1시간 클로드 입문서

> **약속:** 코딩을 한 번도 해본 적 없어도, **60분 안에 Claude Code를 자기 손에 익힌다.**
> 매주 반복되던 일 하나를 자동화하는 것까지 책 한 권으로.

이 책은 누구를 위해 썼나

- ChatGPT는 한 번 써봤지만 **"Claude Code"는 처음** 듣는 사람
- 터미널·파일 경로·JSON이 무서운 사람
- 영수증·메일·일정·회의록처럼 **매주 반복되는 일을 줄이고 싶은** 사람
- 코딩 경험은 0이지만, AI 도구로 자기 일을 자동화하고 싶은 사람

이 책은 **"코딩을 가르치는 책"이 아니다.** 자기 일을 Claude에게 시키는 법을 가르친다.

11장의 여정 — 60분이면 손에 익는다

Part	장	무엇을 얻나	분량
Part 1. 입문	Ch1~8	Claude Code 켜기 → 자기 일 시키기 → 토큰 절약까지	약 35,000자
Part 2. 활용	Ch9~11	외부 연결·스킬·에이전트 — 더 멀리 더 자동	약 17,000자

총 약 52,000자 / 한 챕터당 4,000~6,000자.

> Part 1을 끝낸 시점에 이미 일상 작업의 70%는 Claude로 옮겨와 있다. Part 2는 그 위에 외부 서비스 연결·자동 호출 일꾼을 얹는 단계다.

학습 동선 원칙

1. **공포 → 친숙 → 활용** 순서. 절대 건너뛰지 말 것.
2. **각 장 끝에 5분 핸즈온**. 따라하지 않으면 다음 장이 어렵다.
3. **막힘은 정상**. 함정 박스와 트러블슈팅이 즉시 회복시킨다.
4. **이전 장을 모르면 못 읽는 구조 금지** — 각 장 첫 단락에 필요한 선행 개념을 30초로 요약해둠.

시작 전 체크리스트

- [ ] 인터넷이 연결되어 있다
- [ ] 브라우저(Chrome / Edge / Safari) 한 개가 깔려 있다
- [ ] Anthropic 계정 가입 가능 (Google 계정 또는 이메일)

책의 첫 약속

이 책의 마지막 페이지(Ch11)를 닫는 순간 다음 5가지가 손에 있다.

장	무엇을 가졌나
Ch1~3	Claude Code 켜고 첫 응답 + 4공식으로 시키는 손
Ch4~6	CLAUDE.md·자동화·확장 도구의 큰 그림
Ch7~8	슬래시 50+·토큰 절약 — 매일 30분 절약
Ch9~10	외부 서비스 연결·스킬로 일을 자동화하는 능력
Ch11	에이전트·서브에이전트로 메인 컨텍스트 보호

준비됐다면 **Ch1. 5분 안에 Claude Code 켜기** 으로 시작.

Ch1. 5분 안에 Claude Code 켜기

> **이 장을 끝내면 할 수 있는 것:** Claude Code를 켜고 "안녕"이라 입력해 응답을 받을 수 있다.

---

1. Claude Code의 세 가지 입구

먼저 한 가지를 분명히 짚자. **Claude Code의 "Code"는 "코드"가 아니라 "코드명" 같은 약칭**이다. 코딩만 되는 도구가 아니라, **내 컴퓨터의 파일을 직접 다루는 Claude의 모드**다. 글·표·일정·이미지·폴더 정리 — 일상 작업이 다 된다.

Claude Code를 켜는 방법은 세 가지다. 비개발자에게는 **웹 → 데스크톱 앱 → 터미널** 순서로 친숙해진다.

- **웹 (claude.ai/code)** — 브라우저에서 바로 쓰는 모드. 설치 없음. 가장 진입장벽이 낮다.
- **데스크톱 앱** — Mac/Windows에 설치하는 전용 앱. 웹과 비슷하지만 내 컴퓨터의 폴더를 더 자유롭게 다룬다.
- **터미널 (CLI)** — 검은 화면에 `claude` 라고 입력하면 켜진다. 가장 강력하지만 익숙해지는 데 가장 오래 걸린다.

이 책의 비개발자 독자는 **웹부터** 시작한다. 웹 한 가지로 책의 절반을 끝낼 수 있다. 데스크톱과 터미널은 손에 익은 다음에 자연스럽게 옮겨가면 된다.

이 장이 책에서 다뤄질 가치는 한 가지다. **켜는 데 5분이면 충분하다.** 첫 응답을 받은 사람은 절반 이상이 그날 안에 두 번째·세 번째 질문을 던진다.

---

2. 무엇과 어떻게 다른가

2-1. 웹 vs 데스크톱 앱 vs 터미널

축	웹 (claude.ai/code)	데스크톱 앱	터미널 (`claude`)
설치	없음 (브라우저만)	한 번 설치 (5분)	한 번 설치 (10분)
진입 장벽	매우 낮음	낮음	보통
폴더 접근	첨부 + Projects	폴더 통째 가능	폴더 통째 + 자동화 강력
다중 작업	탭으로 분리	창으로 분리	동시 여러 세션
이 책의 비중	**Ch1~5 전체**	Ch6 후반	Ch9 이후

처음 몇 주는 웹만 써도 된다. 익숙해지면 데스크톱 앱을, 자동화가 필요해지면 터미널을 추가한다.

2-2. "터미널 (CLI)"은 다시 셋으로 나뉜다

비개발자에게 가장 헷갈리는 게 터미널이다. 사실 우리가 만나는 검은 입력창은 세 종류로 분리해 보면 무섭지 않다.

입구	무엇인가	어디서
① 운영체제 기본 터미널	macOS의 "터미널" 앱, Windows의 "PowerShell"	자체 앱으로 따로 띄움
② **VS Code 안의 터미널**	VS Code 에디터 하단에 붙어 있는 같은 종류의 검은 창	VS Code 메뉴 → Terminal → New Terminal
③ **VS Code 확장 앱 (Claude Code)**	VS Code 안에서 채팅 UI로 Claude Code를 쓰는 패널	VS Code 좌측 사이드바 → Claude Code 아이콘

세 입구 모두 **같은** `claude` **명령**을 받는다. 다만 ③은 검은 화면 대신 채팅창으로 감싼 형태라 비개발자에게 가장 부드럽다. 책의 후반부는 ②번을 표준으로 가정한다.

2-3. 켜는 순서 — 비개발자 표준

단계	행동	시간
1	브라우저에서 https://claude.ai/code 접속	10초
2	Anthropic 계정 로그인 (Google 또는 이메일)	1분

단계	행동	시간
3	첫 입력창에 "안녕" 입력 → 엔터	5초
4	응답 확인	5초
5	모델 선택 메뉴 위치 확인 (좌상단 또는 입력창 옆)	30초

총 2분 미만. 5분이라 한 건 로그인이 처음인 경우의 여유 시간이다.

---

3. ⚠️ 처음 사람들이 자주 막히는 곳

함정 1. 로그인이 안 되어 있어 다른 페이지로 흘러간다

https://claude.ai/code 주소를 처음 누르면 로그인 페이지로 리다이렉트된다. 새 창에서 결제 페이지처럼 보일 수 있다 (**Pro 플랜** 권유). **Free 플랜으로도 Claude Code 웹의 기본 사용은 가능**하니, 결제 페이지가 떠도 작은 글씨의 "Free로 시작" 또는 "건너뛰기"를 찾아 누른다.

함정 2. 모델 선택 메뉴를 못 찾는다

UI는 가끔 바뀌지만, 보통은 **좌상단** 또는 **입력창 바로 옆**에 작은 모델 이름(`Sonnet 4.6` 같은)이 적혀 있다. 클릭하면 Opus·Sonnet·Haiku 중 고를 수 있는 메뉴가 펼쳐진다. 첫 사용자는 **Sonnet**이 기본이고, 그대로 두면 된다.

함정 3. 답이 길어 어디부터 봐야 할지 모를 때

답이 5~6 문단을 넘어가면 어디부터 읽어야 할지 헷갈린다. 두 가지 회피책:

- (a) 입력창에 **"위 답을 표 한 장으로 정리해줘"** 한 줄 추가
- (b) 처음 메시지에 **"한 문단으로 답해줘"** 같은 형식 지시를 함께 넣기

Ch3의 4공식 중 **FORMAT**이 정확히 이 문제를 푸는 도구다.

---

4. 5분 미니 실습 — 회의 안건 정리로 첫 응답 받기

전제

- 인터넷 연결과 브라우저
- Google 계정 또는 이메일 (가입용)

단계

1. 브라우저에서 https://claude.ai/code 를 연다.

   ✓ 체크포인트: 로그인 페이지가 뜨거나, 이미 로그인되어 있으면 입력창이 보인다.

2. 로그인한다 (처음이면 가입).

   - Google 계정으로 가입이 가장 빠르다 (10초)
   - 이메일이라면 인증 메일을 확인한다

   ✓ 체크포인트: Claude Code 메인 화면(입력창 + 좌측 사이드바)이 보인다.

3. 입력창에 다음을 친다.

   ```
   안녕
   ```

   엔터.

   ✓ 체크포인트: 1~3초 후 응답이 뜬다.

4. 좌상단(또는 입력창 옆)의 모델 이름을 클릭해 메뉴를 펼친다.

   ✓ 체크포인트: Opus / Sonnet / Haiku 중 선택할 수 있는 메뉴가 보인다.

   다시 닫는다 (선택은 그대로 둔다).

5. 두 번째 메시지로 자기 일 한 줄을 시킨다. 회의 안건 정리가 이 책의 표준 예시다.

   ```
   내일 회의 안건 5개를 정해야 해.
   일반적인 스타트업 주간 회의에서 다룰 만한 항목 5개만 표로 만들어줘.
   ```

   엔터.

   ✓ 체크포인트: 표 형태의 응답이 뜬다.

결과 인증

이 실습을 끝냈다면:

- Claude Code 계정과 첫 대화 이력이 생겼다
- 모델 선택 메뉴 위치를 안다
- **Ch2로 넘어갈 준비가 끝났다.** Ch2부터 "무엇을 어떻게 시켜야 하나"를 다룬다.

⚠️ 트러블슈팅

문제 1: 로그인 후 아무 화면도 안 뜹니다. → 브라우저 캐시 문제. 페이지 새로고침(F5) 또는 시크릿 창에서 다시 시도.

문제 2: 모델 선택 메뉴가 어디에도 안 보입니다. → 좁은 화면(모바일·작은 노트북)에서는 입력창 우측의 ⚙ 또는 햄버거 메뉴(≡) 안에 들어가 있을 수 있다. 클릭해 펼친다.

문제 3: "안녕" 응답이 영어로 옵니다. → 두 번째 메시지에 "한국어로 답해줘" 한 줄을 덧붙인다. 또는 처음부터 "안녕하세요"처럼 한국어 인사로 시작한다.

---

5. 한 가지 더 — 권한·비용 빠르게

권한 모드 (allow / ask / deny)

Claude Code가 처음 자기 폴더를 만지려 할 때 묻는 한 줄. 어떤 명령을 허용할지 사용자가 결정한다.

모드	의미	추천 대상
`allow`	이 명령은 묻지 말고 즉시 실행	자주 쓰는 안전한 명령 (`ls`, `pwd` 등)
`ask`	매번 한 번씩 확인	**비개발자 첫 사용 — 기본값 권장**
`deny`	이 명령은 절대 실행 금지	위험한 명령 (`rm -rf`, `sudo` 등)

설정 위치: `~/.claude/settings.json`의 `permissions` 항목, 또는 Claude Code 데스크톱 앱의 설정 메뉴.

비용 한 장 비교 (2026 기준)

플랜	가격(월)	무엇이 열림
Free	$0	웹 채팅 + 메시지 한도 (가벼운 사용)
Pro	$20/월 (연간 결제 시 $17/월)	메시지 한도 ↑, 데스크톱 앱, 큰 파일 첨부, claude.ai/design 사용
Max	$100~$200/월	가장 높은 한도 + 우선 액세스 + 모든 고급 기능

→ **비개발자 첫 시작은 Free**. 익숙해지면 Pro로. 디자인·자동화가 본격화되면 Max를 검토한다.

가격은 시점·지역에 따라 변동 — Anthropic 공식 페이지에서 최신 확인.

> 📌 비개발자의 첫 시작은 **Free 플랜으로 충분**하다. 익숙해진 뒤 더 큰 컨텍스트·자동화가 필요할 때 Pro로 올라가면 된다.

다음 장으로: Ch2에서는 Claude에게 **무엇을 어떻게 시켜야 결과가 좋은지**를 다룬다. "잘 해줘"라는 모호한 요청 대신 한 문장으로 결과 형태까지 그려넣는 법.

Ch2. 첫 대화 — 무엇을 어떻게 시켜야 하나

> **이 장을 끝내면 할 수 있는 것:** 자기 일 한 가지를 Claude에게 시켜 결과물을 받을 수 있다.

---

1. "잘 해줘"가 가장 위험하다

Claude Code를 켠 첫날 가장 흔한 사고는 입력창에 이렇게 치는 것이다.

> "내일 발표 자료 잘 만들어줘."

답은 분명히 온다. 그런데 받아본 사람은 안다 — **자기가 원하던 것과 다르다.** 이유는 한 가지다. "잘"이라는 단어는 사람마다 다르게 읽힌다. 어떤 사람에게는 8슬라이드짜리 깔끔한 PPT, 어떤 사람에게는 한 문단의 핵심 메시지다.

이 장에서는 첫 대화의 기본기 세 단어를 정의한다.

- **프롬프트** — Claude에게 시키는 한 마디. 채팅창에 입력하는 요청 문장이 곧 프롬프트다.
- **컨텍스트** — Claude가 지금 보고 있는 것들. 현재 대화 + 열어둔 파일 + 첨부 자료를 합한 것.
- **모델** — AI의 두뇌 종류. Opus(어려운 일) / Sonnet(보통 일) / Haiku(짧은 일).

