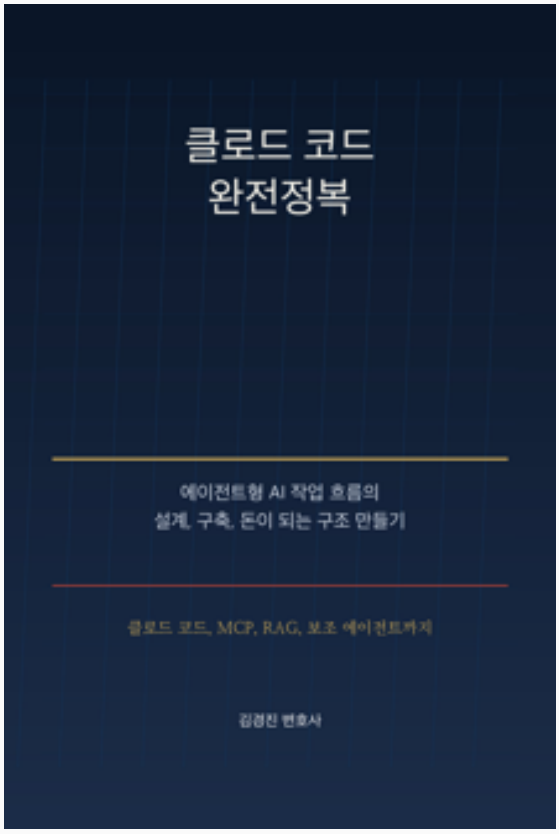
클로드 코드 완전정복

목차, 40장



김경진 변호사

Korean PDF edition

클로드 코드 완전정복

김경진 변호사

클로드 코드를 처음 쓰는 사람을 위한 실전 입문서입니다. 터미널, 권한, 프로젝트 구조, 자동화 흐름을 차근차근 따라갈 수 있도록 구성했습니다.

목차

1장 클로드 코드 실제 사용 사례

전체 목차

책의 모든 장 보기

회의 녹취록 요약과 담당자별 후속 조치 표 만들기

회의가 끝나면 녹취록 파일은 남지만, 그것을 읽고 요약하고 담당자별 후속 조치를 정리하는 일은 사람이 직접 해야 했습니다. 클로드 코드에 녹취록 파일 한 개와 한 줄 지시만 주면, 몇 분 뒤에는 Word 파일로 정리된 회의록이 폴더에 놓여 있습니다.

준비물은 단순합니다. Zoom, Google Meet, Microsoft Teams, 파이어플라이즈, 클로바노트, Granola 중 어떤 도구를 쓰든 회의가 끝나면 녹취록을 텍스트 파일로 내보낼 수 있습니다. Zoom은 "Save Transcript"로 .vtt 또는 .txt 파일을 저장하고, Teams는 회의 상세 페이지에서 .docx로 받을 수 있습니다.

이 파일을 프로젝트 폴더의 transcripts 하위 폴더에 넣는 것이 전부입니다.

터미널에서 프로젝트 폴더로 이동한 뒤 클로드 코드를 실행합니다.

```
cd ~/ai_workspace/meetings

source ~/ai_workspace/venv/bin/activate

mkdir -p transcripts summaries

claude
```

대화가 시작되면 자연어로 지시합니다. "transcripts 폴더의 2026-04-15_임원회의.txt 파일을 읽고 10줄 요약, 결정 사항 목록, 담당자별 후속 조치 표가 포함된 Word 파일을 summaries 폴더에 저장해줘. 파일명은 원본 날짜와 회의명을 따라줘."

클로드 코드는 python-docx 라이브러리를 내부에서 호출하여 실제로 열고 편집 가능한 .docx 파일을 생성합니다. 결과물에는 세 가지 요소가 들어갑니다. 첫째, 회의 성격을 10줄로 압축한 요약입니다. 90분 회의든 20분 회의든 핵심 논지가 같은 밀도로 정리됩니다. 둘째, 결정된 사항만 따로 뽑은 목록입니다.

안건별로 "무엇을 결정했는가, 왜 그렇게 결정했는가"가 붙습니다. 셋째, 담당자/할 일/마감일 세 열로 구성된 후속 조치 표입니다. 녹취록에 "제가 수요일까지 보내드릴게요" 같은 발언이 있으면, 담당자 이름과 마감일이 자동으로 추출됩니다.

결과물의 쓸모는 프롬프트의 구체성에 비례합니다. "회의 요약해줘"보다는 맥락을 같이 주는 편이 낫습니다. "이 녹취록은 2026년 4월 15일 임원회의 기록이고, 참석자는 대표, CTO, 마케팅 이사 세 사람이다. 결정 중심으로 요약하고, 발언자별 성향 분석은 빼라"라고 쓰면 출력이 훨씬 정돈됩니다.

슬랙이나 노션에 바로 올릴 용도라면 형식을 추가로 지시할 수 있습니다. "슬랙에 붙일 거니까 마크다운 없이 300자 이내 평문으로 요약본을 하나 더 만들어줘. 노션에 붙일 거면 체크박스와 소제목이 있는 마크다운으로."

반복 업무라면 이 과정을 CLAUDE.md에 영구 지시로 저장해두는 것이 좋습니다.

cat 〈〈 'EOF' 〉 ~/ai_workspace/meetings/CLAUDE.md

이 프로젝트는 회의 녹취록을 구조화된 회의록으로 변환합니다.

transcripts 폴더에서 .txt 또는 .vtt 파일을 읽고,

summaries 폴더에 Word 파일로 저장합니다.

회의록 형식: (1) 10줄 요약 (2) 결정 사항 목록 (3) 담당자별 후속 조치 표.

담당자 이름은 녹취록에서 언급된 사람으로 맞춥니다.

마감일이 명시되지 않았으면 3영업일 뒤로 기본 설정합니다.

경어체로 씁니다. 요약 과정에서 원문을 임의로 생략하지 않습니다.

EOF

다음 회의부터는 파일을 transcripts 폴더에 떨어뜨리고 "새 녹취록 처리해줘"라고 한 줄만 말하면 됩니다. 세션이 바뀌어도 CLAUDE.md의 규칙이 매번 로드되므로, 품질이 일정하게 유지됩니다.

한 가지 주의할 점이 있습니다. 녹취록은 회의 참석자의 발언이 그대로 담긴 민감 정보입니다. 법률 자문 회의, 인사 논의, 고객 미팅 내용은 조직의 데이터 정책에 따라 외부 AI에 전송 가능한지 먼저 확인해야 합니다. 변호사 업무에서는 의뢰인 정보가 포함된 녹취록은 특히 신중해야 합니다. 내부 규정에 따라 API 경로만 사용하거나, 익명화를 거친 뒤 올리는 절차가 필요합니다.

클로드가 만든 회의록은 초안입니다. 숫자, 날짜, 금액, 법 조항 같은 오류가 치명적인 항목은 반드시 원문 녹취록과 대조해야 합니다. 특히 "몇 퍼센트 인상에 합의했다", "언제까지 납품하기로 했다" 같은 구체적 약속은 녹취록 원문을 다시 찾아 확인하는 과정을 건너뛰면 안 됩니다.

다운로드 폴더 자동 정리

다운로드 폴더는 방치하면 수백 개의 파일이 섞여 쌓입니다. PDF 영수증, 스크린샷, 계약서 초안, 이름이 뒤엉킨 CSV, 오래된 설치 파일까지 모두 한곳에 있습니다. 찾고 싶은 파일을 다시 찾는 데만 몇 분이 걸립니다. 클로드 코드를 쓰면 한 번의 지시로 파일을 유형과 날짜별로 분류하고, 오래된 것은 보관소로 옮기고, 의심스러운 파일은 따로 빼두는 작업이 몇 분 만에 끝납니다.

시작 전 반드시 지켜야 할 원칙이 하나 있습니다. 파일 이동 스크립트는 처음부터 실제 이동시키지 말고, 먼저 드라이런으로 돌려서 무엇이 어디로 가는지 확인해야 합니다. 중요한 파일이 엉뚱한 폴더로 옮겨지면, 그 파일을 다시 찾는 일이 정리 전보다 더 고통스럽습니다.

터미널에서 작업 폴더를 준비합니다.

cd ~/ai_workspace/file_automation

source ~/ai_workspace/venv/bin/activate

claude

클로드 코드가 실행되면 지시합니다. "~/Downloads 폴더를 정리하는 파이썬 스크립트를 만들어줘. 이미지(.jpg .png .gif .webp)는 Pictures 폴더로, PDF는 Documents/PDFs로, 동영상(.mp4 .mov .mkv)은 Videos로, 코드 파일(.py .js .ts)은 Projects/Code로 분류하고, 나머지는 Other로 보내줘.

6개월 이상 된 파일은 Archive로 옮기고, 같은 이름 파일이 둘 이상 발견되면 'DuplicateCheck' 폴더로 분리해줘. 아무것도 삭제하지 말고, --dry-run 플래그를 기본으로 켜줘. 실행 로그를 organize_log.txt에 기록해줘."

클로드 코드가 스크립트를 작성한 뒤, 터미널에서 드라이런으로 먼저 돌려봅니다.

python organize_downloads.py --dry-run

출력 로그를 확인한 뒤, 의도대로 움직일 것 같으면 실제 실행합니다.

python organize_downloads.py --execute

186개 파일 기준으로 약 1분 30초에서 2분 안에 정리가 끝납니다. 파일 내용까지 읽어서 분류하도록 지시하면 시간은 더 걸리지만 정확도가 올라갑니다. 예를 들어 "image_001.jpg"라는 이름의 파일이 실제로는 스캔된 영수증이라면, 확장자만 보고 Pictures로 보내는 대신 내용을 읽어 Documents/Receipts로 보내는 판단이 가능합니다.

이 작업을 한 번 만들어두면 다시 짤 필요가 없습니다. 완성된 스크립트는 버전 관리해두고, 매달 한 번씩 돌리면 됩니다. 크론탭에 등록해 매주 일요일 새벽에 자동 실행되게 할 수도 있습니다.

crontab -e

0 3 0 cd ~/ai_workspace/file_automation && source ~/ai_workspace/venv/bin/activate && python organize_downloads.py --execute >> ~/logs/organize.log 2>&1

여기서 한 걸음 더 나아가면, 내용 기반 분류도 가능합니다. 영수증 PDF를 폴더에 모으고 "각 파일의 PDF 내용을 읽어서 '연도-월-일_업체명_금액' 형식으로 이름을 바꿔줘. 원래 이름과 바뀐 이름을 기록한 changelog.txt도 남겨줘"라고 하면, OCR과 텍스트 추출을 거쳐 일괄 이름 변경이 이루어집니다.

OCR 인식이 불확실한 항목은 파일명에 "〔확인필요〕" 접두어가 붙어서 사람이 검토할 수 있도록 표시됩니다.

변호사 사무실에서 이 방식을 응용하면 쓸모가 더 커집니다. 사건별로 '의뢰인이 보내준 자료' 폴더가 만들어지는데, 여기에는 사진, 스캔 파일, 한글 문서, 녹취록이 뒤섞여 들어옵니다. 분류 규칙을 사건 성격에 맞게 짜두면, 의뢰인이 자료를 보낼 때마다 폴더 구조가 자동으로 정돈됩니다.

"형사/증거_사진, 형사/증인진술서, 형사/의뢰인_메모, 민사/계약서, 민사/거래내역" 같은 분류를 CLAUDE.md에 저장해두면 됩니다.

큰 원칙 하나만 지키면 됩니다. 파일 작업 스크립트는 항상 드라이런 옵션을 기본값으로 두고, 실제 실행은 명시적 플래그를 붙여야만 동작하도록 만듭니다. 클로드 코드에 스크립트 작성을 맡길 때마다 이 규칙을 프롬프트에 넣으면 실수를 줄일 수 있습니다. 5분 걸리는 반복 업무를 2시간 들여 자동화하면, 그 뒤로 매년 28시간이 남습니다. 시간의 복리 이자입니다.

이메일 분류와 답장 초안 작성

받은 편지함이 일을 지배하는 상황은 누구에게나 익숙합니다. 하루 200통, 300통이 쌓이면 중요한 메일이 광고와 자동 알림 사이에 묻혀버립니다. 클로드 코드에 Gmail을 연결하면, 아침에 30초 안에 "답장 필요 / 한 번 볼 것 / 소음" 세 단계로 분류된 요약이 나오고, 중요 메일에 대해서는 내 말투로 쓰인 답장 초안까지 임시보관함에 저장됩니다.

핵심은 두 가지입니다. 첫째, 절대 자동 발송하지 않는다. 둘째, 내 말투를 학습시킨다.

Gmail을 클로드 코드에 연결하는 방법은 여러 가지인데, Gmail MCP 연결 기능를 쓰는 쪽이 가장 안정적입니다. Google Cloud Console에서 OAuth 자격증명을 만들고, 클로드 코드에 등록합니다.

```
claude mcp add --transport stdio google-workspace \

-- npx -y google-workspace-mcp \

--client-id YOUR_GOOGLE_CLIENT_ID \

--client-secret YOUR_GOOGLE_CLIENT_SECRET
```

최초 실행 시 브라우저에서 구글 계정 인증을 마치면 연결이 완료됩니다. 계정이 여러 개면(개인, 업무, 뉴스레터) 각각 다른 이름으로 등록합니다. gmail-personal, gmail-business처럼 이름을 구분해두면 "업무 메일만 분류해줘" 같은 지시가 가능해집니다.

내 말투 프로파일을 먼저 만들어둡니다. 보낸편지함의 최근 20통을 클로드 코드에게 읽히고, 말투의 특징을 뽑아내게 합니다.

```
claude

"내 gmail-personal 보낸편지함에서 최근 20통을 읽어서 내 말투의 특징을 분석해줘.

문장 길이, 인사말 패턴, 자주 쓰는 접속사, 격식 수준, 사인오프 스타일을 정리해서

'나의_이메일_톤.md' 파일로 저장해줘."
```

이 .md 파일이 이후 답장 초안의 기준이 됩니다. 다음부터는 이렇게 지시하면 됩니다. "지난 12시간 동안 받은 메일을 훑어서 (1) 답장 필요 (2) 한 번 볼 것 (3) 소음 세 단계로 분류해줘. 답장 필요 항목에는 '나의_이메일_톤.md'의 톤을 적용한 답장 초안을 만들어서 Gmail 임시보관함에 저장해줘. 절대 발송하지 말고 초안으로만 저장해줘.

VIP 목록은 (대표, 주요 거래처 대표, 법인카드 담당자)다. VIP 메일은 최우선으로 올려줘."

결과는 이렇게 돌아옵니다. 918통의 안 읽은 메일 중 7통이 "답장 필요"로 올라오고, 각 메일 옆에 초안 번호가 붙습니다. "3번 초안 어투를 조금 더 부드럽게 바꿔줘"라고 이어서 말하면 즉시 수정됩니다.

이 워크플로를 매일 아침 자동 실행시키려면, 스케줄 훅을 걸어둘 수 있습니다.

cat 〈〈 'EOF' 〉 ~/.claude/commands/morning-email.md

아침 이메일 브리핑 루틴입니다.

1. gmail-personal, gmail-business 두 계정의 지난 12시간 메일을 확인합니다.

2. VIP 발신자(config에 저장된 목록)의 메일을 최우선으로 올립니다.

3. 답장 필요 / 한 번 볼 것 / 소음으로 분류합니다.

4. 답장 필요 항목에 대해 나의_이메일_톤.md의 말투로 초안을 씁니다.

5. 초안은 반드시 Gmail 임시보관함에만 저장합니다. 자동 발송 절대 금지.

6. 결과 요약을 Morning_Email_Brief.md 파일로 저장합니다.

EOF

이제 매일 아침 "/morning-email"이라고 치면 이 루틴이 실행됩니다. 사람이 하는 일은 초안을 읽고, 고치고, 보내기 버튼을 누르는 것뿐입니다.

변호사 업무에 이 방식을 쓰면 특히 쓸모가 큽니다. 사건별로 의뢰인과 주고받은 메일 스레드가 수백 통 쌓이는데, 아침마다 어느 사건에서 무엇이 진척됐는지 파악하는 데만 30분이 걸립니다. "사건번호 라벨이 붙은 메일을 사건별로 그룹핑하고, 각 사건에서 최근 24시간 안에 일어난 일을 한 문단으로 요약해줘.

의뢰인이 새로 제공한 자료, 상대방이 보낸 내용, 내가 약속한 후속 조치를 구분해서 정리해줘"라고 지시하면, 사건별 브리핑이 즉시 나옵니다.

절대 건너뛰면 안 되는 원칙이 있습니다. AI가 쓴 이메일 초안을 그대로 보내지 않는 것입니다. 친구에게 보낼 편지, 비즈니스 파트너에게 보낼 제안, 의뢰인에게 보낼 회신은 말 한 마디의 뉘앙스가 결과를 바꿉니다. 코드는 AI에게 맡겨도 되지만, 이메일은 마지막에 사람이 읽고 고쳐야 합니다. 버그가 있는 메일은 이미 보낸 뒤 지울 수 없습니다.

법률 자문 메일에서 단어 하나가 잘못 쓰이면, 의뢰인은 그 메일을 근거로 움직입니다. 초안은 빠르게 만들되, 보내기 버튼은 사람이 누른다는 원칙을 한 번도 어기지 않는 것이 변호사의 실무에서는 특히 중요합니다.

파일 일괄 이름 변경 — 내용을 읽어서 바꾸기

영수증, 계약서, 스캔한 세금계산서처럼 내용은 구조화되어 있지만 파일명은 엉망인 경우가 많습니다. IMG_2341.jpg, scan_0019.pdf, 무제-3.pdf 같은 이름들 사이에서 특정 거래를 찾으려면 파일을 하나씩 열어봐야 합니다. 클로드 코드는 파일 안을 직접 읽고, 일관된 규칙에 따라 이름을 다시 붙여줍니다.

작업 순서는 항상 같습니다. 첫째, 백업을 만듭니다. 둘째, 드라이런으로 결과를 먼저 봅니다. 셋째, 실제 이름 변경을 실행합니다. 넷째, 변경 로그를 남깁니다.

터미널에서 작업 폴더를 준비합니다.

cd ~/Documents/receipts_2026Q1

cp -r . ../receipts_2026Q1_backup_$(date +%Y%m%d)

claude

클로드 코드에 지시합니다. "이 폴더의 모든 영수증 PDF와 JPG 파일 내용을 읽고, 상호명, 거래 일자, 금액을 추출해줘. 파일명을 '2026-MM-DD_상호명_금액.확장자' 형식으로 바꿔줘. 상호명에 공백이 있으면 언더스코어로 바꿔줘. 추출이 애매한 파일은 이름 앞에 '〔확인필요〕'를 붙이고 rename_log.csv에 이유를 기록해줘. 먼저 --dry-run으로 돌려서 어떻게 바뀔지 보여줘."

클로드가 만든 스크립트를 드라이런으로 확인합니다. 예상 변경 내역이 표 형태로 출력됩니다.

원본 파일명 → 새 파일명

IMG_2341.jpg → 2026-01-15_스타벅스_4800.jpg

scan_0019.pdf → 2026-01-22_서울가든_87000.pdf

무제-3.pdf → 〔확인필요〕_무제-3.pdf (상호명 불명확)

의도대로면 실제 실행합니다.

python rename_receipts.py --execute

변경 로그에는 원본 파일명, 새 파일명, 추출된 데이터, 신뢰도 점수가 모두 기록됩니다. 이 로그 파일 하나만 있으면 나중에 문제가 생겼을 때 어느 파일을 어떻게 바꿨는지 100% 추적할 수 있습니다.

법률사무소에서는 이 방식이 특히 유용합니다. 사건 기록은 수천 개 PDF가 쌓이는데, 스캔해서 올린 파일은 대부분 무의미한 파일명을 가지고 있습니다.

"각 파일에서 사건번호, 문서유형(소장/답변서/준비서면/증거), 작성일자를 추출해서 '사건번호_문서유형_작성일자.pdf' 형식으로 이름을 바꿔주고, 사건번호별 하위 폴더로 이동시켜줘"라고 지시하면, 며칠 걸릴 분류 작업이 몇 시간 안에 끝납니다. 사건번호 추출이 불가능한 파일은 '미분류' 폴더로 빠집니다.

판별이 애매한 파일은 항상 '〔확인필요〕' 접두어로 표시하도록 지시하는 것이 안전합니다. AI가 "이 정도면 맞겠지"라고 추정한 결과를 그대로 받아들이면, 사건 기록에서 한두 개 파일이 잘못된 사건번호로 분류되는 일이 발생할 수 있습니다. 법률문서에서 이런 실수는 치명적입니다.

응용 범위는 계속 넓어집니다. 사진 폴더에서는 "EXIF 메타데이터를 읽어서 '촬영일자_장소.jpg' 형식으로 바꿔줘"가 가능합니다. 동영상 녹화본은 "화면 녹화 프레임을 분석해서 '날짜_프로그램명_활동내용.mp4'로 바꿔줘"가 됩니다. 논문 PDF는 "파일에서 저자, 연도, 제목을 추출해서 '저자_연도_제목.pdf' 형식으로 바꿔줘"로 끝납니다.

파일을 여는 사람의 습관에 맞춰 규칙만 바꾸면, 같은 엔진이 온갖 폴더를 정리합니다.

가장 중요한 원칙 하나. 이름 변경은 되돌리기가 어렵습니다. 1,000개 파일을 일괄 변경한 뒤 결과가 엉뚱하면, 로그를 보고 하나씩 되돌리는 수밖에 없습니다. 그래서 세 가지를 항상 지킵니다. 백업을 떠둔다. 드라이런으로 먼저 본다. 로그를 남긴다. 이 세 가지를 프롬프트에 미리 못 박아두면, 같은 실수를 반복하지 않습니다. 사소해 보이는 이 세 줄이, 주말 하루를 통째로 잃는 사고를 막아줍니다.

영수증과 세금계산서 엑셀화

스캔 영수증과 PDF 세금계산서가 수십 장 쌓이면, 매달 말 정산이 고통입니다. 엑셀을 열어 상호명, 날짜, 금액, 카테고리를 하나씩 입력하는 일은 단순하지만 지겹고, 실수가 잘 납니다. 클로드 코드는 이 작업을 한 번의 지시로 끝냅니다. 수식까지 살아 있는 실제 .xlsx 파일이 폴더에 생성됩니다.

준비는 간단합니다. 영수증 사진과 PDF를 한 폴더에 모으고, 클로드 코드를 그 폴더에서 실행합니다.

```
mkdir -p ~/Documents/receipts_2026April

# 영수증 파일들을 이 폴더에 넣는다

cd ~/Documents/receipts_2026April

source ~/ai_workspace/venv/bin/activate

claude
```

지시합니다. "이 폴더에 있는 모든 영수증 이미지와 PDF를 읽고, 상호명, 날짜, 금액, 결제 카테고리(식비/교통/사무용품/접대비/기타)를 추출해줘. 엑셀 파일로 저장하되, 날짜순으로 정렬하고, 카테고리별 합계를 SUMIF 수식으로 넣어줘. OCR 인식이 애매한 항목은 비고란에 '확인필요'로 표시해줘. 파일명은 receipts_2026April.xlsx로 해줘."

몇 분 뒤 폴더에 엑셀 파일이 생성됩니다. 열어보면 여섯 열로 구성된 표가 있습니다. 날짜, 상호명, 금액, 카테고리, 결제수단, 비고. 표 아래에는 카테고리별 합계가 수식으로 들어가 있습니다. 엑셀에서 직접 숫자를 바꿔도 합계가 자동으로 갱신됩니다.

수식이 실제로 동작하는 엑셀 파일이라는 점이 핵심입니다. 클로드 코드는 openpyxl 라이브러리를 내부에서 호출해서 셀 값뿐 아니라 수식, 조건부 서식, 피벗 테이블까지 파일에 기록합니다. 열어보면 =SUMIF(D:D,"식비",C:C) 같은 수식이 실제로 남아 있습니다.

정산 카테고리가 회사마다 다르다면, CLAUDE.md에 카테고리 정의를 저장해둡니다.

cat 〈〈 'EOF' 〉 ~/Documents/receipts_template/CLAUDE.md

영수증 카테고리 분류 기준:

- 식비: 음식점, 카페, 편의점 식품

- 교통: 주유, 택시, KTX, 항공권, 주차

- 사무용품: 문구, 잉크, 소프트웨어, 전자기기

- 접대비: 거래처 미팅 식대, 선물

- 통신: 휴대폰, 인터넷, 클라우드 서비스

- 교육: 도서, 세미나, 온라인 강의

- 기타: 위 분류에 해당하지 않는 모든 것

엑셀 출력 규칙:

- 날짜 형식: YYYY-MM-DD

- 금액은 숫자로만 기록 (₩ 기호 제외)

- 부가세 10%는 별도 열로 분리 기록

- 카테고리별 합계를 SUMIF 수식으로 표 하단에 추가

- 월 전체 합계와 부가세 합계를 맨 아래에 기록

EOF

이제 다음 달부터는 파일을 폴더에 떨어뜨리고 "이번 달 영수증 엑셀로 만들어줘"라고 한 줄만 말하면 됩니다.

세금계산서는 한 단계 더 나아갈 수 있습니다. 홈택스에서 받은 전자세금계산서 XML을 폴더에 모으고, "XML을 파싱해서 공급자, 공급받는자, 품목, 공급가액, 세액, 합계를 엑셀로 정리해줘. 매출분과 매입분을 별도 시트로 나눠줘. 월별 부가세 합계를 한 시트에 따로 요약해줘"라고 지시하면, 부가세 신고 기초자료가 바로 나옵니다.

변호사 사무실에서는 이 방식을 수임료 관리에도 씁니다. 의뢰인별로 수임계약서, 입금 확인증, 출장 경비 영수증이 섞여 쌓이면, 사건이 끝난 뒤 정산이 복잡해집니다. "사건번호별로 폴더에 들어있는 영수증과 입금증을 읽어서, 수임료 청구분과 실비 상환분을 별도 열로 분리한 엑셀을 만들어줘.

각 사건별로 총 실비, 총 수임료, 미수 잔액을 계산해줘"라고 지시하면, 의뢰인에게 바로 보낼 수 있는 정산서 초안이 완성됩니다.

OCR 인식률이 떨어지는 영수증은 반드시 존재합니다. 흐릿한 사진, 구겨진 종이, 영수증 뒷면의 광고 문구가 섞인 경우. AI는 이런 파일을 "확인필요"로 표시하지만, 사람이 마지막에 비고란을 훑어서 직접 수정해야 합니다. 법인 정산에서는 건별 금액이 정확해야 하고, 세무조사에서는 원본 영수증과 장부 기록이 일치해야 합니다.

숫자 자동화의 편리함은 숫자 검증의 책임을 줄여주지 않습니다. 엑셀을 만드는 데 30분이 걸리든 30초가 걸리든, 숫자가 맞다고 확신하는 일은 사람의 몫입니다.

여러 문서를 하나의 보고서로 병합

사건 하나, 프로젝트 하나를 마무리할 때쯤 폴더를 열면 파일 형식이 제각각입니다. PDF 자료조사 보고서, 텍스트 메모, 마크다운 노트, CSV 데이터, 엑셀 시트, 워드 초안까지. 이걸 하나의 워드 문서로 합치려면 열어서 복사하고 붙여넣고 형식을 맞추는 지루한 작업이 반나절입니다. 클로드 코드는 파일 형식이 달라도 내용을 읽어서 하나의 보고서로 조립합니다.

원리는 간단합니다. 클로드 코드는 pypdf, python-docx, openpyxl, pandas 같은 라이브러리를 상황에 맞게 골라서 파일을 읽고, 형식이 다른 내용을 텍스트로 통일한 뒤 최종 문서를 조립합니다. 사람이 할 일은 "원본이 어디에 있고, 결과물을 어떻게 만들고 싶다"는 지시를 한 번 주는 것뿐입니다.

작업 폴더를 준비합니다.

cd ~/ai_workspace/reports/case_2026_042

ls

investigation_report.pdf, interview_notes.txt, evidence_log.xlsx,

timeline.md, witness_statements.docx, financial_data.csv

클로드 코드를 실행하고 지시합니다.

claude

"이 폴더의 모든 파일을 읽고, 하나의 사건 종결 보고서를 Word로 조립해줘. 구성은 (1) 사건 개요 (2) 시간순 타임라인 (3) 관련자 진술 요약

(4) 증거 목록 (5) 재무 분석 (6) 결론 및 후속 조치 순서로 해줘. 날짜 표기는 YYYY-MM-DD로 통일하고, 금액은 모두 원화(₩) 기준으로 통일해줘. CSV의 표는 Word 표로 변환해서 넣어줘.

각 섹션 끝에 출처 파일명을 각주로 달아줘. 파일명은 Case_2026_042_Final_Report.docx로 저장해줘."

클로드 코드는 내부적으로 세 단계를 거칩니다. 첫째, 추출 단계. 각 파일을 형식에 맞는 라이브러리로 열어 텍스트로 변환합니다. 둘째, 정제 단계. 날짜 표기, 통화 단위, 이름 표기를 통일합니다. 같은 사람이 "홍길동"과 "홍 길동"으로 섞여 있으면 한 이름으로 정리합니다. 셋째, 조립 단계. 지시받은 섹션 구조에 맞춰 내용을 재배치하고, CSV의 표는 실제 Word 표로 변환해 삽입합니다.

파일 수가 많으면 클로드 코드는 보조 에이전트를 병렬로 가동합니다. 한 에이전트가 PDF를 읽는 동안 다른 에이전트가 엑셀을 파싱하고, 또 다른 에이전트가 녹취록을 요약합니다. 50개 파일이 섞인 폴더도 몇 분 안에 정리됩니다.

변호사 업무에서 이 방식은 특히 유용합니다. 사건이 몇 달에 걸쳐 진행되면, 클라이언트가 보낸 자료, 상대방 답변서, 증인 진술서, 판례 자료, 은행 거래내역이 파일별로 쌓입니다. 재판 준비서면을 쓸 때 이 모든 자료를 훑어보며 사실관계를 정리하는 일이 제일 오래 걸립니다. "이 폴더에 든 모든 자료를 읽고, 시간순 사실관계 정리표를 만들어줘.

각 사실의 출처 파일명과 페이지를 반드시 기재해줘. 상충하는 진술이 있으면 별도 섹션에 표시해줘" 라고 지시하면, 준비서면 초안의 뼈대가 한 시간 안에 나옵니다.

한 가지 명심할 점. 클로드가 조립한 보고서는 초안입니다. 법률문서에서 사실 하나가 틀리면 전체 논리가 무너집니다. 각 섹션 끝에 달아둔 출처 파일명과 페이지를 근거로, 중요한 사실은 반드시 원본을 다시 열어 확인해야 합니다. 특히 금액, 날짜, 당사자 이름, 계약 조항 번호는 눈으로 한 번 더 대조하는 과정을 건너뛸 수 없습니다.

반복되는 보고서 형식이 있다면 CLAUDE.md에 템플릿을 저장해둡니다.

cat 〈〈 'EOF' 〉 ~/ai_workspace/reports/CLAUDE.md

사건 종결 보고서 표준 구성:

1. 사건 개요 (사건번호, 수임일, 종결일, 의뢰인, 상대방, 사건 성격)

2. 시간순 타임라인 (날짜/사건/출처 3열 표)

3. 관련자 진술 요약 (인물별 그룹핑)

4. 증거 목록 (증거번호/제목/제출일/출처 4열 표)

5. 재무 분석 (총 수임료, 실비 내역, 미수금)

6. 결론 및 후속 조치

통일 규칙:

- 날짜: YYYY-MM-DD

- 금액: ₩ 기호 + 천 단위 콤마

- 사람 이름: 띄어쓰기 없이 성+이름

- 법 조항: 「법률명」 제○조 제○항 형식

출처 표기: 각 사실문 끝에 (파일명, 페이지) 형식 각주

EOF

다음 사건부터는 "사건 폴더에 든 자료로 종결 보고서 만들어줘"라고 한 줄만 말하면 같은 형식의 보고서가 나옵니다. 형식이 일정하면 읽는 사람이 사건마다 다시 구조를 파악할 필요가 없고, 사무실 내 자료 관리도 일관성을 얻습니다.

주간 보고서 자동 생성 — Notion, Slack, Google Drive 통합

월요일 아침 주간 보고서를 쓰려고 책상에 앉으면, 지난 한 주 동안 벌어진 일을 다시 끌어모으는 것부터가 일입니다. Notion 프로젝트 보드를 열어 완료 이슈를 세고, Slack을 거슬러 올라가며 결정 사항을 찾고, Google Drive에서 새로 올라온 문서를 확인합니다. 자료 수집에만 한 시간, 정리에 또 한 시간이 걸립니다. 클로드 코드는 세 도구를 동시에 연결해서 이 과정을 몇 분으로 줄입니다.

먼저 세 개 커넥터를 연결합니다. Notion, Slack, Google Drive 각각 공식 MCP 연결 서버가 제공됩니다.

claude mcp add --transport http notion https://mcp.notion.com/mcp

claude mcp add --transport http slack https://mcp.slack.com/mcp

claude mcp add --transport stdio google-workspace \

-- npx -y google-workspace-mcp \

--client-id YOUR_GOOGLE_CLIENT_ID \

--client-secret YOUR_GOOGLE_CLIENT_SECRET

각 커넥터의 최초 인증은 브라우저에서 OAuth 창이 열리고, 계정과 접근 범위를 선택하는 한 번의 절차입니다. Notion은 특정 워크스페이스와 페이지에만 권한을 주고, Slack은 필요한 채널에만 접근하도록 제한할 수 있습니다. 의뢰인 정보가 들어 있는 노션 페이지나 법무팀 전용 슬랙 채널은 의식적으로 범위를 좁혀 연결하는 편이 안전합니다.

연결이 끝나면 클로드 코드 세션에서 지시합니다.

claude

"매주 금요일 오후 5시에 실행될 주간 보고서 스킬을 만들어줘.

Notion의 '프로젝트 보드' 데이터베이스에서 지난 7일 동안 상태가 '완료'로 바뀐 항목을 모두 가져와.

Slack의 #project-updates, #legal-team 채널에서 지난 7일간 결정 사항 메시지를 찾아줘

(이모지 가 붙은 메시지를 결정 사항으로 간주).

Google Drive의 '주간 자료' 폴더에서 이번 주에 수정된 문서 목록을 가져와.

이 세 가지를 합쳐서 주간 보고서 Word 문서를 만들어줘.

구조는 (1) 이번 주 완료 업무 (2) 주요 결정 사항 (3) 새로 올라온 자료 (4) 다음 주 예정 업무.

결과물은 '~/weekly_reports/2026_Week_XX.docx'로 저장해줘."

클로드 코드는 세 도구를 동시에 호출합니다. 노션에서 데이터베이스 필터를 걸어 완료 항목만 가져오고, 슬랙에서 채널별 메시지를 검색해 이모지가 붙은 것만 추려냅니다. 드라이브에서는 modifiedTime 필드로 최근 변경 파일만 필터링합니다. 이 모든 결과를 하나의 보고서로 조립해 Word 파일로 떨어뜨립니다.

더 나아가 이 작업을 스케줄로 올릴 수 있습니다. 2026년 4월 14일 출시된 Claude Code Routines를 쓰면 매주 금요일 오후 5시에 자동 실행됩니다.

claude

"/schedule 매주 금요일 오후 5시에 주간 보고서 루틴 실행"

Routines는 Anthropic 클라우드에서 실행되기 때문에 노트북을 닫아도, 출장 중이어도 돌아갑니다. Pro 플랜은 하루 5회, Max는 15회, Team과 Enterprise는 25회 실행이 가능합니다. 내 컴퓨터 맥에서만 돌리고 싶다면 Desktop의 Scheduled Tasks를 쓰는 쪽이 단순합니다.

이 경우 컴퓨터가 켜져 있고 Claude Desktop 앱이 열려 있어야 한다는 제약이 있지만, 내 컴퓨터 파일 접근은 훨씬 자유롭습니다.

주의점 두 가지가 있습니다. 첫째, 민감 데이터는 연결 범위를 신중하게 정해야 합니다. 의뢰인 이름, 사건 상세, 계약 금액 같은 내용은 연결된 서비스를 거쳐 AI 서버로 오가는 구조이므로, 수임 계약에서 AI 도구 사용을 허용한 경우에만 쓰거나, 익명화 처리를 거친 별도 워크스페이스를 만들어 운영하는 편이 낫습니다.

둘째, 보고서 초안을 그대로 상급자에게 보내지 않는 것이 좋습니다. Notion 데이터베이스 필터가 잘못 걸려서 완료 항목이 누락되는 일이 가끔 생깁니다. 사람이 마지막에 한 번 훑어보고, "이 항목이 빠졌다"고 지적하면 즉시 수정해 다시 생성합니다. 자동화는 사람의 판단을 대체하는 것이 아니라 사람이 판단할 시간을 만들어주는 도구입니다.

경쟁사 뉴스 모니터링과 브리핑

법률시장의 경쟁 환경, 새로 나온 판례, 관심 기업의 공시 같은 정보는 매일 따라가야 쓸모가 있습니다. 하지만 뉴스 사이트 열 개를 돌아다니며 읽고, 중요한 것을 추려내는 일은 하루 한두 시간을 잡아먹습니다. 클로드 코드는 웹 검색을 도구로 품고 있어서, "오늘 이 기업들에 대해 새로 나온 뉴스를 정리해달라"고 한 번 말하면 알아서 검색하고 요약해 폴더에 저장해줍니다.

이 작업의 핵심은 검색 범위와 요약 형식을 구체적으로 지정하는 것입니다.

claude

"매일 아침 7시에 실행될 경쟁사 모니터링 루틴을 만들어줘. 대상 기업은 법무법인 김&장, 광장, 세종, 율촌, 태평양 다섯 곳이고,

확인할 내용은 (1) 새로 영입한 파트너 변호사 (2) 새로운 프랙티스 그룹 출범

(3) 주요 수임 공지 (4) 미디어 인터뷰·칼럼 기고 네 가지야. 지난 24시간 동안 발표된 내용만 가져오고, 한국어 뉴스 위주로 검색해줘.

결과는 '~/intel/YYYY-MM-DD_law_market_brief.md' 파일로 저장하고,

각 항목 끝에 출처 URL을 반드시 적어줘. 중요도는 상/중/하로 분류해서 상 항목만 맨 위에 모아줘."

