               | 쉽게 말하면                                                   |
|------------------|----------------------------------------------------------|
| CLI              | 글자로 명령하는 검은 창 (타이핑으로 시킴)                                 |
| GUI              | 마우스로 클릭하는 보통 화면 (폴더 · 버튼)                                |
| 터미널 / PowerShell | CLI를 입력하는 앱. Mac은 터미널, Windows는 PowerShell               |
| Node.js          | AI 코딩 도구를 돌리기 위한 실행 환경(엔진)                               |
| npm              | Node 도구 설치 관리자. <code>npm install -g</code> 는 컴퓨터 전체에 설치 |
| 에이전트             | 스스로 파일을 읽고 · 만들고 · 실행하는 AI                               |
| Git              | 파일 변경 기록 · 되돌리기 도구                                       |
| GitHub           | Git 저장소를 인터넷에 올려두는 곳 (코드의 클라우드)                          |
| repo (저장소)       | 프로젝트 파일이 담긴 폴더 묶음                                        |
| clone (클론)       | GitHub 저장소를 내 컴퓨터로 복제해 받기                                |
| Markdown / .md   | 기호로 제목 · 표를 표현하는 가벼운 문서 형식                               |
| localhost        | 내 컴퓨터 안에서만 열리는 임시 미리보기 주소                                |
| 포트(port)         | 주소 뒤 숫자. 내 컴퓨터 안에서 서버를 구분하는 번호                           |
| RAG              | AI가 지어내지 않게 신뢰 자료를 먼저 찾아 답하게 하는 방식                       |

# AI 운영 용어

| 용어                       | 쉽게 말하면                           |
|--------------------------|----------------------------------|
| 프롬프트                     | AI에게 주는 요청 · 지시문                 |
| harness (하네스)            | AI가 헤매지 않게 도구 · 규칙 · 절차를 묶어둔 작업대 |
| skill (스킬)               | 반복 작업을 정해둔 절차 묶음                 |
| context / token          | AI가 한 번에 보는 정보(맥락) / 그 처리 단위     |
| context window           | 컨텍스트를 담을 수 있는 최대 용량 (AI의 책상 크기)  |
| compacting               | 길어진 대화를 요약해 공간을 확보하는 것           |
| effort / fast            | AI가 얼마나 깊게 생각할지 조절. 가벼운 일은 fast  |
| MCP                      | AI를 외부 서비스에 연결하는 표준 플러그인 규격      |
| API                      | 프로그램끼리 정해진 규격으로 주고받는 소통 창구       |
| Subagents                | 특정 작업을 전담하는 보조 에이전트              |
| Hooks                    | 특정 이벤트가 생기면 자동 실행되는 스크립트         |
| 환경변수 / <code>.env</code> | API 키 등 비밀 정보를 코드 밖에 보관하는 파일     |
| <code>.gitignore</code>  | Git이 올리지 말아야 할 파일 목록             |

## 창업 · 사업 용어

| 용어              | 쉽게 말하면                                    |
|-----------------|-------------------------------------------|
| MVP             | 검증에 꼭 필요한 최소 기능 제품                        |
| BMC             | 사업 구조를 9칸으로 정리한 한 장 (비즈니스 모델 캔버스)         |
| GTM             | 어떻게 시장 · 첫 고객에게 들어갈지 전략                   |
| 퍼널              | 인지 → 관심 → 구매 → 재구매로 좁아지는 깔때기              |
| CAC             | 고객 1명 데려오는 비용                             |
| LTV             | 고객 1명이 평생 주는 매출 ( $LTV/CAC \geq 3$ 이면 양호) |
| TAM / SAM / SOM | 전체 / 도달 가능 / 1년 내 현실 목표 시장                |
| unit economics  | 고객 1명당 벌고 쓰는 돈이 남는지 (단위 경제성)              |
| WTP             | 지불 의향 — 얼마면 낼지                            |
| 페르소나            | 가상의 대표 고객 1명                              |
| 레드팀 / VC 리뷰     | 투자자처럼 약점을 일부러 공격해 점검                      |
| 합성 소비자          | AI로 만든 가상 고객 패널 (가설 점검용, 진짜 검증은 아님)       |
| 피어 리뷰           | 동료끼리 서로 산출물 점검 · 피드백                      |
| 피치              | 아이디어를 짧고 설득력 있게 발표                        |

# 링크 모음

본문에서는 읽기를 위해 주소를 뺐습니다. 종이에서 옮겨 칠 수 있도록 여기 모았습니다.