이 세 단어가 첫 대화의 90%를 결정한다. **무엇을(프롬프트), 뭘 보고(컨텍스트), 누가(모델)** — 빠지면 결과가 흔들린다. 이 장이 책에서 다뤄질 가치는 한 가지다. 좋은 프롬프트의 감각을 첫날에 잡으면, 그 다음 100번의 대화가 편해진다.

---

2. 무엇과 어떻게 다른가

2-1. 좋은 요청 vs 나쁜 요청 — 세 쌍

항목	나쁜 요청	좋은 요청
발표 자료	"발표 자료 잘 만들어줘"	"신입사원 5명 대상 30분짜리 회사 소개 발표를 8슬라이드 분량, 각 슬라이드 한 문장 핵심 메시지만"
메일 정리	"메일 정리해줘"	"최근 7일간 받은 주문 확인 메일만 발신자/주문번호/금액 표로 정리. 합계 줄 포함"
글 다듬기	"이 글 다듬어줘"	"이 글을 50대 부모님 톤으로 친근하게 + 6문장 이내로 다시 써줘"

세 쌍의 공통점은 분명하다. **나쁜 요청은 결과 형태를 정하지 않는다.** 좋은 요청은 누가 / 무엇을 / 어떻게 / 어떤 형태로를 한 문장에 담는다.

2-2. 일을 시키기 전에 답할 4 질문

질문	예
결과는 **표 / 목록 / 글 / 코드** 중 무엇?	"표로"
분량은 어느 정도?	"한 문단" / "5개 항목"
누가 보나?	"초등학생 수준" / "사장님 보고용"
다음에 뭘 할 건가?	"그대로 카톡에 붙여넣기" / "엑셀에 옮기기"

이 4가지에 답이 있으면 좋은 프롬프트는 자동으로 짜인다. **요청을 적기 전에 머릿속에서 결과물을 한 번 그려본다** — 이게 첫 대화의 핵심 습관이다.

좋은 프롬프트 4 질문 체크리스트

---

3. ⚠️ 처음 사람들이 자주 막히는 곳

함정 1. 한 메시지에 일 5개를 한꺼번에 시킨다

"발표 자료 만들고, 회의록도 정리하고, 메일 답장도 써주고, 그리고 가계부도 보고…"

Claude는 다 답하려 한다. 그래서 **각 항목이 다 부실해진다.** 한 일이 끝나면 다음 일을 새 메시지로. 단순한 규칙이다 — **메시지 1개 = 작업 1개.**

함정 2. "잘 해줘" 같은 모호한 형용사

"잘", "괜찮게", "적당히", "예쁘게", "전문적으로". 이 단어들은 **사람마다 기준이 다르다.** 결과물 형태를 적는 칸이 비면 Claude는 추측하고, 추측한 결과는 대부분 사용자가 원한 것과 다르게 나온다.

대체어:

- "잘" → 구체적 형태 ("표로", "5문장으로")

- "예쁘게" → 톤 명시 ("따뜻하게", "포멀하게")
- "전문적으로" → 독자 명시 ("팀장 보고용", "외부 클라이언트용")

함정 3. 응답이 너무 길어 어디부터 봐야 할지 모른다

좋은 요청을 던졌는데도 답이 5문단을 넘어 길게 올 때가 있다. 회피책 세 가지:

- (a) 다음 메시지에 **"위 답을 표 한 장으로"** 한 줄
- (b) 처음부터 **"한 문단으로"**, **"5개 항목으로"** 같은 분량 지시 추가
- (c) Ch3의 4공식 중 **FORMAT**을 정확히 채우기 — 다음 장에서 다룬다

---

4. 5분 미니 실습 — 자기 일 한 가지 시키기

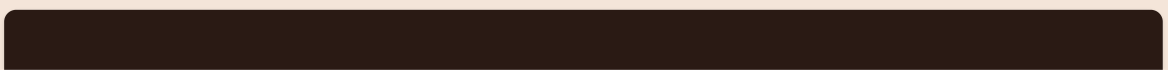
전제

- Ch1을 끝내 Claude Code 웹에 로그인되어 있다
- "안녕" 응답을 받아본 적 있다

단계

1. **자기 일 1개를 정한다.** 다음 중 가장 가까운 것을 고른다.
    - 받은 메일함에서 한 가지 종류 메일만 정리하기
    - 다음 주 일정을 한 문장 메모로 다듬기
    - 친구에게 보낼 안부 카톡 한 줄 만들기
    - 회의록 한 페이지를 5줄 요약으로 압축
2. **결과물 형태를 머리에 그린다.**
    - 표인가, 목록인가, 글인가?
    - 분량은? 5줄? 한 문단?
    - 누가 볼 건가?
3. **한 문장 프롬프트를 짠다.** 위 3가지를 모두 담아서.

    예시:

```
다음 주 월~금 일정 5개를 [요일 / 시간 / 항목] 표로 정리해줘.
원본은 아래 내 메모야:
"월요일 10시 거래처 미팅, 화요일 오후 보고서, ..."
```

4. claude.ai/code 입력창에 붙여넣고 엔터.

   ✓ 체크포인트: 위에서 그린 형태와 비슷한 결과가 온다.

5. **결과를 본다.** 마음에 들면 그대로 사용. 안 들면 한 줄만 더 추가.

   - "조금 더 짧게"
   - "이 컬럼 빼고"
   - "친근한 톤으로"

결과 인증

이 실습을 끝냈다면:

- 자기 일 한 가지를 Claude로 처리해본 첫 경험을 가졌다
- 결과 형태를 명시한 요청과 안 한 요청의 차이를 안다
- Ch3의 4공식을 받을 준비가 끝났다

⚠️ 트러블슈팅

문제 1: 응답이 영어로 옵니다. → 두 번째 메시지에 "한국어로 답해줘" 추가. 또는 처음부터 한국어 인사로 시작.

문제 2: 답이 길어 어디부터 봐야 할지 모릅니다. → "위 답을 표로" 또는 "3줄 요약으로" 한 줄 추가.

문제 3: 결과물이 자기가 원한 것과 영 다릅니다. → 한 문장 프롬프트 안에 "결과 형태"가 있는지 점검. 없으면 추가하고 재요청.

다음 장으로: Ch3에서는 좋은 프롬프트의 표준 — **ROLE / CONTEXT / TASK / FORMAT 4공식**을 다룬다. 이 장에서 머리로 그린 그것을 한 줄에 담는 정형화된 틀이다.

Ch3. 프롬프트 4 공식 — ROLE / CONTEXT / TASK / FORMAT

> **이 장을 끝내면 할 수 있는 것:** ROLE / CONTEXT / TASK / FORMAT 한 문장씩을 채워 요청을 던질 수 있다.

---

1. 잘 시키는 사람은 4가지를 빠뜨리지 않는다

Ch2에서 우리는 "결과 형태부터 머리에 그려야 한다"는 한 가지 원칙을 봤다. 이 장은 그 원칙을 정형화된 틀로 바꾼다. **4공식 — ROLE / CONTEXT / TASK / FORMAT**. 누가 / 뭘 보고 / 뭘 / 어떤 형태로.

- **ROLE (역할)** — Claude에게 어떤 입장으로 답하라고 정해주는 한 줄. 예: "마케터 입장에서", "30년차 출판 편집자 입장에서".
- **CONTEXT (맥락)** — Claude가 답하기 위해 알아야 할 배경. 예: "우리 회사는 1인 출판사고 신간이 다음 주 출간된다."
- **TASK (할 일)** — 실제 시키는 일. 예: "신간 보도자료 초안을 한 페이지로 작성해줘."
- **FORMAT (형태)** — 결과물의 모양. 예: "헤드라인 1줄 + 본문 4문단 + 인용 2줄."

이 4가지를 한 메시지에 다 담는다. 분량은 평소 요청과 거의 같지만, 결과 적중률이 두 배가 된다. 이 장이 책에서 다뤄질 가치는 한 가지다 — **이 4공식이 익숙해지면 두 번 다시 "왜 이런 답이 왔지?" 라고 묻지 않는다.**

4공식 ROLE/CONTEXT/TASK/FORMAT을 채워 넣을 빈 칸이 있는 템플릿

---

2. 무엇과 어떻게 다른가

2-1. 4공식 빠진 요청 vs 채운 요청

축	4공식 빠진	4공식 채운
입력 분량	한 줄	3~5줄
답 분량	들쭉날쭉	요청한 분량 그대로
톤	일반	명시한 톤
재요청 횟수	평균 3~4회	평균 1회
결과 적중률 (체감)	낮음 — 다시 묻기 잦음	높음 — 한 번에 OK

분량은 5배가 아니라 **3배 정도**. 들이는 시간 30초가 결과 30분을 절약한다.

2-2. 4 칸을 한 문장으로 잇는 표준 템플릿

칸	한 문장 형태	인스타 캐러셀 카피로 채워보기
ROLE	"당신은 ___ 입니다."	"당신은 1인 출판사 마케터입니다."
CONTEXT	"지금 상황은 ___ 입니다."	"다음 주 신간 '조용한 일상' 출간. 30대 직장인 독자 대상."
TASK	"___ 를 해주세요."	"인스타그램 캐러셀 8컷 카피를 작성해주세요."
FORMAT	"결과는 ___ 형태로."	"컷마다 한 줄 (15자 이내). 따뜻한 톤. 마지막 컷은 사전 예약 CTA."

영어 약어로 시작했지만 한국어로 그대로 쓴다. 이것을 한 메시지에 4줄로 나열하거나, 한 문단에 자연스럽게 섞어 쓴다.

4 칸 템플릿이 한 사례로 채워진 입력창

---

3. ⚠️ 처음 사람들이 자주 막히는 곳

함정 1. ROLE을 너무 길게 (5줄 페르소나)

"당신은 30년차 베테랑 마케터이며 스타트업 5개를 키운 경력이 있고 매주 콘텐츠 50건을 검토하며 톤은 친근하고 전문성 있는…"

ROLE은 **한 줄**이면 충분하다. 길어진다고 결과가 더 좋아지지 않는다. **"마케터 입장에서"** 한 줄과 5줄 페르소나의 결과 차이는 거의 없다.

함정 2. FORMAT 누락 — 가장 흔한 실수

"보고서 정리해줘. 우리 팀은 ... 회의록은 ..." — ROLE도 없고 FORMAT도 없다. CONTEXT만 길다. 결과물은 글 한 덩어리로 나온다. 표로 받고 싶었는데.

FORMAT은 4공식 중 가장 빠지기 쉬우면서 가장 결정적이다. 한 문장만 추가한다 — "결과는 표로." 이것만 있어도 답이 표로 온다.

함정 3. CONTEXT 너무 적게 — 일반론적 답이 온다

"광고 카피 만들어줘"만 던지면 Claude는 일반 광고 카피 5개를 만든다. 우리 제품·우리 톤·우리 채널을 모르고 만든 거라 **그대로 쓸 수 없다.**

CONTEXT 한 문단을 추가한다.

> "CONTEXT: 우리는 30대 직장인 대상 1인 일정 관리 앱이다. 톤은 따뜻한 친구. 채널은 인스타그램 캐러셀."

이 한 문단으로 결과가 갑자기 쓸 만해진다.

---

4. 5분 미니 실습 — 자기 일을 4공식으로

전제

- Ch2를 끝내 자기 일 1개를 시켜본 적 있다
- 그 결과가 살짝 아쉬웠다 (재시킬 만한 일)

단계

1. Ch2에서 시켰던 자기 일 1개를 다시 가져온다.

2. 4 칸을 한 문장씩 채운다.

```
ROLE: 나는 ___이다. (또는 당신은 ___이다.)
CONTEXT: 지금 상황은 ___.
TASK: ___ 를 해줘.
FORMAT: 결과는 ___ 형태로 (분량 ___, 톤 ___).
```

3. 채운 4줄을 한 메시지로 묶는다.

   예시 (회의록 정리):

```
ROLE: 당신은 스타트업 PM입니다.
CONTEXT: 우리 팀 월요일 주간 회의록 1페이지가 아래에 있습니다. 참석자 5명, 주제는 다음 주 출시 준비.
TASK: 이 회의록을 (1) 5줄 요약 + (2) 다음 액션 표로 정리해주세요.
FORMAT:
- 5줄 요약은 불릿 5개
- 액션 표 컬럼: 담당자 / 액션 / 기한
- 한국어, 존댓말 사용

회의록:
[회의록 본문 붙여넣기]
```

4. claude.ai/code에 붙여넣고 엔터.

   ✓ 체크포인트: Ch2보다 결과 적중률이 눈에 띄게 올라간다.

5. **Ch2 결과와 Ch3 결과를 나란히 비교**한다. 어느 쪽이 더 쓸 만한가?

결과 인증

이 실습을 끝냈다면:

- Ch2 결과와 Ch3 결과를 **눈으로 비교**해봤다
- 30초 더 들여서 얻은 결과의 차이를 안다
- 4공식이 자기 손에 어느 정도 익숙해졌다

같은 일에 대한 Ch2 결과(공식 없이)와 Ch3 결과(4공식)를 나란히 비교

⚠️ 트러블슈팅

문제 1: ROLE이 떠오르지 않습니다. → "전문가" 같은 일반어로 시작 (예: "마케팅 전문가", "PM 전문가"). 점점 구체화.

문제 2: 4공식이 다 들어갔는데 결과가 그대로다. → FORMAT을 더 구체적으로. "표로"가 아니라 "3컬럼 표 (이름/액션/기한)"처럼.

문제 3: 4줄 쓰는 게 매번 귀찮다. → Ch4의 CLAUDE.md와 Ch7의 슬래시 커맨드가 정확히 이 문제를 푼다. 자주 쓰는 4공식을 단축어·자동 메모로 만든다.

다음 장으로: Ch4에서는 4공식을 매번 안 적어도 되도록 해주는 도구 — **CLAUDE.md와 폴더 구조**를 다룬다. 자기 폴더 자체를 Claude의 작업장으로 만드는 법.

Ch4. 파일·프로젝트로 맥락 누적

> **이 장을 끝내면 할 수 있는 것:** 자기 폴더에 CLAUDE.md를 작성해 Claude에게 "이 폴더 안에서 일해줘"라고 시킬 수 있다.