클로드는 웹 검색 도구를 써서 각 기업명과 키워드 조합으로 검색합니다. "김&장 파트너 영입 2026", "광장 새 그룹 출범"처럼 날짜를 포함해 쿼리를 만들고, 상위 결과를 읽어 핵심만 뽑아냅니다. 결과는 마크다운 파일로 저장되어 Obsidian이나 Notion에서 바로 열어볼 수 있습니다.

기업 뉴스만이 아니라 판례·법령 모니터링에도 같은 패턴이 들어맞습니다.

claude

"매주 월요일 아침 8시에 실행될 신규 판례 브리핑을 만들어줘. 관심 영역은 (1) AI·데이터 관련 판례 (2) 개인정보보호법 위반 (3) 스타트업 투자계약 분쟁 세 가지. 국내 법원 사이트(law.go.kr, scourt.go.kr)와 주요 로펌 블로그에서

지난주 선고·공개된 판례를 찾아줘.

각 판례에 대해 사건번호, 쟁점, 판시사항을 정리하고,

왜 실무에 중요한지 2에서 3문장 해설을 덧붙여줘. 결과는 '~/intel/YYYY-MM-DD_precedent_brief.md'로 저장하고,

'~/blog_ideas.md' 파일에 블로그 포스팅 아이디어 3개를 제안해줘."

여기서 한 걸음 나아가면, 수집한 정보를 단순히 저장하는 데 그치지 않고 바로 글로 만들 수도 있습니다. 경쟁사 동향 요약 뒤에 "이 정보 중 블로그 포스팅 주제로 좋을 만한 것 3개를 뽑고, 각 주제에 대해 개요를 써줘"라고 덧붙이면, 매일 아침 콘텐츠 기획 자료까지 함께 나옵니다.

개인 영역에서도 같은 패턴이 쓰입니다. 투자 포트폴리오에 담긴 기업들, 내가 쓰는 서비스들의 약관 변경, 관심 있는 기술 분야의 주요 논문을 매일 훑어 하루치 요약을 받는 식입니다. "삼성전자, SK하이닉스, 네이버의 지난 24시간 주가 관련 뉴스와 공시를 확인해줘. 가격 변동률이 ±3% 이상인 종목은 맨 위로 올려줘.

중요한 공시는 전문을 요약해줘"라고 지시하면, 개별 종목 분석을 위한 기초자료가 매일 아침 폴더에 쌓입니다.

웹 검색 결과에는 한 가지 맹점이 있습니다. 출처의 신뢰도가 들쭉날쭉합니다. AI가 검색해서 가져온 뉴스는 가끔 블로그 요약본이나 두세 단계 걸러진 기사이기도 하고, 드물게는 잘못된 날짜가 붙은 과거 기사일 때도 있습니다. 그래서 결과 파일에 반드시 원본 URL을 출처로 남기도록 지시합니다. 중요한 내용은 사람이 원본을 한 번 더 열어 확인합니다.

특히 "누가 어디로 옮겼다"는 인사 동향 뉴스는 같은 이름의 다른 사람과 혼동되는 경우가 종종 있으므로, 원문의 사진과 경력 소개를 함께 보는 습관이 필요합니다.

시장 분석 보고서나 경제 지표를 다룰 때는 또 다른 주의가 필요합니다. AI는 수치를 그럴듯하게 말하는 데 능하지만, 수치 자체의 정확성을 보증하지는 않습니다. "이번 주 원유 가격이 전주 대비 4.2% 상승했습니다"라는 문장이 브리핑에 나와도, 실제로는 3.8%일 수 있습니다.

숫자가 의사결정의 근거가 된다면, 반드시 원본 데이터 소스(한국은행, 통계청, 거래소 공시)로 한 번 더 내려가서 확인해야 합니다.

경쟁사 모니터링은 정보의 가치가 시간에 민감합니다. 어제 나온 뉴스를 일주일 뒤에 보면 쓸모가 반으로 줄어듭니다. 그래서 이 작업을 매일 자동으로 돌리는 것이 중요하고, Routines에 올려두면 노트북이 꺼져 있어도 클라우드에서 실행됩니다. 매일 아침 커피를 타러 부엌에 갔다 오는 사이에 오늘의 시장 브리핑이 책상 위 폴더에 놓여 있는 상태가 됩니다.

매일 아침 자동 브리핑 생성

하루를 시작하는 첫 30분이 어떻게 쓰이느냐가 그날 전체의 밀도를 결정합니다. 받은 편지함을 열어 어수선한 메일에 끌려다니기 시작하면, 오전 내내 반응만 하다가 정작 집중해야 할 일에 도달하지 못합니다. 아침 브리핑은 이 시작점을 뒤집습니다. 책상에 앉기 전에, 오늘 중요한 것들이 한 페이지로 요약되어 있는 상태를 만드는 일입니다.

구조는 이렇습니다. 오늘 일정, 답장이 시급한 메일, 지난 회의에서 약속한 후속 조치, 방해받지 말아야 할 집중 시간대. 이 네 가지가 한 눈에 보이는 마크다운 파일 하나만 있으면 됩니다.

터미널에서 브리핑 스킬을 만듭니다.

```
mkdir -p ~/.claude/skills/morning_briefing

cat 〈〈 'EOF' 〉 ~/.claude/skills/morning_briefing/SKILL.md

# Morning Briefing Skill

## Trigger

/morning 또는 매일 오전 7시 자동 실행.

## Workflow

1. Google Calendar에서 오늘과 내일 일정을 가져온다.

2. 각 일정에 대해 (a) 누가 참석하는지 (b) 지난번 이 사람/팀과 만났을 때 녹취록·메모에서 미완료

후속 조치가 있는지 확인한다.

3. Gmail에서 지난 12시간 동안 도착한 메일 중 VIP 발신자(config에 저장)의 것과

'긴급' 키워드가 포함된 것을 추린다.
```

4. 최근 녹취록 파일에서 "제가 ~하겠습니다", "~까지 보내드리겠습니다" 같은

내가 한 약속을 찾아서, 아직 실행되지 않은 것을 나열한다.

5. 위 내용을 합쳐서 '~/briefings/YYYY-MM-DD_morning.md' 파일을 만든다.

6. 맨 마지막에 "오늘 최소 90분 딥 워크 블록 1개를 확보하세요" 문구를 넣는다.

Output format

- 일정은 시간순으로, 참석자와 사전 준비 메모 함께 표기

- 메일은 긴급도 3단계로 분류, 각 메일에 왜 긴급한지 한 줄 해설

- 후속 조치는 원래 약속 날짜와 함께 표기

- 모든 내용은 한 페이지 안에 들어가도록 압축

EOF

Claude Code Desktop의 Scheduled Tasks에서 새 작업을 만들고, 프롬프트에 "/morning 스킬 실행해줘"라고 적은 뒤 매일 오전 7시로 설정합니다. 노트북이 켜져 있고 앱이 열려 있으면 매일 아침 자동으로 브리핑 파일이 생성됩니다. Routines를 쓰면 클라우드에서 실행되기 때문에 노트북 상태와 무관하게 돌아갑니다.

claude

"/schedule 매일 오전 7시에 morning_briefing 스킬 실행"

결과 파일을 열면 이런 형태가 됩니다. 맨 위에는 오늘의 일정 세 건. 10시 의뢰인 미팅 옆에는 "지난번 만남에서 약속한 의견서 초안, 아직 전달 안 됨"이라는 메모가 붙어 있습니다. 14시 자문 통화 옆에는 지난 통화 요약 두 줄과, 상대 회사의 최근 보도자료 한 줄이 붙어 있습니다. 그 아래에는 답장이 시급한 메일 세 통.

대표가 보낸 내부 메일, 주요 거래처 대표의 계약 문의, 관할 법원의 공판 일정 알림. 각각 왜 시급한지 한 문장으로 설명이 붙어 있습니다. 그 아래에는 지난 회의에서 약속했던 후속 조치 네 가지. "수요일까지 보내겠다고 한 자료", "금요일까지 회신하기로 한 메일" 같은 것들이 원래 약속일과 함께 나열됩니다.

맨 아래에는 "오늘 최소 90분 딥 워크 블록 1개 확보" 문구.

변호사 업무에서 이 방식은 특히 가치가 큽니다. 하루에 사건 여러 개를 동시에 진행하면, 각 사건의 현재 상황을 다시 파악하는 데 시간이 많이 걸립니다. 브리핑에 "사건별 오늘의 할 일" 섹션을 추가하면, 각 사건의 마감일과 다음 단계가 자동으로 갱신되어 올라옵니다. "X사건: 내일 서면 마감", "Y사건: 증인 신문 준비 필요" 같은 식으로 사건 이름만 보고도 우선순위를 정할 수 있게 됩니다.

브리핑이 제 역할을 하려면 한 가지가 중요합니다. 너무 길면 안 됩니다. 한 페이지, 3분 안에 읽을 수 있어야 브리핑입니다. 20가지 항목이 나열된 브리핑은 브리핑이 아니라 그냥 또 다른 할 일 목록이 됩니다. 프롬프트에 "한 페이지, 읽는 데 3분 이내로 압축"이라는 제약을 꼭 넣어두는 편이 좋습니다. 자동화는 필요한 것을 빠르게 내놓는 데 쓰지, 더 많이 내놓는 데 쓰는 것이 아닙니다.

흩어진 메모를 논리적 초안으로 정리

글 쓰는 사람의 폴더는 어느 시점이든 메모가 어수선하게 쌓여 있습니다. 아이디어가 떠오른 순간 아이폰에 받아적은 한 줄 메모. 회의 중간에 종이에 갈겨쓴 뒤 스캔한 스케치. 강연을 들으며 타이핑한 마크다운 파일. 책을 읽다가 옮겨 적은 인용문. 카카오톡 '나에게' 방에 던져둔 한 단락.

이 파편들 사이에는 분명 연결고리가 있는데, 한 편의 글로 엮으려면 시간과 집중력이 필요합니다. 클로드 코드는 이 파편들을 읽고, 주제별로 묶고, 논리 흐름을 잡아 초안으로 재구성합니다.

시작은 파일을 한 폴더에 모으는 일입니다. 형식은 뒤섞여도 상관없습니다. 텍스트, 마크다운, 워드, PDF, 음성 메모의 전사본까지 모두 한 폴더에 넣습니다.

```
mkdir -p ~/Documents/writing/raw_notes_ai_law_book

# 그동안 쌓인 메모들을 이 폴더에 모은다

cd ~/Documents/writing/raw_notes_ai_law_book

ls

# idea_20260301.txt, lecture_notes.md, book_excerpt.pdf,

# conference_memo.docx, voice_transcript.txt, kakao_clip.md...
```

클로드 코드를 실행하고 지시합니다.

```
claude

"이 폴더의 모든 파일을 읽어줘. 같은 주제끼리 묶고,

중복되는 아이디어는 하나로 합쳐줘. 단, 원문의 미묘한 뉘앙스 차이는 유지해줘. 그 다음 주제별로 소제목을 붙이고, 논리적 흐름에 따라 재배치해줘. 초안 형태의 한글 원고로 조립해서 'AI법_원고초안_v1.md'로 저장해줘. 각 문단 끝에는 출처 파일명을 대괄호로 표시해서, 나중에 원본을 찾아갈 수 있게 해줘.

필자 말투는 경어체('~입니다', '~합니다')로 통일하고,

금지된 표현 목록은 project_style.md를 참고해줘."
```

결과물은 한 편의 완성된 원고가 아닙니다. 조각들이 분류되어 주제별로 모이고, 논리적으로 연결될 만한 순서로 재배치된 상태입니다. 이것을 필자가 다시 읽으며 흐름을 고치고, 자기 문장으로 다듬는 것이 다음 단계입니다.

이 방식이 의미 있는 이유는 기억의 작업을 덜어주기 때문입니다. 반년 전에 써둔 한 줄 메모와 지난 달에 들은 강연 노트가 같은 주제로 묶였을 때, 필자는 "아, 이 둘이 연결될 수 있겠구나"라는 연결을 비로소 인식합니다. 사람은 이런 연결을 우연히 발견하면 기뻐하지만, 파일이 수백 개 쌓이면 우연조차 일어나지 않습니다. AI가 파편을 한 번 훑어 분류해주면, 연결의 가능성 자체가 눈앞에 펼쳐집니다.

변호사의 집필에는 특별한 규칙이 필요합니다. 인용과 출처가 글의 생명이기 때문입니다.

cat 〈〈 'EOF' 〉 ~/Documents/writing/CLAUDE.md

집필 폴더 공통 규칙:

- 원문의 한 글자도 생략·요약·수정하지 않는다. 재배치만 한다.

- 인용문은 큰따옴표로 감싸고, 출처 파일명과 해당 라인을 반드시 함께 기록한다.

- 같은 아이디어의 다른 표현 두 개가 있으면, 둘 다 보존하고 차이점을 주석으로 남긴다.

- 경어체로 통일: '~입니다', '~합니다', '~했습니다'

- 금지 표현: '단순', '특히', '활용', '극대화', '강화', '다양한 관점이 존재하지만', '결론적으로'

- 소제목은 꼭 필요할 때만 붙인다. 글의 2/3 이상이 문단으로 흘러야 한다.

- 번호 목록, 불릿 포인트는 5개 이상 연속 사용 금지.

- 사실 먼저 놓고 해석은 뒤에 붙인다. '필자가 중요성을 선언하는' 문장을 쓰지 않는다.

EOF

이 규칙이 있으면 폴더에 메모를 던져두고 "초안 만들어줘"라고만 해도 필자의 문체가 유지된 결과물이 나옵니다. 다른 필자의 폴더에는 그 필자의 규칙이 적혀 있고, AI는 매번 그 규칙에 맞춰 결과를 조립합니다.

한 가지 경계할 점이 있습니다. AI는 '논리적으로' 재배치하는 데 능하지만, 논리적인 것이 꼭 좋은 글은 아닙니다. 좋은 글은 때로 논리를 깨뜨리고, 장면으로 시작하고, 결론을 유보합니다. AI가 정리해준 초안을 그대로 쓰면 글이 매끈하지만 밋밋해지는 일이 흔합니다.

필자가 해야 할 일은 AI의 정리본을 받은 뒤, "어디서 장면으로 시작할까", "어느 구간을 일부러 건너뛸까", "어떤 문장을 끝에서 두 번째 줄로 옮겨 여운을 만들까"를 고민하는 것입니다. AI는 파편을 줄 세워 주고, 필자는 줄을 흐트러뜨려 리듬을 만듭니다. 이 두 작업의 구분이 명확해질수록 결과물이 좋아집니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

2장 에이전트 작업 흐름이 바꾸는 일터

전체 목차

책의 모든 장 보기

도입

금요일 오후 4시, 마케팅 팀장 이수진 씨는 모니터 앞에서 한숨을 쉽니다. 경쟁사 5곳의 가격 정책이 바뀌었다는 소식이 슬랙에 올라왔고, CEO는 월요일 아침 회의 전까지 경쟁 분석 보고서를 요청했습니다. 예전이라면 각 경쟁사 웹사이트를 하나하나 열어 가격표를 복사하고, 엑셀에 옮기고, 파워포인트에 차트를 그리는 데 주말 절반을 써야 했을 겁니다.

그런데 이수진 씨는 노트북을 열고 클로드 코드에 한 문장을 입력합니다. "우리 회사 브랜드 가이드라인에 맞춰 경쟁사 5곳의 최신 가격 정책을 분석하고, PDF 보고서를 만들어줘." 15분 뒤, 회사 로고와 색상이 반영된 12쪽짜리 보고서가 화면에 나타납니다.

이것이 에이전트 작업 흐름(Agentic Workflow)의 모습입니다. 사람이 각 단계를 수동으로 연결하는 대신, AI 에이전트가 스스로 판단하고, 도구를 선택하고, 오류를 수정하면서 결과물을 완성합니다.

80억 달러에서 500억 달러로: 에이전트형 AI 시장의 폭발적 성장

숫자부터 확인하겠습니다. 에이전트형 AI 시장 규모는 2025년 기준 약 80억 달러(한화 약 11조 원)로 추산됩니다. 업계 전망에 따르면 2030년까지 400억~500억 달러(한화 약 55조~69조 원) 규모로 성장할 것으로 예측됩니다. 작은 도약이 아닙니다. 하나의 산업이 눈앞에서 만들어지고 있는 것입니다.

이 수치가 단지 전망치에 불과하다고 생각할 수 있습니다. 하지만 현장의 움직임은 이미 시작되었습니다. 세계 기업의 약 25%가 에이전트 시험 운영 프로젝트를 가동하고 있으며, 2027년에는 그 비율이 50%에 이를 것으로 전망됩니다. 주요 기업 두 곳 중 한 곳이 에이전트 작업 흐름을 운영하게 된다는 뜻입니다.

이에 따라 큰 예산 투입, 새로운 보안 요구 사항, 그리고 이 시스템을 구축할 줄 아는 인재에 대한 수요가 동시에 폭증하고 있습니다.

기업의 25%가 이미 도입한 에이전트 시험 운영

에이전트 시험 운영(Agentic Pilot)이란 기업이 본격 도입에 앞서 소규모로 실험하는 에이전트 작업 흐름 프로젝트를 말합니다. 전사적으로 배포하기 전에, 특정 부서나 업무에 먼저 적용해 보는 것입니다. 마케팅 팀에서 경쟁사 모니터링을 자동화해 보거나, 영업 팀에서 잠재 고객 목록 생성을 에이전트에게 맡겨 보는 식입니다.

이 파일럿 단계에서 기업들이 공통으로 발견하는 사실이 있습니다. 에이전트 작업 흐름은 구축 과정에서 스스로 문제를 해결합니다. 기존 자동화 도구로 워크플로우를 만들 때는, 예외 상황 하나하나를 개발자가 미리 예측하고 처리 로직을 넣어야 했습니다.

에이전트 작업 흐름에서는 에이전트가 실행 도중 오류를 만나면 원인을 분석하고, 접근 방식을 바꾸고, 도구를 수정한 뒤 계속 진행합니다. 이 자가 치유(스스로 고치기) 능력이 구축 단계의 생산성을 극적으로 끌어올립니다.

전통적 자동화의 천장과 에이전틱 전환의 동력

n8n이나 Zapier 같은 자동화 도구로 워크플로우를 만들어 본 경험이 있다면, 그 과정을 잘 알고 있을 겁니다. 각 단계를 설계하고, 노드를 연결하고, 예외 상황을 직접 처리합니다. 잘 돌아갑니다. 예상하지 못한 입력이 들어오기 전까지는요. 전통적 워크플로우는 예상 밖의 상황을 만나면 멈춥니다.

누군가가 로그를 열어 어디에서 어떤 오류가 발생했는지 확인하고, 수동으로 고쳐야 합니다. 그것이 유지보수이고, 시간이고, 비용입니다.

기업들이 에이전틱 전환을 서두르는 배경이 여기에 있습니다. 전통적 자동화가 도달할 수 있는 천장에 부딪힌 것입니다. 단순 반복 업무는 자동화로 해결했지만, 판단이 필요한 업무, 맥락에 따라 경로가 달라지는 업무, 예외가 빈번한 업무 앞에서는 기존 도구가 한계를 드러냅니다. 에이전트 작업 흐름은 그 한계 너머에서 작동합니다.

대형 언어 모델(LLM)이 추론하고, 판단하고, 여러 단계의 작업을 일관되게 수행할 수 있는 수준에 도달했기 때문입니다.

여기에 MCP(Model Context Protocol) 같은 표준화된 도구 연결 프로토콜, Vercel이나 Modal 같은 배포 인프라, 그리고 클로드 코드 같은 개발자가 아닌 사람용 에이전트 도구가 등장하면서, 에이전트 작업 흐름을 구축하는 데 필요한 기술 장벽이 빠르게 낮아지고 있습니다.

열차 궤도 비유: 수작업 건설에서 건설 크루 위임으로

에이전트 작업 흐름의 가치를 이해하는 데 도움이 되는 비유가 있습니다.

전통적 자동화는 열차 궤도를 손으로 깔아가는 것과 같습니다. 레일 하나, 분기점 하나, 연결부 하나를 전부 직접 설치합니다. 에이전트 작업 흐름은 건설 크루에게 "여기서 저기까지 궤도를 깔아줘"라고 말하는 것에 가깝습니다. 건설 크루가 작업 도중 암반을 만나면 알아서 우회로를 찾습니다. 결과적으로 더 나은 궤도가, 더 빨리, 더 적은 실수로 완성됩니다.

에이전트가 구축 과정에서 사람이 미처 떠올리지 못한 예외 상황을 처리해 주기 때문입니다.

배포 전에는 이 궤도를 철저하게 시험합니다. 무게가 다른 열차, 길이가 다른 열차, 바퀴 규격이 다른 열차를 번갈아 달려 봅니다. 다양한 종류의 열차가 문제 없이 달릴 수 있다는 확신이 들어야 비로소 궤도를 운행에 투입합니다. 이것이 에이전트 작업 흐름의 구축-검증-배포 사이클입니다.

자가 치유와 배포의 경계: 에이전트를 배포하는 것이 아니라 코드를 배포한다

에이전트 작업 흐름에 대한 온라인 담론에는 과장이 섞여 있습니다. "한번 만들면 영원히 스스로 고쳐가며 돌아간다"는 식의 이야기입니다. 이 말은 절반만 맞습니다. 어떤 맥락에서 실행되느냐에 따라 달라지기 때문입니다.

클로드 코드 안에서 직접 워크플로우를 실행하면, 에이전트가 곁에 있습니다. 작업 도중 오류가 발생하면 에이전트가 즉시 감지하고, 접근 방식을 수정하고, 도구 코드를 고친 뒤 이어서 진행합니다. 자가 치유가 실시간으로 일어납니다.

그러나 워크플로우를 예약 실행으로 전환하거나, 웹훅(웹훅) 트리거에 연결하여 자동으로 돌아가게 만들면 상황이 달라집니다. 이때 배포되는 것은 워크플로우(W)와 도구(T)의 코드이지, 에이전트(A) 자체가 아닙니다. WAT 프레임워크에서 W와 T만 클라우드로 올라가고, A는 남겨집니다. 에이전트 없이 실행되는 코드는 전통적 자동화와 비슷하게 같은 입력이면 같은 결과가 나오는 방식으로 동작합니다.

이것이 약점처럼 들릴 수 있지만, 오히려 강점입니다. 예측 가능하고, 반복 가능하며, 결과가 일정한 자동화야말로 운영 환경에서 가장 신뢰할 수 있는 형태이기 때문입니다. 에이전트 작업 흐름의 진짜 이점은 "배포 후 영원히 자가 치유되는 것"이 아니라, "구축 과정에서 훨씬 빠르고 정교하게 만들 수 있다는 것"에 있습니다.

건설 크루가 궤도를 깔아준 뒤, 그 궤도 자체는 견고하고 예측 가능한 레일 위에서 열차를 달리게 합니다.

이수진 씨의 경쟁 분석 보고서는 월요일 아침 회의 테이블 위에 올라갔습니다. CEO는 보고서의 차트를 넘기며 "이걸 혼자 다 했어요?"라고 물었습니다. 이수진 씨는 고개를 끄덕이면서도, 머릿속으로는 다른 질문을 굴리고 있었습니다. 보고서를 만든 이 에이전트, 우리 회사의 다른 부서에서도 쓸 수 있지 않을까. 그리고 이 기술을 가르쳐달라고 하는 사람이 나타나면, 거기에도 사업 기회가 있지 않을까.

〔그림 2-01: 에이전트형 AI 시장 규모 성장 추이. 2025년 약 80억 달러에서 2030년 400억~500억 달러로 성장하는 곡선을 보여주는 그래프.〕

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

3장 클로드 코드, 누구나 쓸 수 있다

전체 목차

책의 모든 장 보기

도입

매일 아침 카페에서 노트북을 펼치는 프리랜서 컨설턴트가 있습니다. 그는 n8n으로 클라이언트의 이메일 자동 분류 시스템을 만들어 왔고, Zapier로 CRM 데이터를 구글 시트에 동기화하는 워크플로우도 수십 개 운영합니다. 자동화의 맛을 충분히 본 사람입니다. 그런데 요즘 클라이언트들이 묻기 시작합니다.

"에이전트형 AI로 할 수 있는 건 없나요?" 그는 에이전틱이라는 단어를 들을 때마다 슬쩍 불안해집니다. 코딩을 배워야 하나, 파이썬부터 시작해야 하나. 그런데 막상 클로드 코드를 열어 보니, 그가 입력하는 것은 코드가 아니라 한국어 문장이었습니다.

n8n·Zapier 경험자가 가진 숨겨진 강점

에이전트 작업 흐름이 주목받으면서, 기존 자동화 도구 경험이 쓸모없어지는 것 아닌가 하는 걱정이 생길 수 있습니다. 정반대입니다. n8n이나 Zapier로 워크플로우를 만들어 본 사람은 에이전트 시스템을 다룰 때 결정적인 이점을 가집니다.

웹훅(웹훅)이 무엇인지 압니다. API가 어떻게 작동하는지 감각적으로 이해합니다. 데이터가 한 노드에서 다른 노드로 흘러가는 구조를 머릿속에 그릴 수 있습니다. 이 기초가 없는 사람이 에이전틱 도구에 뛰어들면, 에이전트가 만들어 놓은 결과물이 좋은 것인지 나쁜 것인지 판별할 수 없습니다. 에이전트가 잘못된 판단을 내렸을 때, 그것이 잘못되었다는 사실 자체를 알아차리지 못합니다.

자동화의 기본 원리를 체득한 사람은 에이전트에게 지시할 때도 정확합니다. "이 웹사이트에서 리드를 긁어줘"라는 막연한 요청 대신, "이 URL에서 영업직 채용 공고를 웹 정보 수집하되, 직종·지역·경력 필드를 포함해서 엑셀로 정리하고, 200건까지만 가져와"라고 말합니다. 에이전트는 이런 구체적 지시를 받았을 때 훨씬 나은 결과를 내놓습니다.

기초 없이 뛰어들면 생기는 함정: 웹훅을 모르면 실수를 못 본다

에이전트 작업 흐름 시장이 뜨거워지면서, 자동화 경험 없이 곧바로 클로드 코드에 뛰어드는 사람이 늘고 있습니다. "코딩 필요 없이 앱을 만들 수 있다"는 메시지에 끌려서입니다. 클로드 코드가 개발자가 아닌 사람에게 강력한 도구인 것은 분명합니다. 하지만 기초 개념을 완전히 건너뛰면 위험합니다.

에이전트에게 "리드 스크래퍼를 만들어줘"라고 지시했다고 가정해 보겠습니다. 에이전트는 열심히 코드를 작성하고, 도구를 만들고, 워크플로우를 구성합니다. 결과물이 나옵니다. 그런데 그 결과물이 정말로 괜찮은 것인지, 보안에 취약한 부분은 없는지, 데이터를 올바르게 처리하고 있는지 어떻게 판단할 수 있을까요.

웹훅이 무엇인지 모르는 사람은 에이전트가 웹훅 설정을 잘못해도 알아차리지 못합니다. API 호출의 기본 구조를 모르는 사람은 에이전트가 불필요한 API 호출을 반복하며 비용을 낭비해도 깨닫지 못합

니다.

이 책은 코딩 경험이 없는 독자를 위해 쓰였습니다. 프로그래밍 언어를 배우라는 이야기가 아닙니다. 다만, 자동화의 기본 개념 — 데이터가 어디서 어디로 흐르는지, 트리거가 무엇인지, API 키가 왜 필요한지 — 을 이해하는 것이 에이전트 작업 흐름을 제대로 다루는 데 필수라는 점을 강조합니다. 이 책의 앞부분에서 그 기초를 함께 다질 것입니다.

코딩을 모른 채 코딩 도구를 다루는 시대의 도래

전통적으로 소프트웨어를 만들려면 프로그래밍 언어를 배워야 했습니다. 변수를 선언하고, 반복문을 쓰고, 함수를 정의하는 법을 익혀야 했습니다. 그 과정이 진입 장벽이었습니다. n8n과 Zapier가 그 장벽을 한 단계 낮췄습니다. 노드를 드래그 앤 드롭으로 연결하면 워크플로우가 만들어졌으니까요. 수동 코딩에 비해 구축 시간이 획기적으로 줄었습니다.

클로드 코드는 그 장벽을 한 단계 더 낮춥니다. 드래그 앤 드롭조차 필요 없습니다. 자연어로 원하는 것을 설명하면, 에이전트가 코드를 작성하고, 파일을 생성하고, 오류를 수정합니다. 사용자는 파이썬 코드를 읽을 줄 몰라도 됩니다. 에이전트가 만든 도구(Tool) 파일은

.py

확장자를 달고 있지만, 사용자가 그 안의 코드를 직접 편집할 일은 거의 없습니다.

사용자의 역할은 관리자, 감독자, 아키텍트에 가깝습니다. 무엇을 만들지 결정하고, 결과물을 검토하고, 방향을 수정하는 것입니다.

이 변화의 속도를 실감하려면 타임라인을 보면 됩니다. 수동 코딩으로 자동화 시스템을 만드는 데 걸리던 시간을 기준선으로 잡으면, 코드 없이 쓰는 도구의 등장으로 구축 시간이 크게 단축되었고, 에이전틱 도구의 등장으로 다시 한번 대폭 줄어들었습니다. 이 추세는 가속하고 있습니다.

클로드 코드가 제공하는 말로 하는 제어의 힘

클로드 코드에서 워크플로우를 만드는 과정은, 지시를 잘하는 사람이 결과를 잘 받는 과정입니다. "경쟁사를 분석해줘"라고 말하면 에이전트는 일단 무언가를 만들어 냅니다. 하지만 "우리 회사는 AI 리드 생성 플랫폼이고, 주 타깃은 마케팅 에이전시이며, 월 구독료는 200에서 500달러 범위입니다. 경쟁사의 가격 정책, 핵심 기능, 타깃 시장을 비교 분석해서, 우리 로고와 브랜드 색상이 적용된 PDF 보고서로 만들어줘"라고 말하면 결과물의 수준이 완전히 달라집니다.

에이전트와의 소통에서 핵심은 두 가지입니다. 목표를 구체적으로 서술하는 것, 그리고 완료 기준을 명확히 하는 것입니다. "잠재 고객 목록을 만들어줘"보다 "미국과 유럽의 영업직 채용 공고 500건을 웹 정보 수집해서, 직종·지역·연봉 컬럼이 포함된 엑셀 파일로 저장해줘"가 낫습니다. 에이전트는 후자의 지시에서 언제 작업을 멈춰야 하는지, 어떤 데이터를 수집해야 하는지 분명하게 파악합니다.

n8n에서 노드의 파라미터를 하나하나 설정하던 작업이, 클로드 코드에서는 자연어 한 단락으로 대체됩니다. API 문서를 읽고, 올바른 엔드포인트를 찾고, JSON을 구조화하는 일을 에이전트가 대신합니다. 사용자에게 남는 것은 "무엇을 원하는가"를 정확하게 표현하는 능력이며, 그것이야말로 코딩보다

더 보편적이고 훈련 가능한 기술입니다.

프리랜서 컨설턴트는 카페에서 첫 번째 에이전트 작업 흐름을 완성합니다. n8n에서 3시간 걸리던 워크플로우를 클로드 코드에서는 20분 만에 만들었습니다. 에이전트가 생성한 파이썬 코드를 한 줄도 읽지 않았지만, 결과물은 정확했습니다. 그는 노트북을 닫으며 생각합니다. 이 도구의 진짜 힘은 코드를 작성하는 데 있지 않습니다.

내가 무엇을 원하는지 명확하게 말할 수 있느냐에 있습니다. 그리고 그건, 자동화를 몇 년간 해온 자신에게 이미 익숙한 근육이었습니다.

〔그림 3-01: 자동화 구축 시간의 변화 타임라인. 수동 코딩 → 코드 없이 쓰는 도구(n8n, Zapier) → 에이전틱 도구(클로드 코드)로 이행하면서 구축 시간이 단계적으로 단축되는 과정을 보여주는 도표.〕

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

4장 첫 에이전트 작업 흐름: 경쟁사 보고서 자동화

전체 목차

책의 모든 장 보기

도입

모니터 왼쪽에는 빈 폴더가 하나 열려 있습니다. 파일이 하나도 없는 깨끗한 프로젝트입니다. 모니터 오른쪽에는 작은 게 아이콘이 달린 채팅창이 보입니다. 이것이 클로드 코드의 에이전트 인터페이스입니다. 이 빈 폴더와 채팅창만으로, 지금부터 경쟁사 분석 보고서를 자동으로 생성하는 워크플로우를 구축할 것입니다. 코드를 한 줄도 직접 작성하지 않고요.

VS Code 설치와 클로드 코드 확장 프로그램 연결

클로드 코드를 사용하려면 먼저 비주얼 스튜디오 코드(Visual Studio Code, 이하 VS Code)를 설치해야 합니다. VS Code는 통합 개발 환경(IDE)이라는 도구인데, 개발자들이 코드를 작성하고 관리하는 작업 공간이라고 생각하면 됩니다. 무료로 내려받을 수 있으며, 윈도우와 맥 모두에서 작동합니다.

설치가 끝나면 VS Code를 열어 봅니다. 화면 왼쪽 세로 메뉴바에서 사각형 네 개가 모인 아이콘을 찾습니다. 확장(Extensions) 메뉴입니다. 이 아이콘을 클릭하면 검색창이 나타나고, 여기에 "Claude Code"를 입력합니다. 앤트로픽(Anthropic)이 배포한 공식 확장 프로그램이 상단에 표시됩니다.

이 확장을 설치하면 VS Code 안에서 클로드 코드 에이전트와 대화할 수 있는 창구가 열립니다.

설치 버튼을 누르면 곧바로 앤트로픽 계정 로그인을 요청합니다. 클로드 코드를 사용하려면 유료 구독이 필요합니다. 프로(Pro) 플랜은 월 17달러(한화 약 2만 3천 원)부터 시작하며, 클로드 코드와 최신 오퍼스(Opus) 모델을 포함합니다. 사용량이 늘어나면 맥스(Max) 플랜으로 업그레이드할 수 있습니다.

맥스 플랜은 사용 한도가 높아서, 여러 워크플로우를 동시에 구축하거나 긴 대화를 이어갈 때 유리합니다.

〔그림 4-01: VS Code 왼쪽 확장 메뉴에서 Claude Code를 검색한 화면. Install 버튼이 강조 표시되어 있다.〕

프로젝트 폴더 열기와 에이전트 인터페이스 이해

클로드 코드 확장이 설치되면, 화면 오른쪽 상단에 작은 앤트로픽 로고 버튼이 나타납니다. 하지만 이 버튼을 누르기 전에 먼저 프로젝트 폴더를 열어야 합니다. 왼쪽 메뉴에서 탐색기(Explorer) 아이콘을 클릭하면 "아직 폴더를 열지 않았습니다"라는 메시지가 보입니다.

컴퓨터의 바탕화면이나 문서 폴더에 빈 폴더를 하나 만듭니다. "첫번째_워크플로우"라는 이름이면 충분합니다. VS Code에서 이 폴더를 열면, 왼쪽 탐색기에 폴더 이름이 표시되고 그 아래는 텅 비어 있습니다. 이 빈 공간이 앞으로 에이전트가 만들어내는 파일들로 채워질 것입니다.

이제 오른쪽 상단의 앤트로픽 로고 버튼을 클릭합니다. 환영 화면이 뜨면 닫아 줍니다. 화면이 두 영역으로 나뉩니다. 왼쪽은 파일 탐색기, 오른쪽은 에이전트와의 대화창입니다. 이 대화창은 카카오톡이나 챗GPT 인터페이스와 비슷합니다. 메시지를 입력하고 전송하면, 에이전트가 생각하고, 행동하고, 결과를 보여줍니다. 왼쪽에서는 에이전트가 실시간으로 생성하는 파일과 폴더를 확인할 수 있습니다.

〔그림 4-02: VS Code 화면 분할. 왼쪽 파일 탐색기에 빈 프로젝트 폴더, 오른쪽에 클로드 코드 채팅 인터페이스가 표시된 화면.〕

계획 모드로 에이전트와 대화하기: 경쟁사 분석 워크플로우 기획

대화창 하단에 모드 선택 옵션이 있습니다. 네 가지 모드 중 "계획 모드(Plan Mode)"를 선택합니다. 계획 모드에서 에이전트는 코드를 작성하거나 파일을 만들지 않습니다. 대신 계획을 세우고, 질문을 던지고, 접근 방식을 제안합니다. 집을 짓기 전에 설계도를 먼저 그리는 것과 같습니다.

에이전트에게 이렇게 말합니다.

"경쟁사 분석 워크플로우를 만들고 싶어. 최종 결과물은 우리 회사 로고와 브랜드 색상이 적용된 PDF 보고서야. 우리 회사 정보를 입력하면, 에이전트가 경쟁사를 찾아서 분석하고, 기회 영역과 개선점을 정리해 줬으면 해. 계획을 세워줘. 질문이 있으면 해."

이 메시지를 보내면 에이전트는 생각하기 시작합니다. 대화창에 에이전트의 사고 과정이 실시간으로 표시됩니다. "기존 워크플로우와 도구를 탐색합니다", "브랜딩과 PDF 생성 방법을 조사합니다" 같은 메시지가 흐릅니다. 잠시 뒤, 에이전트가 질문을 던집니다.

에이전트의 질문은 구체적입니다. "경쟁사를 어떻게 식별할까요? 직접 목록을 주시겠습니까, 아니면 제가 자동으로 발견하게 할까요?" "어떤 측면을 분석할까요? 가격 정책, 제품 기능, 마케팅 메시지 중 선택하거나 전부 분석할 수 있습니다." "보고서를 얼마나 자주 만들까요? 월간, 주간, 필요할 때?" 이런 질문에 하나씩 답하면, 에이전트는 답변을 반영하여 계획을 다듬습니다.

계획이 완성되면 에이전트가 전체 구현 계획서를 보여줍니다. 어떤 기술을 사용할 것인지(클로드 소네트 모델, 파이어크롤 API, 리포트랩 PDF 생성 라이브러리), 어떤 폴더 구조로 정리할 것인지, 어떤 도구 파일을 만들 것인지가 모두 정리되어 있습니다. 비용 추정도 포함됩니다. 첫 실행에 약 1.5달러, 이후 반복 실행은 캐싱된 데이터 덕분에 비용이 절감됩니다.