## 설치 · 시작

|                          |                                                                                             |
|--------------------------|---------------------------------------------------------------------------------------------|
| Claude Code 설치 가이드 (한국어) | <a href="https://ai.retn.kr/claudecode">ai.retn.kr/claudecode</a>                           |
| Claude Code 공식 퀵스타트      | <a href="https://code.claude.com/docs/en/quickstart">code.claude.com/docs/en/quickstart</a> |

## 가입해야 하는 서비스

|             |                                                           |
|-------------|-----------------------------------------------------------|
| GitHub      | <a href="https://github.com">github.com</a>               |
| Vercel      | <a href="https://vercel.com">vercel.com</a>               |
| Supabase    | <a href="https://supabase.com">supabase.com</a>           |
| Vercel 문서   | <a href="https://vercel.com/docs">vercel.com/docs</a>     |
| Supabase 문서 | <a href="https://supabase.com/docs">supabase.com/docs</a> |

## 디자인 레퍼런스

|                      |                                                                                                 |
|----------------------|-------------------------------------------------------------------------------------------------|
| Mobbin               | <a href="https://mobbin.com">mobbin.com</a>                                                     |
| Screensdesign        | <a href="https://screensdesign.com">screensdesign.com</a>                                       |
| Wwit.design          | <a href="https://wwit.design">wwit.design</a>                                                   |
| The Shape of AI      | <a href="https://shapeof.ai">shapeof.ai</a>                                                     |
| KoruUX — AI Patterns | <a href="https://koruux.com/ai-patterns-for-ui-design">koruux.com/ai-patterns-for-ui-design</a> |
| Awwwards             | <a href="https://awwwards.com/websites/ui-design">awwwards.com/websites/ui-design</a>           |
| Dribbble — AI UI     | <a href="https://dribbble.com/tags/ai-ui">dribbble.com/tags/ai-ui</a>                           |
| Behance — AI App     | <a href="https://behance.net/search/projects/ai%20app">behance.net/search/projects/ai app</a>   |
| Design.md            | <a href="https://getdesign.md">getdesign.md</a>                                                 |

## 도구

|                         |                                                                                             |
|-------------------------|---------------------------------------------------------------------------------------------|
| Ghostty (Mac 터미널)       | <a href="https://ghostty.org">ghostty.org</a>                                               |
| Wave Terminal (Windows) | <a href="https://waveterm.dev">waveterm.dev</a>                                             |
| Wispr Flow (음성 입력)      | <a href="https://wisprflow.ai">wisprflow.ai</a>                                             |
| Playwright (브라우저 자동화)   | <a href="https://github.com/microsoft/playwright">github.com/microsoft/playwright</a>       |
| browser-use             | <a href="https://github.com/browser-use/browser-use">github.com/browser-use/browser-use</a> |
| Chromux (Mac 브라우저 자동화)  | <a href="https://github.com/team-attention/chromux">github.com/team-attention/chromux</a>   |

## 스킬 · Harness

|                      |                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|
| 스킬 모음                | <a href="https://skills.sh">skills.sh</a>                                                                               |
| k-skill (한국어 스킬)     | <a href="https://github.com/NomaDamas/k-skill">github.com/NomaDamas/k-skill</a>                                         |
| Ouroboros (Harness)  | <a href="https://github.com/Q000/ouroboros">github.com/Q000/ouroboros</a>                                               |
| Ouroboros 한국어 README | <a href="https://github.com/Q000/ouroboros/blob/main/README.ko.md">github.com/Q000/ouroboros/blob/main/README.ko.md</a> |
| Compound Engineering | <a href="https://github.com/EveryInc/compound-engineering-plugin">github.com/EveryInc/compound-engineering-plugin</a>   |
| insane-search        | <a href="https://github.com/fivetaku/insane-search">github.com/fivetaku/insane-search</a>                               |
| 용어집 원본               | <a href="https://github.com/erickimme/ai-startup-roadmap">github.com/erickimme/ai-startup-roadmap</a>                   |

AI 코딩 셀프스터디 핸드북 · Get Ship Done

부트캠프 교육 문서에서 운영·행사 정보를 덜어내고, 집에서 혼자 복습할 수 있는 내용만 남겨 재구성한 판입니다.

도구와 화면은 바뀝니다. 막히면 이 문서의 문단을 그대로 AI에게 붙여넣고 물어보세요.