---

1. Claude의 메모리는 짧다 — 외장 SSD가 필요하다

Claude의 가장 큰 약점은 한 가지다. **대화창을 닫으면 다 잊어버린다.** 어제 두 시간 들여 가르친 우리 회사 톤·고객 페르소나·금기어를 오늘 다시 처음부터 가르쳐야 한다. 매일.

해결책은 **폴더 안에 메모를 적어두는 것**이다. 이 메모를 Claude가 스스로 읽도록 약속된 두 가지 방법이 있다.

- **CLAUDE.md** — 폴더 단위 메모. Claude가 그 폴더에서 일을 시작할 때 가장 먼저 읽는다. "이 프로젝트는 무엇이고, 어떤 규칙을 지키며, 결과물은 어떤 형태로 내야 하는지" 적는다.
- **Projects** — Claude 웹의 작업공간. CLAUDE.md와 비슷한 역할이지만 웹 한정이고 클릭 인터페이스로 관리한다.

이 두 가지가 곧 Claude의 **외장 SSD**다. 폴더와 파일이 그의 짧은 메모리를 보완한다. 이 장이 책에서 다뤄질 가치는 한 가지다 — **CLAUDE.md 한 번 적어두면 그 폴더에서 시작하는 모든 대화에 그 컨텍스트가 자동 적용된다.** 매일 가르치는 시간이 0이 된다.

---

2. 무엇과 어떻게 다른가

2-1. 일회 첨부 vs CLAUDE.md vs Projects

축	일회 첨부	CLAUDE.md	Projects (Claude 웹)
위치	채팅 메시지	폴더 루트 파일	Claude 웹의 작업공간
지속성	한 메시지만	**그 폴더 안 모든 대화**	그 Project 안 모든 대화

축	일회 첨부	CLAUDE.md	Projects (Claude 웹)
용량	첨부 한도	마크다운 텍스트 (수천 줄 가능)	Project 자료 한도
자동성	매번 첨부	**자동 로드**	자동 로드
협업	본인만	폴더 공유 시 모두	Project 멤버
적합한 일	한 번만 쓸 자료	**장기 프로젝트**	Claude 웹 한정 작업

핵심: **장기로 가는 일은 CLAUDE.md, 한 번만 보면 되는 자료는 첨부.**

2-2. Claude가 읽는 일반 폴더 구조

비개발자가 Claude 작업용 폴더를 처음 만든다면 이 모양을 가정한다.

```
my-work-2026/
├── CLAUDE.md            ← Claude가 가장 먼저 읽음
├── README.md            ← 사람이 처음 보는 안내문
├── data/                ← 입력 자료 (회의록·영수증 등)
│   ├── meeting-2026-05-03.md
│   └── meeting-2026-05-10.md
├── output/              ← Claude가 만든 결과물
│   ├── summary-2026-05-03.md
│   └── summary-2026-05-10.md
├── .claude/             ← Claude 설정·스킬·에이전트 (자동 생성)
│   ├── skills/
│   └── agents/
└── .env                 ← 비밀 (API 키 등) — .gitignore 필수
```

5가지만 외운다.

위치	역할
`CLAUDE.md`	이 폴더의 규칙 메모 (Claude 자동 로드)
`README.md`	사람을 위한 첫 안내문
`data/`	입력 자료. 회의록·영수증·메모
`output/`	Claude의 결과물

위치	역할
`.env`	비밀 — 절대 공개 폴더에 안 둠

이 한 장의 폴더 구조를 머리에 담으면 다음 챕터들의 예시가 모두 같은 모양으로 읽힌다.

2-3. CLAUDE.md의 단골 3 섹션

섹션	무엇을 적나	예
이 프로젝트는	한 줄 정체성	"주간 회의록 자동 정리 폴더"
규칙	지킬 것·금기	"톤은 존댓말. 영문 약어는 풀어쓰기. 참석자 실명은 결과에 노출하지 않기"
출력 형식	결과물 모양	"결과물은 마크다운. 5줄 요약 + 액션 표 (담당자/액션/기한)"

3 섹션 각각 1~3줄이면 충분하다. **5줄짜리 CLAUDE.md 한 장이 매일 4공식 4줄 적기를 대신한다.**

---

3. ⚠️ 처음 사람들이 자주 막히는 곳

함정 1. 너무 큰 파일을 첨부 → 컨텍스트 윈도우 초과

Claude가 한 번에 기억할 수 있는 분량(컨텍스트 윈도우)에는 한계가 있다. 100MB짜리 PDF를 통째로 첨부하면 앞부분이 잘려나간다.

대처:

- 큰 파일은 **요약**해서 첨부 (Claude에게 먼저 요약을 시킨 뒤 그 요약을 CLAUDE.md에)
- CLAUDE.md는 **마크다운 텍스트만** 넣기 — PDF·이미지는 별도

함정 2. 비밀(API 키·개인정보)을 CLAUDE.md에 적는다

CLAUDE.md는 폴더에 들어가는 텍스트 파일이라 **GitHub에 올리면 모두에게 공개된다.** API 키, 비밀번호, 개인 식별정보를 CLAUDE.md에 적으면 평생 검색된다.

대처: 비밀은 `.env` **파일**에 따로 두고, CLAUDE.md에는 *"API 키는 .env에 있음"* 한 줄만 적는다. `.gitignore`에 `.env` 추가는 필수.

함정 3. 일회성 일을 매번 영구 메모리에 넣는다

"오늘 점심 메뉴 추천해줘" 같은 일회성 질문에 CLAUDE.md를 만들 필요는 없다. 일회성은 일회성으로. CLAUDE.md는 **3번 이상 반복될 일** 일 때 만든다.

기준: *"이 일을 다음 주에 또 할 것 같다"* — YES면 CLAUDE.md, NO면 첨부.

---

4. 5분 미니 실습 — 첫 CLAUDE.md 작성

전제

- 데스크톱에 작업 폴더 1개를 정한다 (회의록·일정·가계부 등 매주 반복할 만한 것)
- 텍스트 에디터 1개 (메모장 / VS Code / TextEdit 무엇이든)

단계

1. 그 폴더로 이동한다 (Finder/탐색기 또는 터미널

   ```
   cd
   ```

).

2. 텍스트 에디터로 새 파일

   ```
   CLAUDE.md
   ```

   를 만든다.

   - **macOS**: 터미널에서 `touch CLAUDE.md` 후 `open CLAUDE.md`
   - **Windows**: 메모장에서 빈 파일을 `CLAUDE.md`로 저장 (확장자 `.md` 주의 — 아래 트러블슈팅 참고)

3. 다음 3 섹션을 채운다 (각 1~3줄).

   ```
   # [내 프로젝트 이름]
   ```

```
## 이 프로젝트는
___ 를 자동 정리하는 폴더다.

## 규칙
- 톤은 ___
- ___ 정보는 결과에 포함하지 않는다

## 출력 형식
- ___ 형태로
- 분량은 ___
```

4. 저장한다.

5. 그 폴더 안에서 Claude Code를 켜고(또는 폴더를 첨부해) 짧은 질문을 던진다.

   ```
   이 폴더에서 시작하자. CLAUDE.md를 읽고 한 줄로 요약해줘.
   ```

   ✓ 체크포인트: Claude가 위에 적은 3 섹션을 한 줄로 요약해 답한다.

결과 인증

이 실습을 끝냈다면:

- 자기 폴더에 CLAUDE.md 한 장이 있다
- Claude가 그 파일을 자동으로 읽었다
- 같은 폴더 안에서 던지는 모든 질문이 그 컨텍스트를 자동 적용한다

⚠️ 트러블슈팅

문제 1: 메모장에서 저장하니 파일이 `CLAUDE.md.txt`가 됩니다. → "다른 이름으로 저장" 다이얼로그에서 파일 형식을 "모든 파일 (.)"로 바꾸고 이름은 `CLAUDE.md`. 또는 VS Code 같은 에디터를 사용한다.

문제 2: Claude가 CLAUDE.md를 안 읽은 것 같습니다. → 메시지에 "이 폴더의 CLAUDE.md 먼저 읽어줘" 명시. 또는 Claude Code 데스크톱·터미널 사용 (웹은 폴더 인식이 제한적).

문제 3: CLAUDE.md를 어디에 둬야 할지 모르겠습니다. → **프로젝트 폴더 루트**(가장 위). 하위 폴더가 아니라 프로젝트 폴더의 첫 자식.

다음 장으로: Ch5에서는 Claude Code가 일하는 **3가지 방식 — 로컬·원격·디스패치**를 한눈에 본다. 책상 앞에 있을 때, 책상 떠나 있을 때, 자고 있을 때 Claude가 어떻게 다르게 도는지.

Ch5. 자동화 3종 — 로컬 루틴·원격 루틴·디스패치

> **이 장을 끝내면 할 수 있는 것:** Claude를 자동화하는 3 방식(로컬 루틴 / 원격 루틴 / 디스패치)을 1초 안에 골라 자기 일에 얹을 수 있다.

---

1. Claude를 자동화한다 — 3 방식

여기까지 우리는 Claude Code를 **내가 직접 시키는 방식**으로 썼다. 회의록을 던지면 답이 오고, 끝나면 노트북을 닫는다.

그런데 사람의 일은 노트북을 닫고도 진행된다. 자고 있을 때·외출 중일 때·다음 주 월요일 아침에도 일이 일어난다. Anthropic은 이걸 위해 **자동화 3 종**을 공식으로 분류해 두었다.

종류	어디서 실행	트리거	누가 결과 받음	대표 예
💻 **로컬 루틴**	내 컴퓨터	일정 (cron)	내 컴퓨터	매일 9시 코드·문서 리뷰
☁️ **원격 루틴**	Anthropic 클라우드	스케줄·GitHub·API	저장소·커넥터	PR 자동 검토
📱→💻 **디스패치**	클라우드	**모바일 사용자 명령**	데스크탑 (페어링)	출근길 모바일→사무실

핵심 차이 두 축: ① 어디서 실행되나(내 컴퓨터 vs 클라우드) ② 어디서 결과를 받나(같은 기기 vs 디바이스 동기화). 셋은 경쟁이 아니라 **보완 관계**라 조합해서 쓴다.

이 장이 책에서 다뤄질 가치는 한 가지다 — **3 방식의 자리를 한 번 잡으면, "이 일은 어디에 얹어야 하지?" 망설일 시간이 사라진다.**

> 출처: Routines 공식 · Desktop scheduled tasks · Desktop Dispatch

---

2. 로컬 루틴 — 내 컴퓨터에서 자동 실행

로컬 루틴은 내 컴퓨터(Claude 데스크탑 앱)에서 정해진 시간에 자동으로 작업을 수행한다.

특징	내용
실행 환경	내 컴퓨터 (Claude 데스크탑 앱)
작동 조건	**컴퓨터가 깨어 있을 때만**
접근 가능	로컬 폴더·파일·워크트리
일정 옵션	시간별 · 매일 · 평일 · 주간 · 사용자 정의

> 공식 안내 (Desktop scheduled tasks): *"Local scheduled tasks run on your machine, with direct access to your local files and tools."*

비개발자 시나리오

- 직장인: 매일 8시 → 어제 회의록 5분 요약 · output/ 에 저장
- 작가: 매주 일요일 → 초고 폴더 정리 · 챕터 진행도 점검
- 마케터: 평일 7시 → 어제 캠페인 결과 정리 · 다음 일정 알림
- 1인 사업자: 시간별 → 영수증 폴더 신규 파일 자동 분류

👉 **원격 루틴과 차이**: 로컬 파일·폴더에 접근해야 하면 **로컬 루틴**, 공유 자료·외부 트리거가 필요하면 **원격 루틴**.

---

3. 원격 루틴 (Routines) — Anthropic 클라우드

원격 루틴은 Anthropic이 직접 운영하는 클라우드에서 자동 실행되는 작업이다 (공식 Routines). research preview 단계, **Pro·Max·Team·Enterprise** 가입자만 사용 가능.

> 공식 정의: *"A routine is a saved Claude Code configuration: a prompt, one or more repositories, and a set of connectors, packaged once and run automatically."*

트리거 3종

트리거	설명
🕑 **Schedule**	반복 cron 또는 미래 1회
🔗 **GitHub**	PR · Issue · Push 이벤트
⚡ **API**	HTTP POST 요청

> 한 routine에 여러 트리거 동시 부착 가능. 최소 1시간 간격, daily 계정 cap 있음.

공식 사용 사례

- **Backlog maintenance** — 야간 issue 분류·라벨링
- **Alert triage** — Sentry alert → 자동 PR 초안
- **Bespoke code review** — PR opened 시 팀 체크리스트 적용
- **Deploy verification** — CD pipeline → smoke tests
- **Docs drift** — 주간 merged PR 스캔 → 문서 업데이트 PR

생성 위치: claude.ai/code/routines 또는 CLI `/schedule`.

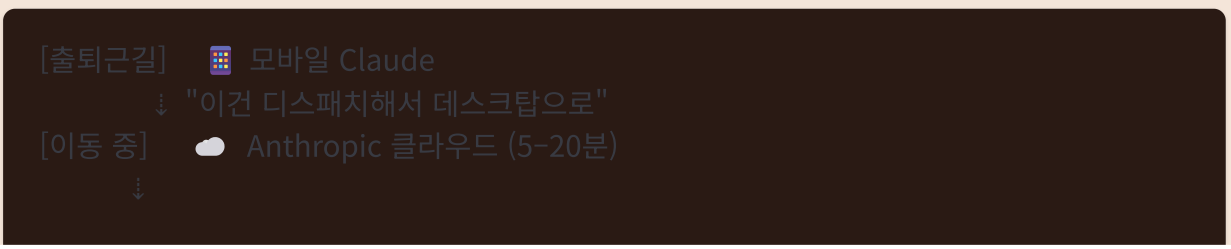
> 💡 "스케줄"이라 부르는 게 사실 Routines의 trigger type 중 하나(*Schedule trigger* — recurring 또는 one-off)다. 별도 기능 아님.

---

4. 디스패치 (Dispatch) — 모바일 → 데스크탑

디스패치는 모바일에서 던진 작업을 클라우드에서 처리하고 데스크탑에 결과를 받는 동기화 패턴이다.