〔그림 4-03: 에이전트가 제시한 경쟁사 분석 워크플로우 구현 계획서. 기술 스택, 파일 구조, 비용 추정이 포함된 화면.〕

브랜드 자산 투입: 로고와 색상 가이드라인 반영

계획을 승인하기 전에 한 가지를 조정합니다. 에이전트가 임의의 색상과 타이포그래피를 적용하는 대신, 실제 회사 브랜드를 반영하도록 하는 것입니다.

컴퓨터에 저장된 회사 로고 파일(PNG 형식)과 브랜드 가이드라인 이미지를 왼쪽 파일 탐색기 영역으로 드래그합니다. 파일이 프로젝트 폴더에 추가됩니다. 그런 다음 에이전트에게 말합니다.

"방금 로고 파일과 브랜드 가이드라인을 넣었어. 이 파일들을 확인하고, 로고와 색상과 타이포그래피를 PDF 보고서에 반영해줘."

에이전트는 투입된 이미지를 분석합니다. 로고를 인식하고, 브랜드 가이드라인에서 주 색상, 보조 색상, 서체 정보를 추출합니다. "로고를 확인했습니다. 브랜드 색상을 추출하여 PDF에 적용하겠습니다"라는 응답과 함께, 계획서가 업데이트됩니다. 이제 계획을 승인합니다.

워크플로우 실행과 PDF 보고서 생성

계획 승인 후, 에이전트는 투두 리스트(To-Do List)를 생성하고 작업을 시작합니다. 왼쪽 파일 탐색기에 폴더와 파일이 하나씩 나타나기 시작합니다.

brand_assets

폴더가 생기고,

workflows

폴더 안에 경쟁사 분석 워크플로우 파일이 만들어지고,

tools

폴더 안에 다섯 개의 파이썬 도구 파일이 생성됩니다.

비즈니스 정보 수집, 경쟁사 발견, 경쟁사 조사, 경쟁사 분석, PDF 보고서 생성 — 각각 하나의 독립된 도구입니다.

에이전트가 회사 정보를 물어옵니다. "주요 타깃 시장은 무엇입니까?" "가격대는 어떻게 되나요?" "핵심 차별화 요소가 있습니까?" 이 질문에 답하면, 에이전트는

business_profile.json

이라는 파일을 만들어 회사 정보를 저장합니다. 다음에 이 워크플로우를 실행할 때는 같은 질문을 반복하지 않습니다. 에이전트가 이미 알고 있기 때문입니다.

이어서 에이전트는 자동으로 경쟁사를 탐색합니다. 웹을 검색하고, 경쟁사 목록을 만들고, 각 경쟁사의 제품·가격·강점을 조사합니다. 조사 결과가

competitors

폴더에 경쟁사별 개별 파일로 저장됩니다. 분석이 끝나면 최종 PDF 보고서가 생성됩니다.

〔그림 4-04: 워크플로우 실행 후 왼쪽 파일 탐색기에 생성된 폴더 구조. brand_assets, workflows, tools, competitors 폴더와 그 안의 파일들이 보인다.〕

오류 수정과 반복 개선: 로고 미표시 문제 해결 사례

첫 번째 결과물이 완벽할 가능성은 낮습니다. 이것은 실패가 아니라 구축 과정의 자연스러운 일부입니다.

PDF 보고서를 열어 봅니다. 브랜드 색상과 타이포그래피가 적용되어 있습니다. 경쟁사 프로필, 위협 수준 분류, 전략적 권고사항이 정리되어 있습니다. 그런데 로고가 보이지 않습니다. 가격 분석 차트도 빠져 있습니다.

에이전트에게 말합니다.

"보고서 괜찮은데, 로고가 안 보여. 아마 흰색 배경 위에 흰색 로고라서 그런 것 같아. 차트도 없어. 확인하고 고쳐줘."

에이전트는 문제를 진단합니다. "PDF 생성기에 몇 가지 문제가 있습니다"라고 응답하고, 문제 목록을 나열합니다. 그런 다음 도구 파일의 코드를 수정합니다. 로고 배경에 컬러 패드를 추가하고, 차트 생성 로직을 보완합니다. 수정된 도구로 보고서를 다시 생성합니다. 이때 경쟁사 데이터를 다시 수집하지는 않습니다. 이전 실행에서 캐싱한 데이터를 재활용합니다.

두 번째 보고서를 열면, 로고가 선명하게 보이고 가격 분석 차트가 포함되어 있습니다. 이 과정이 자가 치유입니다. 사람이 오류를 알려주면, 에이전트가 원인을 파악하고, 코드를 수정하고, 워크플로우를 갱신하여 같은 오류가 반복되지 않도록 만듭니다.

여기서부터는 미세 조정의 영역입니다. "경쟁사 프로필에 더 자세한 정보를 넣어줘", "권고사항을 세 개에서 다섯 개로 늘려줘" 같은 요청을 자연어로 전달하면, 에이전트가 반영합니다. 워크플로우를 여러 차례 실행하고 수정 의견을 줄수록, 결과물의 품질이 올라갑니다.

비용 분석: 첫 실행 1.43달러의 의미

이 워크플로우의 첫 실행 비용은 약 1.43달러(한화 약 2천 원)였습니다. 경쟁사 8곳을 조사하고, 각 경쟁사의 제품·가격·마케팅 전략을 분석하고, 브랜드가 적용된 PDF 보고서를 생성하는 전체 과정에 들어간 API 비용입니다.

이 작업을 사람이 수동으로 수행한다면 어떨까요. 경쟁사 웹사이트를 하나씩 방문하고, 정보를 정리하고, 엑셀이나 파워포인트로 보고서를 만드는 데 최소 반나절, 길면 하루가 소요됩니다. 시간당 인건비를 고려하면, 2천 원은 파격적인 투자 수익률입니다.

반복 실행의 비용은 더 낮습니다. 경쟁사 기본 정보와 회사 프로필이 이미 저장되어 있기 때문에, 에이전트는 변동 사항만 확인합니다. 데이터 캐싱 덕분에 API 호출 횟수가 줄어들고, 비용도 자연스럽게 절감됩니다.

이 비용 구조는 에이전트 작업 흐름의 경제적 매력을 압축해서 보여줍니다. 처음 한 번의 구축에 시간과 비용을 투입하면, 이후에는 낮은 비용으로 반복 실행이 가능합니다. 그리고 이 "처음 한 번의 구축"조차, 에이전트가 대부분의 작업을 수행하기 때문에 소요 시간이 짧습니다.

PDF 보고서가 화면에 열려 있습니다. 회사 로고가 좌측 상단에 선명하게 찍혀 있고, 경쟁사별 위협 수준이 색상으로 구분되어 있으며, 가격 분석 차트가 데이터를 시각적으로 보여줍니다. 이 보고서를 만드는 데 사용한 모든 도구와 워크플로우는 프로젝트 폴더에 남아 있습니다.

다음 달에 "이번 달 경쟁 분석 돌려줘"라고 한 마디만 하면, 에이전트는 워크플로우를 읽고, 도구를 실행하고, 최신 데이터가 반영된 새 보고서를 내놓을 것입니다. 그런데 이 워크플로우를 돌리기 위해 에이전트에게 준 지시서, 그러니까 에이전트의 업무 매뉴얼이라 할 수 있는 그 문서를 어떻게 작성해야 하는지 살펴보겠습니다.

〔그림 4-05: 완성된 경쟁사 분석 PDF 보고서. 회사 로고, 브랜드 색상, 경쟁사 프로필, 가격 분석 차트가 포함된 보고서의 주요 페이지.〕

〔도표 03-01: 에이전트 작업 흐름의 구축-실행-개선 사이클. 계획 모드(계획) → 구축(구축) → 검증(검증) → 수정 의견(수정) → 반복이 순환하는 흐름도.〕

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

5장 개발 환경을 겁내지 않는 법

전체 목차

책의 모든 장 보기

도입

처음 비행기 조종석 사진을 보면 압도됩니다. 수백 개의 버튼, 다이얼, 스위치, 화면이 빼곡합니다. 하지만 조종사에게 물어보면 이렇게 말합니다. "실제로 이착륙할 때 보는 계기는 여섯 개예요." VS Code도 마찬가지입니다. 화면 곳곳에 버튼과 패널이 있지만, 클로드 코드를 사용하는 데 필요한 것은 셋뿐입니다. 파일 탐색기, 클로드 코드 채팅창, 그리고 하단의 모드 선택기.

VS Code 설치부터 클로드 코드 확장 설치까지 단계별 안내

3장에서 설치 과정을 간략하게 다뤘습니다. 이 장에서는 각 단계에서 마주칠 수 있는 상황을 좀 더 세밀하게 살펴봅니다.

웹 브라우저를 열고 "Visual Studio Code"를 검색합니다. 공식 사이트에서 자신의 운영체제(윈도우, 맥, 리눅스)에 맞는 설치 파일을 내려받습니다. 다운로드가 끝나면 설치 프로그램을 실행합니다. 별다른 옵션을 건드릴 필요 없이 기본 설정으로 진행하면 됩니다.

VS Code를 처음 열면 환영 탭이 나타납니다. 새 파일 만들기, 폴더 열기, 연습 과정 안내 같은 링크가 보입니다. 이 화면은 닫아도 됩니다. 화면 왼쪽 세로 메뉴바에서 네모 네 개가 모인 아이콘(확장 메뉴)을 클릭합니다. 검색창에 "Claude Code"를 입력합니다. 앤트로픽 공식 확장이 검색 결과 상단에 나타나면, "Install" 버튼을 누릅니다.

설치가 완료되면 화면 오른쪽 상단에 앤트로픽 로고 모양의 작은 버튼이 생깁니다. 이 버튼이 클로드 코드 채팅창을 여는 입구입니다. 클릭하면 로그인을 요청합니다.

〔그림 5-01: VS Code 왼쪽 확장 메뉴에서 Claude Code를 검색한 화면. Install 버튼이 강조 표시되어 있다.〕

유료 구독 선택: Pro와 Max의 차이

클로드 코드는 앤트로픽의 유료 구독에 포함된 기능입니다. 무료 계정으로는 사용할 수 없습니다.

프로(Pro) 플랜은 월 17달러(한화 약 2만 3천 원)입니다. 클로드 코드와 오퍼스(Opus) 모델이 포함됩니다. 처음 시작하기에 충분합니다. 다만 일일 사용량에 제한이 있어서, 여러 워크플로우를 연이어 구축하거나 긴 대화를 이어가면 한도에 도달할 수 있습니다.

맥스(Max) 플랜은 사용 한도가 높습니다. 여러 세션을 동시에 열어 두거나, 큰 규모의 프로젝트를 지속적으로 작업하는 사용자에게 적합합니다. 한도에 신경 쓰지 않고 작업에 집중할 수 있다는 것이 가장 큰 이점입니다.

처음에는 프로 플랜으로 시작하고, 사용량이 늘어나면 맥스로 전환하는 것을 권장합니다. API 키를 직접 사용하는 방법도 있지만, 사용량에 따라 비용이 변동하므로 예측이 어렵습니다. 고정 비용의 구독 플랜이 비용 관리 측면에서 더 안정적입니다.

프로젝트 폴더 만들기와 탐색기 패널 이해

클로드 코드는 항상 프로젝트 폴더 안에서 동작합니다. 폴더를 열지 않으면 에이전트에게 작업 공간이 없는 것과 같습니다.

왼쪽 세로 메뉴바 최상단의 서류 아이콘(탐색기)을 클릭합니다. "아직 폴더를 열지 않았습니다(You have not yet opened a folder)"라는 문구가 보입니다. "Open Folder" 버튼을 누르고, 컴퓨터에서 빈 폴더를 선택합니다. 폴더가 없다면 바탕화면이나 문서 폴더에 새 폴더를 하나 만들어 둡니다.

폴더를 열면 왼쪽 탐색기 패널에 폴더 이름이 표시됩니다. 아직 파일은 없으니 비어 있습니다. 에이전트가 작업을 시작하면, 이 영역에 파일과 하위 폴더가 실시간으로 생겨납니다. 워크플로우 파일, 도구 파일, 설정 파일, 임시 데이터 — 에이전트가 만드는 모든 것이 여기에 나타납니다. 탐색기 패널에서 파일을 클릭하면 오른쪽에 파일 내용이 열리므로, 에이전트가 무엇을 만들었는지 직접 확인할 수 있습니다.

클로드 코드 인터페이스: 왼쪽은 파일, 오른쪽은 에이전트

화면 구성을 머릿속에 확실히 그려 두면 이후의 모든 작업이 편해집니다.

왼쪽 영역은 파일의 세계입니다. 에이전트가 생성한 워크플로우, 도구, 데이터, 설정 파일이 모두 이곳에 쌓입니다. 오른쪽 영역은 에이전트의 세계입니다. 메시지를 입력하고, 에이전트의 사고 과정을 관찰하고, 질문에 답하고, 실행 결과를 확인하는 곳입니다.

VS Code에는 VS Code 자체의 AI 에이전트 기능(코파일럿(Copilot) 등)이 내장되어 있을 수 있습니다. 이것은 클로드 코드와 별개입니다. VS Code의 자체 채팅 패널이 열려 있다면 닫아 두는 것이 혼동을 줄입니다. 오른쪽 상단의 앤트로픽 로고 버튼을 눌러 열리는 패널만 사용하면 됩니다.

여러 개의 탭을 오른쪽 영역에 열어 둘 수도 있습니다. 에이전트 채팅 탭 하나, 파일 내용 탭 하나를 동시에 열어서 에이전트의 대화를 보면서 생성된 파일 내용을 확인하는 것도 가능합니다. 필요에 따라 에이전트 채팅 세션을 여러 개 동시에 실행할 수도 있습니다. 각 세션은 독립적인 대화 맥락을 가집니다.

〔그림 5-02: VS Code 화면 레이아웃 안내. 왼쪽 파일 탐색기, 오른쪽 클로드 코드 채팅 패널, 하단 모드 선택기가 표시된 전체 화면.〕

권한 모드: 계획, 확인, 자동 편집, 바이패스

클로드 코드 채팅창 하단에 모드 선택 메뉴가 있습니다. 에이전트에게 부여하는 자율성의 수준을 조절하는 장치입니다.

계획 모드(Plan Mode)

에서 에이전트는 읽기와 조사만 수행합니다. 파일을 생성하거나 코드를 작성하지 않습니다. 계획을 세우고, 질문을 하고, 접근 방식을 제안합니다. 새로운 워크플로우를 기획할 때 가장 먼저 사용해야 하는 모드입니다.

편집 전 확인(Ask Before Edits)

모드에서는 에이전트가 파일을 생성하거나 수정하기 전에 매번 승인을 요청합니다. 변경 내용을 하나하나 검토하고 싶을 때 적합합니다.

자동 편집(Edit Automatically)

모드에서는 에이전트가 파일 수정은 자동으로 수행하되, 시스템 명령어(터미널 명령어) 실행은 승인을 요청합니다.

바이패스 권한(Bypass Permissions)

모드는 에이전트에게 전권을 위임합니다. 파일 생성, 수정, 삭제, 터미널 명령어 실행 모두를 승인 없이 수행합니다. 이 모드는 VS Code 설정에서 별도로 활성화해야 합니다.

설정(Settings)에서 "Claude Code"를 검색하고, "Allow Dangerously Skip Permissions"를 체크합니다. 이름에 "dangerously"가 포함되어 있는 이유는, 에이전트가 모든 작업을 사용자 확인 없이 진행하기 때문입니다.

권장되는 작업 흐름은 이렇습니다. 계획 모드에서 계획을 세우고, 계획이 만족스러우면 바이패스 권한 모드로 전환하여 실행합니다. 이때 에이전트가 작업하는 과정을 화면에서 지켜보면서, 방향이 어긋나면 즉시 개입합니다. 에이전트를 밤새 혼자 돌려놓고 자리를 비우는 것은 아직 권장되지 않습니다. 에이전트 곁에서 감독하되, 매 단계마다 버튼을 누르는 번거로움을 줄이는 것이 바이패스 권한 모드의 올바른 활용법입니다.

조종석의 계기판이 더 이상 두렵지 않은 순간이 옵니다. 탐색기 패널에서 파일이 생겨나는 것을 보고, 채팅창에서 에이전트의 생각을 읽고, 모드를 전환하여 자율성의 수위를 조절하는 것이 자연스러워지는 순간입니다. 그 다음에 알아야 할 것은, 에이전트가 워크플로우와 도구를 어떤 구조로 정리하는가 하는 문제입니다. 그것을 설명하는 프레임워크의 이름은 세 글자입니다.

전체 목차

책의 모든 장 보기

6장 WAT 틀: 작업 흐름, 에이전트, 도구

전체 목차

책의 모든 장 보기

도입

냉장고를 열고 재료를 꺼내고, 레시피를 보면서 케이크를 굽는 장면을 떠올려 봅니다. 레시피가 있습니다. 요리사가 있습니다. 밀가루, 달걀, 설탕 같은 재료가 있습니다. 이 세 가지를 잘 분리해 두면 누구라도 같은 케이크를 만들 수 있습니다.

레시피를 바꾸면 다른 케이크가 나오고, 재료를 바꾸면 맛이 달라지고, 요리사가 바뀌어도 레시피만 따르면 비슷한 결과물을 낼 수 있습니다. 클로드 코드에서 에이전트 작업 흐름을 구축하는 구조가 정확히 이 원리를 따릅니다.

케이크 비유: 레시피(워크플로우)와 재료(도구)와 셰프(에이전트)

WAT 프레임워크는 세 글자의 약어입니다. W는 워크플로우(Workflow), A는 에이전트(Agent), T는 도구(Tool)입니다.

워크플로우

는 레시피입니다. "이 케이크를 만들려면, 먼저 달걀 두 개를 깨뜨리고, 밀가루 한 컵을 넣고, 180도에서 30분간 굽는다"처럼, 작업의 목표·순서·조건을 자연어로 기술한 문서입니다. 파일 형식은 마크다운(Markdown,

.md

)이며, 프로그래밍 언어가 아닌 일상적인 문장으로 작성됩니다.

도구

는 재료이자 주방 기구입니다. 달걀을 깨뜨리는 행위, 오븐을 예열하는 행위, 반죽을 젓는 행위 — 각각이 하나의 도구입니다. 기술적으로는 파이썬(Python) 스크립트 파일(

.py

)로 작성되며, API 호출, 데이터 변환, 파일 생성 같은 실제 실행 동작을 담당합니다.

에이전트

는 셰프입니다. 클로드 코드 그 자체입니다. 레시피(워크플로우)를 읽고, 필요한 재료(도구)를 골라서, 올바른 순서로 사용합니다. 실행 도중 문제가 생기면 판단을 내립니다. 오븐 온도가 너무 높으면 낮추고, 밀가루가 부족하면 대체 재료를 찾습니다.

핵심은 이 셋이 분리되어 있다는 것입니다. 레시피를 바꾸면 다른 결과물을 만들 수 있고, 도구를 교체하면 같은 레시피로도 다른 서비스에 연결할 수 있으며, 셰프(에이전트)는 어떤 레시피와 도구 조합이든 다룰 수 있습니다.

워크플로우는 마크다운, 도구는 파이썬: 왜 분리하는가

워크플로우 파일 하나를 열어 보면 이런 구조입니다.

경쟁사 분석 워크플로우

목표

경쟁사 데이터를 수집하고 분석하여 브랜드가 적용된 PDF 보고서를 생성한다.

필요 입력

- 회사 정보 (business_profile.json)

- 브랜드 자산 (logo, brand guidelines)

사용할 도구

1. discover_competitors.py

2. research_competitors.py

3. analyze_competitors.py

4. generate_report.py

예상 출력

- 경쟁사별 프로필 파일

- 분석 결과 요약

- 브랜드 적용 PDF 보고서

예외 처리

- 경쟁사 웹사이트 접근 차단 시: 대체 소스 검색

- API 호출 한도 초과 시: 배치 요청으로 전환

일상어로 쓰여 있습니다. 프로그래밍을 모르는 사람도 읽을 수 있습니다. 워크플로우 파일은 에이전트에게 주는 업무 지시서이자, 사람이 검토할 수 있는 문서이기도 합니다.

반면 도구 파일은 파이썬 코드입니다.

discover_competitors.py

를 열면 API 호출 로직, 데이터 파싱 코드, 오류 처리 루틴이 담겨 있습니다. 이 코드를 직접 읽거나 수정할 필요는 거의 없습니다. 에이전트가 생성하고, 에이전트가 수정합니다. 사용자가 신경 쓸 것은 워크플로우 파일의 내용, 즉 "무엇을 하고 싶은가"입니다.

분리의 이유는 역할 분담에 있습니다. 워크플로우는 "무엇을" 지시하고, 도구는 "어떻게" 실행합니다. 이 둘을 섞어 놓으면, 작업 지시를 바꾸고 싶을 때 코드를 건드려야 하고, 코드를 수정하고 싶을 때 지시 문서가 꼬입니다. 분리되어 있기 때문에, 워크플로우만 수정하여 분석 범위를 바꾸거나, 도구만 교체하여 다른 API를 연결하는 것이 가능합니다.

확률적 추론과 결정적 실행의 분리가 만드는 신뢰성

WAT 프레임워크가 워크플로우와 도구를 나누는 데는 기술적으로 더 깊은 이유가 있습니다. AI 에이전트의 추론은 매번 조금씩 달라질 수 있는 방식입니다. 같은 질문에 대해 매번 약간 다른 답을 내놓을 수 있습니다. 반면 코드의 실행은 같은 입력이면 같은 결과가 나오는 방식입니다. 같은 입력을 주면 항상 같은 출력을 냅니다.

에이전트가 모든 단계를 직접 처리하게 하면, 확률적 추론이 누적되면서 정확도가 급격히 떨어집니다. 한 단계의 정확도가 90%라고 가정해 보겠습니다. 두 단계를 거치면 81%(0.9 × 0.9)입니다. 다섯 단계를 거치면 59%(0.9)로 곤두박질칩니다. 열 단계라면 35%에 불과합니다.

WAT 프레임워크는 이 문제를 해결합니다. 에이전트는 "어떤 도구를 어떤 순서로 사용할 것인가"를 판단하는 역할만 맡습니다. 실제 데이터 처리, API 호출, 파일 생성 같은 실행은 결정적 코드(도구)가 담당합니다. 에이전트의 확률적 판단은 소수의 핵심 결정 지점에만 집중되고, 나머지 실행 과정은 코드가 일관되게 수행합니다. 그 결과, 전체 워크플로우의 신뢰성이 올라갑니다.

5단계 연속 작업의 정확도 함정: 90%가 59%로 떨어지는 이유

이 수학을 좀 더 구체적으로 들여다보겠습니다.

AI 모델에게 이메일을 읽고, 핵심을 요약하고, 카테고리를 분류하고, 우선순위를 매기고, 응답 초안을 작성하게 한다고 합시다. 다섯 단계입니다. 각 단계에서 모델이 올바른 판단을 내릴 확률이 90%라면, 다섯 단계를 모두 올바르게 통과할 확률은 약 59%입니다. 열 번 중 네 번은 어딘가에서 틀린다는 뜻입니다.

이 문제의 해결책이 바로 "관심사의 분리(Separation of Concerns)"입니다. 이메일을 읽는 행위, 텍스트에서 키워드를 추출하는 행위, 카테고리 태그를 붙이는 행위를 각각 독립된 도구로 만듭니다. 각 도구는 결정적으로 동작합니다. 에이전트는 "이 이메일에 대해 분류 도구를 실행하고, 그 결과를 바탕으로 우선순위 도구를 실행해"라는 판단만 내립니다.

판단의 복잡도가 낮아지면서, 전체 정확도가 올라갑니다.

이것이 WAT 프레임워크를 사용하는 실무적 이유입니다. 에이전트에게 "알아서 다 해줘"라고 던지는 것보다, 각 단계를 독립된 도구로 쪼개고 에이전트가 그 도구들을 조합하게 하는 것이 훨씬 안정적인 결과를 냅니다.

자기 개선 루프: 오류에서 배우고 워크플로우를 갱신하는 구조

WAT 프레임워크에는 자기 개선 루프(Self-Improvement Loop)가 내장되어 있습니다. 에이전트가 도구를 실행하다가 오류를 만나면, 오류 메시지를 읽고 원인을 분석합니다. 도구의 코드를 수정합니다. 수정된 도구가 정상 동작하는지 확인합니다. 그런 다음 워크플로우 파일에 "이 상황에서는 이렇게 처리한다"는 예외 처리 항목을 추가합니다.

예를 들어, 경쟁사 웹사이트를 웹 정보 수집하는 도구가 API 호출 한도를 초과했다고 합시다. 에이전트는 오류 메시지에서 "Rate Limit Exceeded"를 읽습니다. API 문서를 검색하여 배치 엔드포인트(Batch Endpoint)가 있는지 확인합니다. 배치 엔드포인트를 발견하면, 도구 코드를 수정하여 배치 요청 방식으로 전환합니다. 수정된 도구를 검증합니다.

정상 동작을 확인한 뒤, 워크플로우 파일에 "API 한도 초과 시 배치 엔드포인트 사용"이라는 항목을 추가합니다.

다음에 같은 워크플로우를 실행하면, 에이전트는 업데이트된 워크플로우를 읽고 처음부터 배치 방식을 사용합니다. 같은 오류를 두 번 겪지 않습니다. 워크플로우를 반복 실행할수록, 예외 처리가 축적되고, 도구의 코드가 다듬어지며, 전체 시스템이 견고해집니다.

파일 구조 설계: workflows, tools, temp 폴더의 역할

에이전트가 만드는 파일이 늘어날수록 정리 체계가 필요합니다. WAT 프레임워크는 세 개의 핵심 폴더를 사용합니다.

workflows/

폴더에는 워크플로우 파일이 들어갑니다. 경쟁사 분석 워크플로우, 뉴스레터 생성 워크플로우, 리드 웹 정보 수집 워크플로우 — 각각 하나의 마크다운 파일입니다.

tools/

폴더에는 도구 파일이 들어갑니다. 웹 웹 정보 수집 도구, PDF 생성 도구, 이메일 발송 도구 — 각각 하나의 파이썬 파일입니다.

temp/

폴더는 임시 파일을 위한 공간입니다. 작업 도중 생성되는 중간 데이터, 디버깅용 로그, 일회성 출력물이 여기에 저장됩니다. 작업이 끝나면 정리해도 되는 파일들입니다.

이 폴더 구조를 에이전트에게 알려주는 것이

claude.md

파일의 역할 중 하나입니다.

claude.md

에 "워크플로우는

workflows/

폴더에, 도구는

tools/

폴더에, 임시 파일은

temp/

폴더에 저장하라"고 적어 두면, 에이전트는 이 규칙을 따릅니다.

이 파일 없이 작업하면, 에이전트가 모든 파일을 프로젝트 최상위에 무질서하게 쌓아 놓을 수 있습니다. 파일이 늘어날수록 혼란이 커지고, 에이전트도 어떤 도구가 어디 있는지 찾는 데 시간을 낭비합니다.

claude.md

파일을 프로젝트에 넣고 "이 파일을 읽고 프로젝트를 초기화해줘"라고 에이전트에게 말하면, 에이전트가 폴더 구조를 자동으로 생성합니다. 왼쪽 탐색기에

workflows

,

tools

,

temp

폴더가 나타나고, 각 폴더 안에 안내 파일(README)이 배치됩니다. 이 초기화 과정은 새로운 프로젝트를 시작할 때마다 한 번만 수행하면 됩니다.

〔도표 05-01: WAT 프레임워크 구조도. 에이전트(A)가 워크플로우(W)를 읽고, 도구(T)를 실행하는 삼각 관계. 워크플로우 폴더, 도구 폴더, 임시 폴더의 배치가 표시된다.〕

케이크 비유로 돌아가 보겠습니다. 레시피를 잘 정리해 두면 같은 케이크를 언제든 다시 만들 수 있고, 재료만 바꾸면 변형된 케이크를 시도할 수 있습니다. 셰프가 새로운 사람으로 바뀌어도, 레시피를 따르면 비슷한 결과를 낼 수 있습니다. 워크플로우와 도구를 분리해 두는 것이 바로 이 효과를 만듭니다.

그런데 셰프에게 레시피를 건네기 전에, 셰프가 일하는 방식 자체를 설정해 주어야 합니다. 그것이

claude.md

의 역할이며, 에이전트의 작업 기억이 어떻게 작동하는지 이해해야 그 설정을 잘 쓸 수 있습니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

7장 모델이 세는 글 조각과 작업 기억: 한 번에 읽는 범위

전체 목차

책의 모든 장 보기

도입

클로드 코드에게 경쟁사 분석을 맡기고, 이런저런 수정 사항을 주고받다 보니 대화가 길어졌습니다. 처음에는 정확한 차트와 깔끔한 PDF를 만들어 주던 에이전트가, 어느 순간 이상한 데이터를 끌어오기 시작합니다. 분명 앞에서 알려 준 브랜드 색상을 무시하고, 이미 답한 질문을 다시 묻습니다.

화면 하단에 눈길을 돌리면 "23% of your context remaining"이라는 문구가 보입니다. 에이전트의 메모장이 거의 꽉 찬 것입니다. 이 현상을 이해하려면, 에이전트가 우리의 말을 어떻게 읽는지부터 알아야 합니다.

모델이 세는 글 조각이란 무엇인가

사람은 글을 단어 단위로 읽습니다. AI는 다릅니다. AI는 모델이 세는 글 조각(Token)이라는 조각 단위로 읽습니다.

모델이 세는 글 조각 하나가 정확히 단어 하나는 아닙니다. 대략 3에서 4개의 영문 글자가 하나의 모델이 세는 글 조각에 해당합니다. 쉼표 하나도 모델이 세는 글 조각 하나이고, "in"이라는 짧은 전치사도 모델이 세는 글 조각 하나입니다. 공백조차 모델이 세는 글 조각 계산에 포함됩니다. 일관된 규칙이 있는 것처럼 보이지만, 실제로는 꽤 들쭉날쭉합니다.

대략적인 기준을 하나 제시하면, 영어 기준으로 모델이 세는 글 조각 하나는 단어의 약 75%에 해당합니다.

한국어는 상황이 조금 다릅니다. 한국어 한 글자가 여러 모델이 세는 글 조각으로 쪼개지는 경우가 잦아서, 같은 의미를 담은 문장이라도 영어보다 모델이 세는 글 조각을 더 많이 소모합니다.

왜 이것이 중요합니까? 모델이 세는 글 조각은 돈이기 때문입니다. 클로드 코드를 포함한 모든 큰 규모의 언어 모델은 모델이 세는 글 조각 단위로 비용을 계산합니다. 구독 요금제를 사용하더라도, 일정 기간 내에 사용할 수 있는 모델이 세는 글 조각 총량에는 한계가 있습니다. 모델이 세는 글 조각을 낭비하면 한도를 빨리 소진합니다.

〔그림 7-1〕 영어 문장이 모델이 세는 글 조각으로 분해되는 예시. "Hello, how are you?"가 〔"Hello", "," , " how", " are", " you", "?"〕로 쪼개지는 시각적 도해〕

한 번에 읽는 범위의 크기와 한계

클로드에게 메모장이 하나 있다고 상상해 보겠습니다. 대화를 시작하면, 클로드는 이 메모장에 사용자가 말한 것, 자기가 답한 것, 시스템 프롬프트, MCP 도구 목록, 프로젝트 파일 내용을 전부 빼곡히 적어 넣습니다. 이 메모장이 바로 한 번에 읽는 범위(Context Window)입니다.

이 메모장의 크기에는 물리적 한계가 있습니다. 현재 클로드 모델의 표준 한 번에 읽는 범위는 약 200,000모델이 세는 글 조각입니다. 숫자만 보면 꽤 커 보이지만, 생각보다 빠르게 채워집니다. 메모장에 들어가는 항목을 나열해 보겠습니다.

시스템 프롬프트

: claude.md 파일 전체

MCP 연결 서버

: 연결된 각 MCP 연결 서버의 도구 정의와 설명

대화 기록

: 사용자가 보낸 모든 메시지와 에이전트의 모든 응답

파일 내용

: 에이전트가 읽은 코드, 워크플로우, 데이터 파일

도구 호출 결과

: 스크립트 실행 결과, 검색 결과, 웹 웹 정보 수집 결과

프로젝트 규모가 커질수록, MCP 연결 서버를 여러 개 연결할수록, 이 항목들이 메모장을 빠르게 잠식합니다.

/context

명령어를 입력하면 현재 모델이 세는 글 조각 사용 내역을 확인할 수 있습니다. 예를 들어, 한 세션에서 모델이 클로드 오퍼스 4.6이고 모델이 세는 글 조각이 22,000/200,000이라면, 이미 전체의 11%를 소비한 셈입니다.

〔그림 7-2〕 /context 명령어 실행 결과 예시. 시스템 프롬프트, 시스템 도구, MCP 도구, 대화 기록이 각각 몇 모델이 세는 글 조각을 차지하는지 막대그래프로 보여주는 화면 캡처〕

맥락이 새는 현상

메모장이 절반쯤 차면 미묘한 변화가 시작됩니다. 처음에는 정확하던 에이전트가 점차 엉뚱한 답을 내놓습니다. 이미 제공한 정보를 다시 요청하거나, 없는 사실을 지어내기도 합니다. 이 현상을 컨텍스트 로트(Context Rot)라고 부릅니다.

원인은 큰 규모의 언어 모델이 긴 문맥을 처리하는 방식에 있습니다. 모델이 세는 글 조각이 쌓일수록 모델의 품질과 정확도가 눈에 띄게 떨어집니다. 대화의 맨 처음과 맨 끝에 있는 정보는 비교적 잘 기억하지만, 중간에 있는 정보는 묻히기 쉽습니다. 이것을 "중간 소실(Lost in the Middle)" 현상이라고 합니다.

그래프를 하나 그려 본다면, 가로축은 모델이 세는 글 조각 사용량이고 세로축은 응답 품질입니다. 사용량이 50%를 넘어서면 품질 곡선이 완만한 내리막을 타다가, 70%를 지나면 가파르게 추락합니다. 이 현상은 클로드뿐 아니라 모든 큰 규모의 언어 모델에서 관찰됩니다.

〔그림 7-3〕 컨텍스트 사용량(가로축)과 응답 품질(세로축)의 관계를 보여주는 곡선 그래프. 60% 지점에 수직 점선을 그어 "경고 구간" 표시〕

실제 작업 중에 겪는 증상은 이렇습니다. 에이전트가 갑자기 허구의 정보를 만들어 냅니다. 앞서 저장해 둔 파일의 경로를 엉뚱하게 기억합니다. 같은 질문을 반복합니다. 이런 신호가 포착되면, 모델이 세는 글 조각 메모장을 정리할 때가 된 것입니다.

60% 규칙

경쟁사 분석 워크플로우를 실행하고, 결과를 검토하고, 수정 사항을 주고받다 보면 화면 하단의 컨텍스트 잔여량 표시가 어느새 50% 아래로 내려가 있습니다. "45% of your context remaining until auto-compact"라는 문구를 본 적이 있을 것입니다.

실무에서 유용한 기준이 하나 있습니다. 컨텍스트의 60%가 채워지면 정리를 시작하라는 것입니다. 이것을 60% 규칙이라 부릅니다.

왜 하필 60%입니까? 그 시점을 넘기면 컨텍스트 로트가 실질적으로 체감되기 시작하기 때문입니다. 물론 클로드 코드에는 자동 압축(Auto-compact) 기능이 내장되어 있어서, 한계에 도달하면 스스로 대화를 압축합니다. 하지만 자동 압축은 에이전트가 어떤 정보를 보존하고 어떤 정보를 버릴지 자체적으로 판단합니다. 중요한 결정 사항이 압축 과정에서 소실될 수 있습니다.

따라서 능동적으로, 60% 지점에서 직접 정리를 수행하는 편이 안전합니다. 그렇다면 정리는 어떻게 합니까? 두 가지 도구가 있습니다.

/compact: 대화를 압축하되 기억은 남긴다

/compact

명령어를 입력하면, 클로드 코드는 지금까지의 대화 기록을 요약본으로 압축합니다. 긴 대화가 핵심만 추린 짧은 요약으로 바뀌면서, 모델이 세는 글 조각 사용량이 크게 줄어듭니다. 마치 장문의 회의록을 핵심 결정 사항 목록으로 정리하는 것과 비슷합니다.

압축 이후에도 에이전트는 이전에 무슨 작업을 했는지 대략적으로 기억합니다. 만든 파일, 실행한 도구, 도달한 결론 — 이런 것들이 요약본 안에 남아 있어서, 대화를 이어 갈 수 있습니다.

그런데 여기서 한 걸음 더 나아갈 수 있습니다.

/compact

명령어 뒤에 보존하고 싶은 정보를 직접 지정하는 것입니다.

/compact keep API decisions and schema

이렇게 입력하면, 에이전트는 API 관련 결정 사항과 스키마 정보를 압축 과정에서 우선적으로 보존합니다. 브랜드 자산 경로를 잃고 싶지 않다면

/compact keep brand asset paths

라고 쓸 수 있습니다. 프로젝트의 핵심 맥락이 무엇인지 사용자가 직접 알려 주는 셈입니다.

이것이 선택적 보존 전략입니다. 에이전트에게 "전부 기억해"라고 하는 대신, "이것만은 반드시 기억해"라고 말하는 것이 훨씬 효과적입니다.

/clear: 완전한 초기화

때로는 압축이 아니라 백지 상태가 필요합니다.

/clear

명령어는 대화 기록을 통째로 지웁니다. 에이전트는 이전 대화에서 무엇을 했는지 전혀 기억하지 못합니다.

하지만 중요한 점이 있습니다.

/clear

를 실행해도 프로젝트의 파일은 그대로 남습니다. claude.md 파일, 워크플로우 문서, 도구 스크립트, 비즈니스 프로필 JSON — 디스크에 저장된 모든 파일은 건드리지 않습니다. 에이전트가 새 대화를 시작하면, claude.md를 처음부터 다시 읽고, 필요한 파일을 다시 탐색합니다. 마치 새로 출근한 비서가 업무 매뉴얼을 펼치는 것과 같습니다.

/clear

를 사용하기 좋은 시점은 다음과 같습니다.