동작 흐름



[출근 후] 💻 데스크탑 알림 → 완성된 결과

시작 4단계

1. 모바일에 Claude 앱 설치 (iOS / Android)
2. 앱 하단 **Dispatch** 탭
3. 데스크탑 Claude와 페어링 (QR 또는 계정 동기화)
4. 모바일에서 명령 던지면 클라우드 처리 후 데스크탑 결과

비개발자 시나리오

- 출근길: 어제 들어온 이메일 50개 분류 → 사무실 도착 시 데스크탑 정리
- 통학 시간: 수업 음성·녹음 핵심 5줄 요약 → 집에 가서 노트북에서 확인
- 카페 외출: 다음 주 제안서 초안 3개 → 집에 와서 편집
- 이동 중: 경쟁사 5곳 리서치 → 작업실 도착 시 합쳐진 보고서

👉 **원격 루틴과 차이**: 트리거가 **사람의 자발적 명령**이면 디스패치, **자동 일정·이벤트**면 원격 루틴.

> ⚠️ 병렬 실행(Task 도구·parallel agents)은 Dispatch와 별개 — 그건 Ch11 [에이전트·서브에이전트·에이전트 팀] 참고.

---

5. 선택 가이드 — 1초 결정

질문	답
GitHub · API · 정기 일정 트리거?	YES → ☁️ **원격 루틴**
폴더·로컬 파일 접근 필요?	YES → 💻 **로컬 루틴**
모바일에서 던지고 싶음?	YES → 📱 **디스패치**

셋은 보완 관계 — 한 일을 여러 방식으로 동시에 얹을 수도 있다.

---

6. 조합 레시피

레시피	조합	무엇을
월간 리포트	원격 루틴 + 디스패치	월말 클라우드 집계 → 출근길 모바일에서 결과 확인
외출 중 수신	디스패치	카페에서 모바일로 던짐 → 집 도착 시 데스크탑에 자료
콘텐츠 엔진	원격 루틴 + 스킬 + 디스패치	매일 자동 → 스킬로 형식 고정 → 어디서든 결과 수신

---

7. ⚠️ 처음 사람들이 자주 막히는 곳

함정 1. 루틴 과다 등록 — 결과물 소화 못 함

처음에 신나서 루틴 20개를 등록해두면 매일 아침 받는 결과가 너무 많아 다 못 본다. **5~7개 선에서 유지**.

함정 2. 로컬·원격 선택 착각

로컬 파일이 필요한데 원격 루틴으로 만들면 작동 안 된다. 반대도 마찬가지. **단순한 작업으로 한 번 테스트** 후 선택.

함정 3. 디스패치 페어링 불안정

모바일 신호 약한 곳·기기 절전 모드일 때 페어링이 끊긴다. **안정적인 환경**에서 던지기.

---

8. 5분 미니 실습 — 같은 작업을 두 방식으로 얹기

전제

- Ch1~Ch4 끝남 (로컬 작업 폴더 + CLAUDE.md 작성)
- claude.ai 웹 로그인 가능
- (선택) 모바일 Claude 앱 설치

단계

1. 자기 폴더에 매주 반복할 작업 1개를 정한다 (예: 주간 회의록 요약).
2. **로컬 루틴 셋업** — Claude Code 데스크탑에서:

```
/schedule weekly monday 08:00
data/meeting-*.md 가장 최신 1장을 5줄 요약해서
output/summary-YYYYMMDD.md 에 저장
```

   ✓ 체크포인트: 다음 월요일 8시에 자동 실행 예약됨.

3. **디스패치 시도** (선택, 모바일 앱 있을 때):
   - 모바일 Claude 앱 열기 → Dispatch 탭
   - 데스크탑과 페어링
   - 모바일에서 던지기: *"이번 주 회의 핵심 3개만 짧게"*
   - 데스크탑에 알림이 오는지 확인

   ✓ 체크포인트: 데스크탑 알림에 결과가 도착.

4. (참고) **원격 루틴**은 Pro 이상 가입자 한정 + GitHub 등 외부 트리거가 있을 때만 의미가 있어, 이 단계는 본 책에서는 생략한다.

결과 인증

이 실습을 끝냈다면:

- 자기 일 1개를 자동 실행에 처음 얹어본 경험을 가졌다
- 로컬과 디스패치의 작동 방식 차이를 안다
- 어느 일에 어느 방식이 맞을지 1초 안에 고를 수 있다

⚠️ 트러블슈팅

문제 1: `/schedule` 명령이 안 보입니다. → Claude Code 데스크탑·CLI 버전 확인 (구버전은 미지원). `/update`로 최신 갱신.

문제 2: 디스패치 페어링이 안 됩니다. → 모바일·데스크탑 같은 Anthropic 계정 로그인 확인. 또는 앱 재로그인.

문제 3: 로컬 루틴이 실행은 됐는데 결과가 없습니다. → 컴퓨터가 그 시간에 켜져 있었는지 확인. 절전·종료 상태면 로컬 루틴은 동작하지 않는다.

다음 장으로: Ch6에서는 Claude를 한 단계 키우는 **확장 6장치(커넥터·MCP·스킬·플러그인·에이전트·마켓플레이스)**를 한 장 지도로 본다.

Ch6. 확장 6장치 한눈에 — 커넥터·MCP·스킬·플러그인·에이전트·마켓플레이스

> **이 장을 끝내면 할 수 있는 것:** Claude의 6 확장 도구가 각각 무엇을 하는지 한 줄로 구분하고, 자기 일에 어느 것이 필요한지 고를 수 있다.

---

1. 6장치 한눈

Claude 기본 채팅으로도 많은 일을 할 수 있지만, 2026년 실전 유저들은 거의 확장 기능 하나 이상을 붙여 쓴다. **밖에서 안으로**(외부 서비스 연결) → **안에서 밖으로**(Claude 내부 확장) 순서로 6 장치를 배치한다.

#	장치	방향	만든 주체	핵심 한 줄	언제
1	**커넥터**	외부 → Claude	Anthropic 공식	앱 ↔ Claude 공식 연결	Gmail·Drive·Slack 쓸 때
2	**MCP**	외부 → Claude	누구나 (오픈 표준)	도구 연결 프로토콜	커넥터에 없는 툴 붙일 때
3	**스킬**	Claude 내부	나 or 커뮤니티	특정 작업 매뉴얼	반복 작업 규격화할 때
4	**플러그인**	Claude 내부	커뮤니티·회사	스킬+에이전트+MCP 번들	주제별 패키지 한 번에
5	**에이전트**	Claude 내부	나 or 플러그인	독립 실행 작업자 (컨텍스트 분리)	병렬 탐색·전문 위임
6	**마켓플레이스**	—	Anthropic·커뮤니티	플러그인 앱스토어	남의 플러그인 찾을 때

핵심 차이 한 줄:

- 커넥터·MCP는 *Claude 밖 도구를 연결*

- 스킬은 *Claude에게 일하는 법을 가르침*
- 에이전트는 *독립 실행자를 띄움*
- 플러그인은 위 셋을 묶음
- 마켓플레이스는 유통

이 장이 책에서 다뤄질 가치는 한 가지다 — **6 장치 자리를 먼저 머리에 그리면, Ch9~Ch11에서 깊이 들어갈 때 길을 잃지 않는다.**

---

2. 커넥터 — Claude의 공식 앱 스토어

커넥터는 Anthropic이 직접 검수해 Claude 앱에 공식 내장한 외부 서비스 연결 기능이다. 2024년 출시되어 2026년 현재 Gmail·Google Drive·Slack·Notion·Linear·Zapier 등 ~20개가 등록돼 있고, 설치는 Claude 앱에서 **Connectors 메뉴 → 토글 ON → OAuth 로그인**, 끝이다.

구조적 특징은 인증·보안·API 변경을 전부 Anthropic이 관리한다는 점. 대신 확장성은 닫혀 있어서 — 내가 쓰는 앱이 목록에 없으면 못 쓴다(이때가 MCP 차례).

비개발자 시나리오

- 대학생: `@Google Drive 지난 학기 레포트 PDF 3개 찾아 요약`
- 직장인: `@Gmail 이번 주 회의 요청 메일만 정리`
- 프리랜서: `@Slack 이번 달 클라이언트 채널 미답변 체크`

👉 **MCP와의 차이**: 내가 쓰는 앱이 커넥터 목록에 있으면 무조건 커넥터(보안·편의). 없으면 MCP.

---

3. MCP (Model Context Protocol) — 오픈 도구 표준

MCP는 Anthropic이 2024년 11월 공개한 **AI 도구 연결 오픈 표준**이다. 2026년 현재 Claude·Cursor·Windsurf·Zed가 전부 지원해서 업계 표준이 됐고, 역할은 "AI가 외부 도구를 부를 때 쓰는 공용 어댑터". USB-C가 충전기 난립을 끝낸 것과 같은 포지션.

👉 **커넥터와 다른 점 두 가지**:

1. 커넥터는 Anthropic 검수·목록 ~20개의 **닫힌** 생태계. MCP는 누구나 서버를 만들 수 있는 **열린** 생태계(2026 현재 수천 개).
2. 커넥터는 OAuth 자동, MCP는 API 키·토큰을 직접 관리.

2026년 인기 MCP 서버 TOP 6

서버	무엇을	누가 많이 쓰나
Chrome MCP	실제 크롬 제어	크리에이터·마케터
Notion MCP	노션 DB·페이지	지식노동자 전반
Figma MCP	디자인 → 코드	디자이너·프론트엔드
Filesystem MCP	로컬 파일 CRUD	개발자·연구자·작가
Context7	최신 라이브러리 문서	개발자 전반
Exa·Perplexity	심층 웹 검색	리서치·대학원생

👉 **스킬과의 차이**: MCP는 *도구 연결*, 스킬은 *작업 방법*. 둘을 짝지어 쓸 때 가장 강하다 — Figma MCP(파일 열기) + "디자인→컴포넌트 변환 스킬"(일하는 규칙).

---

4. 스킬 — Claude의 전문 매뉴얼

스킬은 특정 작업을 할 때 Claude가 따라야 할 **마크다운 매뉴얼**이다 (공식 명칭 *Agent Skills*). 2025년 10월 베타로 공개돼 2026년 정식 출시됐고, `Workflow / Capability / Hybrid` 자동 분류·프론트매터 hook·A/B 평가까지 붙으면서 지금 가장 빠르게 크는 확장 장치가 됐다.

👉 **MCP와 헷갈리기 쉬운데 역할이 정반대.** MCP는 "Notion을 연결할게"(도구), 스킬은 "이 템플릿대로 회의록 작성"(절차). 두 개는 **같이 쓸 때** 가장 강하다 — Notion MCP 연결 + 회의록 스킬 = "회의 음성 → 규격화된 노션 페이지".

공식 vs 커뮤니티 — 실제로 많이 쓰이는 스킬

구분	스킬	무엇을
🟢 Anthropic 공식	**pdf**	PDF 양식 작성·보고서·텍스트/표 추출
🟢 Anthropic 공식	**xlsx**	Excel 차트·수식·피벗
🟢 Anthropic 공식	**pptx · docx**	PowerPoint 슬라이드 / Word 문서
🟢 Anthropic 공식	**webdev · skill-creator**	HTML 인터랙티브 / 스킬 만드는 메타 스킬
🟡 superpowers	**brainstorming**	창의 작업 전 인텐트 탐색
🟡 superpowers	**systematic-debugging**	버그·테스트 실패 분석
🟡 superpowers	**TDD**	테스트 우선 개발 워크플로우
🟡 superpowers	**writing/executing-plans**	다단계 작업 설계·실행
🟡 superpowers	**parallel-agents**	병렬 서브에이전트 발사
🟡 superpowers	**verification-before-completion**	완료 주장 전 증거 검증

비개발자가 직접 만들 때:

- 블로거: `blog-post` (제목→훅→본문→CTA)
- 학생: `study-note` (개념→예시→퀴즈)
- 마케터: `threads-post` (훅·길이·해시태그)

👉 **플러그인과의 차이**: 스킬 1개 = 작업 1개. 여러 스킬을 묶어 "한 주제 전체"를 커버하려면 플러그인.

---

5. 플러그인 — 스킬·에이전트·MCP 묶음

플러그인은 스킬 + 에이전트 + MCP 설정 + 슬래시 커맨드를 하나로 묶은 **주제 패키지**다. `/plugin install <이름>` 한 줄로 수십 개 스킬이 한꺼번에 활성화되는, 2026년 현재

Claude 생태계의 실질적 배포 단위.

2026년 인기 플러그인

플러그인	주제	누구용
anthropic-skills	🟢 Anthropic 공식 — PDF·Excel·PPT·Webdev·Docx	입문자 — 시작점
anthropics/cookbook	🟢 Anthropic 공식 — 예제·튜토리얼·레시피	학습자 — 따라하며 익힘
superpowers	🟡 커뮤니티 인기 — 브레인스토밍·디버깅·TDD	개발자 전반
oh-my-claudecode	🟡 커뮤니티 인기 — 에이전트 35종	Claude Code 파워유저

👉 **스킬과의 차이**: 한 가지 작업만 자동화 → 스킬. 한 주제 전체(예: AI 네이티브 개발 전 과정)를 세팅 → 플러그인.

---

6. 에이전트 · 서브에이전트 — 독립 실행자

에이전트는 특정 역할만 전담하는 **독립된 Claude 작업자**다. 일반 대화와 달리 (1) 별도의 지시사항·역할 정의, (2) 자체 도구 접근권, (3) 자체 컨텍스트 창을 갖고 움직이고, 작업이 끝나면 **결과만** 메인 Claude에게 돌려준다. 2026년 Claude Code는 `.claude/agents/*.md` 파일로 에이전트를 정의하고, 메인 세션이 `Task` 도구로 호출한다.

서브에이전트는 메인 Claude가 실행 도중 호출하는 **1회성 보조 에이전트**다. "서브"라는 말처럼 메인 대화의 하위 계층에서 돌아가고, 자체 컨텍스트에서 작업을 끝낸 뒤 **결과 요약만** 올려준다 — 메인 대화가 거대한 탐색 결과로 오염되지 않는 것이 핵심.

왜 필요한가:

1. Claude에게 30만 토큰짜리 코드베이스를 탐색시키면 메인 세션이 그 로그로 가득 차버린다. 서브에이전트에 위임하면 메인은 *"3개 파일에 로그인 로직이 있다"* 한 줄만 받는다.
2. 여러 전문 영역(보안·QA·리서치)을 **병렬로** 굴릴 수 있다.

에이전트 vs 스킬 vs 플러그인 — 한 표

축	스킬	에이전트	플러그인
정의	작업 매뉴얼	독립 실행자	스킬+에이전트+MCP 번들
컨텍스트	메인과 공유	**분리**	혼합
호출 방식	자동 매칭	Task 도구	설치 후 내장
병렬 실행	없음	**있음**	없음 (단독)
예시	blog-post	security-reviewer	anthropic-skills · cookbook

한 줄: 스킬은 *"어떻게"*를 가르치고, 에이전트는 *"누가"*를 따로 앉힌다.

---

7. 마켓플레이스 — 플러그인 앱스토어

마켓플레이스는 플러그인을 배포·검색·설치하는 공간. Anthropic이 운영하는 공식 마켓과 커뮤니티 3rd-party 마켓이 공존한다. 핵심은 **발견**(검색·순위·리뷰)과 **배포**(버전·업데이트) 두 가지.

검색 방법 3 경로

A. 데스크탑 앱 (가장 쉬움)

1. Claude 데스크탑 좌측 하단 ⚙️ 설정 → **Plugin Marketplace** 또는 **Extensions**
2. 검색창에 키워드 입력 (예: `debugging`, `pdf`, `notion`)
3. 평점·다운로드수·최근 업데이트 정렬
4. **Install** 버튼 클릭 → OAuth/권한 동의 → 끝

B. CLI (Claude Code 개발자 위주)

```
/plugin install <name>@<marketplace>
/plugin install superpowers@<marketplace-name>
claude plugin list
claude plugin remove <이름>
```

C. GitHub 직접 (가장 자유로움)

```
github.com/search?q=topic:claude-code-plugin-marketplace
github.com/search?q=topic:claude-skills
github.com/anthropics/skills        # 공식 번들
```

안전 설치 기준 (커뮤니티 휴리스틱)

- GitHub 별⭐ **1k+** 또는 다운로드 **1만+**
- 최근 **1개월 이내** 커밋
- README에 **Permissions/Tools** 명시 (코드 실행 권한 확인)
- 리뷰 **5개 이상** + 평점 **3.5/5 이상**

→ 위 4 조건 만족 시 양면 안전.

---

8. 한 가지 더 — 훅(Hooks) 박스

위 6 장치 외에 알아두면 강한 **7번째 자리**가 있다. **훅(Hooks)**.

훅은 **사건이 일어날 때 자동 발동되는 알람**이다. 사용자가 키워드를 던질 필요 없이, 파일 저장·세션 시작·커밋 직전 같은 *이벤트* 자체가 트리거가 된다.

도구	호출 시점
슬래시 커맨드 (Ch7)	사용자가 `/clear` 같은 단어 입력
스킬	메시지에 트리거 표현이 등장
에이전트	Task 도구로 명시 호출
훅	**파일 저장·세션 시작·커밋 직전 같은 이벤트**

훅 시나리오

- **PreToolUse**: 파일을 쓰기 전 자동 보안 점검 (API 키·시크릿 노출 검사)
- **SessionStart**: 세션 시작 시 자동 메모(MEMORY.md) 로드
- **PostToolUse**: 명령 실행 후 자동 로깅·알림
- **PreCommit**: git commit 직전 자동 lint·테스트

훅은 `~/.claude/settings.json`의 `hooks` 항목에 시점과 명령을 짝지어 둔다. 비개발자에게는 **자주 안 쓰지만 한 번 알아두면 강력**한 도구. 매주 같은 시점에 자동으로 도는 일이 늘어나면 훅을 떠올린다.

---

9. 실전 설치 순서

순	단계	무엇
①	커넥터 먼저	쓰는 앱이 목록에 있으면 OAuth로 즉시
②	MCP 보충	목록에 없는 툴은 MCP 서버 추가
③	스킬 1개	가장 반복되는 작업부터 매뉴얼화
④	에이전트 1개	위임하고 싶은 전문 영역 (리뷰·검수·리서치)
⑤	플러그인	주제 전체 세팅이 필요할 때
⑥	마켓플레이스 탐색	월 1회 신규 플러그인 둘러보기
⑦	훅 (선택)	매주 같은 시점 자동화가 필요하면

비개발자 첫 시도 순서: 커넥터 → 스킬 → 플러그인 → 에이전트 → MCP → 훅. 위에서 아래로 갈수록 진입 장벽이 높다.

---

10. ⚠️ 흔한 함정 TOP 4

- **커넥터 + MCP 중복**: 같은 Notion을 둘 다 켜면 Claude가 혼동 → 하나만
- **보안 미검수 MCP**: GitHub star 수·최근 커밋일 확인 후 설치
- **스킬 과다**: 20개 넘어가면 Claude가 상황 매칭 실패 → 월 1회 정리
- **에이전트 남용**: 단순 작업까지 에이전트로 위임하면 응답 지연↑. 메인에서 1분 내 끝날 작업은 그냥 대화

---

11. 5분 미니 실습 — 6 장치 자리 외우기

전제

- 종이 한 장 또는 디지털 노트

단계

1. 종이 한 장에 6 자리를 그린다.

```
외부 → Claude ：  커넥터        MCP
                (공식)      (오픈)
Claude 내부   ：  스킬          플러그인        에이전트
                (매뉴얼)    (번들)          (독립 실행자)
유통          ：  마켓플레이스
```

2. 각 자리 옆에 자기 일 1개씩 적어 본다.
    - 커넥터: 가장 자주 쓰는 외부 앱 (Notion·Drive·Gmail 등)
    - MCP: 커넥터에 없는 도구 (있다면)
    - 스킬: 매주 반복할 작업 1개
    - 플러그인: 받고 싶은 주제 번들 (anthropic-skills부터)
    - 에이전트: 전담 직원이 있으면 좋겠는 일 1개
    - 마켓플레이스: 한 달에 한 번 둘러볼 캘린더 메모

    ✓ 체크포인트: 6 장치 중 적어도 **3개에 자기 일이 있다**.

3. 6 장치 중 가장 빨리 만들고 싶은 1개를 동그라미.

    ✓ 체크포인트: 1개를 골랐다.

4. 그 1개를 어느 챕터에서 다루는지 확인한다.
    - 커넥터 / MCP / 플러그인 → **Ch9**
    - 스킬 → **Ch10**
    - 에이전트 → **Ch11**

결과 인증

이 실습을 끝냈다면:

- 6 장치의 자리를 머리에 그릴 수 있다
- 자기 일에 가장 먼저 만들 1개를 골랐다

- Ch9~Ch11이 어디에 무엇을 더 채울지 안다

⚠️ 트러블슈팅

문제 1: 6 장치가 다 비슷해 보입니다. → "공식 어댑터 / 오픈 어댑터 / 매뉴얼 / 번들 / 독립 실행자 / 앱스토어" 6 비유를 외운다.

문제 2: 자기 일에 어느 장치가 맞을지 모르겠습니다. → "3번 이상 반복?" — NO면 만들지 말 것. YES면 위 §9 실전 설치 순서 따라.

문제 3: 커넥터에 노션이 없거나 안 보입니다. → 커넥터 목록은 Anthropic이 추가/제거한다. 노션이 안 보이면 MCP로 우회 가능 (Ch9에서).

다음 장으로: Ch7에서는 6 장치 중 가장 진입 장벽이 낮은 **슬래시 커맨드 50+**를 한 번에 정리한다. 외울 필요 없다 — 카테고리별 표 한 장으로 충분.

Ch7. 슬래시 커맨드 — 한 장으로 끝내는 단축어 사전

> **이 장을 끝내면 할 수 있는 것:** 자주 쓰는 슬래시 커맨드를 카테고리별 한 장으로 훑고, 자기에게 필요한 7~10개만 골라 손에 익힐 수 있다.

---

1. `/`로 시작하는 약속어, 외우지 말고 펼쳐 보기

Claude Code에 익숙해지면 같은 일을 하루에 5번 시키게 된다. 같은 4공식을 매일 다시 적기는 비효율적이다.

슬래시 커맨드는 이 문제를 푸는 약속어다. `/`로 시작하는 한 단어를 입력하면 **이미 약속된 일이 한 번에 실행된다.**

이 장은 외우라고 하지 않는다. **카테고리별 한 장**을 보여줄 뿐. 외울 건 카테고리 7개 이름과 자기에게 맞는 7~10개. 나머지는 `/help`로 언제든 펼쳐 본다.

- **사용 위치** — 모든 입력창의 **첫 글자**에 `/`를 누르면 자동완성 메뉴가 펼쳐진다.
- **자동완성** — `/c` 까지만 쳐도 `/clear`, `/config` 같은 후보가 떠 클릭만으로 선택.
- **두 종류** — 기본 내장 슬래시(Anthropic 공식)와 플러그인이 추가하는 슬래시. 이 장은 둘 다 다룬다.

> 📌 아래 표 중 일부는 **Claude Code 데스크톱·CLI 기준**이거나 **특정 플러그인 설치 시** 활성화된다. 첫 화면에서 `/help`로 실제 사용 가능한 명령을 확인.

---

2. 슬래시 한 장 — 7 카테고리

A. 세션·대화 관리

슬래시	한 줄
`/clear`	대화 초기화 — 컨텍스트 깔끔히
`/help`	도움말 / 사용 가능한 명령 목록
`/compact`	긴 대화를 자동 요약해 컨텍스트 줄이기
`/resume`	이전 세션 이어가기
`/save`	현재 대화를 저장
`/export`	대화 내용을 .md 파일로
`/history`	최근 대화 목록
`/title`	현재 대화에 제목 달기

B. 프로젝트·파일

슬래시	한 줄
`/init`	폴더 초기화 — CLAUDE.md 자동 생성
`/cd`	작업 폴더 변경
`/pwd`	현재 작업 폴더 확인
`/ls`	현재 폴더 파일 목록
`/open`	파일을 컨텍스트에 추가
`/close`	컨텍스트에서 파일 제거
`/files`	현재 컨텍스트의 파일 목록
`/search`	폴더 안 텍스트 검색

C. 사고·통찰 ⭐ insight 류

슬래시	한 줄
`/think`	깊이 사고 모드 — 추론 시간 길어짐
`/insight`	핵심 통찰 뽑기 — 결론·시사점 한 줄로
`/brainstorm`	아이디어 발산 — 10개 이상
`/critique`	비판적 검토 — 약점·반례 찾기
`/explain`	어려운 용어·코드를 풀어 설명
`/summarize`	핵심 요약
`/translate`	한국어 ↔ 영어 변환

D. 검수·정리

슬래시	한 줄
`/review`	파일·폴더 검토 (글·코드)
`/simplify`	받은 답·코드를 더 깔끔하게 정리
`/format`	결과를 표·목록·코드 블록으로 다시
`/lint`	문법·스타일 검사
`/diff`	두 버전 차이 보기
`/security-review`	보안 점검 (API 키 노출·취약점)
`/verify`	결과 검증 — 증거 기반

E. 자동화·예약 (Pro+ 일부)

슬래시	한 줄
`/loop`	같은 작업 반복 실행
`/schedule`	정기 자동 실행 (매주 월요일 등)

슬래시	한 줄
`/dispatch`	백그라운드 작업 시작 (모바일→데스크탑)
`/status`	진행 중인 작업 상태
`/cancel`	진행 중인 작업 취소
`/run`	저장된 스킬·에이전트 호출

F. 외부 연결 (커넥터·MCP)

슬래시	한 줄
`/connect`	커넥터 연결
`/notion`	노션 페이지 직접 다루기
`/drive`	구글 드라이브 검색·열기
`/gmail`	메일 분류·요약
`/github`	깃허브 이슈·코드
`/slack`	슬랙 메시지 검색
`/mcp`	MCP 상태 / 추가
`/plugin`	플러그인 설치·관리

G. 확장·운영

슬래시	한 줄
`/skill`	스킬 호출·관리
`/agent`	에이전트 호출
`/memory`	자동 메모 관리
`/config`	Claude Code 설정 변경

슬래시	한 줄
`/model`	모델 변경 (Opus / Sonnet / Haiku)
`/usage`	토큰·메시지 사용량
`/cost`	누적 비용
`/version`	버전 확인
`/update`	최신 버전 갱신
`/login` `/logout`	로그인 / 로그아웃

다음 장으로: Ch8에서는 `/clear` · `/compact`만으로는 부족한 실전 — **토큰 절약 8가지 꿀팁**을 다룬다. 컨텍스트 다이어트가 비용·속도·정확도를 동시에 잡는다.

Ch8. 토큰 절약 8가지 꿀팁 — 비용·속도·정확도 동시에

> **이 장을 끝내면 할 수 있는 것:** 같은 일에 토큰을 50% 이상 줄여, Claude의 답이 더 빠르고 더 정확해지는 8가지 패턴을 자기 작업에 적용할 수 있다.

---

1. 토큰은 셋을 동시에 결정한다 — 비용·속도·정확도

비개발자에게 "토큰" 한 단어는 살짝 무서워 보인다. 사실 단순하다.

- **토큰** — Claude가 한 번에 다루는 글자 단위. 한국어 1글자가 보통 토큰 1~2개. "안녕하세요"는 약 3 토큰.
- **컨텍스트 윈도우** — 한 대화에 들어갈 수 있는 토큰 한도. Claude는 200K 토큰 (한국어 약 10만 자) 정도까지 한 번에 본다.

토큰을 아껴야 하는 이유는 셋이다.

차원	토큰을 줄이면
비용	Pro/Max 플랜의 메시지 한도 안에서 더 오래 쓴다
속도	답이 빨리 온다 (긴 컨텍스트일수록 느림)
정확도	**컨텍스트가 깔끔할수록 답이 정확해진다**

세 번째가 가장 결정적이다. 토큰을 줄이는 건 절약 차원이 아니라 **품질 관리**다. 늘어진 컨텍스트는 Claude에게도 사람에게도 부담이다.