작업의 맥락이 완전히 달라질 때 (경쟁사 분석에서 채용 공고 웹 정보 수집으로 전환하는 경우)

컨텍스트 로트가 심각하여 에이전트가 반복적으로 오류를 낼 때

새로운 워크플로우를 깨끗한 상태에서 시작하고 싶을 때

/compact와 /clear, 언제 무엇을 쓸 것인가

두 명령어의 차이를 정리합니다.

구분

/compact

/clear

대화 기록

요약본으로 압축

완전 삭제

이전 작업 맥락

대략적으로 유지

소실

파일(claude.md 등)

유지

토큰 절감 효과

중간~높음

최대

권장 시점

같은 작업을 계속할 때

작업 주제가 바뀔 때

〔그림 7-4〕 /compact와 /clear의 동작 차이를 좌우 비교로 보여주는 다이어그램. 왼쪽은 대화 기록이 요약본으로 압축되는 모습, 오른쪽은 대화 기록이 사라지고 claude.md만 남는 모습〕

실전 패턴

실무에서 이 도구들을 활용하는 흐름은 이렇습니다.

작업을 시작합니다. 에이전트와 여러 차례 대화를 주고받습니다. 화면 하단의 잔여량 표시가 40% 아래로 내려갑니다. 여기서

/compact keep brand assets and competitor list

를 실행합니다. 모델이 세는 글 조각이 확보되면 작업을 이어갑니다. 작업이 끝났습니다. 다음 작업은 전혀 다른 주제입니다. 이때는

/clear

를 실행하고 새로 시작합니다.

이 리듬이 몸에 익으면, 에이전트의 품질이 대화 길이에 관계없이 일정하게 유지되는 것을 체감할 수 있습니다. 모델이 세는 글 조각과 한 번에 읽는 범위는 처음 접하면 기술적으로 보이지만, 핵심은 결국 하나의 습관입니다. 메모장이 반쯤 찼으면 정리한다.

에이전트의 기억을 관리하는 방법을 익혔으니, 이제 그 기억의 첫 페이지에 해당하는 문서 — 에이전트가 매번 가장 먼저 읽는 그 파일 — 를 어떻게 작성하면 좋을지 살펴볼 차례입니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

8장 claude.md: 에이전트 지침서 쓰기

전체 목차

책의 모든 장 보기

도입

새로 채용한 직원이 첫 출근을 합니다. 사무실 문을 열고 들어온 그에게 아무런 안내 없이 "일해"라고만 말한다면 어떻게 될까요. 화장실이 어디인지, 주요 거래처 연락처가 어디에 있는지, 회사의 톤 앤 매너가 어떤지 — 아무것도 모른 채 이메일을 쓰기 시작할 것입니다. 결과물은 엉뚱할 수밖에 없습니다. 클로드 코드를 새 프로젝트에서 처음 실행하는 상황이 정확히 이것입니다.

에이전트는 폴더를 열고, 가장 먼저 하나의 파일을 찾습니다.

claude.md

입니다. 이 파일이 없으면 에이전트는 안내 없이 출근한 신입 직원과 다를 바 없습니다.

시스템 프롬프트의 역할

n8n에서 AI 에이전트를 만들어 본 적이 있다면, 노드 설정 창에서 시스템 프롬프트(System Prompt)를 입력하는 칸을 본 적이 있을 것입니다. ChatGPT에서 커스텀 GPT를 만들어 본 적이 있다면, "Instructions" 탭에 역할과 규칙을 적어 넣는 경험을 했을 것입니다. claude.md는 그것과 동일한 역할을 합니다.

이름만 다를 뿐, 본질은 같습니다. 에이전트가 사용자의 메시지를 읽기 전에 반드시 먼저 읽는 문서입니다.

작동 순서를 따라가 보겠습니다. 사용자가 클로드 코드에 메시지를 보냅니다. 에이전트는 그 메시지를 처리하기 전에, 프로젝트 루트에 있는 claude.md 파일을 통째로 읽어 들입니다. 그다음에야 비로소 사용자의 메시지를 확인하고 응답을 생성합니다. 이 순서는 매번 동일합니다. 대화를 새로 시작할 때마다,

/clear

를 실행한 뒤에도, 에이전트는 claude.md부터 읽습니다.

이것이 의미하는 바는 분명합니다. claude.md에 적혀 있는 내용은 에이전트의 행동 전체를 관통하는 기준선이 됩니다. 여기에 "모든 응답은 한국어로 한다"고 적으면 에이전트는 한국어로 답합니다. "파이썬 스크립트를 만들 때는 반드시 에러 핸들링을 포함한다"고 적으면 에이전트는 try-except 블록을 빠뜨리지 않습니다.

〔그림 8-1〕 에이전트가 메시지를 처리하는 순서를 보여주는 플로차트. claude.md 읽기 → 사용자 메시지 확인 → 필요 시 파일 탐색 → 응답 생성〕

담아야 할 것

claude.md에 들어가야 하는 정보는 크게 세 가지 층위로 나뉩니다.

무엇(What)

: 프로젝트의 정체성입니다. 이 프로젝트가 무엇인지, 기술 스택은 무엇인지, 핵심 패키지나 스킬이 무엇인지를 명시합니다. 실제 예시를 보면 이렇습니다.

Executive Assistant

You are the user's executive assistant. Your job is to help them spend less time

on operations, people management, and admin, so they can focus on learning

AI tools and making YouTube videos. This is their #1 priority.

Everything else supports it.

한 문단입니다. 에이전트의 역할과 사용자의 최우선 목표가 담겨 있습니다.

왜(Why)

: 각 구성 요소의 목적입니다. "tools 폴더에는 파이썬 스크립트가 들어 있다"라고만 적으면 에이전트는 그 폴더의 존재를 알 뿐, 왜 거기를 먼저 봐야 하는지는 모릅니다. "새 작업을 받으면 tools 폴더를 먼저 확인하라, 이미 만들어 둔 도구가 있을 수 있다"라고 적으면 에이전트의 행동이 달라집니다.

어떻게(How)

: 에이전트에게 기대하는 작업 방식입니다. "오류가 발생하면 스크립트를 수정한 뒤 워크플로우 문서에 해당 오류 대응 방법을 추가하라"는 식의 행동 규칙이 여기에 해당합니다.

담지 말아야 할 것

팀원 전원의 이력서를 넣으면 안 됩니다. 프로젝트의 전체 회의록을 복사해 붙이면 안 됩니다. 모든 API 엔드포인트의 상세 문서를 넣으면 안 됩니다.

이유는 앞 장에서 다룬 모델이 세는 글 조각 때문입니다. claude.md는 매 대화 시작 시 통째로 읽힙니다. 파일이 500줄이면, 매번 500줄에 해당하는 모델이 세는 글 조각이 한 번에 읽는 범위를 잠식합니다. 질문 하나에 대한 답을 구하는 데, 대화를 시작하자마자 컨텍스트의 상당 부분이 시스템 프롬프트에 소모되는 셈입니다.

규칙은 간결합니다. claude.md에는 에이전트가

매번

알아야 하는 것만 넣습니다. 가끔 필요한 정보는 별도 파일에 두고, claude.md에서 그 파일의 위치를 가리킵니다.

150에서 200줄 원칙

그렇다면 claude.md의 적정 길이는 어느 정도입니까?

실전에서 검증된 기준은 150줄에서 200줄 사이입니다. 이 범위 안에서 프로젝트 정체성, 핵심 규칙, 파일 위치 안내를 모두 담을 수 있습니다. 실제로 에이전트 비서 프로젝트를 세팅한 직후의 claude.md가 약 87줄이었다는 사례가 있습니다. 프로젝트가 커지면서 스킬과 도구가 추가되고, 참조 파일이 늘어나면, 자연스럽게 150줄을 향해 불어납니다.

200줄을 넘어가기 시작하면 경계해야 합니다. 그 시점에서 claude.md 안에 있는 내용 중 "정말 매번 읽어야 하는가?"라는 질문에 "아니오"인 항목이 분명히 있습니다. 그 항목은 별도 파일로 빼야 합니다.

시간이 흐르면서 프로젝트에 새로운 스킬이 추가되고, 참조 문서가 늘어나고, 규칙이 세분화됩니다. claude.md도 자연스럽게 비대해집니다. 이때 필요한 것이 라우팅 전략입니다.

라우팅 전략: claude.md에서 다른 파일로 안내하기

에이전트 비서 프로젝트의 claude.md를 열어 보면, 중간 부분에 이런 구절이 있습니다.

Quick Reference

- About Me → read context/me.md

- Business details → read context/work.md

- Team info → read context/team.md

- Current focus → read context/current-priorities.md

- Project: Website Launch → read projects/website-launch/README.md

- Project: West Coast Expansion → read projects/west-coast-expansion/README.md

이것이 라우팅(Routing)입니다. claude.md 자체에는 "사용자가 누구인지"에 대한 상세 정보가 없습니다. 대신 "사용자에 대해 알고 싶으면 이 파일을 읽어라"라는 안내만 있습니다. 에이전트는 사용자의 질문이 사용자의 개인 정보와 관련될 때에만

context/me.md

를 읽습니다. 관련이 없으면 읽지 않습니다.

이렇게 하면 매번 소모되는 모델이 세는 글 조각은 claude.md의 짧은 안내 문구뿐입니다.

라우팅의 효과를 수치로 살펴보겠습니다. 사용자 프로필, 업무 정보, 팀 정보, 우선순위, 프로젝트 설명을 전부 claude.md에 담으면 500줄이 넘을 수 있습니다. 반면 라우팅을 적용하면 claude.md는 100에서 150줄 수준을 유지하면서, 에이전트는 필요한 순간에만 해당 파일을 불러옵니다. 한 번에 읽는 범위의 여유 공간이 그만큼 늘어납니다.

라우팅 대상이 될 수 있는 파일의 예시입니다.

파일

내용

claude.md 안내 예시

me.md

사용자 개인 정보, 성향, 배경

"About the user → context/me.md"

work.md

사업 정보, 제품, 고객

"Business context → context/work.md"

team.md

팀원 정보, 역할, 연락처

"Team info → context/team.md"

current-priorities.md

현재 분기 목표, 긴급 업무

"Current focus → context/current-priorities.md"

brand-assets/

로고, 색상 가이드라인

"Brand files → brand-assets/"

〔그림 8-2〕 claude.md를 허브로, me.md·work.md·team.md 등을 스포크로 연결한 허브-앤-스포크(Hub-and-Spoke) 다이어그램〕

프로젝트가 성장할 때 claude.md를 갱신하는 습관

claude.md는 한 번 작성하고 끝나는 문서가 아닙니다. 프로젝트와 함께 숨을 쉬는 문서입니다.

최초 생성: /init 명령어

기존 프로젝트에 claude.md가 없다면,

/init

명령어를 실행합니다. 에이전트가 프로젝트의 폴더 구조와 코드를 스캔한 뒤, 자동으로 claude.md 초안을 생성합니다. 완벽하지는 않지만, 빈 문서에서 시작하는 것보다 훨씬 빠릅니다. 초안을 훑어보고, 빠진 규칙을 추가하고, 불필요한 부분을 삭제하면 됩니다.

일상적 갱신

새로운 스킬이나 워크플로우를 추가했을 때, 에이전트에게 직접 말하면 됩니다. "방금 brand-assets 라는 폴더를 추가했어. claude.md를 업데이트해서, 로고나 브랜드 가이드라인이 필요하면 그 폴더를 참조하라고 안내해 줘." 에이전트는 claude.md 파일을 열어 해당 내용을 추가합니다. 이런 습관이 쌓이면, claude.md는 프로젝트의 현재 상태를 정확히 반영하는 지도가 됩니다.

정기적 압축

프로젝트가 몇 주, 몇 달간 운영되면 claude.md에 누적된 항목이 200줄을 넘기기 시작합니다. 이때 에이전트에게 이렇게 요청합니다. "claude.md가 좀 길어졌어. 현재 상태를 반영해서 정리해 줘. 더 이상 사용하지 않는 도구나 프로젝트 참조는 제거하고, 비슷한 규칙은 통합해." 에이전트가 파일을 분석하고 정리합니다. 결과를 검토한 뒤 승인하면 됩니다.

claude.md와 .claude/ 디렉터리의 관계

프로젝트 루트에는 claude.md 외에

.claude

라는 숨김 디렉터리가 생기는 경우가 있습니다. 이 디렉터리 안에는 커뮤니케이션 스타일 같은 세부 규칙 파일이 저장됩니다. claude.md가 큰 틀의 지침이라면,

.claude

안의 파일들은 보조 규칙이라고 생각하면 됩니다.

프로젝트 루트/

├── CLAUDE.md ← 핵심 지침 (150~200줄)

├── .claude/

│ └── rules/

│ └── communication-style.md ← 세부 스타일 규칙

├── context/

│ ├── me.md

│ ├── work.md

│ └── team.md

├── workflows/

├── tools/

└── .env

〔그림 8-3〕 프로젝트 폴더 구조에서 claude.md와 .claude/ 디렉터리, 그리고 context 폴더의 관계를 보여주는 트리 구조 시각화〕

요약하면, claude.md는 에이전트의 직무기술서입니다. 역할을 정의하고, 핵심 규칙을 명시하고, 상세 정보가 담긴 파일의 위치를 가리킵니다. 길이는 150에서 200줄 이내로 유지합니다. 프로젝트가 변하면 함께 업데이트합니다.

에이전트가 누구인지, 무엇을 기억하는지, 어떤 규칙을 따르는지를 정리했습니다. 그런데 에이전트가 아무리 똑똑해도, 외부 세계와 상호작용할 수단이 없으면 할 수 있는 일은 제한적입니다. 이메일을 보내거나, 웹 사이트를 웹 정보 수집하거나, 캘린더에 일정을 추가하려면 에이전트에게 "손"이 필요합니다.

9장 MCP 연결 서버: 에이전트 확장하기

전체 목차

책의 모든 장 보기

도입

토요일 아침, 케이크를 만들기로 마음먹었습니다. 레시피에는 달걀, 밀가루, 프로스팅이 필요합니다. 달걀은 양계장에서 사야 하고, 밀가루는 제분소에서 사야 하고, 프로스팅은 제과 재료 전문점에서 사야 합니다. 세 군데를 돌아다니며 각각의 주문 방식을 익히고, 각각의 결제 시스템을 통과해야 합니다. 번거롭습니다. 그런데 동네에 슈퍼마켓이 하나 있다면 어떨까요.

한 곳에서 달걀도 사고 밀가루도 사고 프로스팅도 삽니다. 주문 방식도 하나, 결제 시스템도 하나입니다. MCP 연결 서버가 바로 이 슈퍼마켓입니다.

MCP란 무엇인가: 슈퍼마켓 비유

MCP는 모델 컨텍스트 프로토콜(Model Context Protocol)의 약자입니다. 에이전트가 외부 서비스에 접근할 때 사용하는 표준화된 연결 방식입니다.

Gmail을 예로 들어 보겠습니다. Gmail에는 이메일을 보내는 기능이 있고, 읽는 기능이 있고, 검색하는 기능이 있고, 초안을 만드는 기능이 있습니다. 하나의 서비스 안에 여러 가지 도구가 들어 있습니다. Gmail MCP 연결 서버를 클로드 코드에 연결하면, 에이전트는 이 모든 도구를 한꺼번에 사용할 수 있게 됩니다.

"이메일 보내기"를 위해 별도의 스크립트를 짤 필요가 없습니다. "검색하기"를 위해 API 문서를 뒤질 필요도 없습니다. 에이전트가 MCP 연결 서버에 연결되어 있으면, 어떤 도구가 있는지 자동으로 파악하고, 상황에 맞는 도구를 골라서 사용합니다.

슈퍼마켓 비유를 좀 더 확장해 보겠습니다.

MCP 연결 서버 없이

: 달걀이 필요하면 양계장(이메일 전송 API)에 직접 가야 합니다. 밀가루가 필요하면 제분소(웹 웹 정보 수집 API)에 직접 가야 합니다. 각 가게마다 주문서 양식이 다르고, 영업시간이 다르고, 결제 방식이 다릅니다.

MCP 연결 서버 사용

: 슈퍼마켓(Gmail MCP 연결 서버, Firecrawl MCP 연결 서버 등) 하나에 들어가면 필요한 재료를 전부 구할 수 있습니다. 에이전트는 슈퍼마켓의 진열대(도구 목록)를 둘러보고, 필요한 것만 장바구니에 담습니다.

〔그림 9-1〕 왼쪽에 "MCP 없이"로 에이전트가 개별 API 세 곳에 각각 연결하는 모습, 오른쪽에 "MCP 사용"으로 에이전트가 MCP 연결 서버 하나를 통해 여러 도구에 접근하는 모습을 비교하는 다

이어그램〕

MCP의 핵심은 표준화입니다. 어떤 서비스든 MCP 규격에 맞춰 서버를 만들어 두면, 클로드 코드는 동일한 방식으로 연결하고 동일한 방식으로 도구를 호출합니다. Gmail이든 Firecrawl이든 Google Calendar든, 연결 방법의 골격이 같습니다.

Firecrawl MCP 연결 서버 설치 실습

이론만으로는 감이 잡히지 않습니다. 실제로 MCP 연결 서버를 하나 설치해 보겠습니다. Firecrawl은 웹사이트의 데이터를 수집하는 서비스입니다. 웹 웹 정보 수집(Web Scraping), 크롤링(Crawling), 검색, 스크린샷 캡처 등 여러 기능을 제공합니다. 이 서비스의 MCP 연결 서버를 클로드 코드에 연결하면, 에이전트가 자유롭게 웹 데이터를 가져올 수 있게 됩니다.

1단계: Firecrawl API 키 발급

Firecrawl 웹사이트(firecrawl.dev)에 접속하여 계정을 만듭니다. 무료 플랜으로 시작하면 500크레딧을 받습니다. 실습하기에 충분한 양입니다. 대시보드에 들어가면 API 키가 표시되어 있습니다. 이 키를 복사해 둡니다.

2단계: .env 파일에 API 키 저장

프로젝트 루트에

.env

파일이 없으면 생성합니다. 이미 있다면 해당 파일을 엽니다. 여기에 API 키를 다음과 같은 형식으로 저장합니다.

FIRECRAWL_API_KEY=fc-xxxxxxxxxxxxxxxxxxxxxxxx

왜

.env

파일에 저장합니까? 이유는 다음 절에서 자세히 다루겠지만, 핵심만 말하면 이렇습니다. API 키는 비밀번호와 같습니다. 대화 기록에 직접 입력하면, 그 키가 한 번에 읽는 범위 안에 평문으로 남습니다.

.env

파일에 저장하면 에이전트가 필요할 때 파일에서 읽어 오되, 대화 기록 자체에는 키가 노출되지 않습니다.

3단계: MCP 연결 서버 설치 명령 실행

클로드 코드에게 설치를 요청합니다. Firecrawl의 공식 문서에는 클로드 코드 명령줄 방식용 설치 명령어가 안내되어 있습니다. 에이전트에게 이렇게 말합니다.

Firecrawl MCP 서버를 설치하고 싶어. 아래 명령어를 사용해서 설치해 줘.

〔Firecrawl 공식 문서에서 복사한 CLI 명령어 붙여넣기〕

단, API 키는 직접 넘기지 마. .env 파일에서 읽어 와.

에이전트는 명령어를 실행하여 MCP 연결 서버를 프로젝트에 등록합니다. 설치가 완료되면,

.env

파일에 저장된 API 키를 참조하도록 설정을 마무리합니다.

4단계: 설치 확인

설치가 끝났으면

/context

명령어를 입력해 봅니다. MCP 도구 항목에 Firecrawl 관련 도구들이 표시되어야 합니다. 웹 웹 정보 수집, 크롤링, 검색 등의 도구가 목록에 나타나면 정상적으로 연결된 것입니다.

〔그림 9-2〕 /context 명령어 실행 결과에서 MCP Tools 섹션에 Firecrawl 도구 목록이 표시된 화면 캡처〕

이 시점부터 에이전트는 "이 URL에서 직업 공고를 웹 정보 수집해 줘"라는 자연어 요청을 받으면, Firecrawl MCP 연결 서버의 적절한 도구를 자동으로 선택하여 실행합니다. 어떤 엔드포인트를 호출해야 하는지, 어떤 파라미터를 넘겨야 하는지를 사용자가 알 필요 없습니다.

API 키 보안: .env 파일에 비밀을 저장하는 이유

Firecrawl 설치 과정에서

.env

파일을 언급했습니다. 왜 API 키를 대화창에 직접 입력하면 안 되는지, 좀 더 깊이 살펴보겠습니다.

API 키(API Key)는 외부 서비스에 접근하기 위한 인증 수단입니다. 비밀번호와 본질적으로 같습니다. 이 키가 노출되면 누군가가 내 계정으로 서비스를 사용할 수 있고, 비용이 청구될 수 있습니다.

클로드 코드 대화창에 API 키를 직접 붙여 넣으면 어떤 일이 벌어집니까? 그 키는 대화 기록의 일부가 됩니다. 한 번에 읽는 범위 안에 평문으로 존재합니다. 에이전트가 Bash 명령어를 실행할 때 키를 인자로 전달하면, 실행 로그에도 키가 남을 수 있습니다. 보안 관점에서 바람직하지 않습니다.

.env

파일을 사용하면 이 문제가 완화됩니다. 키는 내 컴퓨터 파일에만 존재합니다. 에이전트는 스크립트 내에서

os.environ

같은 방식으로 환경 변수를 읽어 오고, 키 값 자체는 대화 기록에 남지 않습니다.

여기에 한 가지 안전장치를 더합니다. 프로젝트에

.gitignore

파일을 만들고,

.env

를 목록에 추가합니다. 이렇게 하면 프로젝트를 깃허브(GitHub) 같은 공개 저장소에 올리더라도

.env

파일은 업로드되지 않습니다. 비밀이 인터넷에 노출되는 사고를 방지할 수 있습니다.

.gitignore

.env

이 두 줄이 보안의 기본 방어선입니다.

〔그림 9-3〕 API 키의 안전한 관리 흐름을 보여주는 다이어그램. ".env 파일 → 환경 변수로 로드 → 스크립트에서 참조" 경로와, "대화창에 직접 입력 → 컨텍스트 기록에 노출" 경로를 비교〕

실습 중에 실수로 API 키를 대화창에 입력했다면, 해당 키를 즉시 교체(Rotate)하는 것이 안전합니다. 서비스 대시보드에서 기존 키를 폐기하고 새 키를 발급받아

.env

파일에 저장하면 됩니다.

MCP로 확장되는 에이전트의 행동 반경

Firecrawl 하나를 연결했을 뿐인데, 에이전트가 할 수 있는 일의 범위가 크게 넓어졌습니다. 구직 사이트에서 수백 개의 공고를 웹 정보 수집하여 엑셀 파일로 정리할 수 있습니다. 옐로우 페이지에서 치과 의사 연락처를 수집하여 영업용 리드 목록을 만들 수 있습니다. 경쟁사 웹사이트를 정기적으로 모니터링할 수 있습니다.

하나의 MCP 연결 서버가 하나의 슈퍼마켓이라면, 여러 MCP 연결 서버를 연결하는 것은 여러 슈퍼마켓 회원권을 동시에 보유하는 것과 같습니다. Gmail MCP 연결 서버를 추가하면 이메일 읽기, 보내기, 검색이 가능해집니다. Google Calendar MCP 연결 서버를 추가하면 일정 조회와 생성이 가능해집니다.

이들을 조합하면 "오늘 받은 이메일 중 회의 요청을 찾아서 캘린더에 일정을 추가해 줘"라는 복합적인 요청을 에이전트가 처리할 수 있습니다.

MCP 서버

제공하는 도구(예시)

Firecrawl

웹 스크래핑, 크롤링, 검색, 스크린샷

Gmail

이메일 보내기, 읽기, 검색, 초안 생성

Google Calendar

일정 조회, 생성, 수정, 삭제

Notion

페이지 생성, 데이터베이스 조회, 코멘트

Slack

메시지 전송, 채널 조회, 파일 업로드

〔그림 9-4〕 에이전트를 중심에 두고, Firecrawl·Gmail·Calendar·Notion 등 여러 MCP 연결 서버가 방사형으로 연결된 다이어그램. 각 서버 옆에 대표 도구 2에서 3개를 작은 아이콘으로 표시〕

MCP 연결 서버를 추가할 때마다, 해당 서비스의 API 키를

.env

파일에 저장하고, 설치 명령어를 실행하는 패턴은 동일합니다. 한 번 익힌 절차를 반복할 뿐입니다.

여기서 한 가지 유의할 점이 있습니다. MCP 연결 서버를 연결할수록 한 번에 읽는 범위에서 MCP 도구 정의가 차지하는 비중이 커집니다. 앞 장에서 다룬

/context

명령어로 확인하면, MCP Tools 항목의 모델이 세는 글 조각 소모량이 늘어나 있는 것을 볼 수 있습니다. 에이전트의 손은 많아졌지만, 기억할 수 있는 공간은 줄어든 셈입니다.

필요 없는 MCP 연결 서버를 무분별하게 연결하면, 정작 중요한 대화 내용을 담을 공간이 부족해질 수 있습니다.

MCP 연결 서버의 선택은 프로젝트의 목적에 맞춰야 합니다. 경쟁사 분석을 하는 프로젝트에 Gmail MCP 연결 서버가 필요하지는 않습니다. 이메일 자동화를 하는 프로젝트에 Firecrawl이 꼭 있어야 하는 것도 아닙니다. 필요한 손만 달아 주는 것이 효율적입니다.

에이전트에게 기억(한 번에 읽는 범위)과 지침(claude.md)과 손(MCP 연결 서버)을 갖추어 주었습니다. 이 세 가지가 어우러지면, 에이전트는 비로소 실질적인 업무를 수행할 준비가 됩니다. 이제 이

도구들을 결합하여 실제 프로젝트를 구축하는 과정으로 넘어갑니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

10장 기본 명령어와 프로젝트 설정 계층

전체 목차

책의 모든 장 보기

도입

터미널에

/status

를 입력하는 순간, 화면에 클로드 코드의 버전, 현재 연결된 모델명, 인증된 계정 정보가 한꺼번에 쏟아집니다. 마치 자동차 계기판을 한 번 들여다보는 것처럼, 지금 이 에이전트가 어떤 상태인지를 단번에 파악할 수 있습니다. 이 명령어를 외워야 했을까요? 아닙니다.

"지금 내 클로드 코드 상태가 어떤지 알려줘"라고 자연어로 물어도 에이전트가 알아서

/status

를 실행합니다. 기본 명령어(Built-in Command)의 세계는 이렇게 시작됩니다.

핵심 명령어 모음

클로드 코드에는 크게 두 부류의 기본 명령어가 있습니다. 하나는 현재 상태를 확인하는

진단·상태 명령어

이고, 다른 하나는 에이전트의 행동 방식을 바꾸는

설정 명령어

입니다.

진단·상태 명령어부터 살펴봅니다.

/model

— 사용 중인 모델을 확인하거나 다른 모델로 전환합니다. Opus에서 Sonnet으로, 혹은 Haiku로 바꿀 때 사용합니다.

/help

— 사용 가능한 모든 명령어 목록을 표시합니다. 명령어 이름이 기억나지 않을 때 가장 먼저 떠올릴 명령어입니다.

/doctor

— 설치 상태 진단을 실행합니다. 의존성 누락, 인증 문제, 네트워크 연결 상태를 점검하여 "이상 없음" 혹은 구체적인 문제 원인을 알려 줍니다.

/status

— 클로드 코드의 버전, 연결된 모델, 계정 정보를 한 번에 보여 줍니다.

〔그림 10-1〕 /status 명령어 실행 결과 예시〕

여기서 핵심은

이 명령어들을 외울 필요가 없다

는 점입니다. 클로드 코드는 자연어를 해석하여 적절한 명령어를 자동으로 실행할 수 있습니다. "진단 한번 돌려봐"라고 말하면

/doctor

가 실행되고, "지금 무슨 모델 쓰고 있어?"라고 물으면

/status

가 응답합니다. 명령어의 존재를 아는 것은 유용하지만, 암기는 에이전트의 몫입니다.

설정 명령어

설정 명령어는 에이전트의 작동 방식을 조율하는 제어판입니다.

/memory

— 자동 메모리(Auto Memory)를 편집합니다. 에이전트가 대화 중에 학습한 사항을 영구 저장하거나, 저장된 내용을 수정·삭제할 수 있습니다. "나는 항상 경어체를 선호해"라고 말하면 에이전트가 이를 기억하도록 저장합니다.

/config

— 전반적인 설정을 관리합니다. 어떤 모델을 기본값으로 사용할지, 모델이 세는 글 조각 사용량 경고 임계치를 얼마로 잡을지 같은 선택을 여기서 합니다.

/permissions

— 에이전트가 파일을 읽거나 쓰거나 삭제하는 권한을 제어합니다. 민감한 디렉터리에 대한 접근을 차단하거나, 반대로 자유롭게 허용하는 설정을 이곳에서 다룹니다.

/mcp

— MCP 연결 서버(Model Context Protocol Server) 연결을 관리합니다. 외부 도구나 서비스와 에이전트를 이어 주는 통로입니다.

/agents

— 보조 에이전트(Sub-agent) 목록을 확인하거나 관리합니다.

이 명령어들 역시 자연어로 접근할 수 있습니다. "내 권한 설정 보여줘"라고 하면

/permissions

내용이 표시됩니다. 그러나 직접 슬래시 명령어를 입력하는 편이 더 빠를 때도 있습니다. 에이전트가 의도를 해석하는 단계를 건너뛰기 때문입니다. 상황에 따라 자연어와 명령어를 섞어 쓰는 것이 실용적입니다.

설정 계층구조: 사용자, 프로젝트, 내 컴퓨터

기본 명령어가 에이전트와 대화하는 방식이라면, 설정 파일은 에이전트의 기본 행동 양식을 정의하는 청사진입니다. 클로드 코드의 설정은 세 개의 층으로 이루어져 있으며, 각 층이 서로 다른 범위를 담당합니다.

사용자 수준(User Level, 글로벌)

설정은 홈 디렉터리(

~/

)에 위치합니다. 이 설정은 컴퓨터에 존재하는 모든 프로젝트에 영향을 미칩니다. 어떤 폴더를 열든, 어떤 프로젝트에서 작업하든 항상 적용되는 기본값입니다.

프로젝트 수준(Project Level)

설정은 프로젝트 폴더 안의

.claude/settings.json

에 위치합니다. 이 파일은 깃허브(GitHub)에 커밋되어 팀 전체가 공유할 수 있습니다. 팀원 누구든 이 프로젝트를 열면 동일한 설정을 적용받습니다.

내 컴퓨터 프로젝트 수준(Local Project Level)

설정은

.claude/settings.local.json

에 위치합니다. 이 파일은 오직 내 컴퓨터에서만 적용됩니다. 깃허브에 올라가지 않으며, 같은 프로젝트를 쓰는 팀원에게는 보이지 않습니다.

〔그림 10-2〕 설정 계층 구조 다이어그램: 사용자 → 프로젝트 → 내 컴퓨터〕

세 계층은 위에서 아래로 우선순위가 결정됩니다. 에이전트가 어떤 동작을 수행하려 할 때, 내 컴퓨터 설정을 먼저 확인합니다. 내 컴퓨터에 해당 규칙이 없으면 프로젝트 설정을 봅니다. 프로젝트 설정에도 없으면 글로벌 설정을 따릅니다.

이 우선순위가 실제로 어떻게 작동하는지 예를 들어 보겠습니다.

read

라는 파일 읽기 명령이 있다고 합시다. 내 컴퓨터 설정에서 "read를 절대 사용하지 마라"고 지정했다면, 에이전트는 즉시 멈춥니다. 프로젝트와 글로벌에서 허용했더라도 소용없습니다. 반대로 내 컴퓨터과 프로젝트에서는 허용했지만 글로벌에서 금지했다면, 글로벌 단계에서 차단됩니다.

내 컴퓨터이 가장 좁은 범위이면서 가장 먼저 확인되고, 글로벌이 가장 넓은 범위이면서 마지막 관문 역할을 합니다.

settings.json과 settings.local.json의 차이

두 파일은 같은

.claude

폴더 안에 나란히 존재하지만, 용도가 명확히 다릅니다.

구분

settings.json

settings.local.json

위치

.claude/settings.json

.claude/settings.local.json

공유 여부

깃허브에 커밋, 팀 전원 공유

로컬 전용, 깃허브에 올리지 않음

용도

팀 표준(Team Standard)

개인 오버라이드(Personal Override)

예시

"코드 리뷰 시 반드시 ESLint 실행"

"나는 Vim 키바인딩을 선호"

settings.json

에는 팀 전체가 합의한 규칙을 넣습니다. 코딩 컨벤션, 필수 도구 목록, 공통 MCP 연결 서버 연결 정보 같은 것들입니다. 새로운 팀원이 프로젝트를 클론(clone)하면 이 설정이 자동으로 적용되므로, 온보딩 과정에서 "설정 맞춰 줘"라고 따로 안내할 필요가 줄어듭니다.

settings.local.json

에는 나만의 개인 취향을 넣습니다. 선호하는 편집기 동작, 개인 API 키 경로, 내 컴퓨터 전용 디버깅 옵션 같은 것들입니다. 팀 설정과 충돌이 생기면 내 컴퓨터이 우선하므로, 팀 표준을 따르면서도 자기만의 작업 환경을 유지할 수 있습니다.

한 가지 주의할 점이 있습니다.

settings.local.json

은

.gitignore

에 등록하여 깃허브에 절대 올라가지 않도록 해야 합니다. 개인 API 키나 민감한 경로 정보가 포함될 수 있기 때문입니다.

.claude 폴더의 구조와 역할

프로젝트 폴더를 열었을 때 눈에 들어오는

.claude

디렉터리는 클로드 코드의 심장부입니다. 이 폴더 안에 에이전트가 프로젝트를 이해하고 작동하는 데 필요한 모든 구성 요소가 담겨 있습니다.

프로젝트 폴더/

├── .claude/

│ ├── settings.json # 팀 공유 설정

│ ├── settings.local.json # 개인 전용 설정

│ ├── rules/ # 에이전트 행동 규칙

│ ├── skills/ # 재사용 가능한 스킬 파일

│ ├── agents/ # 서브에이전트 정의

```
│  └──── commands/ # 커스텀 명령어
├──── CLAUDE.md # 프로젝트 두뇌 (메인 컨텍스트)
├──── CLAUDE.local.md # 개인 전용 컨텍스트
├──── .env # 환경 변수 (API 키 등)
├──── .gitignore # 깃 제외 목록
└──── (프로젝트 파일들)
```

〔그림 10-3〕 .claude 폴더의 내부 구조와 각 파일의 역할〕

CLAUDE.md

는 에이전트가 대화를 시작할 때마다 가장 먼저 읽는 파일입니다. 프로젝트의 목적, 핵심 규칙, 다른 파일의 위치를 안내하는 지도 역할을 합니다.

CLAUDE.local.md

는

CLAUDE.md

의 개인 버전으로, 나만의 추가 지시사항을 담습니다.

.gitignore

파일은 깃이 추적하지 않을 파일 목록을 지정합니다.

.env

파일(API 키 보관),

settings.local.json

, 개인용 이미지나 비밀번호 파일 등을 여기에 등록합니다. VS Code에서 이 파일들은 회색으로 표시되어 시각적으로 구분됩니다. 초록색은 깃이 아직 추적하지 않는 새 파일, 노란색은 수정된 파일입니다.

이 색상 체계는 "지금 커밋해야 할 것이 있는가"를 한눈에 알려 줍니다.

rules/

폴더에는 에이전트의 커뮤니케이션 스타일, 코딩 규칙 같은 행동 지침이 들어갑니다.

skills/

폴더에는 재사용 가능한 워크플로우 레시피가,

agents/

폴더에는 보조 에이전트 정의 파일이 저장됩니다.

글로벌 설정도 이와 유사한 구조를 가집니다. 홈 디렉터리의

.claude

폴더 아래에

settings.json

,

skills/

,

agents/

,

rules/

등이 위치하며, 여기에 저장된 내용은 어떤 프로젝트를 열든 적용됩니다. 프로젝트마다 반복하고 싶지 않은 공통 설정 — 예컨대 "항상 경어체를 사용하라" 같은 규칙 — 은 글로벌에 두는 편이 효율적입니다.

각 계층에 무엇을 넣을지 정리하면 이렇습니다.

계층

저장 위치

넣을 내용

글로벌 (사용자)

~/

개인 기본값, 글로벌 스킬, 공통 규칙

프로젝트

.claude/settings.json

팀 표준, 프로젝트 규칙, 공유 MCP 설정

O

로컬

.claude/settings.local.json

개인 오버라이드, 로컬 API 경로, 디버깅 옵션

이 구조를 처음 접하면 복잡해 보입니다. 폴더 안에 폴더, 파일 안에 파일. 그러나 실제로 프로젝트를 하나 열고

.claude

폴더를 직접 탐색해 보면, 각 파일의 역할이 이름만으로도 드러납니다. 설정의 계층은 "팀이 합의한 것 / 프로젝트가 요구하는 것 / 내가 원하는 것"이라는 세 겹의 렌즈와 같습니다. 에이전트는 이 렌즈를 안쪽부터 바깥쪽으로 훑으며, 자신이 어떻게 행동해야 하는지를 결정합니다.

기본 명령어가 에이전트와의 즉각적인 대화 수단이라면, 설정 계층은 에이전트의 장기적인 성격을 형성하는 골격입니다. 그런데 에이전트의 기억에는 한 가지 근본적인 한계가 있습니다. 한 번에 읽는 범위(Context Window)라 불리는, 한 번에 처리할 수 있는 정보량의 천장입니다. 이 천장을 넘어서는 방대한 데이터를 에이전트가 활용하려면, 외부 기억 장치가 필요합니다.

전체 목차

책의 모든 장 보기

11장 검색 결합 생성과 의미 저장소: 긴 기억 만들기

전체 목차

책의 모든 장 보기

도입

68페이지짜리 청소기 매뉴얼 PDF를 클로드 코드에게 던져 주고, "필터 청소하는 법 알려줘"라고 물었습니다. 에이전트는 잠시 검색을 수행한 뒤, 단계별 설명을 텍스트로 보여 주었습니다. 그 아래에는 매뉴얼 속 분해 도면 이미지가 함께 표시되었습니다. 물리적 장치를 다룰 때는 글보다 그림이 훨씬 명확할 수 있다는 판단을 에이전트가 스스로 내린 것입니다.