이 장이 책에서 다뤄질 가치는 한 가지다 — **8가지 패턴을 손에 익히면 같은 일에 토큰이 절반으로 줄고, 답은 더 빠르고 정확해진다.**

---

2. 8가지 꿀팁 한 장

꿀팁 1. 주제 바뀌면 `/clear`

가장 단순. 주제가 바뀌면 직전 컨텍스트는 다 잊어도 된다.

```
/clear
```

한 번이면 끝.

언제 쓰나: 회의록 정리 → 영수증 분류 같은 다른 도메인으로 넘어갈 때.

꿀팁 2. 대화 길어지면 `/compact`

`/clear`는 다 비우기, `/compact`는 **요약해서 남기기**. 이전 컨텍스트의 핵심을 자동 요약해 5분의 1 정도로 줄여 보관한다.

상태	추천
같은 주제 계속 가야 하지만 대화가 길어짐	`/compact`
다른 주제로 완전히 옮길 때	`/clear`

꿀팁 3. 큰 파일은 요약을 첨부

100MB PDF·1만 줄 회의록을 통째로 첨부하면 컨텍스트의 절반을 잡아먹는다. 그러면 Claude는 **자기 답할 여유 공간**이 줄어 답이 부실해진다.

대처: 큰 파일은 먼저 Claude에게 1페이지 요약을 시킨 뒤, 그 요약본만 본 작업에 첨부. 원본은 폴더에 두고 *"필요하면 data/원본.md 를 직접 읽어"* 한 줄로 안내.

꿀팁 4. CLAUDE.md에 "출력 형식"을 좁게

CLAUDE.md(Ch4)에 "출력 형식: 마크다운, 5줄 이내, 표 우선" 같은 분량·형태 제한을 넣어두면 매 답이 자동으로 짧아진다. 토큰이 절반으로 줄고 가독성도 올라간다.

예시:

```
## 출력 형식
- 한 응답 5문장 이내. 더 길면 표나 불릿으로 분할.
- 코드 답변은 핵심 함수만. 나머지는 "필요시 추가" 안내.
- 인사·서론·결론 문장 생략.
```

꿀팁 5. 결과만 받고 추론은 끄기

기본 모드에서 Claude는 답하기 전에 자기 생각(reasoning)을 펼친다. 이게 길면 토큰을 많이 쓴다.

짧고 명확한 작업(예: 표 한 줄 정리)은 처음부터 *"바로 결과만, 설명 없이"* 한 줄을 추가. 답이 3분의 1로 줄어든다.

일	추천
회의록 5줄 요약	"바로 결과만"
어려운 의사결정 (어느 도구를 써야?)	추론 켜기 (설명 포함 요청)

꿀팁 6. 한 메시지 = 한 작업

Ch2 함정 1과 같다. "메일 정리하고, 회의록도, 영수증도..." 한 번에 시키면 각 작업이 부실해진다. 토큰도 작업 수에 비례해 늘어난다.

한 메시지 한 작업 원칙. 작업이 끝나면 `/clear` 또는 `/compact` 후 다음 작업.

꿀팁 7. 모델을 작업 크기에 맞추기

Opus는 Sonnet의 5배, Haiku의 10배 토큰을 쓴다 (대략).

작업	추천 모델	이유
어려운 의사결정·디자인·전략	**Opus**	5배 비싸도 1번에 끝남
일상 작업의 95% (회의록·메일·정리)	**Sonnet**	균형형 — 책의 표준
한 줄 답·번역·간단 분류	**Haiku**	매우 빠름·저렴

규칙: 모르겠으면 Sonnet. Opus는 진짜 어려운 일에만.

꿀팁 8. 자주 쓸 컨텍스트는 스킬·CLAUDE.md로 영구 보관

매번 같은 5줄을 채팅에 붙여넣고 있다면 (예: 우리 회사 톤·금기어·고객 페르소나) — 그 5줄은 컨텍스트가 아니라 **영구 메모**여야 한다.

대처: CLAUDE.md(폴더 단위) 또는 SKILL.md(키워드 단위)로 옮긴다. 매 대화에서 5줄을 다시 안 보내도 자동 적용.

---

3. ⚠️ 처음 사람들이 자주 막히는 곳

함정 1. `/compact` 만 믿고 무한 누적

`/compact`는 강력하지만, 매 메시지마다 누적된 컨텍스트를 5분의 1로 줄이는 것뿐이다. 한 시간 동안 누적하면 결국 한도 근처. 5~10 메시지마다 한 번씩 `/compact` 또는 `/clear`로 초기화.

함정 2. 모델을 한 단계만 쓴다 (다 Opus 또는 다 Haiku)

비개발자에게 흔한 실수. 다 Opus면 비용이 5배. 다 Haiku면 어려운 작업에서 답이 부정확.

대처: 기본은 Sonnet. 어려운 의사결정만 Opus, 단순 분류·번역만 Haiku.

함정 3. 대화 안에서 첨부를 반복한다

같은 PDF를 한 메시지에서 한 번, 다음 메시지에서 또 첨부하는 실수. 컨텍스트에 두 번 쌓인다.

대처: 한 번 첨부한 자료는 같은 대화 안에서 다시 안 첨부. *"위에서 첨부한 회의록 기준으로"* 한 줄로 충분.

---

4. 5분 미니 실습 — 자기 작업의 토큰 다이어트

전제

- Ch5에서 작업 폴더 1개 + CLAUDE.md 작성 완료
- 그 폴더에서 한 시간 이상 작업해본 경험

단계

1. 그 폴더에서 어제·오늘 작업한 대화 1개를 떠올린다.

2. 8가지 꿀팁 중 자기가 안 쓰고 있는 것 3개를 표시한다.

- [ ] 주제 바뀌면 `/clear`
- [ ] 대화 길어지면 `/compact`
- [ ] 큰 파일은 요약을 첨부
- [ ] CLAUDE.md에 "출력 형식" 좁게
- [ ] 결과만 받고 추론은 끄기
- [ ] 한 메시지 = 한 작업
- [ ] 모델을 작업 크기에 맞추기
- [ ] 자주 쓸 컨텍스트는 스킬·CLAUDE.md로

3. 표시한 3개 중 가장 쉬운 1개를 골라 다음 작업에 적용한다.

4. 적용 후 토큰 사용량을 확인한다.

```
/usage
```

✓ 체크포인트: 직전 작업 대비 토큰이 줄었는지 확인.

5. 다음 주에 표시한 나머지 2개도 차례로 적용.

결과 인증

이 실습을 끝냈다면:

- 8가지 꿀팁 중 자기가 안 쓰던 패턴을 안다
- 한 패턴을 적용해 토큰 차이를 눈으로 확인했다
- 매주 한 패턴씩 추가할 길이 보인다

⚠️ 트러블슈팅

문제 1: `/usage`가 토큰 수치를 안 보여줍니다. → `/usage` 미지원 환경(웹의 일부)에서는 Settings → Usage에서 확인. 또는 `/cost`로 누적 비용 확인.

문제 2: `/compact`가 답을 너무 많이 잘라냅니다. → 메시지에 *"중요한 사실 X·Y는 보존하고 compact"* 한 줄 추가. 또는 핵심을 CLAUDE.md에 옮긴 뒤 `/compact`.

문제 3: 모델을 바꿔도 결과 차이를 모르겠습니다. → 같은 일을 Sonnet과 Haiku로 한 번씩 시켜 비교. 차이가 없으면 Haiku 유지 (저렴·빠름).

다음 장으로: Ch9부터는 Part 2 활용 편. 첫 주제는 **MCP·플러그인·커넥터** — Claude를 노션·구글드라이브·깃허브 같은 외부 세계에 연결하는 3 다리.

Ch9. 커넥터·MCP·플러그인 — 바깥 세상 연결

> **이 장을 끝내면 할 수 있는 것:** Notion·구글드라이브·GitHub 중 하나를 Claude에 연결해 첫 페이지를 읽거나 쓸 수 있다.

---

1. Claude는 자기 컴퓨터 안에서만 살지 않는다

Ch1~Ch8에서 우리는 **내 컴퓨터 안의 일**을 다듬었다. 회의록·메일·일정. 그런데 우리 일의 절반은 컴퓨터 바깥에 있다.

- **노션**에 적어둔 회사 위키
- **구글드라이브**에 쌓인 자료
- **GitHub** 저장소의 코드와 이슈
- **이메일**(Gmail) 받은 편지함
- **카카오톡·Slack** 대화

이걸 매번 복사해서 Claude에 붙여넣는 건 비효율적이다. 차라리 Claude가 직접 들어가 페이지를 읽거나 글을 쓰면 된다.

이걸 가능하게 하는 다리가 3종이다 — **커넥터 → MCP → 플러그인** 순서로 진입 장벽이 올라간다.

- **커넥터** — Anthropic이 직접 검수해 Claude 앱에 공식 내장한 외부 서비스 연결 기능. **클릭 한 번**.
- **MCP** — Model Context Protocol. Anthropic이 공개한 AI 도구 연결 **오픈 표준**. 누구나 서버를 만들어 붙일 수 있다.
- **플러그인** — 스킬·에이전트·MCP·슬래시 커맨드를 한 봉투에 묶은 **주제 패키지**.

이 장이 책에서 다뤄질 가치는 한 가지다 — **커넥터 1개를 연결하는 순간 Claude가 다루는 자료의 범위가 10배 늘어난다.** 비개발자에게 가장 큰 변화는 여기서 한 번 더 일어난다.

 ch9-three-bridges

커넥터 / MCP / 플러그인 3종이 Claude를 외부 서비스에 연결하는 다이어그램

---

2. 무엇과 어떻게 다른가

2-1. 커넥터 vs MCP vs 플러그인

축	커넥터 (클릭 한 번)	MCP (직접 셋업)	플러그인 (번들)
진입 장벽	**매우 낮음 (클릭만)**	높음 (설정 파일 작성)	보통 (마켓플레이스에서 설치)
자동성	**자동**	사용자가 트리거 명시	스킬/에이전트와 함께 자동
적용 범위	Claude 웹·데스크톱	모든 Claude 환경	플러그인 호환 환경
누가 만드나	Anthropic 공식	본인 또는 외부 개발자	마켓플레이스 등록자
비개발자 첫 시도	◯ **(이걸로 시작)**	×	△

비개발자 진입 순서: 커넥터 → 플러그인 → MCP. 커넥터로 외부 연결의 감을 잡고, 필요해지면 플러그인 설치, 가장 마지막에 MCP 직접 셋업.

2-2. 비개발자가 자주 쓸 커넥터 (2026 기준)

커넥터	무엇을 가능하게	활용 예
Notion	페이지 읽기·쓰기	"이 회의록 노션에 정리해줘"
Google Drive	파일 검색·읽기	"최근 받은 PDF 5개 요약해줘"
Gmail	메일 분류·답장 초안	"주문 확인 메일만 표로"
GitHub	이슈·코드 검색	"내 레포 이슈 3개 정리"
Slack	채널 메시지 검색·요약	"이번 주 #general 핵심 5줄"

이 책의 비개발자 독자는 **노션 또는 구글드라이브 1개부터** 시작한다. 한 개 연결해 한 번 써보면 다음 연결은 쉽다.

2-3. 인기 플러그인 (2026 기준)

플러그인	주제	누구용
anthropic-skills	🟢 공식 — PDF·Excel·PPT·Webdev·Docx	입문자 — 시작점
anthropics/cookbook	🟢 공식 — 예제·튜토리얼	학습자
superpowers	🟡 커뮤니티 — 브레인스토밍·디버깅·TDD	개발자
oh-my-claudecode	🟡 커뮤니티 — 에이전트 35종	Claude Code 파워 유저

---

3. ⚠️ 처음 사람들이 자주 막히는 곳

함정 1. API 키 노출

MCP를 직접 셋업할 때 API 키를 설정 파일에 적는다. 이 파일이 GitHub에 올라가거나 누군가에게 공유되면 **API 키가 평생 검색되고**, 누가 그 키로 우리 계정을 마음대로 쓸 수 있다.

대처:

- API 키는 `.env` **파일**에 분리, `.gitignore`에 `.env` 추가 (Ch4 함정 2와 같은 패턴)
- MCP 설정 파일에는 *"API 키는 환경변수에서"* 같은 참조만
- 노출되면 **즉시 해당 서비스에서 키 무효화** 후 새로 발급

함정 2. 권한을 너무 넓게 — "전체 공유" 클릭

커넥터 연결 시 권한 화면에서 *"내 노션 전체에 접근"* 같은 옵션을 무심코 승인한다. 그러면 Claude가 회사 위키·개인 일기·비밀 문서까지 모두 볼 수 있다.

대처:

- **필요한 페이지·폴더만 공유**. 노션은 페이지 단위, 구글드라이브는 폴더 단위로 권한 가능

- 정기적으로 권한 점검 (한 달에 한 번 Connectors 페이지에서 연결 목록 확인)
- 더 이상 안 쓰는 커넥터는 즉시 해제

함정 3. 커넥터 + MCP 중복

같은 Notion을 커넥터로도 켜고 MCP로도 켜면 Claude가 어느 쪽을 써야 할지 헷갈린다. 응답이 느려지거나 권한 충돌이 생길 수 있다.

대처: **하나만 켠다.** 커넥터 목록에 있으면 커넥터, 없을 때만 MCP.

---

4. 5분 미니 실습 — Notion 커넥터 1개 연결

전제

- 노션 계정이 있다 (없으면 notion.so에서 무료 가입)
- 노션에 테스트용 빈 페이지 1개가 있거나 만들 수 있다

단계

1. claude.ai에서 우상단 프로필 → **Settings** → **Connectors**.

   ✓ 체크포인트: 사용 가능한 커넥터 목록이 보인다.

2. **Notion** 옆 **"Connect"** 버튼 클릭.

   ✓ 체크포인트: 노션 OAuth 권한 화면이 새 창에 뜬다.

3. 권한을 **"선택한 페이지만"**으로 좁히고, 테스트용 빈 페이지 1개만 선택.

   ✓ 체크포인트: 권한 승인 후 Claude로 돌아와 "Connected" 표시.

4. claude.ai 메인으로 돌아가 새 메시지를 던진다.

   ```
   내 노션의 [페이지 이름] 에 "Hello Claude!" 한 줄 추가해줘.
   ```

   ✓ 체크포인트: Claude가 노션 페이지를 갱신했다는 응답이 온다.

5. **노션 앱·웹에서 확인**한다. 그 페이지에 새 줄이 들어가 있어야 한다.

결과 인증

이 실습을 끝냈다면:

- Claude가 외부 서비스(노션)에 직접 쓰기 동작을 한 첫 경험을 가졌다
- Claude의 자료 범위가 내 컴퓨터를 넘어섰다
- 다음 일에 같은 패턴을 적용할 준비가 됐다 (구글드라이브, GitHub 등)

⚠️ 트러블슈팅

문제 1: 권한 화면에서 페이지가 안 보입니다. → 노션 워크스페이스가 비어 있을 수 있다. 노션에서 빈 페이지 1개를 먼저 만든 뒤 다시 시도.

문제 2: Claude가 "노션을 찾을 수 없다"고 답합니다. → 커넥터 연결이 시간 차로 적용될 수 있다. 1~2분 후 새 대화창에서 다시 시도.

문제 3: 의도하지 않은 페이지를 갱신해버렸습니다. → 커넥터 권한을 다시 점검 (Settings → Connectors → Notion → "Manage permissions"). 페이지 단위로 좁힌다.

다음 장으로: Ch10에서는 같은 노션 작업을 **자동 호출되는 매뉴얼 — 스킬(Skills)**로 바꾼다. 매주 같은 4공식을 다시 안 적어도, 한 키워드로 자동 발동.

Ch10. 스킬 (Skills) — 자주 시키는 일을 한 단어로

> **이 장을 끝내면 할 수 있는 것:** 자주 시키는 일을 SKILL.md 한 장으로 묶어, 다음번엔 키워드 한 마디로 자동 호출되도록 만들 수 있다.

---

1. 한 번 가르치면 매번 안 가르쳐도 되는 게 스킬이다

Ch4에서 CLAUDE.md로 "이 폴더의 규칙"을 적어뒀다. 그 폴더 안에서 시작하는 모든 대화에 그 규칙이 자동 적용된다.

스킬(Skill)은 한 단계 더 나아간다. 폴더 단위가 아니라 **작업 단위로 묶어둔 매뉴얼**이다.

예를 들어 "주간 회의록을 5줄 요약 + 다음 액션 표로 정리하기"라는 일을 매주 한다. 매번 4공식을 새로 적는 대신, 이 절차를 한 번 SKILL.md 한 장으로 적어두면 — 다음번 회의록을 던지는 순간 Claude가 **알아서 그 매뉴얼을 꺼내 따른다.**

- **스킬** — 자주 시키는 작업의 절차를 마크다운 파일로 적어둔 것. 트리거 조건이 맞으면 Claude가 자동 로드.
- **SKILL.md** — 스킬의 정의 파일. 마크다운 본문 + 위에 짧은 메타정보(YAML frontmatter)로 구성.
- **자동 트리거** — 사용자가 명시 호출하지 않아도, description의 키워드가 메시지에 등장하면 Claude가 스스로 그 스킬을 불러온다.

이 장이 책에서 다뤄질 가치는 한 가지다 — **스킬 5개가 익숙해진 순간부터 같은 일에 들이는 시간이 1/5로 준다.** 매주 1시간 작업이 12분으로.

.claude/skills/ 폴더 안에 여러 스킬 폴더(skill-name/SKILL.md 구조)가 보이는 화면

---

2. 무엇과 어떻게 다른가

2-1. 4공식 vs CLAUDE.md vs 스킬

축	4공식 (매번 적기)	CLAUDE.md	스킬
적용 범위	한 메시지	한 폴더 안 모든 대화	**모든 폴더, 키워드 매치 시**
호출 방식	매번 4줄	자동 (폴더 진입 시)	**자동 (메시지 키워드 매치 시)**
누가 만드나	사용자 직접	사용자	사용자 또는 마켓플레이스
재사용	×	그 폴더에서만	**모든 폴더, 평생**
만드는 시간	30초 (매번)	5분 (한 번)	10분 (한 번)

핵심 차이: 4공식은 매 메시지, CLAUDE.md는 한 폴더, 스킬은 **온 컴퓨터**. 위로 갈수록 한 번 들이는 시간이 길지만 평생 절약하는 시간이 길다.

2-2. SKILL.md의 표준 구조

```
---
name: meeting-summarizer
description: 주간 회의록 1페이지를 5줄 요약 + 다음 액션 표로 정리하는 스킬.
             "회의록 정리", "이번 주 미팅 요약" 같은 표현이 나오면 반드시 사용.
---

# meeting-summarizer — 주간 회의록 한 장 요약 스킬

## 절차
1. 입력 폴더(data/)의 가장 최신 meeting-*.md 1장을 읽는다
2. 핵심 5줄을 추출 (불릿 5개)
3. "다음 액션" 항목을 (담당자 / 액션 / 기한) 3컬럼 표로 정리
4. output/summary-YYYYMMDD.md 로 저장

## 출력 형식
- 한국어 존댓말
- 5줄 요약은 불릿
- 액션 표 마지막에 "다음 회의" 한 줄 추가
- 참석자 실명 대신 직책으로 표기 (예: 한지훈 → PM)
```

부분	설명
`---` ~ `---` (YAML frontmatter)	name과 description. **description이 자동 트리거의 핵심**
본문	구체적 절차. 4공식의 ROLE·CONTEXT·TASK·FORMAT을 풀어쓴 매뉴얼

---

3. ⚠️ 처음 사람들이 자주 막히는 곳

함정 1. description이 약해 자동 트리거 안 됨

가장 흔한 실패. SKILL.md는 만들었는데 메시지를 던져도 Claude가 알아채지 못한다. 원인은 description이 너무 모호하거나 짧아서.

나쁜 예:

```
description: 회의록 처리 스킬
```

좋은 예:

```
description: 주간 회의록 1페이지를 5줄 요약 + 다음 액션 표로 정리하는 스킬.
             "회의록 정리", "이번 주 미팅 요약", "월요일 회의 정리" 같은 표현이 나오면 반드시 사용.
```

규칙: description은 **하는 일 + 구체적 트리거 표현**을 모두 담는다. 모호함을 피하고 적극적으로 — Claude가 빠뜨리지 않게 명확하게 — 적는다.

좌: 약한 description SKILL.md, 우: 강한 description SKILL.md 두 화면 비교

함정 2. 본문이 500줄 초과 — 컨텍스트 무거움

스킬은 자동 로드되기 때문에 본문이 무거우면 매번 그 분량이 컨텍스트에 들어간다. **500줄을 넘기면 다른 스킬·작업의 여유 공간이 줄어든다** (Ch8 토큰 절약과 같은 원리).

대처:

- 핵심 절차만 SKILL.md에 (보통 100~200줄로 충분)
- 부가 자료(예시 파일·참조 문서)는 같은 폴더 안 `references/`에 두고, SKILL.md에서 *"필요할 때 references/x.md를 읽어"* 라고만 안내

함정 3. 너무 좁은 스킬 — 한 번 쓰고 안 쓰게 됨

"2026년 5월 첫째 주 보고서 작성 스킬" 같은 너무 좁은 스킬은 한 번 쓰고 만다. 스킬은 **3번 이상 반복될 일**일 때만 만든다.

기준: *"이 일을 다음 분기에도 또 할 것 같다"* — YES면 스킬, NO면 4공식 한 번으로 끝.

---

4. 5분 미니 실습 — 첫 SKILL.md 작성 (회의록 정리 스킬)

전제

- Ch4를 끝내 CLAUDE.md를 한 번 작성해봤다
- 매주 반복하는 작업 1개를 정한다 (회의록 정리, 메일 분류, 일정 표 만들기 등)

단계

1. 사용자 스킬 폴더를 만든다.
   - **macOS / WSL**: `mkdir -p ~/.claude/skills/meeting-summarizer`
   - **Windows PowerShell**: `New-Item -ItemType Directory -Path "$HOME\.claude\skills\meeting-summarizer"`

   ✓ 체크포인트: 그 경로에 폴더가 만들어졌다.

2. 그 폴더 안에

   `SKILL.md`

   파일을 만든다.

3. SKILL.md에 다음 템플릿을 채운다.

```
---
name: meeting-summarizer
description: 주간 회의록 1페이지를 5줄 요약 + 다음 액션 표로 정리하는 스킬.
             "회의록 정리", "이번 주 미팅 요약" 같은 표현에 반드시 사용.
---

# meeting-summarizer — 회의록 한 장 요약 스킬

## 절차
1. 회의록 본문을 읽는다
```

```
2. 핵심 5줄 추출
3. "다음 액션" 항목을 (담당자/액션/기한) 표로

## 출력 형식
- 5줄 불릿 + 3컬럼 표
- 한국어 존댓말
- 참석자 실명 대신 직책으로 표기
```

4. 저장한다.

5. 새 대화창에서 description의 트리거 표현 중 하나를 던져본다.

```
이번 주 미팅 요약해줘. 회의록은 아래야...
```

✓ 체크포인트: Claude가 meeting-summarizer를 자동 로드했다는 표시 또는 그 절차에 따른 응답이 나온다.

결과 인증

이 실습을 끝냈다면:

- 자기 첫 스킬이 `~/.claude/skills/meeting-summarizer/SKILL.md`에 있다
- 트리거 표현으로 자동 호출이 동작한다
- 같은 일을 다음 주에 또 시킬 때 4공식을 다시 안 적어도 된다

⚠️ 트러블슈팅

문제 1: 트리거 표현을 던졌는데 Claude가 알아채지 못합니다. → description을 더 구체적으로. 키워드 2~3개 명시 (예: "회의록 정리", "미팅 요약", "월요일 회의").

문제 2: 스킬이 작동하지만 결과가 SKILL.md 절차와 다릅니다. → Claude가 절차를 흘려봤을 가능성. SKILL.md 본문에 *"위 절차를 그대로 따라"* 한 줄 추가.

문제 3: 같은 폴더에 여러 스킬이 있는데 다른 스킬이 호출됩니다. → description의 키워드가 겹치는 경우. 더 구체적인 표현으로 차별화.

---

5. 한 가지 더 — 훅(Hooks) 한 박스

스킬은 **트리거 표현이 메시지에 등장하면 호출**되는 자동화. 그런데 어떤 자동화는 트리거 없이 **사건이 일어날 때마다** 작동해야 한다 — 파일을 저장할 때 자동 검사, 세션이 시작될 때 자동 메모 로드, 커밋 직전 자동 보안 점검 같은 일. 이때 쓰는 게 **훅(Hooks)**이다.

도구	호출 시점
슬래시 커맨드	사용자가 직접 `/clear` 같은 단어 입력
스킬	메시지에 트리거 표현이 등장
훅	**파일 저장·세션 시작·커밋 직전 같은 사건이 일어남**

훅은 `~/.claude/settings.json`의 `hooks` 항목에 *"PreToolUse / PostToolUse / SessionStart"* 같은 시점을 적고, 그때 실행할 명령을 짝지어 둔다. 비개발자에게는 **자주 안 쓰지만 한 번 알아두면 강력**한 도구. 매주 같은 시점에 자동으로 도는 일이 늘어나면 훅을 떠올린다.

다음 장으로: Ch11에서는 스킬을 한 단계 더 발전시킨 도구 — **에이전트·서브에이전트**를 다룬다. 스킬이 매뉴얼이라면 에이전트는 그 매뉴얼을 들고 일하는 일꾼이다.

Ch11. 에이전트·서브에이전트

> **이 장을 끝내면 할 수 있는 것:** 짧은 일은 메인 에이전트로, 큰 일은 서브에이전트로 — 두 자리를 1초 안에 구분해서 시킬 수 있다.

---

1. 두 컨셉 한눈 — 자리가 다르다

"에이전트"라는 말은 한 단어가 아니다. Anthropic 공식 분류로 두 자리가 있고, **자리만 다르고 동작은 같다.**

컨셉	한 줄 정의	일하는 자리	결과를 누가 보나
메인 에이전트	메인 Claude가 직접 자율 동작	**사용자와 같은 컨텍스트**	사용자가 모든 단계 봄
서브에이전트	메인이 위임한 별도 워커	**격리된 자기 컨텍스트**	사용자는 요약만 봄

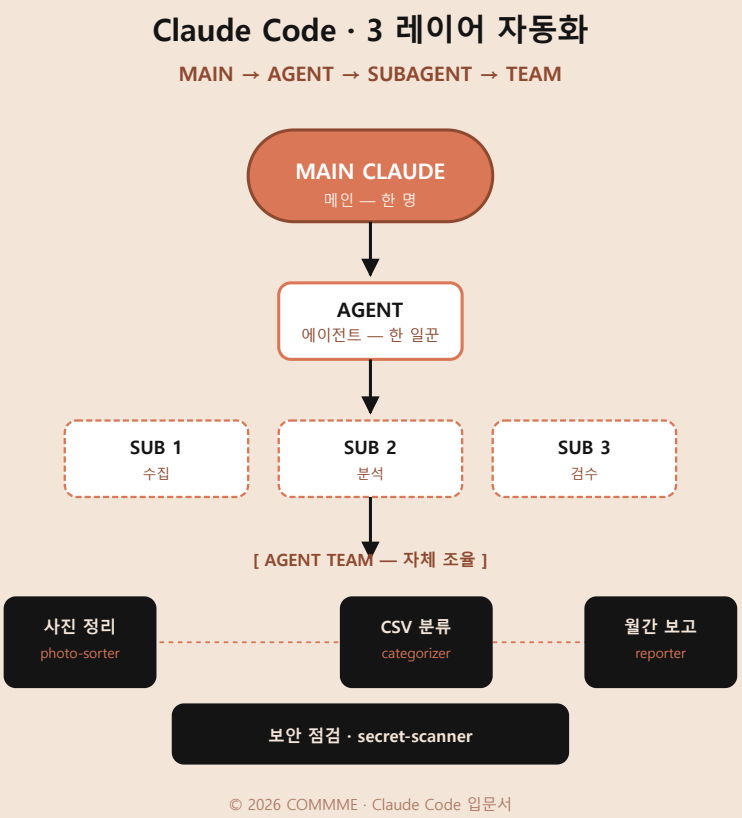
> 공식 한 줄: *"Subagents work within a single session; agent teams coordinate across separate sessions."* — Anthropic Sub-agents

비유로 한 줄:

- **메인 에이전트 = 사장님이 직접 책상 앞에서 일하는 모드**
- **서브에이전트 = 사장님이 직원에게 일을 맡기고 결과 보고만 받는 모드**

> 📌 **에이전트 팀**(여러 세션 사이 협업)은 이 책에서 다루지 않는다. 정상 작동하려면 `TeamCreate` · `TaskCreate` · `SendMessage` + 별도 **오케스트레이터**까지 갖춰야 하는 영역으로, 비개발자가 한 시간 안에 손에 익히기엔 무겁다. 익숙해진 뒤 공식 Agent Teams 가이드로.