68페이지 분량의 문서가 에이전트의 한 번에 읽는 범위에 통째로 들어갈 수 없었는데, 어떻게 정확한 페이지에서 정확한 정보를 찾아낸 걸까요? 그 비밀이 검색 결합 생성(Retrieval-Augmented Generation, 검색 증강 생성)입니다.

검색 증강 생성이란 무엇인가

AI 에이전트에게는 태생적 한계가 있습니다. 학습 데이터에 포함되지 않은 정보는 알지 못합니다. 자사의 내부 매뉴얼, 지난주에 작성된 회의록, 오늘 아침에 촬영한 현장 사진 — 이런 데이터는 모델의 훈련 과정에 존재하지 않습니다. 한 번에 읽는 범위에 모든 문서를 넣어 주는 방법도 있지만, 수십 페이지를 넘어서면 모델이 세는 글 조각 한도에 걸립니다.

검색 결합 생성는 이 문제를 우회하는 구조입니다. 세 단계로 이루어져 있습니다.

검색(Retrieval)

— 사용자의 질문과 관련된 정보를 외부 데이터 저장소에서 찾아옵니다. 전체 문서를 읽는 것이 아니라, 질문에 해당하는 조각만 골라냅니다.

증강(Augmentation)

— 검색된 조각을 에이전트의 프롬프트에 덧붙입니다. 에이전트는 이제 원래 알지 못했던 정보를 "방금 읽은 것처럼" 활용할 수 있습니다.

생성(Generation)

— 증강된 컨텍스트를 바탕으로 답변을 생성합니다. 에이전트의 추론 능력과 외부 데이터가 결합되어, 단독으로는 불가능했던 수준의 응답이 나옵니다.

〔그림 11-1〕 검색 결합 생성의 세 단계: 검색 → 증강 → 생성 흐름도〕

비유하자면, 오픈북 시험과 같습니다. 학생(에이전트)이 모든 교과서를 암기하고 있을 필요는 없습니다. 시험 중에 필요한 페이지를 펼쳐 볼 수 있으면 됩니다. 다만 어떤 페이지를 펼쳐야 하는지를 빠르게 판단하는 능력, 그리고 그 페이지의 정보를 자신의 지식과 결합하여 답안을 구성하는 능력은 학생의 몫입니다.

검색 결합 생성에서 "어떤 페이지를 펼칠 것인가"를 결정하는 기술이 바로 임베딩(Embedding)입니다.

임베딩의 개념

임베딩은 텍스트를 숫자 벡터(Vector)로 변환하는 과정입니다. "벡터"라는 단어가 수학적으로 느껴질 수 있지만, 핵심은 간단합니다. 문장의 의미를 숫자 목록으로 표현하는 것입니다.

"커피 한 잔 마시고 싶다"라는 문장과 "카페라테를 주문하고 싶다"라는 문장은 단어가 다릅니다. 그러나 의미는 비슷합니다. 임베딩 모델(Embedding Model)은 이 두 문장을 비슷한 숫자 벡터로 변환합니다. 반면 "오늘 주식 시장이 하락했다"는 전혀 다른 벡터로 변환됩니다. 의미가 다르기 때문입니다.

이 벡터들을 다차원 공간에 배치하면, 의미가 비슷한 문장끼리 가까이 모이고 의미가 다른 문장끼리 멀리 떨어집니다. 이것이

의미적 유사도(Semantic Similarity)

기반 검색의 원리입니다. 사용자가 "필터 청소 방법"이라고 질문하면, 이 질문을 벡터로 변환한 뒤, 미리 저장해 둔 문서 조각 벡터들 중에서 가장 가까운 것을 찾습니다.

키워드가 정확히 일치하지 않아도 의미가 통하면 검색됩니다. "먼지받이 세척 절차"라고 쓰여 있어도 "필터 청소 방법"에 대한 답으로 끌어올 수 있습니다.

〔그림 11-2〕 임베딩 공간에서 유사한 문장끼리 클러스터를 이루는 시각화〕

검색 결합 생성에서 임베딩이 담당하는 역할을 다시 정리하면 이렇습니다. 문서를 작은 조각(청크, Chunk)으로 나눕니다. 각 조각을 임베딩 모델에 통과시켜 벡터로 변환합니다. 변환된 벡터를 데이터베이스에 저장합니다. 질문이 들어오면 질문도 벡터로 변환합니다. 저장된 벡터 중에서 질문 벡터와 가장 가까운 것들을 찾아 에이전트에게 전달합니다.

Google Gemini 멀티모달 임베딩 활용

여기까지는 텍스트에 대한 이야기였습니다. 그런데 현실의 데이터는 텍스트만이 아닙니다. 매뉴얼에는 분해 도면이 있고, 현장 보고서에는 사진이 첨부되고, 교육 자료에는 영상이 포함됩니다.

Google이 출시한 Gemini 임베딩 2(Gemini Embedding 2)는 텍스트, 이미지, 영상, 오디오를 모두 같은 벡터 공간에 배치할 수 있는 멀티모달 임베딩 모델(Multimodal Embedding Model)입니다.

이 모델이 가져오는 변화를 구체적인 사례로 살펴봅니다.

청소기 매뉴얼 사례

— 68페이지 PDF를 통째로 임베딩합니다. 텍스트 조각뿐 아니라 다이어그램 이미지도 벡터로 변환됩니다. "필터 청소 방법"을 질문하면 텍스트 설명과 함께 해당 다이어그램 이미지가 반환됩니다. 텍스트만으로는 파악하기 어려운 부품 위치를 이미지로 확인할 수 있습니다.

지붕 수리 업체 사례

— 과거 시공 사진 13장을 임베딩합니다. 각 사진에는 비용, 소요 기간, 투입 인원 같은 메타데이터(Metadata)가 연결되어 있습니다. 새로운 지붕 사진을 업로드하면, 유사한 과거 시공 사례 다섯 건이 유사도 점수와 함께 반환됩니다. 견적 산출의 참고 자료로 쓸 수 있습니다.

〔그림 11-3〕 멀티모달 임베딩 공간: 텍스트, 이미지, 영상이 의미 기반으로 배치된 2D 시각화〕

멀티모달 임베딩의 진가는 서로 다른 유형의 데이터가 같은 공간에 공존한다는 점입니다. 스마일리 감자튀김 사진은 "음식" 범주에, 기타 치는 강아지 영상은 "엔터테인먼트" 범주에, 클로드 코드 사용 영상은 "기술" 범주에 각각 배치됩니다. 데이터 유형이 달라도, AI가 의미를 파악하여 올바른 위치에 놓습니다. 모든 데이터가 지붕 사진이라면, 수해 피해·노후화·구조적 결함 같은 세부 범주로 자동 분류됩니다.

현재 기준으로 영상은 120초 이내의 MP4 또는 MOV 파일을 지원하며, 이미지는 요청당 최대 6장의 PNG 또는 JPEG를 처리할 수 있습니다. 오디오도 지원되며, 오디오의 경우 정확한 설명 메타데이터를 함께 제공하면 검색 정확도가 높아집니다.

Pinecone 의미 기반 데이터 저장소 연동 실습

임베딩된 벡터를 어딘가에 저장하고 검색할 수 있어야 합니다. 그 저장소가 의미 기반 데이터 저장소(Vector Database)이며, Pinecone은 가장 널리 쓰이는 의미 기반 데이터 저장소 서비스 중 하나입니다.

실습의 전체 흐름은 다음과 같습니다.

1단계: 환경 준비

VS Code에서 새 폴더를 만들고 클로드 코드를 엽니다. 계획 모드(Plan Mode)로 전환한 뒤, 에이전트에게 이렇게 지시합니다.

"Google Gemini의 새 임베딩 2 모델을 사용하여 Pinecone 의미 기반 데이터 저장소에 텍스트, 이미지, 영상을 저장하고 검색하는 시스템을 만들어 줘. .env 파일에 Pinecone API 키, Gemini API 키, OpenRouter API 키 자리를 만들어 줘."

에이전트는 프로젝트 구조를 설계하고, 의존성 목록을 정리하고, 단계별 계획을 제시합니다.

세 개의 API 키가 필요합니다. Pinecone은 벡터 저장소 접근을 위해, Gemini는 임베딩 모델 호출을 위해, OpenRouter는 채팅 모델(답변 생성용)을 위해 사용됩니다. Pinecone은 무료 스타터 플랜으로 시작할 수 있고, Gemini API 키는 Google AI Studio에서 발급받으며, OpenRouter는 여러 모델을 하나의 API로 접근할 수 있게 해 주는 서비스입니다.

2단계: 데이터 임베딩

.env

파일에 세 개의 키를 입력하고 저장합니다.

data

폴더에 임베딩할 파일들을 넣습니다. 텍스트 파일, 이미지, 영상을 섞어 넣어도 됩니다. 에이전트에게 "데이터가 준비됐으니 Pinecone에 넣어 줘"라고 말하면, 에이전트가 Pinecone에 인덱스를 생성하고 각 파일을 임베딩하여 저장합니다.

이 과정에서 에이전트가 하는 일은, 각 파일의 유형을 자동 인식하고 적절한 임베딩 방식을 적용하는 것입니다. 텍스트는 청크로 나누어 임베딩하고, 이미지는 시각적 의미를 벡터로 변환하며, 영상은 프레임과 오디오를 분석하여 벡터로 변환합니다.

〔그림 11-4〕 실습 파이프라인: 원본 파일 → 임베딩 모델 → Pinecone 인덱스〕

3단계: 채팅 인터페이스 구축

에이전트에게 "내 컴퓨터에서 검증할 수 있는 채팅 웹 앱을 만들어 줘"라고 요청합니다. 에이전트는 웹 애플리케이션을 구축하고 내 컴퓨터호스트(localhost)에서 실행합니다. 브라우저에서 질문을 입력하면 Pinecone에서 관련 벡터를 검색하고, 검색 결과를 바탕으로 답변을 생성합니다.

실제 검증에서 "워크플로우 클라이언트를 어떻게 확보해야 할까?"라고 물으면, 텍스트 파일에서 관련 내용을 찾아 답변합니다. "기타 치는 골든 리트리버 영상 보여줘"라고 요청하면, 해당 영상의 메타데이터를 찾아 인라인으로 영상을 재생합니다.

4단계: 반복 개선

첫 번째 결과가 완벽하지 않을 수 있습니다. 이미지가 반환되지 않거나, 영상 설명이 부족할 수 있습니다. 이때 에이전트에게 문제를 설명하면, 메타데이터를 보강하거나 앱을 수정합니다. "영상에 더 나은 설명을 달아서 다시 임베딩해 줘"라고 요청하면, 기존 벡터를 삭제하고 개선된 메타데이터와 함께 새로 저장합니다.

이 전체 과정이 30분 안에 이루어집니다. n8n 같은 코드 없이 쓰는 도구에서 동일한 멀티모달 벡터 저장소를 구축하려면 수 시간에서 수 일이 걸리는 작업입니다. 청킹(Chunking) 전략 설계, 이미지 캡처와 저장 방식 설정, 검색 결과 포맷팅을 모두 수동으로 구성해야 하기 때문입니다. 클로드 코드는 이 모든 과정을 자연어 지시만으로 처리합니다.

검색 결합 생성가 에이전트 작업 흐름에 가져오는 변화

검색 결합 생성가 없는 에이전트는 자신이 학습한 범위 안에서만 답합니다. 한 번에 읽는 범위에 들어오는 정보만 참고합니다. 그 바깥의 세계는 존재하지 않는 것과 같습니다.

검색 결합 생성를 장착한 에이전트는 다릅니다. 회사의 68페이지 매뉴얼을 읽을 수 있습니다. 수백 장의 시공 사진 아카이브를 검색할 수 있습니다. 지난 분기의 회의록에서 특정 결정의 맥락을 찾아낼 수 있습니다. 에이전트의 지식 범위가 학습 데이터의 경계를 넘어, 조직이 보유한 모든 데이터로 확장됩니다.

에이전트 작업 흐름에서 검색 결합 생성는 여러 시나리오로 활용됩니다.

고객 지원 자동화

— 제품 매뉴얼과 FAQ를 임베딩하여, 고객 질문에 대해 정확한 페이지와 이미지를 참조한 답변을 생성합니다. "이전에 비슷한 문의가 있었나요?"라는 질문에 과거 티켓 기록을 검색하여 답할 수도 있습니다.

내부 지식 관리

— 팀의 프로젝트 문서, 의사결정 로그, 브랜드 가이드라인을 임베딩합니다. 신규 팀원이 "우리 회사의 로고 사용 규칙이 뭐야?"라고 물으면, 브랜드 가이드라인에서 해당 섹션을 찾아 답변합니다.

연구 보조

— 논문, 보고서, 시장 조사 자료를 임베딩합니다. "이 분야의 최근 동향이 뭐야?"라고 물으면, 관련 자료를 검색하여 요약본을 생성합니다. 원본 출처와 신뢰도 점수를 함께 제공하므로, 사실 확인이 가능합니다.

〔그림 11-5〕 검색 결합 생성 적용 전후 에이전트 지식 범위 비교 다이어그램〕

여기서 중요한 것은 주제 전문성(Subject Matter Expertise)의 역할입니다. 검색 결합 생성 파이프라인을 구축하는 기술적 능력보다, 어떤 데이터를 어떻게 설명하느냐가 결과 품질을 좌우합니다. 지붕 수리 사례에서 보았듯이, 사진에 달리는 메타데이터가 빈약하면 검색 결과도 빈약합니다.

"이 사진은 10년 된 아스팔트 싱글 지붕의 우박 피해 사례이며, 수리 비용은 450만 원, 소요 기간은 3일이었다"라는 설명이 달린 사진과, "지붕 사진"이라고만 태그된 사진의 검색 유용성은 차원이 다릅니다.

기술 구현 역량의 가치는 줄어들고 있습니다. n8n에서 수 일 걸리던 파이프라인을 클로드 코드가 30분에 구축하는 것을 보았습니다. JSON 본문을 구성하고 HTTP 요청을 설정하는 기술적 세부사항은 에이전트가 처리합니다. 반면 프로세스를 명확하게 설명하는 능력, 데이터의 의미를 정확하게 기술하는 능력, 빈틈을 발견하고 명시적으로 지적하는 능력은 여전히 사람의 영역입니다.

에이전트에게 장기 기억을 부여하는 것은 데이터베이스를 연결하는 일이었습니다. 그렇다면 에이전트에게 반복적으로 수행할 수 있는 숙련된 기술을 부여하는 방법을 살펴보겠습니다. 한 번 가르쳐 놓으면 언제든 동일한 품질로 실행할 수 있는, 재사용 가능한 행동 양식 말입니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

12장 스킬: 재사용 가능한 워크플로우 레시피

전체 목차

책의 모든 장 보기

도입

아침 7시, 클로드 코드에 "모닝 커피 스킬 실행해 줘"라고 입력합니다. 에이전트는 캘린더를 확인하고, 프로젝트 관리 도구에서 이번 주 태스크를 가져오고, 분기 목표 진행률을 점검한 뒤, 오늘 하루의 시간표를 제안합니다. 동시에 다른 창에서는 "최근 유튜브 댓글 분석해 줘"라고 보냈고, 또 다른 창에서는 "팀 펄스 체크 돌려 줘"라고 보냈습니다.

세 개의 에이전트가 병렬로 작동합니다. 각각이 서로 다른 스킬을 실행하고 있습니다. 이 모든 것을 입력하는 데 걸린 시간은 1분 남짓이고, 수동으로 했다면 최소 25분은 소요되었을 작업들입니다.

이것이 스킬(Skill)의 세계입니다.

스킬과 워크플로우의 관계

앞서 WAT 프레임워크에서 워크플로우(Workflow)를 다루었습니다. 마크다운으로 작성된 작업 지시서, 에이전트가 읽고 실행하는 SOP(Standard Operating Procedure). 스킬은 이 워크플로우와 본질적으로 같은 것입니다.

이름만 다릅니다. 워크플로우라는 단어가 "일회성 작업 흐름"의 느낌을 준다면, 스킬은 "반복적으로 사용할 수 있는 숙련된 기술"이라는 뉘앙스를 가집니다. 워크플로우에는 도구(Tool)로서 파이썬 스크립트가 붙었습니다. 스킬에도 파이썬 스크립트가 붙습니다. 워크플로우는 마크다운 파일이었고, 스킬도 마크다운 파일입니다. 구조가 동일합니다.

차이가 있다면 스킬에는

YAML 프론트매터(YAML Front Matter)

가 추가된다는 점입니다. 파일 상단에

로 감싸인 영역이 있고, 그 안에 스킬의 이름(name)과 설명(description)이 기록됩니다. 이 프론트매터가 에이전트의 스킬 탐색 효율을 극적으로 높입니다. 에이전트는 모든 스킬 파일의 전체 내용을 읽지 않습니다.

프론트매터의 이름과 설명만 훑어보고, 현재 요청에 맞는 스킬을 골라낸 뒤, 선택된 스킬의 본문만 상세히 읽습니다. 이를

점진적 컨텍스트 로딩(Progressive Context Loading)

이라 합니다.

name: morning-coffee

description: 캘린더, 프로젝트 현황, 분기 목표를 종합하여 오늘의 일정을 계획합니다.

모닝 커피 스킬

언제 사용하는가

사용자가 하루 계획을 세우고 싶을 때, 아침 루틴을 시작할 때

실행 절차

1. 오늘의 캘린더 이벤트를 확인한다

2. ClickUp에서 이번 주 태스크를 가져온다

…

스킬을 워크플로우의 진화형이라고 볼 수도 있고, 같은 개념에 대한 다른 이름이라고 볼 수도 있습니다. 중요한 것은 둘 다 "에이전트에게 주는 자연어 지시서"이며, 사람이 읽을 수 있고 사람이 수정할 수 있다는 점입니다. 프로그래밍 언어가 아닌 일상 문장으로 에이전트의 행동을 정의합니다.

.claude/skills 디렉터리에 스킬 파일 만들기

스킬 파일의 물리적 위치는 프로젝트의

.claude/skills/

디렉터리입니다. 각 스킬은 하위 폴더로 구분되며, 폴더 안에

skill.md

파일이 핵심입니다.

프로젝트/

└── .claude/

└── skills/

├── morning-coffee/

│ └── skill.md

├── research/

│ ├── skill.md

│ └── scripts/

```
│   └── perplexity_search.py
├── infographic-builder/
│   ├── skill.md
│   ├── scripts/
│   │   └── overlay_logo.py
│   └── references/
│   └── api_reference.md
└── excalidraw-diagram/
└── skill.md
```

〔그림 12-1〕.claude/skills 디렉터리 구조 예시〕

스킬 폴더 안에는

skill.md

외에 보조 파일들이 들어갈 수 있습니다. 파이썬 스크립트를

scripts/

폴더에, 참조 문서를

references/

폴더에 넣는 식입니다. 이렇게 스킬 내부에 모든 것을 담는 방식을

자체 포함(Self-Contained)

구조라 합니다.

다른 방법도 있습니다. 스크립트나 참조 파일이 프로젝트의 다른 위치에 있을 수 있습니다. 예를 들어 유튜브 분석 스킬의

skill.md

에서

../../references/youtube_channel.md

를 참조하도록 경로를 지정하면, 파일이 스킬 폴더 바깥에 있어도 문제없습니다. 에이전트는

skill.md

에 기록된 경로를 따라가서 필요한 파일을 읽습니다.

이를

외부 참조(External Reference)

구조라 합니다.

어느 쪽이든, 핵심 원칙은 하나입니다.

skill.md

안에 올바른 경로가 기록되어 있으면 됩니다.

스킬을 처음 만드는 과정은 직접 마크다운 파일을 작성하는 것이 아닙니다. 에이전트와의 대화를 통해 만듭니다. 계획 모드에서 "리서치 스킬을 만들어 줘. Perplexity API를 사용하고, .env에 API 키 자리를 만들어 줘"라고 요청하면, 에이전트가 폴더 구조를 설계하고,

skill.md

를 작성하고, 필요한 스크립트를 생성하고,

CLAUDE.md

를 업데이트하여 새 스킬의 존재를 등록합니다.

왜 이 과정을 수동이 아닌 에이전트에게 맡기는 것이 나을까요? 에이전트는 프론트매터 형식, 파일 경로 규칙, 모델이 세는 글 조각 효율적인 구조 같은 모범 사례를 이미 알고 있기 때문입니다. 다만, 에이전트가 항상 완벽한 구조를 만들지는 않습니다. YAML 프론트매터 없이 순수 마크다운으로만 스킬을 생성하는 경우가 있습니다.

이때는 직접 프론트매터를 추가하거나, Anthropic의 공식 문서 URL을 에이전트에게 주고 "이 문서를 참고해서 모범 사례대로 만들어 줘"라고 지시하면 됩니다.

skill.md

의 권장 분량은 500줄 이하입니다. 상세한 참조 자료는 별도 파일로 분리하는 것이 모델이 세는 글 조각 절약에 유리합니다.

스킬 설명문 작성법

스킬이 올바르게 작동하려면 에이전트가 그 스킬을 찾을 수 있어야 합니다. 에이전트가 사용자의 요청을 분석한 뒤 수십 개의 스킬 중에서 적합한 것을 고르는 과정에서,

설명문(Description)

이 결정적 역할을 합니다.

점진적 컨텍스트 로딩의 첫 번째 단계에서, 에이전트는 각 스킬의 이름과 설명문만 읽습니다. 이 단계에서 소모되는 모델이 세는 글 조각은 스킬당 대략 100개 내외입니다. 설명문이 모호하면 에이전트가 잘못된 스킬을 선택하거나, 적합한 스킬이 있는데도 지나칠 수 있습니다.

좋은 설명문의 조건은 세 가지입니다.

트리거 상황을 명시합니다.

"사용자가 하루 계획을 세우거나, 모닝 커피라고 말할 때 사용"처럼, 이 스킬이 호출되어야 하는 구체적인 상황을 기술합니다. 에이전트가 사용자의 자연어 요청과 설명문을 대조하여 매칭 여부를 판단하기 때문입니다.

결과물을 명시합니다.

"캘린더, 태스크, 목표를 종합하여 시간 블록이 할당된 일정표를 출력한다"처럼, 스킬 실행의 결과물이 무엇인지 적어 둡니다. 이것이 사용자의 기대와 일치하는지를 에이전트가 확인하는 근거가 됩니다.

범위를 한정합니다.

"이 스킬은 일정 계획만 다루며, 콘텐츠 생성이나 리서치는 포함하지 않음"처럼 경계를 설정합니다. 유사한 스킬이 여러 개 있을 때, 에이전트가 혼동하지 않도록 돕습니다.

〔그림 12-2〕 점진적 컨텍스트 로딩의 3단계: 프론트매터 스캔 → 스킬 본문 로드 → 참조 파일 로드〕

스킬의 호출 방식은 두 가지입니다. 첫째,

슬래시 명령어

입니다.

/morning-coffee

라고 입력하면 해당 스킬이 바로 실행됩니다. 둘째,

자연어

입니다. "오늘 하루 계획 좀 세워 줘"라고 말하면 에이전트가 설명문을 검색하여 모닝 커피 스킬을 찾아냅니다.

슬래시 명령어는 빠르고 정확합니다. 해석 과정이 없으니 즉시 실행됩니다. 자연어는 유연합니다. 스킬 이름을 몰라도 의도만 전달하면 됩니다. 둘 다 활성화해 두는 것이 보통이지만, 스킬이 의도치 않게 너무 자주 호출된다면 YAML 프론트매터에서

disable_model_invocation: true

를 설정하여 자연어 트리거를 끄고 슬래시 명령어로만 호출되도록 제한할 수 있습니다.

스킬이 호출되지 않는 문제가 발생하면, 대부분 설명문이 원인입니다. "유용한 스킬"처럼 추상적인 설명은 어떤 요청과도 매칭되지 않습니다. "사용자가 인포그래픽 제작을 요청하거나, 시각적 콘텐츠가 필요할 때 Krea AI의 Nano Banana 모델을 호출하여 브랜드 가이드라인에 맞는 PNG 이미지를 생성한다"처럼 구체적으로 쓸수록 정확도가 올라갑니다.

GitHub에 커밋하여 팀과 공유하기

스킬이

.claude/skills/

폴더에 존재하는 한, 그 스킬은 해당 프로젝트에서만 유효합니다. 프로젝트 폴더를 깃허브(GitHub)에 푸시하면, 팀원 누구든 이 레포지터리를 클론하여 동일한 스킬을 사용할 수 있습니다.

이것이 의미하는 바를 생각해 봅니다. 한 사람이 리서치 스킬을 만들고 수십 번 반복 실행하며 다듬었습니다. 프롬프트를 조정하고, 출력 형식을 개선하고, 불필요한 API 호출을 줄이는 최적화를 거쳤습니다. 그 결과물을 깃허브에 커밋하는 순간, 팀 전원이 같은 품질의 리서치를 수행할 수 있게 됩니다. 한 사람의 학습 곡선이 팀 전체의 역량으로 전환됩니다.

커밋과 공유의 절차는 간단합니다. 프로젝트에서

git init

으로 깃 레포지터리를 초기화하거나, 에이전트에게 "이 프로젝트를 깃허브에 올려 줘"라고 요청합니다. 에이전트가 레포지터리를 생성하고, 커밋하고, 푸시합니다. 이후 스킬을 수정할 때마다 커밋을 남기면, 버전 관리가 자동으로 이루어집니다. 어떤 버전의 스킬이 더 나았는지 비교할 수 있고, 문제가 생기면 이전 버전으로 돌릴 수 있습니다.

〔그림 12-3〕 스킬의 공유 흐름: 개인 작성 → Git 커밋 → GitHub 푸시 → 팀원 클론〕

스킬의 저장 위치에 대해 한 가지 선택이 더 있습니다.

프로젝트 스킬

과

글로벌 스킬

의 구분입니다.

프로젝트 스킬은

.claude/skills/

에 위치하며, 해당 프로젝트에서만 작동합니다. 팀원과 공유할 수 있습니다. 프론트엔드 디자인 스킬이 웹 개발 프로젝트에만 필요하다면, 프로젝트 스킬로 두는 것이 맞습니다.

글로벌 스킬은 홈 디렉터리의

~/.claude/skills/

에 위치하며, 어떤 프로젝트를 열든 사용 가능합니다. 팀원과 공유되지 않습니다. 나만 사용하는 개인 스킬, 혹은 모든 프로젝트에서 공통으로 쓰이는 스킬이 여기에 해당합니다. 프론트엔드 디자인 스킬이 어떤 프로젝트에서든 필요하다면, 글로벌에 두는 편이 편리합니다.

스킬의 진정한 가치는 완성된 스킬 하나에 있지 않습니다. 수정 의견 루프(Feedback Loop)에 있습니다. 스킬을 실행하고, 에이전트의 작업 과정을 관찰하고, "이 부분은 불필요한 API 호출이었어", "출력 형식이 마음에 안 들어"라고 수정 의견하고, 에이전트가

skill.md

를 수정합니다.

이 순환을 열 번, 스무 번 반복하면, 처음에 어색했던 스킬이 자신의 업무 방식에 꼭 맞는 도구로 변모합니다. 에이전트에게 완벽한 스킬을 처음부터 기대하는 것은 비현실적입니다. 완벽은 반복에서 나옵니다.

스킬의 최적화 과정에서 발견하게 되는 패턴이 있습니다. 에이전트가 매번 같은 데이터를 검색하느라 시간을 소모한다면, 자주 참조하는 ID나 경로를

skill.md

에 하드코딩합니다. 에이전트가 한 번에 읽는 범위를 너무 많이 소모한다면, 무거운 작업을 보조 에이전트에게 위임하도록 스킬에 지시를 추가합니다.

에이전트가 외부 API 문서를 매번 검색한다면, 그 문서를 웹 정보 수집하여 참조 마크다운 파일로 저장합니다. 마크다운 파일을 읽는 것이 API 문서를 크롤링하는 것보다 훨씬 빠르고 저렴합니다.

이 모든 최적화의 방향은 하나입니다. 스킬을 실행한 뒤 다른 일을 하다가 10분 후 돌아와서 완성된 결과물을 확인하는 것. 그 경지에 도달하면, 한 사람이 네 개의 에이전트를 동시에 굴리며 하루치 업무를 오전 중에 끝내는 풍경이 비현실이 아닌 일상이 됩니다.

그리고 이 스킬들이 서로 연결되고, 보조 에이전트가 서로를 호출하고, 외부 서비스가 MCP를 통해 에이전트와 대화하기 시작하면, 개인 비서 수준을 넘어선 무언가가 만들어집니다.

전체 목차

책의 모든 장 보기

13장 웹사이트 클론과 커스터마이징

전체 목차

책의 모든 장 보기

참조 사이트를 스크린샷으로 분석하기

프랑스어로 작성된 어느 커뮤니티 사이트가 화면에 떠 있습니다. 배색이 세련되고, 레이아웃의 균형이 눈에 들어옵니다. 직접 만들고 싶은 웹사이트의 완성 이미지가 머릿속에 그려지는 순간, 손이 먼저 키보드로 향하는 것은 자연스러운 반응입니다. 그러나 코드를 한 줄이라도 작성하기 전에 해야 할 일이 있습니다. 참조 사이트의 구조를 정확히 파악하는 것입니다.

클로드 코드에게 웹사이트를 만들어 달라고 요청할 때, 말로 설명하는 것과 시각 자료를 제공하는 것 사이에는 결과물의 품질 차이가 뚜렷하게 나타납니다. "애플 홈페이지처럼 만들어 줘"라고 말하면 AI는 자기가 학습한 데이터에서 애플 사이트의 일반적인 인상을 끌어옵니다. 그러나 참조 사이트의 스크린샷(screenshot)을 직접 전달하면 이야기가 달라집니다.

색상 코드, 여백의 비율, 버튼의 위치, 타이포그래피(typography)의 크기 관계까지 시각 정보에서 구체적으로 읽어낼 수 있기 때문입니다.

실무에서 참조 사이트를 분석하는 절차는 다음과 같습니다.

먼저 참조 사이트의 전체 페이지 스크린샷을 확보합니다. 브라우저의 전체 페이지 캡처 기능을 사용하면 스크롤 아래에 숨어 있는 영역까지 한 장의 이미지로 담을 수 있습니다. 이 스크린샷은 클로드 코드가 "목표 상태"를 인식하는 기준점이 됩니다.

다음으로 브랜드 에셋(brand asset)을 준비합니다. 로고 파일, 주요 색상 코드(프라이머리, 세컨더리, 배경색, 강조색), 사용할 서체 이름을 정리해 둡니다. 참조 사이트의 레이아웃은 빌려오되 브랜드 아이덴티티는 자신의 것으로 교체해야 하기 때문입니다.

〔그림 13-1〕 참조 사이트 스크린샷과 브랜드 에셋을 함께 전달한 프롬프트 구성 예시

그리고 클로드 코드에게 프롬프트를 작성합니다. "이 스크린샷의 레이아웃과 구조를 참조해서, 우리 브랜드 색상과 로고를 적용한 웹사이트를 만들어 줘. 텍스트는 모두 영어로, 우리 커뮤니티의 맥락에 맞게 작성해 줘." 이렇게 요청하면 클로드 코드는 프랑스어 원문 사이트의 구조를 따르되, 색상과 서체와 콘텐츠는 완전히 새로운 것으로 채워 넣습니다.

한 가지 주의할 점이 있습니다. 참조 사이트를 "복제"하는 것과 "참조"하는 것은 다릅니다. HTML 소스 코드를 그대로 가져오는 것이 아니라, 디자인 패턴과 레이아웃의 흐름을 학습 자료로 활용하는 것입니다. 결과물에는 자신만의 콘텐츠, 자신만의 색상, 자신만의 메시지가 담겨야 합니다.

스크린샷 루프: 구축→캡처→비교→수정 반복 전략

첫 번째 구축가 끝나고 내 컴퓨터호스트(localhost)에 웹사이트가 나타났습니다. 색상은 지정한 대로 들어갔고, 로고도 제자리에 있습니다. 그런데 히어로 섹션(hero section)의 여백이 참조 사이트보다

넓고, 카드 레이아웃의 정렬이 미묘하게 어긋나 있습니다. 이 차이를 어떻게 좁힐 수 있을까요?

스크린샷 루프(screenshot loop)라는 전략이 여기서 등장합니다. 이 전략의 핵심은 AI가 자기 자신의 결과물을 눈으로 확인하고, 참조 이미지와 비교한 뒤, 차이점을 스스로 수정하는 반복 과정입니다.

작동 방식은 이렇습니다.

클로드 코드가 웹사이트 코드를 작성합니다. 내 컴퓨터 서버를 띄운 뒤, 자체적으로 스크린샷을 캡처합니다. 캡처한 이미지를 참조 스크린샷과 섹션별로 비교합니다. 불일치 항목을 목록화하고, 코드를 수정합니다. 다시 스크린샷을 찍어 비교합니다. 이 과정을 최소 두 차례 반복한 뒤에야 사용자에게 결과를 보여줍니다.

〔그림 13-2〕 스크린샷 루프의 흐름도: 빌드 → 캡처 → 비교 → 수정 → 재캡처

이 루프가 강력한 이유는 AI가 코드만 보는 것이 아니라 렌더링된 시각적 결과물을 판단 기준으로 삼기 때문입니다. CSS에서

margin-top: 2rem

이라고 적었을 때 실제 화면에서 그 여백이 어떤 느낌을 주는지는 코드만으로는 알 수 없습니다. 스크린샷을 통해 "이 부분이 참조 사이트보다 20픽셀 정도 넓다"는 판단이 가능해집니다.

그러나 스크린샷 루프에도 한계가 있습니다. 동적 요소에서 문제가 발생합니다.

배경 애니메이션이나 스크롤 기반 효과처럼 정지 화면으로 포착할 수 없는 요소가 있을 때, 스크린샷은 그 요소의 일부만 담거나 아예 빈 화면을 보여줍니다. 이때 클로드 코드는 "아직 제대로 구현되지 않았다"고 오판하고, 이미 잘 작동하는 코드를 과도하게 수정하는 악순환에 빠질 수 있습니다.

이런 상황에서는 스크린샷 루프를 의도적으로 끄는 것이 정답입니다. 프롬프트에 "이것은 애니메이션 배경이므로 스크린샷 도구를 사용하지 말고 코드만 작성해 줘. 수정이 필요하면 내가 직접 확인한 뒤 알려줄게"라고 명시합니다. AI에게 시각적 자가 점검 권한을 일시적으로 회수하는 셈입니다.

반대로, 정적 레이아웃이 중심인 랜딩 페이지에서는 스크린샷 루프를 최대한 활용하는 것이 좋습니다. 루프 횟수를 세 차례 이상으로 설정하면, 첫 번째 결과물과 세 번째 결과물 사이에 눈에 띄는 품질 차이가 나타납니다.

〔그림 13-3〕 스크린샷 루프 1회차 vs 3회차 결과물 비교

21st.dev 컴포넌트 라이브러리 활용

전체 레이아웃이 만족스러운 상태에서, 히어로 섹션의 버튼이 밋밋하게 느껴집니다. 둥근 모서리의 파란 버튼 하나가 덩그러니 놓여 있을 뿐입니다. 이 버튼 하나를 무지개빛 테두리가 흐르는 버튼으로 바꿀 수 있다면, 사이트 전체의 인상이 달라질 것입니다.

21st.dev는 웹사이트의 개별 컴포넌트(component)를 모아 놓은 라이브러리입니다. 버튼, 배경 애니메이션, 네비게이션 바, 카드 UI, 토글 스위치 등 다양한 요소가 미리 코드화되어 있습니다. 사용자는 원하는 컴포넌트를 골라 프롬프트 형태로 복사한 뒤, 클로드 코드에게 전달하면 됩니다.

작업 흐름은 이렇습니다.

21st.dev 사이트에 접속해서 카테고리를 탐색합니다. 버튼, 배경, 히어로 섹션, 셰이더(shader) 효과 등 다양한 분류가 있습니다. 원하는 컴포넌트를 선택하면, 해당 컴포넌트의 프롬프트 코드가 생성됩니다. 이 코드를 복사합니다.

클로드 코드로 돌아와서 프롬프트를 구성합니다. "히어로 텍스트 뒤에 이 배경 요소를 적용해 줘"라고 요청하고, 복사한 코드를 붙여넣습니다. 클로드 코드는 이 코드의 구조를 파악하고, 기존 프로젝트의 맥락에 맞게 통합합니다.

〔그림 13-4〕 21st.dev에서 배경 애니메이션 컴포넌트를 선택하는 화면

이 방식의 장점은 전체 사이트를 하나의 참조에서 가져오는 대신, 여러 출처의 개별 요소를 조합할 수 있다는 점입니다. 버튼은 A 사이트에서 영감을 받고, 배경은 21st.dev의 웨이브 애니메이션을 사용하고, 카드 레이아웃은 B 사이트의 구조를 참조하는 식입니다. 하나의 사이트를 통째로 모방하는 것이 아니라, 다양한 요소를 큐레이션해서 고유한 결과물을 만들어 내는 것입니다.

주의할 점이 하나 있습니다. 21st.dev에서 가져온 컴포넌트 중 상당수가 애니메이션을 포함하고 있습니다. 앞서 스크린샷 루프의 한계에서 다뤘듯이, 동적 요소를 추가할 때는 스크린샷 비교를 비활성화하는 것이 안전합니다. "이것은 애니메이션 배경이므로 스크린샷 도구로 비교하지 말고, 코드를 직접 작성한 뒤 내가 확인하겠다"는 지시를 함께 전달해야 합니다.

그렇지 않으면 클로드 코드가 정지 화면에서 애니메이션의 중간 프레임만 포착하고, "구현이 불완전하다"고 판단해서 과도한 수정을 반복하는 상황이 벌어집니다.

컴포넌트를 적용한 뒤에는 반드시 반복적인 미세 조정이 필요합니다. "배경 애니메이션이 너무 산만하다. 히어로 텍스트 뒤에 반투명 배경을 깔아서 가독성을 높여 줘. 'Earn More' 텍스트의 색상을 오렌지에서 파란색으로 바꿔 줘." 이런 수정 의견을 하나씩 전달하면서 다듬어 가는 과정이 완성도를 결정합니다.

〔그림 13-5〕 컴포넌트 적용 전후 히어로 섹션 비교

바이패스 권한 모드와 자율 구축의 장단점

클로드 코드가 코드를 작성하다가 멈춥니다. "이 명령을 실행해도 될까요?"라는 승인 요청이 화면에 뜹니다. 파일을 생성할 때, 패키지를 설치할 때, 서버를 시작할 때, 매번 사용자의 허락을 구합니다. 안전한 설계이지만, 웹사이트를 구축하는 과정에서 이런 중단이 수십 차례 반복되면 작업 흐름이 끊깁니다.

바이패스 권한 모드(bypass permissions mode)는 이 중단을 제거하는 설정입니다. VS Code의 설정에서 "Claude Code"를 검색한 뒤 "Allow Dangerously Skip Permissions" 옵션을 활성화하면, 클로드 코드가 별도의 승인 없이 명령을 연속 실행합니다.

이 모드의 이점은 분명합니다. 스크린샷 루프 전체가 자동으로 돌아갑니다. 코드 작성, 서버 시작, 스크린샷 캡처, 비교, 수정, 재시작이라는 여섯 단계가 사용자의 개입 없이 연속으로 진행됩니다. 할 일 목록(to-do list)을 만들고 하나씩 체크해 가며, 완료된 결과물만 사용자에게 보여줍니다.

그러나 이 모드의 이름에 "Dangerously"가 붙어 있는 데는 이유가 있습니다. 클로드 코드가 어떤 명령이든 실행할 수 있다는 뜻이기 때문입니다. 파일 삭제, 시스템 설정 변경, 외부 네트워크 요청 등이 승인 없이 일어날 수 있습니다.

실용적인 관점에서 위험을 관리하는 방법은 이렇습니다.

바이패스 권한 모드를 켠 상태에서 작업할 때, 화면을 완전히 떠나지 않는 것이 원칙입니다. 옆에서 다른 작업을 하면서 간간이 클로드 코드의 진행 상황을 확인하는 정도면 충분합니다. 밤새 자율 구축를 시키고 잠자리에 드는 것은 권장하지 않습니다.

더 정교한 방법도 있습니다. 바이패스 권한 모드 대신, 권한 설정(permissions configuration)에서 안전한 명령만 허용 목록(allow list)에 넣고, 위험한 명령은 거부 목록(deny list)에 넣는 것입니다. 예를 들어 파일 생성과 읽기는 허용하되,

rm -rf

같은 삭제 명령은 차단합니다.

거부 목록에 있는 명령은 허용 목록보다 우선순위가 높으므로, 실수로 위험한 작업이 실행되는 것을 방지할 수 있습니다. 이 방식을 사용하면 바이패스 권한 모드와 거의 같은 속도를 얻으면서도 안전망을 유지할 수 있습니다.

〔그림 13-6〕 바이패스 권한 모드 vs 선택적 권한 설정 비교표

항목

바이패스 권한 모드

선택적 권한 설정

설정 난이도

토글 한 번

명령별 분류 필요

작업 속도

최고

거의 동일

안전성

낮음

높음

적합한 상황

민감 데이터 없는 프로젝트

API 키, 개인정보 포함 프로젝트

웹사이트를 처음 구축하는 단계에서는 민감한 데이터가 포함되지 않는 경우가 많으므로, 바이패스 권한 모드를 사용해도 실질적인 위험은 크지 않습니다. 그러나 프로젝트에 API 키나 데이터베이스 연결 정보가 포함되기 시작하면, 선택적 권한 설정으로 전환하는 것이 현명합니다.

완성된 웹사이트가 내 컴퓨터호스트에서 매끄럽게 돌아가고 있습니다. 색상과 레이아웃이 마음에 들고, 애니메이션도 자연스럽습니다. 이제 이 결과물을 내 컴퓨터 안에만 가둬 둘 수 없습니다. 다른 사람도 접속할 수 있는 곳에 올려야 합니다. 그 과정에서 코드의 변경 이력을 안전하게 관리하는 도구가 필요합니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

14장 깃과 깃허브: 프로젝트의 시간 여행 장치

전체 목차

책의 모든 장 보기

깃의 핵심 개념: 리포지토리, 커밋, 브랜치, 푸시, 풀, 머지

새벽 두 시, 웹사이트의 히어로 섹션을 수정하던 중 무언가 잘못됐습니다. 버튼의 글로우 효과를 바꾸려다가 전체 레이아웃이 무너져 버렸습니다. 세 시간 전의 상태로 돌아가고 싶지만, 그때의 코드가 어디에도 남아 있지 않습니다. Ctrl+Z를 수십 번 눌러도 이미 저장해 버린 파일은 되돌릴 수 없습니다.

깃(Git)은 이 문제를 해결하기 위해 만들어진 버전 관리 시스템(version control system)입니다. 프로젝트에 일어나는 모든 변경 사항을 추적하고, 원하는 시점으로 되돌릴 수 있게 해 줍니다. 비디오 게임의 체크포인트 저장과 비슷합니다. 의미 있는 변경을 할 때마다 스냅샷을 찍어 두면, 언제든 그 지점으로 되감을 수 있습니다.

깃의 세계를 구성하는 핵심 개념을 하나씩 살펴보겠습니다.

리포지토리(repository)

는 깃이 감시하고 있는 폴더입니다. 프로젝트의 모든 파일과 그 파일들에 일어난 변경의 전체 역사가 이 안에 담겨 있습니다. 은행 계좌에 비유할 수 있습니다. 잔액뿐 아니라 모든 입출금 내역이 기록되어 있는 것처럼, 리포지토리에는 현재 코드뿐 아니라 과거의 모든 변경 기록이 보존됩니다.

커밋(commit)

은 프로젝트의 특정 시점 스냅샷입니다. 각 커밋에는 무엇이 바뀌었는지와 왜 바꿨는지가 함께 기록됩니다. 워드 문서에서 "저장"을 누르는 것과 비슷하지만, 차이가 있습니다. 워드의 저장은 이전 상태를 덮어쓰지만, 깃의 커밋은 이전 상태를 그대로 남겨둔 채 새로운 스냅샷을 추가합니다.

"히어로 버튼에 글로우 효과 추가"라는 메시지와 함께 커밋하면, 나중에 이 커밋만 정확히 되돌릴 수 있습니다.

브랜치(branch)

는 프로젝트의 별도 사본으로, 메인 버전에 영향을 주지 않고 실험할 수 있는 공간입니다. 기본 브랜치는 보통 메인(main)이라고 부릅니다. "feature-login"이라는 브랜치를 만들면, 로그인 기능을 실험하되 안정 버전은 건드리지 않겠다는 선언입니다.

푸시(push)

는 내 컴퓨터 컴퓨터의 변경 사항을 원격 저장소로 보내는 행위입니다. 내 컴퓨터에서만 존재하던 커밋이 클라우드에 올라갑니다.

풀(pull)

은 그 반대입니다. 원격 저장소의 최신 변경 사항을 내 컴퓨터로 가져옵니다.

머지(merge)

는 두 브랜치의 변경 사항을 하나로 합치는 작업입니다. 실험이 성공적이었다면, 실험 브랜치를 메인 브랜치에 머지해서 안정 버전에 반영합니다.

〔그림 14-1〕 깃의 핵심 개념 관계도: 리포지토리 안에서 브랜치, 커밋, 머지가 어떻게 연결되는지

열차 궤도의 비유: 안정 버전을 보호하며 실험하는 방법

기차역을 떠올려 보겠습니다. 본선 궤도 위를 열차가 달리고 있습니다. 이 열차가 현재 운행 중인 안정 버전, 즉 메인 브랜치입니다. 승객을 태우고 있으므로, 궤도 위에서 레일을 뜯어내고 새로운 레일을 깔 수는 없습니다.

새로운 노선을 시험하고 싶다면 어떻게 해야 합니까? 본선에서 갈라져 나오는 시험 궤도를 만듭니다. 이 시험 궤도가 바로 브랜치입니다. 시험 궤도 위에서는 자유롭게 실험할 수 있습니다. 레일의 각도를 바꿔 보기도 하고, 새로운 정차역을 추가해 보기도 합니다. 본선의 열차 운행에는 아무런 영향이 없습니다.

시험이 끝나고 결과가 만족스러우면, 시험 궤도를 본선에 합류시킵니다. 이것이 머지입니다. 합류 지점에서 두 궤도가 자연스럽게 만나야 하므로, 충돌(conflict)이 없는지 확인하는 절차가 필요합니다. 만약 본선에서도 같은 구간을 수정했고 시험 궤도에서도 같은 구간을 수정했다면, 어떤 쪽의 변경을 채택할지 사람이 결정해야 합니다.

〔그림 14-2〕 열차 궤도 비유: 메인 브랜치(본선)에서 기능 브랜치(시험 궤도)가 분기되고 다시 합류하는 모습

이 비유가 실제 웹사이트 개발에서 어떻게 적용되는지 살펴보겠습니다.

앞 장에서 만든 웹사이트가 이미 배포되어 실제 사용자가 방문하고 있다고 가정합니다. 히어로 섹션의 버튼을 더 역동적으로 바꾸고 싶습니다. 메인 브랜치의 코드를 직접 수정하면, 변경 중에 사이트가 깨질 위험이 있습니다. 그 대신

feature-dynamic-button

이라는 브랜치를 만들고, 그 안에서 버튼 코드를 수정합니다. 내 컴퓨터호스트에서 검증하고, 결과가 좋으면 메인 브랜치에 머지합니다.

클로드 코드에게 "이 변경 사항이 마음에 드니까 깃허브에 푸시해 줘"라고 말하기 전까지, 변경은 내 컴퓨터 브랜치 안에만 머뭅니다. 라이브 사이트에는 아무런 영향이 없습니다. 이것이 브랜치를 사용하는 근본적인 이유입니다. 안정된 것을 보호하면서 새로운 것을 시도할 수 있는 것입니다.

워크트리(worktree)라는 개념은 이 원리를 한 단계 더 확장합니다. 일반적으로 깃에서는 한 폴더에서 하나의 브랜치만 활성화할 수 있습니다. 다른 브랜치로 전환하려면 현재 작업을 저장하거나 커밋해야 합니다. 워크트리를 사용하면 같은 리포지토리에서 여러 브랜치를 동시에 다른 폴더에 체크아웃(checkout)할 수 있습니다.

클로드 코드에는 워크트리를 위한 내장 명령이 있어서,

--worktree

옵션으로 격리된 작업 공간을 빠르게 만들 수 있습니다.

여러 클로드 코드 세션을 동시에 실행하면서, 각 세션이 서로 다른 기능을 개발하게 할 수 있습니다. 한 세션은 결제 페이지를 만들고, 다른 세션은 회원 가입 폼을 만들고, 또 다른 세션은 대시보드를 수정합니다. 각 세션이 독립된 브랜치와 워크트리에서 작업하므로, 서로의 코드를 덮어쓸 염려가 없습니다. 작업이 끝나면 각 브랜치를 메인에 머지하면 됩니다.

〔그림 14-3〕 워크트리를 활용한 병렬 개발: 세 개의 클로드 코드 세션이 각각 다른 브랜치에서 동시에 작업하는 구조

깃허브가 제공하는 클라우드 백업과 협업 기능

깃은 내 컴퓨터 컴퓨터에서 작동하는 도구입니다. 내 노트북의 하드 디스크에 변경 이력이 저장됩니다. 노트북이 고장 나면, 커밋 이력도 함께 사라집니다. 그리고 혼자 작업할 때는 내 컴퓨터 깃만으로 충분하지만, 다른 사람과 함께 프로젝트를 진행하려면 코드를 공유할 수 있는 공간이 필요합니다.

깃허브(GitHub)는 깃 리포지토리를 클라우드에 호스팅하는 서비스입니다. 깃이 버전 관리 "도구"라면, 깃허브는 그 도구로 관리되는 프로젝트를 저장하고 공유하는 "장소"입니다. 내 컴퓨터 컴퓨터의 워드 문서를 구글 문서도구에 올리는 것과 비슷합니다.

깃허브가 제공하는 핵심 기능은 다음과 같습니다.

클라우드 백업

: 내 컴퓨터 리포지토리를 깃허브에 푸시하면, 코드의 전체 이력이 클라우드에 복제됩니다. 노트북이 고장 나더라도 깃허브에서 전체 프로젝트를 다시 가져올(클론할) 수 있습니다.

버전 이력 관리

: 깃허브의 웹 인터페이스에서 모든 커밋을 시간순으로 확인할 수 있습니다. 각 커밋에서 어떤 파일의 어떤 줄이 바뀌었는지, 누가 바꿨는지, 왜 바꿨는지를 한눈에 볼 수 있습니다. 앞 장에서 "히어로 버튼에 글로우 펄스 효과 추가"라는 커밋을 만들었을 때, 깃허브에서 이 커밋을 클릭하면 정확히 어떤 코드가 변경되었는지 녹색(추가)과 빨간색(삭제)으로 표시됩니다.

풀 리퀘스트(Pull Request)

: 브랜치에서 작업한 변경 사항을 메인 브랜치에 합치기 전에, 다른 사람에게 검토를 요청할 수 있는 기능입니다. "이 코드를 메인에 합쳐도 될까요?"라는 공식적인 제안입니다. 여러 사람이 같은 프로젝트에서 작업할 때, 각자의 변경 사항이 충돌 없이 합쳐질 수 있는지 확인하는 관문 역할을 합니다.

공개와 비공개 설정

: 리포지토리를 공개(public)로 설정하면 누구나 코드를 볼 수 있고, 비공개(private)로 설정하면 초대받은 사람만 접근할 수 있습니다. 웹사이트 코드에 API 키나 비밀번호가 포함되어 있지 않다면 공개로 두어도 되지만, 민감한 정보가 조금이라도 포함된 프로젝트는 반드시 비공개로 설정해야 합니다.

〔그림 14-4〕 깃허브 리포지토리 화면 구성: 커밋 이력, 브랜치 목록, 풀 리퀘스트 탭의 위치

한 가지 중요한 주의 사항이 있습니다. 깃허브에 코드를 푸시할 때,

.env

파일이나 API 키가 포함된 파일이 함께 올라가지 않도록 주의해야 합니다.

.gitignore

라는 파일을 만들어서, 깃이 추적하지 말아야 할 파일 목록을 지정할 수 있습니다.

클로드 코드에게 "깃허브에 푸시할 준비를 해 줘"라고 요청하면, 보통

.gitignore

파일을 자동으로 생성해 줍니다. 그래도 푸시 전에 한 번 확인하는 습관이 필요합니다.

기본 워크플로우: 리포지토리 생성→클론→브랜치→커밋→PR→머지

실제로 깃과 깃허브를 사용하는 전체 흐름을 처음부터 끝까지 따라가 보겠습니다.

1단계: 깃허브에서 리포지토리 생성

깃허브에 로그인하고, "New Repository" 버튼을 클릭합니다. 이름을 "my-website"로 지정합니다. 설명은 선택 사항입니다. 공개 또는 비공개를 선택한 뒤 "Create Repository"를 클릭합니다. 텅 빈 리포지토리가 만들어집니다.

이 작업을 클로드 코드에게 맡길 수도 있습니다. "my-website라는 이름으로 깃허브 리포지토리를 만들고, 현재 프로젝트의 코드를 푸시해 줘"라고 요청하면, 리포지토리 생성부터 초기 커밋, 푸시까지 한 번에 처리합니다. 다만 깃허브 인증(authentication)이 필요합니다. 처음 한 번만 명령줄 방식를 통해 깃허브에 로그인하면, 이후에는 자동으로 인증이 유지됩니다.

2단계: 클론(clone)

이미 내 컴퓨터에서 개발 중인 프로젝트라면 클론 단계가 필요 없습니다. 반대로 다른 컴퓨터에서 작업을 이어가야 한다면, 깃허브의 리포지토리를 클론해서 내 컴퓨터에 복제합니다. 클론은 리포지토리의 모든 파일과 전체 변경 이력을 한꺼번에 가져오는 작업입니다.

3단계: 브랜치 생성과 작업

새로운 기능을 추가하거나 디자인을 변경할 때, 메인 브랜치에서 직접 작업하지 않습니다. 새 브랜치를 만들고 그 안에서 작업합니다. 클로드 코드는 이 관례를 알고 있으며, 브랜치 관리를 자연어로 요

청할 수 있습니다.

4단계: 커밋

의미 있는 변경이 이루어질 때마다 커밋합니다. 커밋 메시지에는 변경의 목적을 간결하게 적습니다. "fix layout bug"보다 "히어로 섹션 카드 정렬 깨지는 문제 수정"이 나중에 이력을 추적할 때 유용합니다.

5단계: 푸시와 풀 리퀘스트

내 컴퓨터에서 만든 커밋을 깃허브에 푸시합니다. 깃허브에서 풀 리퀘스트를 열어, 메인 브랜치에 합칠 것을 제안합니다. 혼자 작업하는 경우에도 풀 리퀘스트를 활용하면, 변경 사항을 한 번 더 검토하는 기회가 됩니다.

6단계: 머지

풀 리퀘스트가 승인되면 메인 브랜치에 머지합니다. 머지가 완료된 기능 브랜치는 삭제해도 됩니다. 이력은 메인 브랜치의 커밋 기록에 남아 있습니다.

〔그림 14-5〕 깃/깃허브 기본 워크플로우 순서도

클로드 코드를 사용하면 이 여섯 단계의 대부분을 자연어로 처리할 수 있습니다. "히어로 버튼 변경이 마음에 든다. 깃허브에 푸시해 줘"라고 말하면, 클로드 코드가 변경된 파일을 스테이징(staging)하고, 적절한 커밋 메시지를 작성하고, 원격 리포지토리에 푸시합니다. 하지만 내부에서 일어나는 일의 원리를 이해하고 있어야, 문제가 생겼을 때 원인을 파악하고 해결할 수 있습니다.

코드가 깃허브에 안전하게 올라갔습니다. 변경 이력도 깔끔하게 기록되어 있습니다. 그러나 깃허브에 코드가 있다고 해서 누군가가 웹 브라우저로 접속해서 사이트를 볼 수 있는 것은 아닙니다. 코드 저장소와 라이브 웹사이트 사이에는 하나의 다리가 더 필요합니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

15장 Vercel 배포: 코드를 세상에 내보내기

전체 목차

책의 모든 장 보기

Vercel이란 무엇인가: 원클릭 배포의 개념

휴대폰 화면에 자기가 만든 웹사이트가 떠 있는 것을 처음 보는 순간의 감각은 독특합니다. 내 컴퓨터호스트에서만 존재하던 페이지가, 카페에서 커피를 마시며 열어본 모바일 브라우저에도 동일하게 나타납니다. 그 사이에서 무슨 일이 일어난 것일까요?

내 컴퓨터호스트(localhost)는 내 컴퓨터에서만 접근할 수 있는 주소입니다. 클로드 코드가 개발 서버를 시작하면

localhost:3000

이나

localhost:51006

같은 주소가 부여되는데, 이 주소는 같은 컴퓨터의 브라우저에서만 열립니다. 다른 사람이 이 주소를 입력하면 아무것도 나오지 않습니다. 웹사이트의 코드가 내 컴퓨터의 메모리에서만 실행되고 있기 때문입니다.

이 코드를 세상에 내보내려면, 24시간 가동되는 서버에 코드를 올리고, 그 서버에 고유한 주소를 연결해야 합니다. 전통적으로 이 과정은 서버 임대, 운영 체제 설정, 웹 서버 소프트웨어 설치, SSL 인증서 발급, 도메인 연결 등 수십 단계를 거쳐야 했습니다.

버셀(Vercel)은 이 복잡한 과정을 하나의 버튼 클릭으로 압축한 서비스입니다. 깃허브 리포지토리를 선택하고 "Deploy" 버튼을 누르면, 버셀이 코드를 가져와서 구축하고, 전 세계에 분산된 서버에 배포하고, 자동으로 HTTPS 인증서를 설정하고, 고유한 URL을 부여합니다.

〔그림 15-1〕 로컬 개발에서 라이브 배포까지의 전환 과정 개념도

버셀의 무료 플랜으로도 개인 프로젝트와 포트폴리오 사이트를 운영하는 데 충분합니다. 배포된 사이트에는

프로젝트명.vercel.app

형태의 기본 도메인이 부여됩니다. 이 도메인으로도 누구나 접속할 수 있지만, 자신만의 도메인을 연결하는 것도 가능합니다.

여기서 이해해야 할 핵심은, 버셀이 하는 일은 "코드를 받아서 웹사이트로 만들어 주는 것"이라는 점입니다. 깃허브에 코드가 있고, 버셀에 배포 설정이 있으면, 그 둘을 연결하는 파이프라인이 완성됩니다.

깃허브 리포지토리와 Vercel 연결

버셀에 가입할 때 깃허브 계정으로 로그인하면, 이후의 모든 과정이 간결해집니다. 버셀이 깃허브 리포지토리 목록에 바로 접근할 수 있기 때문입니다.

연결 과정은 다음과 같습니다.

버셀 대시보드에서 "Add New Project"를 클릭합니다. 화면에 본인의 깃허브 리포지토리 목록이 나타납니다. 배포할 리포지토리를 찾아 "Import" 버튼을 클릭합니다. 구축 설정 화면이 나타나는데, Next.js나 React 같은 프레임워크를 자동으로 감지합니다. "Deploy" 버튼을 클릭하면 구축가 시작됩니다.

수십 초에서 수 분 사이에 구축가 완료되고, 고유한 URL이 생성됩니다. 이 URL을 브라우저에 입력하면 내 웹사이트가 나타납니다. 휴대폰에서 열어도, 다른 나라의 컴퓨터에서 열어도 같은 페이지가 나타납니다.

〔그림 15-2〕 버셀 대시보드에서 깃허브 리포지토리를 임포트하는 화면

그런데 첫 배포에서 문제가 발생할 수 있습니다. 앞 장에서 다룬 웹사이트 사례를 떠올려 보겠습니다. 블렌더 제품 랜딩 페이지를 버셀에 배포했더니, 스크롤 애니메이션에 사용되는 프레임 이미지 파일들이 빠져 있었습니다. 내 컴퓨터에서는 완벽하게 작동하던 사이트가, 배포 후에는 블렌더 이미지 없이 텍스트만 덩그러니 나타났습니다.

원인은

.gitignore

파일이었습니다. 이미지 프레임 폴더가 깃의 추적 대상에서 제외되어 있었기 때문에, 깃허브에 푸시되지 않았고, 버셀도 그 파일을 받지 못한 것입니다. 해결 방법은 간단합니다. 클로드 코드에게 "프레임 이미지도 깃허브에 포함시켜서 다시 푸시해 줘"라고 요청하면,

.gitignore

설정을 수정하고, 누락된 파일을 추가 커밋하여 푸시합니다.

이 경험이 보여주는 교훈이 있습니다. 내 컴퓨터 환경과 배포 환경은 다릅니다. 내 컴퓨터에는 있지만 깃 추적 대상이 아닌 파일은 배포 환경에 존재하지 않습니다. 배포 전에 "내 리포지토리에 사이트 구동에 필요한 모든 파일이 포함되어 있는가?"를 확인하는 습관이 필요합니다.

버셀 대시보드의 "Deployments" 탭에서는 모든 배포 이력을 확인할 수 있습니다. 문제가 있는 배포를 이전 버전으로 롤백(rollback)하는 것도 클릭 한 번으로 가능합니다.

〔그림 15-3〕 버셀 배포 이력 화면: 각 배포의 상태와 롤백 옵션

실시간 반영 파이프라인: 코드 변경이 자동으로 반영되는 구조

버셀과 깃허브의 연결이 완성되면, 이후부터는 놀라울 정도로 매끄러운 워크플로우가 작동합니다.

과정은 이렇습니다. 클로드 코드에서 웹사이트 코드를 수정합니다. 내 컴퓨터호스트에서 변경 사항을 확인합니다. 마음에 들면 "깃허브에 푸시해 줘"라고 요청합니다. 깃허브에 새로운 커밋이 도착합니다. 버셀이 이 변경을 자동으로 감지합니다. 새로운 구축가 시작되고, 완료되면 라이브 사이트가 자동으로 업데이트됩니다.

〔그림 15-4〕 라이브 업데이트 파이프라인: 클로드 코드 → 깃허브 → 버셀 → 라이브 사이트

이 파이프라인에서 핵심적인 안전장치는 "명시적 푸시"입니다. 내 컴퓨터에서 아무리 많이 수정해도, "깃허브에 푸시해 줘"라고 말하기 전까지는 라이브 사이트에 아무런 영향이 없습니다. 이것은 의도적인 설계입니다.

CLAUDE.md 파일에 이 원칙을 명시해 두면 안전합니다. "우리는 변경 사항을 깃허브에 동기화하고, 깃허브가 자동으로 버셀에 푸시할 것이다. 그러나 내 컴퓨터에서 변경할 때는 항상 내 컴퓨터호스트에서 먼저 검증하고, 내가 명시적으로 깃허브에 커밋하거나 푸시하라고 말하기 전까지는 절대로 원격에 반영하지 마라." 이런 지침이 있으면 클로드 코드가 실수로 작업 중인 코드를 푸시하는 상황을 방지할 수 있습니다.

실시간 반영의 속도도 인상적입니다. 앞선 사례에서 히어로 버튼에 글로우 효과를 추가한 뒤 깃허브에 푸시하자, 수 분 안에 버셀의 배포가 완료되었고, 라이브 사이트를 새로고침하면 글로우 버튼이 나타났습니다.

깃허브의 커밋 이력에는 "add glowing pulse effect to hero join the community button"이라는 메시지가 남았고, 버셀의 배포 이력에도 대응하는 배포 기록이 생겼습니다.

이 구조를 통해 사실상 세 개의 환경이 자연스럽게 만들어집니다.

개발 환경(내 컴퓨터호스트)

: 자유롭게 수정하고 검증하는 공간입니다. 실패해도 아무 영향이 없습니다.

스테이징 환경(깃허브)

: 코드의 변경 이력이 기록되는 공간입니다. 여기에 올라간 코드는 정리된 상태여야 합니다.

프로덕션 환경(버셀 라이브 사이트)

: 실제 사용자가 접속하는 공간입니다. 깃허브에서 자동으로 배포됩니다.

〔그림 15-5〕 세 가지 환경의 관계: 개발 → 스테이징 → 프로덕션

도메인 연결과 환경 변수 설정

버셀이 자동으로 부여하는

프로젝트명.vercel.app

도메인은 기능적으로 완전하지만, 전문적인 인상을 주기에는 부족합니다. 자신만의 도메인을 연결하면 사이트의 신뢰도가 올라갑니다.

도메인 연결 절차는 버셀의 프로젝트 설정에서 시작합니다. "Settings" 탭의 "Domains" 섹션으로 이동합니다. 여기서 두 가지 선택지가 있습니다. 버셀에서 직접 도메인을 구매하거나, 이미 보유한 도메인을 추가하는 것입니다.

기존 도메인을 연결하려면 DNS 설정(Domain Name System configuration)을 변경해야 합니다. 도메인을 구매한 서비스(예: 가비아, 네임칩, 구글 도메인 등)의 관리 패널에서 DNS 레코드를 버셀이 안내하는 값으로 변경합니다. 버셀의 안내 화면이 어떤 레코드를 어떤 값으로 설정해야 하는지 구체적으로 알려주므로, 기술적 배경 지식이 없어도 따라할 수 있습니다.

〔그림 15-6〕 버셀 도메인 설정 화면과 DNS 레코드 구성 예시

DNS 변경이 전파되는 데에는 수 분에서 최대 48시간이 걸릴 수 있습니다. 대부분의 경우 30분 이내에 적용되지만, 도메인을 연결한 직후에 접속이 안 되더라도 당황할 필요는 없습니다.

환경 변수(environment variable) 설정은 프로젝트가 외부 서비스와 연동될 때 필요합니다. API 키, 데이터베이스 연결 문자열, 인증 모델이 세는 글 조각(token) 같은 민감한 정보는 코드에 직접 적지 않고, 환경 변수로 분리해서 관리합니다.

버셀의 프로젝트 설정에서 "Environment Variables" 섹션을 찾으면, 키-값(key-value) 쌍을 등록할 수 있습니다. 예를 들어

NEXT_PUBLIC_API_KEY

라는 키에 실제 API 키 값을 입력합니다. 이 값은 버셀의 서버에만 저장되며, 깃허브 리포지토리에는 포함되지 않습니다. 코드에서는

process.env.NEXT_PUBLIC_API_KEY

로 이 값을 참조합니다.

환경 변수를 사용하는 이유는 보안입니다. 깃허브 리포지토리가 공개되어 있어도 API 키가 노출되지 않습니다. 그리고 개발 환경과 배포 환경에서 서로 다른 값을 사용할 수 있습니다. 내 컴퓨터에서는 검증용 API 키를, 프로덕션에서는 실제 API 키를 사용하는 식입니다.

〔그림 15-7〕 버셀 환경 변수 설정 화면

환경 변수 이름

용도

예시

DATABASE_URL

데이터베이스 연결

postgresql://user:pass@host/db

API_KEY

외부 서비스 인증

sk-abc123...

NEXT_PUBLIC_SITE_URL

사이트 주소 참조

환경 변수를 변경하면 버셀이 자동으로 재배포를 실행합니다. 코드를 수정하지 않았는데도 사이트가 재배포되는 경우, 환경 변수 변경이 원인일 수 있습니다.

사이트가 라이브로 올라갔고, 도메인도 연결되었고, 환경 변수도 설정되었습니다. 브라우저에 주소를 입력하면 누구에게나 같은 페이지가 나타납니다. 그러나 배포가 끝이 아닙니다. 라이브 사이트는 시간이 지나면서 새로운 종류의 문제를 만나게 됩니다. 개발 환경에서는 보이지 않던 문제들입니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

16장 배포 뒤의 세계: 유지보수와 업데이트

전체 목차

책의 모든 장 보기

배포된 자동화의 한계: 자가 치유가 사라지는 지점

블렌더 랜딩 페이지가 배포된 지 이틀째 되는 날, 사이트에 접속해 봅니다. 잘 작동합니다. 일주일 뒤에도 문제없습니다. 그런데 한 달 뒤, 스크롤 애니메이션이 특정 모바일 브라우저에서 멈추기 시작합니다. 어디서부터 손을 대야 하는지가 문제입니다. 클로드 코드에게 "고쳐 줘"라고 말하면 됩니까?

개발 단계에서는 클로드 코드가 강력한 자가 점검 능력을 발휘합니다. 스크린샷을 찍어 비교하고, 에러 로그를 읽고, 코드를 수정하고, 다시 검증합니다. 이 반복 과정이 실시간으로 일어나기 때문에, 문제가 생기면 즉시 감지하고 수정할 수 있습니다.

그러나 배포된 사이트에서는 이 자가 치유 루프(self-healing loop)가 작동하지 않습니다. 클로드 코드는 내 컴퓨터 개발 환경에서 실행되는 도구이지, 배포된 서버를 상시 감시하는 도구가 아닙니다. 사이트가 버셀에 올라간 뒤에는 클로드 코드의 세션이 종료되어 있을 가능성이 높습니다. 문제가 발생해도 자동으로 감지하거나 수정하는 주체가 없습니다.

이 차이를 명확히 인식해야 합니다. 개발 중에는 AI가 "능동적 수호자" 역할을 합니다. 배포 후에는 AI가 "요청 시 출동하는 수리공" 역할로 전환됩니다.

〔그림 16-1〕 개발 단계 vs 배포 후 단계에서의 AI 역할 변화

배포 후 발생할 수 있는 문제의 유형은 개발 중의 버그와 성격이 다릅니다.

외부 의존성 변경

: 사이트가 사용하는 외부 API의 응답 형식이 바뀌거나, CDN(Content Delivery Network)에서 호스팅하던 폰트 파일의 URL이 변경될 수 있습니다.

브라우저 업데이트

: 크롬이나 사파리가 업데이트되면서 특정 CSS 속성이나 자바스크립트 API의 동작이 달라질 수 있습니다. 개발 당시에는 완벽하게 작동하던 애니메이션이 새 브라우저 버전에서 깨지는 경우가 실제로 일어납니다.

트래픽 변화

: 갑자기 방문자가 몰리면 이미지 로딩이 느려지거나, 애니메이션 프레임이 끊길 수 있습니다. 내 컴퓨터호스트에서는 혼자만 접속하므로 이런 문제를 경험할 수 없습니다.

인증서와 도메인

: SSL 인증서의 만료, 도메인 갱신 누락 같은 인프라 수준의 문제가 발생할 수 있습니다. 버셀은 이런 부분을 상당 부분 자동으로 관리해 주지만, 커스텀 도메인을 사용하는 경우 도메인 등록 갱신은 사용자의 몫입니다.

이 문제들에 대처하는 기본 원칙은, 정기적으로 사이트를 직접 방문해서 확인하는 것입니다. 자동화 도구가 없는 상태에서 가용한 최선의 모니터링은 사람의 눈입니다.

예약 실행과 웹훅 트리거의 차이

사이트를 정기적으로 확인하는 작업 자체를 자동화할 수는 없을까요? 두 가지 방식이 있습니다.

예약 실행(scheduled execution)

은 정해진 시간 간격으로 특정 작업을 반복하는 방식입니다. "매일 아침 9시에 사이트의 주요 페이지에 접속해서 응답 시간을 측정하라"는 식의 작업입니다. 크론 작업(cron job)이라고도 불리는 이 방식은 시간 기반입니다. 문제가 발생한 시점과 다음 점검 시점 사이에 시간 간격이 존재하므로, 문제를 즉시 발견하지 못할 수 있습니다.

클로드 코드에서는

/loop

명령으로 간단한 예약 실행을 설정할 수 있습니다. "5분마다 배포 상태를 확인해 줘"라고 요청하면, 같은 세션 안에서 반복적으로 체크합니다. 다만 이 루프는 세션이 유지되는 동안만 작동하며, 최대 3일간 지속됩니다. 장기적인 모니터링이 필요하다면 별도의 모니터링 서비스를 구축해야 합니다.

〔그림 16-2〕 예약 실행의 시간 기반 점검 흐름

웹훅 트리거(웹훅 trigger)

는 다른 방식으로 작동합니다. 특정 이벤트가 발생했을 때 즉시 알림을 보내거나 작업을 실행하는 구조입니다. "깃허브에 새 커밋이 푸시되면 버셀이 자동으로 재배포한다"는 것이 웹훅의 대표적인 예입니다. 시간이 아니라 사건에 반응합니다.

두 방식의 차이를 정리하면 이렇습니다.

구분

예약 실행

웹훅 트리거

작동 기준

시간 (매 N분, 매일 N시)

이벤트 (커밋, 에러 발생, 폼 제출)

지연

점검 간격만큼 존재

즉시 반응

적합한 용도

정기 상태 점검, 보고서 생성

배포 자동화, 알림, 데이터 동기화

비용 구조

아무 일 없어도 실행

이벤트 발생 시에만 실행

실무에서는 두 방식을 조합해서 사용합니다. 웹훅으로 배포 자동화와 즉각적 알림을 처리하고, 예약 실행으로 정기적인 건강 점검(health check)을 수행하는 식입니다.

버셀 자체가 깃허브 웹훅을 활용하는 대표적인 사례입니다. 버셀이 깃허브 리포지토리에 웹훅을 등록해 두면, 새 커밋이 푸시될 때마다 깃허브가 버셀에게 "새로운 변경이 있다"고 알려줍니다. 버셀은 그 신호를 받자마자 코드를 가져와서 구축하고 배포합니다. 사용자가 별도의 설정을 하지 않아도 이 과정이 자동으로 연결되어 있는 것입니다.

배포 전 배틀 검증: 다양한 입력으로 검증하기

배포 버튼을 누르기 전에 한 가지 절차를 더 거치면, 배포 후의 문제를 상당 부분 예방할 수 있습니다. 배틀 검증(battle test)입니다.

어느 개발자가 결제 폼을 만들었습니다. 내 컴퓨터에서 이름, 이메일, 카드 번호를 입력하고 제출 버튼을 눌렀습니다. 잘 작동합니다. 배포합니다. 그런데 실제 사용자 중 누군가가 이메일 칸에 한글 이름을 입력합니다. 다른 사용자는 카드 번호 칸에 공백을 넣습니다. 또 다른 사용자는 뒤로 가기 버튼을 누른 뒤 다시 제출합니다. 이 중 하나에서 에러가 발생하고, 사이트가 멈춥니다.

배틀 검증는 사이트를 "정상적으로" 사용하는 것이 아니라, "비정상적으로" 사용해 보는 과정입니다. 예상치 못한 입력, 비논리적인 순서의 클릭, 극단적인 화면 크기, 느린 네트워크 환경 등을 의도적으로 시험합니다.

클로드 코드를 활용한 배틀 검증 방법이 있습니다.

할 일 목록에 검증 단계를 포함시키는 것입니다. 앞선 장에서 다룬 것처럼, 클로드 코드가 스스로 만드는 할 일 목록에 "웹사이트를 구축한 뒤 스크린샷을 찍어서 확인하라"는 항목을 넣을 수 있었습니다. 같은 원리로 "다양한 브라우저 크기에서 렌더링을 확인하라", "빈 값으로 폼을 제출해 보라", "이미지가 로드되지 않는 상황을 시뮬레이션하라"는 항목을 추가할 수 있습니다.

〔그림 16-3〕 배틀 테스트 체크리스트 예시

크롬 개발자 도구(Chrome DevTools)를 활용하는 방법도 있습니다. 클로드 코드는 브라우저를 열고 직접 상호작용할 수 있습니다. 버튼을 클릭하고, 폼을 채우고, 콘솔 에러를 읽습니다. 이 기능을 활용해서 "사이트를 열고, 모든 버튼을 클릭해 보고, 콘솔에 에러가 있는지 확인해 줘"라고 요청할 수 있습니다.

모바일 대응 확인도 빠뜨리면 안 됩니다. 앞선 사례에서 블렌더 사이트를 휴대폰으로 열었을 때, 모바일 최적화가 되지 않아 레이아웃이 어색했던 것을 기억할 것입니다. "모바일 뷰포트(viewport) 크기에서 레이아웃이 올바르게 표시되는지 확인해 줘"라는 지시를 배포 전 점검 항목에 포함시키면, 이런 문제를 미리 잡을 수 있습니다.