이 장이 책에서 다뤄질 가치는 한 가지다 — **자기 일을 두 자리 중 어디에 얹어야 하는지 1초 안에 결정한다.** 결정의 기준은 단순하다 — **자료가 큰가, 작은가.**



같은 자율 동작이 메인 자리(직접 일함)와 서브 자리(위임받아 일함)에서 다르게 작동한다

---

2. 메인 에이전트 — 사용자와 같은 자리에서 직접

메인 에이전트는 메인 Claude가 자기 컨텍스트 안에서 자율적으로 일하는 모드다. 단순 1턴 응답이 아니라 *컨텍스트 파악 → 도구 선택 → 실행 → 평가 → 다음 단계* 의 루프를 돈다.

비교	챗봇	메인 에이전트
동작	1턴 응답	다중 턴 자율
도구 사용	없거나 제한	자유
컨텍스트	휘발	누적
사용자 시야	답 한 번	**모든 단계 보임**

메인 에이전트에 맞는 일 — "작고 짧은 작업"

데이터가 컨텍스트에 들어와도 부담 없을 정도의 작은 일. 사용자가 중간 단계를 보고 싶은 일.

일	메인이 자율로 하는 5 단계
다음 주 일정 표 만들기	① 메모 읽기 → ② 요일·시간 추출 → ③ 표 생성 → ④ CLAUDE.md 형식 적용 → ⑤ 사용자에게 보여주기
받은 메일 50개 분류	① 메일 읽기 → ② 카테고리 자동 분류 → ③ 표 정리 → ④ 다음 액션 추천
회의록 1장 5줄 요약	① 파일 1개 읽기 → ② 핵심 5줄 → ③ 액션 표 → ④ output/ 저장
CSV 한 장 통계 내기	① 파일 읽기 → ② 합계·평균 → ③ 카테고리 분포 표

기준 한 줄: 입력 자료가 **5만 토큰 안쪽**이면 메인. 사용자가 *"방금 뭐 했지?"* 묻고 싶으면 메인.

---

3. 서브에이전트 — 메인이 위임한 격리 워커

서브에이전트는 메인 Claude가 자기 일을 더 큰 일로 위임하는 별도 워커다. 자기만의 컨텍스트·시스템 프롬프트·도구로 작업하고, 작업이 끝나면 **결과 요약만** 메인에 반환한다.

축	메인 에이전트	서브에이전트
컨텍스트	사용자와 공유	**자기만의 윈도우 (격리)**
사용자 시야	모든 단계 보임	결과 요약만 보임
호출 방식	직접 메시지	메인이 `Task` 도구로 위임
도구	메인의 전체 권한	**화이트리스트로 제한**
적합한 자료	5만 토큰 안쪽	**5만 토큰 초과·전문 작업**

서브에이전트의 4가지 진짜 역할

서브에이전트는 단순히 "메인을 덜 무겁게" 하는 게 아니라 **4가지 분명한 역할**이 있다. 각각 다른 에이전트 예시.

역할 ① 큰 자료 탐색 — `code-explorer`

```
---
name: code-explorer
description: 큰 코드베이스에서 특정 기능의 위치를 찾는다. "어디에 로그인 로직 있어" 같은 표현에 사용.
tools: Read, Glob, Grep
model: sonnet
---
You are a code explorer. Search the entire codebase for the requested feature.
Return ONLY 3 most relevant file paths + 1-line description each. No code dumps.
```

사용 예: 50,000줄짜리 회사 코드. 메인 Claude에게 *"로그인 로직 어디 있어?"* 던지면 메인이 `code-explorer`에 위임.

- 서브에이전트가 50K줄을 자기 컨텍스트에서 다 훑음
- 메인에 반환: *"3 파일:* `auth.js` *(검증) /* `login.tsx` *(UI) /* `session.ts` *(토큰)"*
- 메인 컨텍스트엔 3 줄만 들어옴. 50K 줄 원문은 서브 안에서만 머묾.

역할 ② 전문 작업 격리 — `security-reviewer`

```
---
name: security-reviewer
description: 배포 전 시크릿 노출·취약점을 점검한다. "보안 점검", "git push 전" 표현에 사용.
tools: Read, Grep, Bash
model: opus
---
You are a security reviewer. Check git changes for:
1. API keys, tokens, .env leaks
2. XSS, SQL injection patterns
3. OWASP Top 10 risks
Return: PASS or list of findings with severity.
```

사용 예: GitHub push 직전. 메인 Claude에게 *"보안 점검해줘"* 한 줄.

- 서브에이전트가 모든 변경 파일을 자기 컨텍스트에서 스캔

- 메인에 반환: *"발견 2건: `.env` 누출 + XSS 의심 1건"* 또는 *"PASS"*
- 메인은 그 결과로 다음 행동만 결정 (push할지 멈출지).

역할 ③ 외부 리서치 — `research-runner`

```
---
name: research-runner
description: 경쟁사·논문·시장 자료를 웹에서 수집해 비교 표로 정리. "경쟁사 비교", "리서치" 표현에 사용.
tools: WebFetch, WebSearch
model: sonnet
---
You are a market researcher. Find requested companies/papers.
Return: comparison table only (no full transcripts).
```

사용 예: *"우리 제품의 경쟁사 5곳 가격·기능 표로 정리해줘"* 한 줄.

- 서브에이전트가 5개 웹사이트 풀 텍스트를 자기 컨텍스트에서 처리
- 메인에 반환: *"5 × 4 비교 표 1장"*
- 메인 컨텍스트엔 표만. 원본 5개 HTML은 서브 안에만.

역할 ④ 긴 문서 압축 — `doc-summarizer`

```
---
name: doc-summarizer
description: 100페이지+ PDF·정책 문서에서 우리에게 필요한 5조항만 추출. "긴 문서 요약" 표현에 사용.
tools: Read
model: haiku
---
You are a document summarizer. Extract only the requested clauses.
Return: 5 most relevant clauses, each in 1 sentence.
```

사용 예: 100페이지 회사 정책 PDF. *"우리 부서 출장 규정 5조 항목 5줄로"* 한 줄.

- 서브에이전트(Haiku — 저렴)가 100페이지를 다 읽음
- 메인에 반환: *"출장비 한도 5조 5줄"*
- **비용 절감**: Haiku 라우팅으로 Opus의 1/10 비용.

서브에이전트의 4 이득 (공식)

위 4 역할이 그대로 공식 이득과 매칭된다.

이득	매칭 역할
Preserve context — 메인 대화 깨끗하게	① code-explorer · ④ doc-summarizer
Enforce constraints — tools 제한으로 안전	② security-reviewer (Bash 가능)
Reuse configurations — 모든 프로젝트 재사용	4 가지 모두
Control costs — Haiku 같은 저렴 모델 라우팅	④ doc-summarizer (Haiku)

제작 — YAML frontmatter (공식)

```
---
name: code-explorer
description: 큰 코드베이스 탐색. "어디에 X 있어" 표현.
tools: Read, Glob, Grep
model: sonnet
isolation: worktree
---
You are a code explorer. ...
```

저장 위치: `~/.claude/agents/NAME.md` (모든 프로젝트) 또는 `.claude/agents/NAME.md` (이 프로젝트만).

---

4. 선택 가이드 — 1초 결정

질문	답
입력 자료가 5만 토큰 안쪽인가?	YES → **메인 에이전트**
큰 자료를 훑어 핵심만 받고 싶나?	YES → **서브에이전트** ① (code-explorer 유)
전문 도구 권한이 필요한가? (Bash·WebFetch)	YES → **서브에이전트** ② (security-reviewer 유)

질문	답
같은 작업을 여러 프로젝트에서 반복?	YES → **서브에이전트 ③** (재사용)
비용을 줄이고 싶나?	YES → **서브에이전트 ④** (Haiku 라우팅)

기억할 한 줄: **자료가 작으면 메인, 크면 서브.** 그게 전부.

---

5. ⚠️ 처음 사람들이 자주 막히는 곳

함정 1. 서브에이전트 결과 = 요약만

서브에이전트는 자기 컨텍스트에서 깊이 작업하지만, 메인에 돌려주는 건 **요약만**이다. 디테일이 필요하면 description에 *"5줄 + 원본 인용 3개"* 같이 명시.

함정 2. 작은 일까지 서브로 위임

5천 토큰짜리 회의록 1장에 서브에이전트를 부르면 **위임 오버헤드**가 더 크다. 메인이 직접 하는 게 빠르다.

기준: **자료가 5만 토큰 넘을 때만 서브.**

함정 3. tools 무제한 → 보안 사고

서브에이전트의 `tools:` 필드를 비워두거나 `*`로 두면 모든 도구 접근 가능. 화이트리스트로 좁힌다.

서브에이전트	추천 tools
code-explorer	`Read, Glob, Grep` (수정 X)
security-reviewer	`Read, Grep, Bash` (변경 X)
research-runner	`WebFetch, WebSearch` (로컬 X)
doc-summarizer	`Read`

함정 4. 정의 파일 없이 prompt에 박는 안티패턴

급하다고 정의 파일을 안 만들고 prompt에 *"넌 code-explorer야"* 한 줄로 박으면 다음 세션에 재사용 못 한다. 반드시 `.claude/agents/NAME.md` 파일로.

좌: prompt에 박힌 임시 에이전트, 우: `.claude/agents/NAME.md` 정식 정의

---

6. 5분 미니 실습 — 메인 vs 서브 한 번씩

전제

- Ch10을 끝내 SKILL.md 한 장을 작성해봤다
- 작은 작업 1개 + 큰 작업 1개를 미리 정해 둔다

단계

Step 1 — 메인 에이전트 (작은 일)

새 대화창에 한 줄:

```
data/today-schedule.md 를 읽고 [요일 / 시간 / 항목] 표로 만들어줘.
```

✓ 체크포인트: 메인 Claude가 알아서 읽기 → 추출 → 표 생성의 자율 루프를 돈다. **모든 단계가 메시지 창에 보인다.**

Step 2 — 서브에이전트 (큰 일)

`.claude/agents/doc-summarizer.md`

작성:

```
---
name: doc-summarizer
description: 긴 PDF·정책 문서에서 우리에게 필요한 5조항만 추출. "긴 문서 요약" 표현에 사용.
tools: Read
model: haiku
---

You are a document summarizer.
Read the file, extract only the 5 most relevant clauses.
Return: 5 sentences, each one clause. No full text.
```

같은 대화에서 큰 PDF/긴 마크다운에:

```
data/policy-100p.md 에서 출장 규정만 5조항 뽑아줘.
```

✓ 체크포인트: 메인이 doc-summarizer를 호출했다는 표시 + **메인 메시지창엔 5줄만** 들어옴. 원본 100페이지는 안 보임.

Step 3 — 차이 직접 비교

두 작업의 메시지 창을 나란히 본다.

- Step 1 (메인): 모든 중간 단계가 메시지에 쌓임
- Step 2 (서브): 결과 5줄만. 깔끔.

결과 인증

이 실습을 끝냈다면:

- 같은 도메인의 작은 일·큰 일이 어떻게 다르게 다뤄지는지 안다

- 메인 메시지창의 "무게" 차이를 눈으로 봤다
- 다음 일에 1초 안에 자리 결정 가능

⚠️ 트러블슈팅

문제 1: 서브에이전트가 호출은 됐는데 결과가 어색합니다. → description의 트리거 표현 부족. 키워드 2~3개 명시.

문제 2: 서브에이전트가 권한 없는 도구를 시도합니다. → `tools:` 화이트리스트가 너무 좁음. 필요한 도구만 추가.

문제 3: 작은 일에 서브를 불렀더니 더 느려졌습니다. → 정상. 위임 오버헤드 때문. 5만 토큰 미만은 메인으로.

---

7. 끝맺으며 — 1시간 입문서를 닫는 한 장

여기까지 11장. 첫 페이지에서 약속한 한 줄을 다시 적는다.

> **"코딩을 한 번도 해본 적 없어도, 60분 안에 Claude Code를 자기 손에 익힌다."**

11장의 길이는 책으로는 한 권이지만, 손에 익히는 데 필요한 시간은 60분이다. 매주 한 챕터씩 11주에 나눠 익혀도 된다. 한 권을 끝낸 시점에 손에 남는 건 다음 5가지.

손에 가진 것	어디서
Claude Code 켜고 첫 응답 받는 손	Ch1~Ch3
CLAUDE.md · 자동화 3종 · 6 장치 큰 그림	Ch4~Ch6
슬래시 사전 · 토큰 절약	Ch7~Ch8
커넥터·스킬로 외부 연결·자동화	Ch9~Ch10
메인·서브 에이전트로 자료 크기에 맞게 위임	Ch11

다음 단계는 **자기 일 1개**를 골라 위 5개 중 어울리는 자리에 얹는 것이다. Claude Code는 도구이지 결과물이 아니다. 결과물은 사용자가 자기 일을 줄여 만든 시간이다.

좋은 시간 보내시기를.