배틀 검증의 원칙은 하나입니다. "내가 쓸 때 잘 되니까 괜찮겠지"라는 가정을 의심하는 것입니다. 내가 아닌 다른 사람이, 내가 예상하지 못한 방식으로 사이트를 사용할 때 무슨 일이 벌어지는지 미리 확인합니다.

워크플로우와 도구를 배포하되 에이전트는 배포하지 않는 원칙

웹사이트 배포의 파이프라인을 이해하고 나면, 자연스럽게 드는 생각이 있습니다. "AI 에이전트 자체를 배포해서 24시간 돌릴 수는 없을까?"

클로드 코드로 워크플로우를 만들고, 그 워크플로우가 데이터를 수집하고, 분석하고, 보고서를 만드는 과정을 자동화했다고 가정합니다. 이것을 서버에 올려서 사람 없이 계속 실행되게 할 수 있지 않을까요?

답은 "가능하지만, 대부분의 경우 그러지 않는 것이 낫다"입니다.

워크플로우와 에이전트 사이에는 본질적인 차이가 있습니다. 워크플로우는 정해진 단계를 순서대로 실행합니다. 입력이 들어오면, 정해진 처리를 하고, 출력을 내보냅니다. 경로가 예측 가능합니다. 에이전트는 다릅니다. 상황을 판단하고, 다음 행동을 스스로 결정합니다. 같은 입력이 주어져도 맥락에 따라 다른 행동을 할 수 있습니다.

워크플로우를 배포하는 것은 안전합니다. "매일 아침 이 API에서 데이터를 가져와서, 이 형식으로 변환하고, 이 데이터베이스에 저장하라." 이런 워크플로우는 단계가 명확하고, 실패할 때 어디서 실패했는지 진단하기 쉽고, 예상치 못한 행동을 하지 않습니다.

반면 에이전트를 배포하면 예측할 수 없는 상황이 발생합니다. 에이전트가 외부 API에 요청을 보내는데, 응답이 예상과 다를 때 에이전트가 어떤 판단을 내릴지 미리 알 수 없습니다. 사람이 옆에 있을 때는 에이전트의 판단을 검토하고 교정할 수 있지만, 무인 환경에서는 잘못된 판단이 연쇄적으로 이어질 수 있습니다.

〔그림 16-4〕 워크플로우 vs 에이전트: 배포 적합성 비교

특성

워크플로우

에이전트

실행 경로

고정, 예측 가능

동적, 맥락 의존

실패 시 진단

단계별로 추적 가능

판단 과정 추적 어려움

무인 운영 적합성

높음

낮음

비용 예측

정확

불확실 (토큰 사용량 변동)

배포 권장 여부

권장

사람의 감독 하에만 권장

실용적인 지침은 이렇습니다.

배포해야 할 것은

도구와 워크플로우

입니다. 웹사이트의 코드, 예약된 데이터 수집 스크립트, API 엔드포인트(endpoint), 자동화된 구축 파이프라인 등이 여기에 해당합니다.

배포하지 않는 것이 나은 것은

판단이 필요한 에이전트

입니다. 코드를 분석하고 리팩터링하는 에이전트, 사용자 수정 의견을 해석하고 디자인 변경을 결정하는 에이전트, 여러 API를 조합해서 복합적인 작업을 수행하는 에이전트 등은 사람이 감독하는 환경에서 실행하는 것이 안전합니다.

VPS(Virtual Private Server) 같은 원격 서버에 클로드 코드를 설치해서 항상 켜 두는 방법도 있습니다. 노트북을 닫아도 세션이 유지되므로 장시간 작업에 유리합니다. SSH(Secure Shell)로 접속하거나, 심지어 휴대폰에서 원격으로 제어할 수도 있습니다. 그러나 이 경우에도 "사람이 주기적으로 확인한다"는 전제가 필요합니다.

에이전트가 혼자서 판단하고 실행하는 것과, 에이전트가 작업하고 사람이 검토하는 것 사이에는 결정적인 차이가 있습니다.

〔그림 16-5〕 배포 가능한 요소와 감독이 필요한 요소를 구분하는 의사결정 트리

프로젝트의 성격에 따라 이 경계는 달라질 수 있습니다. 위험이 낮고 복원이 쉬운 작업(예: 블로그 포스트 초안 생성)은 에이전트에게 더 많은 자율성을 줘도 됩니다. 위험이 높고 되돌리기 어려운 작업(예: 데이터베이스 구조 변경, 결제 시스템 수정)은 반드시 사람의 확인을 거쳐야 합니다.

웹사이트가 세상에 나왔고, 업데이트 파이프라인이 갖춰졌고, 유지보수의 원칙이 세워졌습니다. 코드를 작성하고 배포하는 기술적 역량은 도구가 만들어 주었지만, 무엇을 만들지, 누구를 위해 만들지, 어떤 수준의 품질을 유지할지에 대한 판단은 여전히 사람의 영역에 남아 있습니다. 도구를 다루는 능력이 커질수록, 그 도구로 무엇을 할 것인지에 대한 질문이 더 중요해집니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

17장 구직 데이터 웹 정보 수집 워크플로우

전체 목차

책의 모든 장 보기

Firecrawl MCP와 계획 모드를 결합한 작업 설계

화면 왼쪽에는 소셜 미디어 관련 원격 근무 채용 공고가 빼곡히 나열되어 있습니다. 622건. 페이지 수로 치면 21페이지에 달하는 분량입니다. 이 모든 정보를 손으로 하나하나 옮겨 적는다면 하루 종일 걸려도 끝나지 않을 작업입니다. 바로 이 지점에서 에이전트 작업 흐름이 빛을 발합니다.

이번 장에서는 파이어크롤(Firecrawl)이라는 도구와 클로드 코드의 계획 모드를 결합해 큰 규모의 채용 데이터를 자동으로 수집하고 정리하는 과정을 따라갑니다. 실습을 통해 MCP 연결 서버의 실질적인 위력을 체감하게 될 것입니다.

Firecrawl이란 무엇인가

Firecrawl은 웹사이트를 LLM(Large Language Model) 친화적 데이터로 변환해 주는 도구입니다. 단순한 웹 웹 정보 수집을 넘어, 웹사이트의 텍스트를 마크다운 형식으로 추출하고, 스크린샷을 찍고, 사이트 구조를 매핑하고, 데이터를 크롤링하는 기능까지 갖추고 있습니다.

맥도날드 웹사이트에 URL을 넣고 웹 정보 수집을 요청하면, 해당 페이지의 모든 텍스트를 마크다운 형태로 가져옵니다. 그러나 이것은 Firecrawl이 할 수 있는 일의 극히 일부에 불과합니다. 웹 정보 수집, 검색, 추출, 크롤링 등 다양한 기능이 하나의 도구 안에 통합되어 있기 때문입니다.

MCP로 도구를 연결하는 원리

여기서 핵심적인 개념이 등장합니다. MCP(Model Context Protocol, 모델 컨텍스트 프로토콜)입니다. 이 개념을 이해하기 위해 Gmail을 떠올려 보겠습니다. Gmail 안에는 이메일 보내기, 초안 작성, 이메일 가져오기 등 수많은 개별 동작이 존재합니다.

MCP는 이 모든 동작을 하나의 서버로 묶어서, 에이전트가 어떤 도구를 언제 사용할지, 어떤 매개변수를 채워야 할지 스스로 판단하도록 합니다.

케이크를 만드는 비유로 설명하면 이렇습니다. 계란이 필요하면 계란 가게, 밀가루가 필요하면 밀가루 가게, 프로스팅이 필요하면 사탕 가게를 따로 찾아가는 대신, 슈퍼마켓 MCP에 접근 권한만 주면 됩니다. "필요한 재료가 있으면 슈퍼마켓에서 알아서 가져와라." 이것이 MCP의 핵심입니다.

〔그림 17-1〕 MCP 연결 서버의 개념도: 하나의 연결로 여러 도구에 접근하는 구조〕

Firecrawl MCP 연결 서버 설치

Firecrawl의 공식 문서에는 MCP 연결 서버 설치 방법이 안내되어 있습니다. 클로드 코드에서 실행하는 방법도 포함되어 있으므로, 해당 명령어를 복사해 클로드 코드에 전달하면 됩니다.

설치 과정에서 API 키(API Key)를 입력하라는 안내가 나옵니다. API 키는 일종의 비밀번호입니다. 이것을 클로드 코드의 대화 기록에 직접 입력하는 것은 보안상 바람직하지 않습니다. 대신

.env

파일에 저장하는 방식을 사용합니다.

클로드에게 보내는 메시지 예시:

"Firecrawl MCP 서버를 설치하려고 합니다. 이 명령어를 사용해 주세요.

단, API 키는 직접 전달하지 않겠습니다. .env 파일에 넣을 수 있도록 설정해 주세요."

클로드 코드는

.env

파일에 API 키 자리표시자를 만들어 줍니다. 사용자는 Firecrawl 대시보드에서 발급받은 API 키를 해당 자리에 직접 붙여넣고 저장하면 됩니다. Firecrawl은 무료 가입 시 500크레딧을 제공하므로, 실습하기에 충분합니다.

한 가지 주의할 점이 있습니다. 클로드 코드가 배시(Bash) 명령을 실행할 때 API 키가 대화 기록에 노출될 수 있습니다. 무료 키라면 큰 문제가 되지 않지만, 민감한 키라면 클로드 코드에게 설치 방법만 안내받고, 실제 명령은 별도의 터미널에서 직접 실행하는 편이 안전합니다.

계획 모드에서 작업 설계하기

설치가 완료되었으니 본격적인 워크플로우 구축에 들어갑니다. 대화 기록을 정리(

/clear

)한 뒤, 반드시

계획 모드

로 전환합니다. 계획 모드에서 시작하는 이유는 분명합니다. 에이전트가 더 깊이 사고하고, 프로젝트 폴더 전체를 살피며, 미처 생각하지 못한 질문들을 던져 주기 때문입니다.

채용 공고 페이지의 URL을 복사해 클로드 코드에 붙여넣고, 자연어로 요구사항을 설명합니다.

"이 URL에 약 622건의 채용 공고가 있는데, 21페이지에 걸쳐 있습니다.

이 모든 공고를 스크래핑해서 엑셀 시트에 정리해 주세요.

지원 시 필요할 만한 관련 필드를 빠짐없이 포함해 주세요.

프로젝트 요구사항을 더 견고하게 만들 수 있다면 질문해 주세요."

클로드 코드는 이 메시지를 받고 Firecrawl MCP 연결 서버를 활용할 수 있는지 확인합니다. 이어서 구체적인 질문들이 돌아옵니다. 개별 채용 상세 페이지까지 웹 정보 수집할 것인지, 엑셀 파일의 저장 위치는 어디인지, 필터링 조건이 있는지 등입니다.

이 질문들에 답하면 클로드 코드는 종합적인 실행 계획을 제시합니다.

scrape_daily_remote

라는 도구를 생성하고,

scrape_job_listings

라는 워크플로우를 만든 뒤, 실제 웹 정보 수집을 수행하겠다는 내용입니다.

〔그림 17-2〕 계획 모드에서 클로드 코드가 제시한 웹 정보 수집 계획 예시〕

622개 채용 공고를 엑셀로 정리하기

계획을 승인하고 자동 수락(auto accept) 모드를 선택하면, 클로드 코드는 할 일 목록(to-do list)을 생성한 뒤 하나씩 실행해 나갑니다.

실행 결과 확인

작업이 완료되면 결과가 나타납니다. 209건의 채용 공고가 수집되었습니다. 여러 지표와 지역 정보가 함께 정리되어 있습니다. 왼쪽 파일 탐색기를 보면,

tools

폴더에

scrape_daily_remote_jobs

도구가,

workflows

폴더에

scrape_job_listings

워크플로우가 새로 생성된 것을 확인할 수 있습니다.

이 구조가 왜 중요한지 짚어 보겠습니다. 다음번에 같은 유형의 웹 정보 수집을 요청하면, 클로드 코드는 이미 만들어진 도구와 워크플로우를 재활용합니다. 실수가 있었다면 자동으로 수정한 뒤 업데이트합니다. 이것이 WAT 프레임워크의 자기 개선 루프입니다.

엑셀 파일의 구성

temp

폴더에 저장된 엑셀 파일을 열어 봅니다. 클로드 코드가 자체적으로 필터 기능까지 추가해 놓았습니다. 열(column) 구성은 다음과 같습니다.

열 이름

설명

Job Title

직무명

Job Type

근무 형태 (정규직, 계약직 등)

Position

직급

Location

근무 지역

Experience

요구 경력

Categories

직무 분류

Salary

급여 범위

Description Summary

직무 설명 요약

Tags

관련 태그

URL

채용 공고 원본 링크

〔그림 17-3〕 생성된 엑셀 파일의 실제 화면 캡처〕

209건의 채용 정보가 깔끔하게 정리되어 있습니다. 손으로 했다면 반나절이 걸렸을 작업을 클로드 코드가 몇 분 만에 완료한 것입니다.

컨텍스트 관리의 중요성

여기서 잠시 실무적인 팁을 공유합니다. 작업을 진행하다 보면 화면 하단에 "컨텍스트의 45%가 남아 있습니다"라는 표시가 보입니다. 컨텍스트 부패(Context Rot)라는 현상이 있습니다. 대화 기록이 길어질수록 AI 모델의 성능이 저하되는 것입니다. 컨텍스트 사용량이 60%를 넘기 전에 컴팩트(compact) 기능을 사용해 대화를 압축하는 것이 좋습니다.

이렇게 하면 중요한 정보는 유지하면서 불필요한 기록을 정리할 수 있습니다.

필터링과 지역별 분류: 유럽 영업직 372건 추출 사례

이번에는 난이도를 한 단계 올려 봅니다. 검색 필터가 없는 페이지에서 214,000건의 전체 채용 공고 중 특정 조건에 맞는 것만 추출하는 작업입니다.

요구사항 전달

이번에는 의도적으로 바이패스 퍼미션(Bypass Permissions) 모드를 사용합니다. 계획 모드를 거치지 않고 에이전트에게 비교적 모호한 지시를 던져서, 어떤 결과가 나오는지 관찰하기 위해서입니다.

"이 URL에서 영업(sales) 관련 채용 공고를 스크래핑해 주세요.

유럽 지역만 필터링하고, 약 500건을 가져와서 엑셀에 정리해 주세요."

클로드 코드는 이전에 만든 웹 정보 수집 도구를 인식합니다. "이전에 만든 웹 정보 수집 도구를 활용할 수 있겠군요"라고 스스로 판단한 뒤, 계획을 세웁니다. 영업직 공고를 웹 정보 수집하고, 유럽으로 필터링한 다음, 엑셀로 내보내겠다는 것입니다.

현실과의 충돌, 그리고 적응

그런데 예상치 못한 상황이 벌어집니다. 유럽 지역 영업직이 52건밖에 되지 않는 것입니다. 전체 영업직은 409건인데, 유럽으로 한정하면 목표인 500건에 한참 못 미칩니다.

이 지점에서 에이전트의 판단 능력이 드러납니다. 클로드 코드는 자동으로 문제를 인식하고, 사용자에게 선택지를 제시합니다.

유럽 영업직 55건만 유지할 것인지

모든 직종으로 범위를 넓힐 것인지

미국 기반 영업직까지 포함할 것인지

"미국 영업직도 추가해 주세요"라고 답하면, 에이전트는 작업을 이어갑니다. 이 과정에서 임시 파일들이 생성됩니다.

all_sales_jobs

,

sales_jobs_raw

같은 중간 데이터 파일과 필터링용 파이썬 스크립트 3개가

temp

폴더에 만들어집니다. 프로젝트 시작 시 임시 폴더를 별도로 구성해 둔 것이 빛을 발하는 순간입니다.

최종 결과

최종적으로 372건의 영업직 채용 공고가 수집됩니다. 엑셀 파일에는 기존 필드에 더해

Region(지역)

열이 추가되어 있습니다. 이 열을 기준으로 필터링하면 미국, 유럽, 전 세계(worldwide) 공고를 즉시 분류할 수 있습니다. 유럽만 선택하면 약 49건이 남습니다.

〔그림 17-4〕 지역별 필터가 적용된 영업직 엑셀 파일〕

예상치 못한 요청 처리: 미국 치과의사 잠재 고객 목록 생성

에이전트 작업 흐름의 진정한 시험대는 예상 밖의 요청입니다. 지금까지는 특정 URL에서 채용 공고를 가져오는 비교적 정형화된 작업이었습니다. 이번에는 완전히 다른 종류의 요청을 던져 봅니다.

전혀 다른 유형의 요청

"미국의 치과의사들에게 연락하고 싶습니다.

치과의사들의 연락처 정보를 찾아서 리드 리스트를 엑셀로 만들어 주세요."

이 요청에는 URL이 없습니다. 어디서 데이터를 가져올지에 대한 힌트도 없습니다. 에이전트가 스스로 데이터 소스를 찾아야 합니다.

에이전트의 문제 해결 과정

클로드 코드는 먼저 기존 도구와 워크플로우를 점검합니다. 이어서 Firecrawl을 활용해 "치과의사 디렉터리(dentist directory)"를 검색합니다. ADA(미국치과의사협회) 사이트를 발견하지만, 자바스크립트로 동적 렌더링되는 사이트라 정적 웹 정보 수집이 작동하지 않습니다.

여기서 포기하지 않습니다. 다른 방법을 모색해 옐로페이지(Yellow Pages)가 효과적이라는 것을 스스로 파악합니다. 뉴욕시에만 3,000명의 치과의사 정보가 있다는 사실을 확인하고, 새로운 웹 정보 수집 도구를 만들기 시작합니다.

그런데 첫 번째 시도에서 치과의사 2명만 추출됩니다. 파싱 패턴에 문제가 있는 것입니다. 에이전트는 이 오류를 즉시 감지하고, 정규 표현식(Regex) 패턴을 수정합니다. 도구 파일 자체를 업데이트해서 같은 문제가 재발하지 않도록 합니다.

〔그림 17-5〕 에이전트가 웹 정보 수집 오류를 감지하고 도구를 수정하는 과정〕

자기 치유의 실제 모습

이 과정이 바로

자기 치유(스스로 고치기)

의 실제 모습입니다. 전통적인 자동화에서는 오류가 발생하면 개발자가 로그를 읽고, 코드를 수정하고, 다시 검증해야 했습니다. 에이전트 작업 흐름에서는 에이전트가 이 모든 과정을 스스로 수행합니다. 오류를 발견하고, 원인을 분석하고, 코드를 수정하고, 재검증합니다.

최종 결과물

수정된 도구로 다시 실행한 결과, 4개 주요 도시에서 120명의 고유한 치과의사 리드가 수집됩니다. 엑셀 파일의 구성은 다음과 같습니다.

열 이름

설명

Name

치과의사 이름

Phone

전화번호

Address

주소

City

도시

State

주(州)

Zip Code

우편번호

Website

웹사이트

Specialties

전문 분야

Listing URL

옐로페이지 원본 링크

새로운 워크플로우(

scrape_dentist_leads

)와 새로운 도구(

scrape_dentist_leads

)가 자동으로 생성되었습니다. 다음번에 비슷한 요청이 들어오면 이 워크플로우와 도구가 재활용됩니다.

다중 에이전트 전략

여기서 한 가지 고급 전략을 소개합니다. 클로드 코드에서는 여러 에이전트를 동시에 열 수 있습니다. 다섯 개의 에이전트에게 각각 다른 접근 방법을 시도하게 한 뒤, 가장 좋은 결과를 내는 워크플로우만 남기고 나머지를 삭제하는 것입니다. 이 방식은 어떤 데이터 소스가 가장 신뢰할 수 있는지, 어떤 웹 정보 수집 전략이 가장 효율적인지를 병렬로 검증할 수 있게 해 줍니다.

이번 실습에서 확인한 것은 명확합니다. 원하는 결과물의 형태를 알고 있다면, 그것을 달성하기 위한 기술 스택과 세부 구현 방법은 에이전트에게 위임할 수 있습니다. 622건의 채용 공고든, 372건의 영업직이든, 120명의 치과의사 리드든, 자연어로 목표를 설명하는 것만으로 실질적인 결과물이 만들어집니다. 그렇다면 이 강력한 도구를 사용할 때 사람들이 빠지기 쉬운 함정은 무엇인지 살펴보겠습니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

18장 워크플로우 구축의 흔한 실수와 극복법

전체 목차

책의 모든 장 보기

실수 1: 목표를 충분히 명확하게 설명하지 않는 것

"링크드인용 리드 스크래퍼가 필요합니다."

이 한 문장을 클로드 코드에 던지면 어떤 일이 벌어지는지 살펴보겠습니다. 에이전트는 어떤 산업의 리드를 원하는지 알 수 없습니다. 어떤 직책을 타깃으로 하는지도 모릅니다. 결국 무작위 프로필을 끌어모으기 시작합니다. 시간과 모델이 세는 글 조각을 낭비하고, 쓸모없는 엑셀 파일 하나가 남습니다.

앞 장에서 622건의 채용 공고를 웹 정보 수집할 때를 떠올려 보겠습니다. 계획 모드에서 URL을 제공하고, 페이지 수를 알려주고, 원하는 필드를 설명했습니다. 그러자 클로드 코드가 돌아온 질문은 구체적이었습니다. 개별 상세 페이지까지 긁을 것인지, 저장 경로는 어디인지, 필터 조건이 있는지. 이 질문들에 답한 뒤에야 에이전트는 실행 계획을 수립했습니다.

모호함이 만드는 연쇄 반응

목표가 모호하면 에이전트의 판단 하나하나가 불확실해집니다. "리드 스크래퍼"라고만 말하면, 에이전트는 다음과 같은 결정을 혼자 내려야 합니다.

어떤 웹사이트에서 데이터를 가져올 것인가?

이름, 이메일, 전화번호 중 어떤 정보가 필요한가?

몇 건을 수집하면 충분한가?

지역 제한이 있는가?

출력 형식은 엑셀인가, CSV인가, 구글 시트인가?

이 질문들 중 하나라도 잘못 판단하면 전체 결과가 틀어집니다. 에이전트가 질문을 던져 줄 수 있지만, 그것은 계획 모드에서 충분한 맥락을 제공했을 때의 이야기입니다. 바이패스 퍼미션 모드에서 모호한 지시를 내리면, 에이전트는 자체적으로 추측하며 진행합니다.

명확한 목표 설정의 공식

좋은 목표 설정에는 패턴이 있습니다.

모호한 요청:

"링크드인에서 CEO 프로필을 찾아 주세요."

명확한 요청:

"기술 기업 CEO의 링크드인 프로필 75건이 필요합니다. 이름, 회사명, 이메일, 프로필 링크를 스프레드시트에 정리해 주세요. 75건이 모이면 완료입니다."

차이가 보입니다. 후자에는 대상(기술 기업 CEO), 수량(75건), 필요 데이터(이름, 회사명, 이메일, 프로필 링크), 출력 형식(스프레드시트), 완료 조건(75건 달성)이 모두 포함되어 있습니다.

〔그림 18-1〕 모호한 요청 vs. 명확한 요청의 결과물 차이 비교〕

에이전트를 전문가로 대하라

여기서 중요한 관점 전환이 필요합니다. 에이전트는 지시를 기계적으로 수행하는 도구가 아닙니다. 전문가입니다. 사용자는 관리자(manager) 역할을 맡되, 기술적 세부 사항은 에이전트에게 위임하는 것입니다.

실제 소프트웨어 개발자에게 앱 제작을 의뢰한다고 상상해 보겠습니다. "앱 하나 만들어 주세요"라고만 말하면 개발자도 난감합니다. 어떤 기능이 필요한지, 누가 사용할 것인지, 마감은 언제인지 등의 정보가 있어야 일을 시작할 수 있습니다. 에이전트도 마찬가지입니다.

다만, 모든 기술적 세부 사항을 알 필요는 없습니다. "이런 결과를 원하는데, 어떤 접근 방법이 좋을까요?"라고 물으면, 에이전트가 Opus 4.5 같은 추론 모델의 능력으로 다섯 가지 접근 방법을 비교하고 최적의 것을 선택해 줍니다.

실수 2: '완료'의 기준을 정의하지 않는 것

첫 번째 실수가 "무엇을"에 관한 것이라면, 두 번째 실수는 "언제 멈출 것인가"에 관한 것입니다.

에이전트에게 명확한 종료 지점을 주지 않으면, 세 가지 문제가 발생합니다. 과도한 복잡화, 불필요한 반복, 끝없는 리서치입니다. 에이전트는 더 나은 결과를 만들 수 있다고 판단하면 계속 시도합니다. 이것은 강점이면서 동시에 약점입니다.

끝없는 루프의 위험

치과의사 리드 웹 정보 수집 사례를 다시 보겠습니다. "미국의 치과의사를 찾아서 연락처를 모아 주세요"라고만 요청했습니다. 만약 에이전트가 120건을 모은 뒤에도 "더 수집할 수 있다"고 판단하면 어떻게 되겠습니까? 새로운 도시를 추가하고, 새로운 데이터 소스를 탐색하고, 웹 정보 수집 도구를 계속 개선하며 멈추지 않을 수 있습니다.

컨텍스트는 소진되고, 모델이 세는 글 조각 비용은 불어나며, 정작 원하는 결과물은 나오지 않습니다.

완료 조건의 설계

완료 조건은 두 가지 축으로 구성됩니다.

정량적 기준

과

정성적 기준

입니다.

정량적 기준의 예:

수집 건수: "150건이 모이면 중단"

페이지 수: "최대 10페이지까지만 웹 정보 수집"

시간 제한: "5분 내에 가능한 만큼만"

정성적 기준의 예:

데이터 완성도: "전화번호와 주소가 모두 포함된 레코드만 유효"

출력 형식: "엑셀 파일로 저장하되, 열 필터를 적용한 상태"

검증 방법: "저장 후 파일을 열어 첫 10행을 확인"

유럽 영업직 웹 정보 수집 사례에서 이 원칙이 작동하는 것을 보았습니다. "500건"이라는 목표를 제시했고, 에이전트가 유럽에서 52건밖에 찾지 못했을 때 자발적으로 선택지를 제시한 것은 완료 조건이 명확했기 때문입니다. 목표 대비 현재 상태의 괴리를 에이전트가 스스로 인식한 것입니다.

〔그림 18-2〕 완료 조건이 있는 워크플로우와 없는 워크플로우의 실행 경로 비교〕

프로젝트 요구사항 문서(요구사항 문서)를 에이전트와 함께 작성하기

두 가지 실수의 해결책은 하나로 수렴합니다.

프로젝트 요구사항 문서(요구사항 문서, Project Requirements Document)

를 에이전트와 함께 작성하는 것입니다.

요구사항 문서 공동 작성의 실제 과정

계획 모드를 켭니다. 그리고 이렇게 말합니다.

"대략적인 아이디어가 있는데, 이것을 견고한 PRD로 만들어 주세요.

브레인스토밍하고, 리서치하고, 필요한 질문을 해 주세요.

그래야 이 워크플로우를 고품질로 구축할 수 있습니다."

이 요청을 받은 에이전트는 세 단계를 거칩니다.

1단계: 브레인스토밍과 리서치

에이전트는 설명된 목표를 기반으로 관련 API, MCP 연결 서버, 데이터 소스를 조사합니다. 웹 검색을 수행하기도 합니다. 유튜브 분석 워크플로우를 설계할 때 에이전트가 유튜브 데이터 API와 MCP 연결 서버를 비교 조사한 것이 이 단계에 해당합니다.

2단계: 질문과 응답

에이전트는 한 번에 여러 질문을 던집니다. "어떤 채널을 추적할 것인가?", "얼마나 자주 보고서를 받고 싶은가?", "데이터를 시트에도 기록할 것인가?", "보고서를 어느 이메일로 발송할 것인가?" 같은 질문들입니다. 이 질문들은 요구사항 문서의 빈칸을 채우는 역할을 합니다.

3단계: 계획 수립

질문과 응답이 끝나면, 에이전트는 종합적인 실행 계획을 제시합니다. 목표, 워크플로우 구조, 필요한 도구 목록, 입력값, 출력값, 엣지 케이스(Edge Case) 처리 방안이 모두 포함됩니다.

워크플로우 문서의 구조

완성된 워크플로우 문서를 살펴보면, 그 구조가 곧 요구사항 문서입니다.

워크플로우: scrape_job_listings

목표

Daily Remote에서 검색어 기반으로 채용 공고를 스크래핑한다.

필수 입력값

- search_term: 검색 키워드

- max_pages: 최대 스크래핑 페이지 수

- output_path: 엑셀 파일 저장 경로

사용 도구

- scrape_daily_remote_jobs

실행 단계

1. 검색어로 첫 페이지 스크래핑

2. 페이지네이션 감지 후 순차 스크래핑

3. 데이터 정제 및 중복 제거

4. 엑셀 파일 생성 및 필터 적용

예상 출력

- 채용 공고가 정리된 .xlsx 파일

엣지 케이스 처리

- 페이지 로딩 실패 시 3회 재시도

- 데이터 누락 필드는 빈 값으로 처리

이 문서가 존재하기 때문에 다음번에 같은 워크플로우를 실행할 때 에이전트가 일관된 결과를 만들어 냅니다. 문서가 없으면 매번 다른 방식으로 접근하게 되고, 결과의 품질이 들쭉날쭉해집니다.

〔그림 18-3〕 요구사항 문서 기반 워크플로우 문서의 실제 예시〕

에이전트 작업 흐름의 세 이점: 자동 디버깅, 말 제어, 자기 학습

실수를 피하는 방법을 다뤘으니, 이제 에이전트 작업 흐름이 기존 자동화 방식에 비해 본질적으로 다른 점을 세 가지 관점에서 살펴보겠습니다.

자동 디버깅: 더 이상 로그를 읽지 않아도 됩니다

전통적인 워크플로우 자동화의 일상은 이랬습니다. 무언가를 만들고, 실행하고, 예상하지 못한 엣지 케이스에서 시스템이 멈춥니다. 그러면 로그를 열고, 에러 메시지를 분석하고, 실행 데이터를 하나하나 추적하며 원인을 찾아야 했습니다. 한 시간이 훌쩍 지나갑니다.

에이전트 작업 흐름에서는 이 과정이 자동화됩니다. 앞 장에서 치과의사 리드 웹 정보 수집 도구가 2건만 추출했을 때를 기억하실 것입니다. 에이전트는 "파싱 패턴에 문제가 있다"고 스스로 진단하고, 정규 표현식을 수정한 뒤, 도구 파일을 업데이트하고, 다시 실행했습니다. 사람의 개입 없이 이 모든 과정이 이루어졌습니다.

이것은

자기 치유(스스로 고치기)

라는 특성입니다. 실무적으로 이것이 의미하는 바는 큽니다. 오른쪽 모니터에서 에이전트가 워크플로우를 구축하는 동안, 왼쪽 모니터에서 다른 업무를 할 수 있다는 뜻입니다. 가끔 확인하면서 방향을 잡아주기만 하면 됩니다. AI는 비결정적(non-deterministic)이므로 경로를 벗어날 수 있지만, 대부분의 오류는 스스로 복구합니다.

말로 하는 제어: 노드를 배울 필요가 없습니다

n8n 같은 도구에서는 각 노드(Node)가 무엇을 하는지, 언제 사용하는지, 매개변수와 설정값이 무엇을 의미하는지 학습해야 했습니다. API에 연결하려면 문서를 읽고, 엔드포인트(Endpoint)를 찾고, JSON 구조를 올바르게 설정하고, 인증 방식을 파악해야 했습니다. 진입 장벽이 높았습니다.

에이전트 작업 흐름에서는 원하는 것을 자연어로 설명합니다. 에이전트가 사용 가능한 도구를 검토하고, MCP 연결 서버가 있는지 확인하고, 필요하면 API 문서를 스스로 조사합니다. "유럽 영업직을 웹 정보 수집해 주세요"라는 한 문장이 웹 정보 수집 도구 선택, 필터링 로직 구현, 엑셀 출력 형식 결

정을 모두 포함합니다.

이 차이를 표로 정리하면 다음과 같습니다.

구분

기존 자동화 (n8n 등)

에이전틱 워크플로우

도구 선택

사용자가 직접 노드 선택

에이전트가 자동 선택

API 연결

문서 읽기, 엔드포인트 설정, 인증 구성

자연어 요청 한 번

오류 처리

수동 디버깅

자동 감지 및 수정

수정 방법

노드 설정 변경

자연어로 지시

〔그림 18-4〕 기존 자동화와 에이전트 작업 흐름의 작업 흐름 비교 다이어그램〕

자기 학습: 사용할수록 개선됩니다

과거에는 자동화를 업데이트하려면 노드를 직접 수정하고 다시 설정해야 했습니다. 에이전트 작업 흐름에서는 에이전트가 문제를 만날 때마다 학습하고 워크플로우와 도구를 업데이트합니다.

구직 웹 정보 수집 워크플로우의 진행 과정을 돌아보면 이 특성이 분명해집니다. 첫 번째 실행에서 209건을 성공적으로 수집한 뒤, 두 번째 실행에서는 이전에 만든 도구를 재활용했습니다. 세 번째 실행(치과의사 리드)에서는 완전히 새로운 도구를 만들었지만, 이전 경험에서 얻은 웹 정보 수집 패턴을 활용했습니다. 각 워크플로우 파일과 도구 파일이 실행을 거듭하며 정교해집니다.

수동 트리거와 스케줄 트리거의 차이

한 가지 짚어 두어야 할 구분이 있습니다. 클로드 코드 앞에 앉아서 "이 작업을 해 주세요"라고 지시하는 것은

수동 트리거(Human-triggered)

입니다. 이 경우 에이전트가 실시간으로 자기 치유하고 학습합니다.

반면, 매주 월요일 오전 6시에 자동 실행되도록 배포하면 상황이 달라집니다. 배포되는 것은 워크플로우와 도구입니다. 에이전트 자체가 배포되는 것이 아닙니다. 따라서 배포된 워크플로우가 실행 중에 오류를 만나면 스스로 고치지 못합니다. 에이전트가 함께 있지 않기 때문입니다.

이것은 에이전트 작업 흐름의 현재 한계입니다. 스케줄 트리거로 운영 중인 워크플로우를 개선하려면, 클로드 코드로 돌아와서 워크플로우를 수정하고, 업데이트된 버전을 다시 배포해야 합니다.

워크플로우를 견고하게 설계하는 방법과, 에이전틱 접근 방식이 가져다주는 이점을 살펴보았습니다. 목표를 명확히 하고, 완료 조건을 정의하고, 요구사항 문서를 에이전트와 함께 작성하면, 워크플로우의 품질은 비약적으로 높아집니다. 이제 이 원칙들을 가지고, 구글이 제공하는 강력한 도구 생태계를 에이전트에 연결하는 방법을 알아볼 차례입니다.

전체 목차

책의 모든 장 보기

19장 구글 워크스페이스 명령줄 방식: 구글 서비스 한 번에 다루기

전체 목차

책의 모든 장 보기

GWS 명령줄 방식란 무엇인가

유튜브 영상을 기반으로 리소스 가이드 문서를 만들어야 하는 상황을 떠올려 보겠습니다. 기존 방식이라면 이런 과정을 거칩니다. API를 호출해 구글 독스(Google Docs)를 생성하고, 마크다운 텍스트를 삽입합니다. 결과물을 열어 보면 서식 없이 날것의 마크다운이 그대로 드러납니다. 헤더도 없고, 링크도 작동하지 않고, 이미지는 빠져 있습니다.

GWS 명령줄 방식(Google Workspace 명령줄 방식, 구글 워크스페이스 커맨드라인 인터페이스)를 사용하면 이야기가 달라집니다. 유튜브 영상의 링크를 클로드 코드에 넣고 "리소스 가이드를 만들어 주세요"라고 말합니다.

에이전트는 영상의 트랜스크립트를 다운로드하고, 구글 독스를 생성하되, API 호출이 아닌 터미널 명령어(bash command)로 구글과 직접 소통합니다. 완성된 문서에는 헤더 이미지가 삽입되어 있고, 유튜브 채널로 돌아가는 링크가 작동하며, 시장 분석 내용이 체계적으로 정리되어 있고, 하단에는 CTA(Call to Action)까지 포함되어 있습니다.

명령줄 방식와 화면 기반 방식의 차이

여기서 명령줄 방식라는 용어를 풀어 보겠습니다. 명령줄 방식는 커맨드라인 인터페이스(Command Line Interface)의 약자입니다. 우리가 일상적으로 사용하는 것은 화면 기반 방식(Graphical User Interface, 그래피컬 유저 인터페이스)입니다. 버튼을 클릭하고, 폼을 채우고, 메뉴를 탐색하는 방식입니다. 반면 명령줄 방식는 텍스트 명령어로 컴퓨터와 소통합니다.

AI 에이전트에게 명령줄 방식가 유리한 이유는 명확합니다. 에이전트는 버튼을 클릭할 수 없지만, 텍스트 명령어는 자유자재로 작성할 수 있습니다. GWS 명령줄 방식는 이 특성을 활용해 구글 워크스페이스의 모든 서비스를 텍스트 명령어로 제어할 수 있게 해 줍니다.

하나의 도구, 여섯 개의 서비스

GWS 명령줄 방식 하나로 접근할 수 있는 구글 서비스는 다음과 같습니다.

서비스

할 수 있는 일

Google Drive

검색, 목록 조회, 업로드, 다운로드, 이동, 복사, 공유

Gmail

이메일 읽기, 보내기, 분류, 라벨링, 검색

Google Calendar

일정 조회, 생성, 수정, 빈 시간 찾기, 미팅 예약

Google Docs

문서 생성, 편집, 서식 적용, 템플릿 기반 작업

Google Sheets

데이터 읽기, 쓰기, 수식 적용, 보고서 생성

Google Slides

프레젠테이션 생성, 슬라이드 편집, 이미지 삽입

이 모든 것이 하나의 명령줄 방식 도구로 통합되어 있습니다. 별도의 API 엔드포인트를 찾거나, MCP 설정을 서비스마다 반복할 필요가 없습니다. 구글 워크스페이스가 새로운 API 엔드포인트나 메서드를 추가하면, GWS 명령줄 방식가 자동으로 이를 감지합니다(Auto Discovery). 유지보수가 거의 필요 없다는 뜻입니다.

〔그림 19-1〕 GWS 명령줄 방식가 연결하는 구글 워크스페이스 서비스 구조도〕

오픈소스, 그리고 주의사항

GWS 명령줄 방식는 깃허브(GitHub)에 공개된 오픈소스 프로젝트입니다. 무료로 사용할 수 있습니다. 다만 한 가지 알아 두어야 할 점이 있습니다. 깃허브 페이지에 이런 문구가 적혀 있습니다.

"This is not an officially supported Google product."

공식 지원 제품이 아니라는 의미입니다. 이것이 안전하지 않다는 뜻은 아닙니다. 실제 구글 제품이지만, 현재는 오픈소스 베타(beta) 상태입니다. 개발자 실험실에 가까운 포지션이라고 보면 됩니다. 이어지는 문구도 있습니다.

"This product is under active development. Expect breaking changes as we march towards V1.0."

활발하게 개발 중이며, 1.0 버전을 향해 가는 과정에서 호환성이 깨지는 변경이 있을 수 있다는 안내입니다. 현재 상태에서도 충분히 실용적이고, 앞으로 더 나아질 것이라는 의미이기도 합니다.

설치와 인증: 구글 클라우드 프로젝트부터 OAuth까지

GWS 명령줄 방식의 설치 과정은 클로드 코드의 도움을 받으면 수월합니다. 하지만 몇 가지 수동 단계가 있으므로 전체 과정을 순서대로 따라가 보겠습니다.

1단계: 클로드 코드에 설치 요청

깃허브 저장소의 링크를 복사합니다. 클로드 코드에 이렇게 전달합니다.

"이 GWS CLI를 설치하고 싶습니다.

문서를 읽고, 설치에 필요한 모든 것을 안내해 주세요."

에이전트는 저장소의 문서를 분석하고, 이미 설치된 전제 조건(prerequisites)을 확인한 뒤, 명령줄 방식 자체를 설치합니다. 이 과정까지는 자동으로 진행됩니다.

2단계: 구글 클라우드 프로젝트 생성

〔Google Cloud Console〕(https://console.cloud.google.com)에 접속합니다. 오른쪽 상단에서 올바른 계정으로 로그인되어 있는지 확인합니다.

새 프로젝트를 생성합니다. 이름은 자유롭게 지정할 수 있습니다. "Claude-Code-GWS" 같은 이름이면 나중에 식별하기 쉽습니다. 프로젝트 생성 후 해당 프로젝트를 선택해 활성화합니다.

3단계: OAuth 동의 화면 설정

상단 검색창에 "APIs and services"를 입력합니다. OAuth 동의 화면(OAuth consent screen) 설정으로 이동합니다.

"Get started"를 클릭합니다.

앱 이름을 입력합니다.

대상(audience)을 선택합니다. 개인 사용이라면 "Internal"을 선택합니다. "External"을 선택할 경우 검증 모드에서 본인의 이메일을 검증 사용자로 추가해야 합니다.

동의 정보를 입력하고 "I agree"를 체크한 뒤 생성합니다.

4단계: OAuth 클라이언트 ID 생성

APIs and services 〉 Credentials로 이동합니다. "Create Credentials" 〉 "OAuth Client ID"를 선택합니다.

애플리케이션 유형:

Desktop app

이름: "GWS" (또는 원하는 이름)

"Create"를 클릭합니다.

클라이언트 ID와 클라이언트 시크릿이 생성됩니다. JSON 파일로 다운로드합니다. 다운로드한 파일을 GWS의 전역 설정 경로(예:

~/.config/gws/

)에 저장합니다. 경로가 확실하지 않으면 클로드 코드에게 "전체 경로를 알려 주세요"라고 요청하면 됩니다.

〔그림 19-2〕 구글 클라우드 콘솔에서 OAuth 클라이언트 ID를 생성하는 화면〕

5단계: API 활성화

구글 클라우드 프로젝트에서 사용하려는 서비스의 API를 각각 활성화해야 합니다. Gmail API, Google Drive API, Google Sheets API, Google Slides API, Google Calendar API 등을 하나씩 열어 "Enable" 버튼을 누릅니다.

6단계: 인증 로그인

클로드 코드에게

gws auth login

명령을 실행해 달라고 요청합니다. 브라우저 탭이 열리면 사용할 구글 계정을 선택하고, 요청된 권한들을 확인한 뒤 "Allow"를 클릭합니다.

인증이 완료되면 클로드 코드가 자동으로 연결을 검증합니다. "2025년 4월에 만든 구글 독스를 찾아 주세요" 같은 간단한 요청으로 정상 작동을 확인할 수 있습니다.

설치 과정에서 문제가 생기면 당황하지 마세요. 클로드 코드에게 "작동하지 않는데, 이런 에러가 보입니다"라고 전달하면, 문서를 다시 참조하며 해결 방법을 안내해 줍니다. 한두 번의 왕복이면 대부분 해결됩니다.

Gmail 우선순위 자동 분류

GWS 명령줄 방식가 설치되었으니, 실용적인 워크플로우를 만들어 보겠습니다. 첫 번째는 Gmail 이메일 우선순위 자동 분류입니다.

요구사항 전달

"오늘 도착한 읽지 않은 이메일을 가져와 주세요.

내 비즈니스와 우선순위에 대해 알고 있는 내용을 기반으로 각 이메일에 점수를 매겨 주세요.

우선순위 점수가 5점 미만이면 자동으로 읽지 않음 상태로 표시해 주세요."

이 요청에서 주목할 부분은 "내 비즈니스와 우선순위에 대해 알고 있는 내용을 기반으로"라는 구절입니다. 클로드 코드가

claude.md

파일이나 이전 대화에서 축적한 맥락 정보를 활용해 판단하게 됩니다.

실행 결과

에이전트는 30개의 읽지 않은 이메일을 가져옵니다. 각 이메일에 비즈니스 관련성을 기준으로 우선순위 점수를 매깁니다. 뉴스레터 구독 메일은 낮은 점수를, 클라이언트 문의나 파트너십 제안은 높은 점수를 받습니다.

〔그림 19-3〕 30개 이메일에 우선순위 점수가 매겨진 결과 화면〕

점수가 5점 미만인 이메일은 자동으로 읽지 않음 처리됩니다. 아침에 출근해서 이 워크플로우를 한 번 실행하면, 받은편지함에는 실제로 확인이 필요한 이메일만 남게 됩니다. 30개의 이메일을 하나하나 열어 보며 중요도를 판단하는 시간이 절약됩니다.

Google Slides 자동 생성과 시각적 검증 루프

Gmail 분류가 텍스트 기반 작업이라면, Google Slides 생성은 시각적 요소가 개입하는 더 복잡한 작업입니다. 여기서 에이전트 작업 흐름의 흥미로운 한계와 그 극복 방법이 드러납니다.

첫 번째 시도: 프로그래밍 방식의 생성

브랜드 가이드라인과 로고를 제공하고 슬라이드 덱을 만들어 달라고 요청합니다. 결과물은 "괜찮은" 수준입니다. 그런데 간격이 어긋나 있거나, 텍스트가 겹치는 부분이 있습니다.

에이전트에게 "이 부분을 수정해 주세요"라고 말하면 돌아오는 답변이 의미심장합니다.

"저는 슬라이드를 볼 수 없습니다. 프로그래밍 방식으로만 생성할 수 있기 때문에 간격 오류가 발생할 수 있습니다."

에이전트는 구글 슬라이드의 API를 통해 좌표, 크기, 색상 등을 수치로 지정해 슬라이드를 만듭니다. 그러나 그 결과물을 눈으로 확인할 수는 없습니다. 마치 눈을 감고 그림을 그리는 것과 같습니다.

시각적 검증 루프의 구축

이 문제의 해결책은 에이전트에게 "눈"을 달아 주는 것입니다. Chrome DevTools 접근 권한을 부여하면, 에이전트가 슬라이드를 브라우저에서 열고, 스크린샷을 찍고, 그 이미지를 분석할 수 있습니다.

이 과정을

시각적 검증 루프(Visual Validation Loop)

라고 부릅니다. 작동 방식은 이렇습니다.

1. 슬라이드를 프로그래밍 방식으로 생성

2. 브라우저에서 슬라이드를 열고 스크린샷 촬영

3. 스크린샷을 분석해 시각적 문제 감지

4. 문제가 있으면 API로 수정

5. 다시 스크린샷을 찍어 확인

6. 모든 슬라이드가 만족스러울 때까지 반복

〔그림 19-4〕 시각적 검증 루프의 작동 흐름도〕

검증 루프 적용 후의 결과

시각적 검증 루프를 적용하고 슬라이드를 다시 생성하면, 결과가 눈에 띄게 개선됩니다. 브랜드 컬러가 일관되게 적용되고, 로고가 우측 상단에 배치되며, 맞춤형 이미지가 브랜드 색상과 어우러집니다. 마지막 슬라이드에는 CTA가 포함됩니다.

에이전트는 각 슬라이드를 순회하며 스크린샷을 찍고, 간격, 정렬, 텍스트 가독성을 분석합니다. 마지막 슬라이드의 하단 간격이 비정상적이면 이를 감지하고 수정합니다.

완벽하지는 않습니다. 감마(Gamma) 같은 전문 프레젠테이션 도구의 수준에는 아직 미치지 못합니다. 그러나 무료이고, 자동화가 가능하며, 반복할수록 개선됩니다. 에이전트에게 "슬라이드 덱을 다시 감사하고, 향후 어떻게 개선할 수 있을지 알려 주세요"라고 요청하면, 에이전트는 각 슬라이드를 다시 스크린샷으로 캡처하며 개선 방안을 정리합니다.

한 가지 실용적인 팁이 있습니다. 스크린샷을 찍을 때 브라우저 창의 크기가 작으면 해상도가 낮아져 분석 품질이 떨어집니다. 프레젠테이션 모드에서 스크린샷을 찍는 것이 더 정확한 결과를 낳습니다.

같은 시각적 검증 루프를 구글 독스 생성에도 적용할 수 있습니다. 문서의 서식, 이미지 위치, 링크 작동 여부를 자동으로 확인하는 것입니다.

100가지 이상의 내장 스킬 레시피 활용

GWS 명령줄 방식의 강점 중 하나는

스킬 레시피(Skill Recipes)

입니다. 미리 만들어진 다단계 워크플로우 패턴이 100개 이상 내장되어 있습니다.

레시피의 종류

깃허브 저장소의 Skills 섹션에서 전체 목록을 확인할 수 있습니다. 몇 가지 예를 들어 보겠습니다.

시트에서 이벤트 생성

: 구글 시트의 데이터를 읽어 캘린더 이벤트를 자동 생성

프레젠테이션 생성

: 데이터 기반 슬라이드 덱 자동 구성

미팅 공간 확보

: 참석자들의 빈 시간을 찾아 미팅 예약

이메일 라벨링 및 아카이빙

: 조건별 이메일 자동 분류

이 레시피들은 별도 설치 없이 GWS 명령줄 방식가 설치된 상태에서 바로 사용할 수 있습니다. 에이전트가 요청의 성격을 파악해 적절한 레시피를 자동으로 선택합니다.

JSON 우선 설계의 이점

GWS 명령줄 방식의 응답은 JSON 형식으로 구조화되어 있습니다. 이것이 왜 중요한지 설명하겠습니다. AI 에이전트는 구조화된 데이터를 비구조화된 텍스트보다 훨씬 정확하게 처리합니다. API 응답이 JSON으로 돌아오면, 에이전트는 필요한 정보를 정확히 추출하고 다음 단계에 전달할 수 있습니다. 파싱 오류가 줄고, 워크플로우의 신뢰성이 높아집니다.

컨텍스트 효율성

MCP 연결 서버를 서비스마다 개별 연결하면 각각이 컨텍스트를 소비합니다. Gmail MCP, Drive MCP, Calendar MCP를 각각 설정하면 설정 정보만으로도 상당한 컨텍스트가 사용됩니다. GWS 명령줄 방식는 하나의 도구이므로 컨텍스트 소비가 최소화됩니다. 이 차이는 복잡한 워크플로우에서 체감됩니다.

〔그림 19-5〕 GWS 명령줄 방식 깃허브 페이지의 스킬 레시피 목록 캡처〕

현재의 한계와 전망

커뮤니티의 반응은 양분되어 있습니다. 트위터에서는 "놀라울 정도로 강력하다"는 평가와 "다소 불안정하다"는 평가가 공존합니다. 일부 사용자는 인증을 여러 번 반복해야 하는 현상을 보고합니다.

실제 사용 경험에서는 검색, 조회, 예약 같은 기본 기능은 거의 완벽하게 작동합니다. 슬라이드 생성처럼 시각적 정밀도가 필요한 작업에서는 아직 보완이 필요합니다. 그러나 아직 1.0 버전에 도달하지 않은 제품이라는 점을 감안하면, 지금 설치하고 익숙해지는 것이 현명한 선택입니다.

하나의 명령줄 방식 도구로 구글의 주요 서비스를 모두 제어할 수 있다는 것은, 에이전트의 행동 반경이 그만큼 넓어진다는 뜻입니다. 이메일을 분류하고, 슬라이드를 만들고, 캘린더를 관리하는 것이 모두 자연어 한 줄로 가능해집니다. 이 도구를 유튜브 데이터 분석과 결합하면 어떤 워크플로우가 만들어질지 살펴보겠습니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

20장 유튜브 분석 작업 흐름과 이메일 자동 발송

전체 목차

책의 모든 장 보기

유튜브 채널 데이터 웹 정보 수집과 트렌드 분석

VS Code가 열려 있습니다. 왼쪽 파일 탐색기에는 "YouTube Analysis"라는 프로젝트 폴더가 비어 있는 채로 기다리고 있습니다. 오른쪽에는 클로드 코드의 대화창이 깜빡이고 있습니다. 이 빈 폴더에서 출발해, 유튜브 채널 데이터를 수집하고, 트렌드를 분석하며, 전문적인 보고서를 만들어 이메일로 발송하는 자동화 시스템을 구축하려 합니다.

이번 장은 지금까지 다뤄 온 MCP, 스킬, GWS 명령줄 방식, WAT 프레임워크를 하나의 워크플로우 안에서 통합하는 사례입니다.

프로젝트 초기화

가장 먼저 해야 할 일은

claude.md

파일을 프로젝트에 배치하는 것입니다. 이 파일은 에이전트의 시스템 프롬프트(System Prompt) 역할을 합니다. WAT 프레임워크의 구조 — 워크플로우, 에이전트, 도구의 3계층 — 를 에이전트에게 알려 줍니다.

파일을 배치한 뒤 클로드 코드에 "이

claude.md

파일을 기반으로 프로젝트를 초기화해 주세요"라고 요청합니다. 에이전트는

temp

,

tools

,

workflows

폴더를 생성하고,

.env

파일과

.gitignore

를 구성합니다. 프로젝트의 뼈대가 잡혔습니다.

계획 모드에서 요구사항 전달

계획 모드로 전환합니다. 이 단계의 중요성은 이전 장에서 충분히 강조했으므로, 바로 실습에 들어가겠습니다.

"AI와 AI 자동화 분야의 유튜브 영상과 채널을 대량으로 스크래핑하는 자동화를 만들고 싶습니다.

어떤 영상이 트렌딩인지, 무엇이 잘 되고 있는지, AI 업계의 분위기는 어떤지 파악하고 싶습니다.

그래야 사람들이 원하는 콘텐츠를 더 만들 수 있습니다.

이 데이터를 어떻게 가져올 수 있는지 알려 주세요.

API나 MCP 서버를 조사해 주세요.

유용한 스킬이 있는지도 확인해 주세요.

리서치가 끝나면 슬라이드 덱을 만들어 주세요.

차트, 이미지, 그래픽이 포함된 전문적인 슬라이드 덱이어야 합니다.

완성되면 Gmail로 보내 주세요."

이 요청은 의도적으로 "브레인 덤프(brain dump)" 형태입니다. 완벽하게 정제된 요구사항이 아닙니다. 계획 모드의 힘은 바로 이 지점에서 발휘됩니다.

에이전트의 리서치와 질문

에이전트는 단순히 생각만 하지 않습니다. 웹 검색을 수행하며 유튜브 데이터를 웹 정보 수집하는 방법과 MCP 연결 서버 활용법을 조사합니다. 그리고 질문이 돌아옵니다.

추적할 채널

: "특정 유튜브 채널 목록이 있나요, 아니면 제가 AI 자동화 분야의 인기 채널을 자동으로 찾을까요?"

빈도

: "보고서를 얼마나 자주 받고 싶으신가요?"

데이터 저장

: "구글 시트에도 데이터를 기록할까요?"

전달 방식

: "보고서를 어떤 이메일 주소로 보낼까요?"

각 질문에 답합니다. 채널은 자동 발견, 빈도는 주간, 시트 기록은 포함, 이메일은 본인의 Gmail로. 이 답변들이 요구사항 문서의 빈칸을 채워 나갑니다.

〔그림 20-1〕 계획 모드에서 에이전트가 질문을 던지고 답변을 수집하는 화면〕

실행 계획 수립

에이전트가 제시한 계획의 규모가 상당합니다.

7개의 파이썬 도구

를 생성하겠다는 것입니다.

1.

fetch_youtube_data.py

— 유튜브 데이터 수집

2.

analyze_youtube_data.py

— 수집 데이터 분석

3.

generate_charts.py

— 차트 이미지 생성

4.

generate_slides.py

— 슬라이드 덱 구성

5.

send_email_report.py

— 이메일 보고서 발송

6.

export_sheets.py

— 구글 시트 내보내기

7.

discover_channels.py

— AI 채널 자동 발견

워크플로우는

youtube_weekly_report

라는 이름으로 생성됩니다. 이 워크플로우가 7개 도구를 순서대로 호출하는 지휘자 역할을 합니다.

계획을 승인하면 에이전트는 할 일 목록을 만들고 하나씩 실행합니다. 이 동안 다른 모니터에서 다른 작업을 하면서 가끔 진행 상황을 확인하면 됩니다.

의존성 설치와 API 키 설정

도구 생성이 완료되면 에이전트가 다음 단계를 안내합니다.

파이썬 의존성 설치

YouTube Data API 키 등록

Gmail과 구글 시트용 OAuth 인증 설정

API 키는

.env

파일에 저장합니다. 에이전트에게 "의존성을 설치해 주세요. YouTube API 키는 제가 가져오겠습니다"라고 말하면 됩니다. API 키를 클로드 코드에 전달할 때는

.env

파일에만 기록되도록 확인합니다.

OAuth 인증을 위해서는 구글 클라우드 프로젝트에서 YouTube Data API, Gmail API, Google Sheets API를 활성화해야 합니다. 이 과정은 제19장의 GWS 명령줄 방식 설치에서 다룬 것과 동일합니다.

첫 검증 실행

모든 설정이 완료되면 에이전트가 자체적으로 파이프라인을 검증합니다. 결과는 다음과 같습니다.

30개 채널 발견

187개 영상 수집

분석 보고서 생성

6개 차트 이미지 생성

9장 슬라이드의 파워포인트 덱 생성

구글 시트로 데이터 내보내기

이메일로 보고서 발송

〔그림 20-2〕 첫 검증 실행 결과 요약 — 수집된 데이터의 규모〕

슬라이드 덱 자동 생성

이메일로 도착한 보고서를 열어 봅니다. "AI Automation YouTube Analytics"라는 제목의 주간 보고서입니다.

이메일 본문

30개 채널 추적, 187개 영상 분석이라는 핵심 수치가 상단에 배치되어 있습니다. 이번 주의 상위 영상 목록과 콘텐츠 제작 추천 사항이 본문에 포함되어 있습니다.

파워포인트 덱의 구성

첨부된 파워포인트 파일을 열면 다음과 같은 슬라이드들이 나타납니다.

슬라이드 번호

내용

1

표지 — 보고서 제목, 날짜

2

핵심 지표 — 중위 조회수, 중위 참여율

3

트렌딩 토픽 — 주요 키워드 분석

4

상위 영상 — 제목별, 조회수별 정렬

5

상위 채널 — 구독자 수 기준

6

참여율 분석

7

트렌딩 토픽 키워드 분포

8

포스팅 패턴 분석

9

추천 사항

〔그림 20-3〕 자동 생성된 파워포인트 슬라이드 덱 화면 캡처〕

구글 시트의 데이터 구조

보고서와 함께 구글 시트에도 데이터가 기록됩니다. 시트에는 3개의 탭이 있습니다.

채널 통계(Channel Stats)

: 채널 ID, 채널명, 구독자 수, 총 조회수, 영상 수가 날짜별로 기록됩니다. 매주 실행할 때마다 새 행이 추가되므로, 시간 경과에 따른 채널 성장을 추적할 수 있습니다.

상위 영상(Top Videos)

: 영상 ID, 제목, 채널명, 조회수, 좋아요 수, 댓글 수, 참여율, 게시 후 경과 일수가 포함됩니다. 참여율과 경과 일수를 함께 보면 "지금 뜨고 있는" 영상을 정확히 파악할 수 있습니다.

주간 요약(Weekly Summary)

: 실행 일자, 추적 채널 수, 분석 영상 수, 중위 조회수, 중위 참여 점수, 상위 키워드가 한 줄로 요약됩니다. 주간 데이터가 축적되면 업계 트렌드의 변화를 한눈에 파악할 수 있습니다.

PDF 보고서로의 전환

파워포인트 대신 브랜딩이 적용된 PDF 보고서를 원한다면, 캔버스 디자인(Canvas Design) 스킬을 활용할 수 있습니다. 클로드 코드 템플릿 웹사이트에서 이 스킬을 찾아 설치합니다. 설치 명령어를 클로드 코드에 붙여넣으면

.claude/skills

폴더에 스킬 파일과 폰트가 생성됩니다.

"캔버스 디자인 스킬을 방금 설치했습니다.

파워포인트 대신, 이 스킬을 사용해 PDF 보고서를 만들어 주세요.

전문적이면서 보기 좋게 만들어 주세요.

프로젝트 폴더에 넣어 둔 로고 PNG 파일을 모든 페이지에 포함해 주세요."

에이전트는 기존의

generate_slides.py

도구를 새로운 PDF 생성 도구로 교체하고, 워크플로우 파일을 업데이트합니다. 이메일 발송 도구도 첨부 파일 형식을 파워포인트에서 PDF로 변경합니다.

첫 번째 PDF 생성 결과에서 문제가 발생할 수 있습니다. 표지와 마지막 페이지만 포함되고 본문 차트가 빠지는 경우입니다. 이때 자연어로 수정 의견합니다.

"PDF가 2페이지뿐입니다. 표지와 마지막 페이지만 있고 차트가 빠졌습니다."

에이전트는 원인을 찾고, 도구를 수정하고, 워크플로우를 업데이트한 뒤, 9페이지짜리 완성된 PDF를 새로 생성합니다. 로고가 상단에 배치되고, 날짜가 표시되며, 모든 차트와 추천 사항이 포함됩니다.

〔그림 20-4〕 브랜딩이 적용된 PDF 보고서의 완성 화면〕

Gmail 연동: 보고서 자동 발송

워크플로우의 마지막 단계는 완성된 보고서를 Gmail로 발송하는 것입니다.

이메일 발송 도구의 구성

send_email_report.py

도구는 다음 요소들을 처리합니다.

수신자 설정

:

.env

파일에 저장된 이메일 주소

제목 생성

: "AI Automation YouTube Analytics — Weekly Report 〔날짜〕"

본문 구성

: 핵심 수치 요약, 상위 영상 목록, 추천 사항

첨부 파일

: PDF 보고서 (또는 파워포인트)

GWS 명령줄 방식를 통해 Gmail API를 호출하므로, 별도의 SMTP 설정이나 앱 비밀번호가 필요 없습니다. OAuth 인증이 완료된 상태라면

gws

명령 한 줄로 이메일이 발송됩니다.

구글 시트 동시 업데이트

이메일 발송과 동시에 구글 시트에도 데이터가 기록됩니다. 시트를 직접 생성하지 않아도 됩니다. 에이전트가 시트를 새로 만들거나 기존 시트에 탭을 추가하고, 스키마(열 구성)를 설정하며, 데이터를 입력합니다. 매주 실행할 때마다 새 행이 추가되므로 시계열 데이터가 자연스럽게 축적됩니다.

검증와 최적화

워크플로우가 완성되었다고 해서 바로 배포할 수는 없습니다. 검증와 최적화 단계가 남아 있습니다.

반복 개선의 실제

첫 실행에서 모든 것이 완벽하게 작동하는 경우는 드뭅니다. 유튜브 분석 워크플로우에서도 여러 번의 수정이 필요했습니다.

PDF 보고서에 차트가 누락되는 문제 → 도구 코드 수정

폰트 렌더링 오류 → 추가 의존성 설치

슬라이드 간격 불균형 → 시각적 검증 루프 적용

각 문제를 발견할 때마다 에이전트에게 자연어로 전달하면, 에이전트가 원인을 조사하고 수정합니다. 수정 사항은 도구 파일과 워크플로우 파일에 반영되어 다음 실행에서는 같은 문제가 재발하지 않습니다.

보안 검토

배포 전에 반드시 보안 검토를 수행해야 합니다.

"보안 검토를 해 주세요.

API 키가 노출되어 있지 않은지,

취약점은 없는지 확인해 주세요."

에이전트는 코드 전체를 검토하며 다음 항목을 점검합니다.

API 키가

.env

파일 밖에 노출되어 있지 않은지

웹훅(웹훅)이 외부에 공개되어 있지 않은지

인증 모델이 세는 글 조각이 코드에 하드코딩되어 있지 않은지

공개 저장소에 커밋될 위험이 있는 파일은 없는지

보안 검토에서 주의가 필요한 항목이 발견되더라도, 실제로 취약한 상태는 아닐 수 있습니다. 에이전트가 "3개의 주의 항목이 있지만, 모든 시크릿은 안전하게 관리되고 있습니다"라고 보고하면 배포를 진행해도 됩니다.

〔그림 20-5〕 보안 검토 결과 요약 화면〕

배포: Modal을 활용한 스케줄 실행

워크플로우를 주간 스케줄로 운영하려면 클라우드에 배포해야 합니다. Modal(모달)이라는 AI 인프라 플랫폼을 활용합니다. Modal은 자동화가 실행될 때만 과금되므로 비용 효율적입니다. 무료 크레딧도 제공합니다.

배포 과정은 이렇습니다.

"유튜브 분석 워크플로우를 Modal에 배포하고 싶습니다.

매주 월요일 오전 6시에 자동 실행되도록 해 주세요."

에이전트는 다음 작업을 순서대로 수행합니다.

1. 스크립트들의 환경 변수 경로 업데이트

2. Modal 배포 파일 생성

3. 크론(Cron) 스케줄 설정

4. API 키와 인증 모델이 세는 글 조각을 Modal 시크릿으로 등록

5. 배포 실행 및 검증

배포가 완료되면 Modal 대시보드에서 앱을 확인할 수 있습니다. 수동 실행(manual trigger)으로 한 번 더 검증한 뒤, 정상 작동이 확인되면 스케줄 트리거를 활성화합니다.

배포 후의 유지보수

앞 장에서 언급한 중요한 구분을 다시 한 번 상기합니다. 배포되는 것은 워크플로우와 도구입니다. 에이전트는 배포되지 않습니다. 따라서 배포된 워크플로우가 실행 중에 새로운 오류를 만나면 자기 치유가 이루어지지 않습니다.

유지보수의 흐름은 이렇습니다.

1. Modal에서 실행 로그 확인

2. 오류 발생 시 클로드 코드로 복귀

3. 로컬에서 워크플로우 수정 및 테스트

4. 수정된 버전을 Modal에 재배포

워크플로우가 안정화되면 이 주기는 점차 줄어듭니다. 초기에는 주 1에서 2회 확인이 필요하지만, 엣지 케이스가 해소될수록 자동 실행만으로 충분해집니다.

전체 아키텍처 정리

유튜브 분석 워크플로우의 전체 구조를 한눈에 정리합니다.

〔claude.md 시스템 프롬프트〕

|

▼

〔워크플로우: youtube_weekly_report.md〕

|

├── discover_channels.py ─── YouTube Data API

├── fetch_youtube_data.py ─── YouTube Data API

├── analyze_youtube_data.py

├── generate_charts.py

├── generate_pdf_report.py ─── Canvas Design 스킬

├── export_sheets.py ─── GWS CLI → Google Sheets

└── send_email_report.py ─── GWS CLI → Gmail

〔그림 20-6〕 유튜브 분석 워크플로우의 전체 아키텍처 다이어그램〕

MCP(Firecrawl), API(YouTube Data), 명령줄 방식(GWS), 스킬(Canvas Design), 그리고 WAT 프레임워크가 하나의 워크플로우 안에서 유기적으로 결합되어 있습니다. 이 모든 것을 조율한 것은 자연어 대화뿐입니다.

빈 폴더에서 시작해 주간 유튜브 트렌드 보고서가 자동으로 이메일함에 도착하기까지, 전 과정을 밟아 보았습니다. 하나의 프롬프트에서 출발해 에이전트와 질문을 주고받으며 7개의 도구와 하나의 워크플로우를 만들고, 검증하고, 수정하고, 배포했습니다.

여기서 얻은 감각 — 목표를 자연어로 설명하고, 에이전트의 질문에 답하며, 결과물을 확인하고 수정 의견하는 리듬 — 은 이후 어떤 종류의 자동화를 구축하든 동일하게 적용됩니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

21장 4단계 프레임워크: 집, 생명, 손, 성장

전체 목차

책의 모든 장 보기

도입

화면 네 개가 동시에 돌아가고 있습니다. 하나는 오늘 일정을 짜주고, 다른 하나는 링크드인 포스트를 써주며, 세 번째는 팀의 프로젝트 진행 상황을 점검하고, 네 번째는 유튜브 영상용 시각 자료를 만들어냅니다. 미국의 한 AI 자동화 컨설턴트가 매일 아침 출근하면 마주하는 장면입니다. 네 개의 에이전트가 병렬로 돌아가는 데 걸리는 시간은 약 2분. 같은 작업을 손으로 처리하면 25분이 넘게 걸립니다.

이 컨설턴트의 프로젝트 폴더 왼쪽 패널에는 수십 개의 파일과 폴더가 빼곡하게 들어차 있습니다. 컨텍스트 파일, 의사결정 로그, 스킬 폴더, 에이전트 설정까지. 처음 보는 사람은 압도당하기 쉽습니다. 그러나 이 구조물은 하루아침에 만들어진 것이 아닙니다. 매일 조금씩, 네 개의 단계를 밟아 쌓아 올린 결과물입니다.

그 네 단계를 Home(집), Life(생명), Hands(손), Growth(성장)라고 부릅니다.

1단계 '집': 프로젝트 폴더 구조와 claude.md 설계

건물을 짓기 전에 터를 닦듯, AI 이그제큐티브 어시스턴트(Executive Assistant)를 만들려면 먼저 그것이 살아갈 공간을 마련해야 합니다. 여기서 '집'이란 프로젝트 폴더 하나를 의미합니다.

VS Code(Visual Studio Code)를 열고, 바탕화면이든 원하는 위치든 새 폴더를 하나 만듭니다. 이름은 자유롭게 정하되, 이 폴더가 곧 어시스턴트의 전체 작업 공간이 된다는 점을 기억해 두십시오. VS Code 왼쪽 탐색기에서 "Open Folder"를 클릭해 방금 만든 폴더를 여는 순간, 빈 캔버스가 펼쳐집니다. 아직 파일은 하나도 없습니다.

〔그림 21-1〕 VS Code에서 빈 프로젝트 폴더를 열었을 때의 화면 구성〕

이제 이 빈 공간에 첫 번째 파일을 놓아야 합니다. 왼쪽 패널에서 "New File"을 클릭하고 파일 이름을

CLAUDE.md

로 입력합니다. 이 파일이 어시스턴트의 두뇌 역할을 합니다. 에이전트가 사용자의 메시지를 읽기 전에, 매번 이 파일부터 로드하기 때문입니다.

CLAUDE.md

에 모든 정보를 욱여넣으면 어떻게 될까요? 매 대화마다 방대한 텍스트가 로드되어 모델이 세는 글조각(Token)을 낭비하게 됩니다. 그래서 이 파일에는 핵심 규칙과 "어디에 무엇이 있는지"를 알려주는 경로 안내만 적어 둡니다. 실제 상세 정보는 별도 파일에 담고,

CLAUDE.md

는 라우터(Router) 역할을 하도록 설계하는 것입니다.

CLAUDE.md 구조 예시

역할

이 폴더는 사용자의 이그제큐티브 어시스턴트 작업 공간입니다.

컨텍스트 위치

- 사용자 정보: context/me.md

- 업무 정보: context/work.md

- 팀 정보: context/team.md

- 현재 우선순위: context/priorities.md

스킬 위치

- .claude/skills/ 하위 폴더 참조

커뮤니케이션 스타일

- 간결하고 직접적인 어조

〔그림 21-2〕 CLAUDE.md의 라우팅 구조 다이어그램〕

시작 단계에서

CLAUDE.md

는 10에서 20줄이면 충분합니다. 이 파일은 어시스턴트가 성장하면서 함께 진화할 것이므로, 완벽하게 채우려고 애쓸 필요가 없습니다. 87줄 정도가 적정선이며, 150줄, 넉넉잡아 200줄을 넘기지 않는 것이 좋습니다.

에이전트에게 "이 폴더는 네가 나의 이그제큐티브 어시스턴트로 활동할 공간이야.

CLAUDE.md

에 간단한 설명을 넣어 줘"라고 말하면, 에이전트가 알아서 초기 내용을 작성해 줍니다. 이렇게 집의 기초가 놓입니다.

폴더 구조는 에이전트가 자동으로 생성해 주기도 합니다. 일반적으로 다음과 같은 뼈대가 잡힙니다.

프로젝트 루트/

├── CLAUDE.md

```
├── .claude/
│   ├── skills/
│   ├── agents/
│   └── rules/
├── context/
│   ├── me.md
│   ├── work.md
│   ├── team.md
│   └── priorities.md
├── decisions/
├── projects/
├── references/
├── templates/
└── archives/
```

〔그림 21-3〕 이그제큐티브 어시스턴트 프로젝트의 표준 폴더 구조〕

각 폴더의 역할은 이름에서 짐작할 수 있습니다.

context

에는 사용자와 업무에 관한 정보가,

decisions

에는 중요한 의사결정 기록이,

projects

에는 진행 중인 프로젝트별 하위 폴더가 들어갑니다. 이 구조를 외울 필요는 없습니다. 에이전트에게 "이 폴더가 뭘 하는 거야?"라고 물으면 설명해 줄 테니까요.

여기서 한 가지 실용적인 조언이 있습니다. 이 프로젝트를 깃허브(GitHub)에 올려 두면 어떤 기기에서든 리포지토리를 끌어와 어시스턴트를 바로 사용할 수 있습니다. 백업, 롤백, 버전 관리까지 자동으로 따라옵니다. 물론 내 컴퓨터 깃(Git) 커밋만으로 시작해도 괜찮습니다. 깃허브 연동은 나중에 해도 됩니다.

2단계 '생명': 인터뷰를 통한 컨텍스트 구축

빈 집에 가구를 들이고, 냉장고를 채우고, 벽에 사진을 거는 과정이 바로 2단계입니다. 에이전트가 사용자를 알아가는 시간입니다.

이 과정은 인터뷰 형식으로 진행됩니다. 미리 준비된 온보딩 프롬프트(Onboarding Prompt)를 에이전트에게 전달하면, 에이전트가 질문을 던지기 시작합니다. "이름이 무엇인가요?" "역할은요?" "시간대는 어디인가요?" 기본적인 질문부터 시작해서, 사업 내용, 팀 구성, 현재 목표, 소통 방식까지 차근차근 파고듭니다.

이때 핵심 원칙은 '솔직하게, 충분히'입니다. 에이전트가 물어보는 질문에 성의 있게 답할수록, 나중에 "아, 이것도 알려줬어야 했는데"라며 같은 맥락을 반복 설명하는 횟수가 줄어듭니다. 답을 모르는 항목은 "스킵"이라고 말해도 됩니다. 나중에 채워 넣을 수 있으니까요.

에이전트의 질문은 대략 여섯 개 영역을 다룹니다.

1.

개인 정보

: 이름, 역할, 시간대, 업무 스타일

2.

사업과 업무

: 회사명, 사업 분야, 주요 제품이나 서비스

3.

팀

: 함께 일하는 핵심 인물, 역할 분담

4.

우선순위와 목표

: 이번 분기 목표, 긴급 과제, 연간 방향

5.

소통 선호

: 간결한 답변을 좋아하는지, 상세한 설명을 원하는지

6.

자동화 희망 업무

: 반복되는 일, 위임하고 싶은 업무

인터뷰가 끝나면 에이전트는 답변을 바탕으로 네 개의 핵심 파일을 생성합니다.

me.md

: 사용자 개인에 관한 정보. 이름, 배경, 성향, 선호하는 소통 방식.

work.md

: 사업과 회사에 관한 정보. 업종, 제품, 비즈니스 모델.

team.md

: 팀원 정보. 이름, 역할, 담당 업무.

priorities.md

: 현재 집중하고 있는 과제. 분기 목표, 긴급 사안, 마일스톤.

〔그림 21-4〕 인터뷰 기반 컨텍스트 파일 생성 흐름도〕

이 파일들은

context

폴더 안에 저장됩니다. 그리고

CLAUDE.md

는 자동으로 업데이트되어, "사용자 정보가 필요하면

context/me.md

를 읽어라"는 식으로 경로를 안내하게 됩니다.

여기에 추가적인 파일도 생성됩니다.

decisions/log.md

에는 주요 의사결정이 날짜, 결정 내용, 이유, 맥락과 함께 기록됩니다. 프로젝트별 폴더에는 각 프로젝트의 설명 파일이 들어갑니다.

.claude/rules/

폴더에는 소통 스타일 규칙이 저장됩니다. 엠 대시(em dash)를 쓰지 말 것, 내부 소통은 캐주얼하게 할 것 같은 세부 지침들입니다.

이 모든 과정이 끝나면 에이전트는 초기 깃 커밋을 수행합니다. 첫 번째 스냅샷이 저장되는 것입니다.

한 가지 기억해 둘 점이 있습니다. 이 파일들은 고정된 문서가 아닙니다. 사업 방향이 바뀌면

work.md

를, 새 팀원이 합류하면

team.md

를 업데이트하면 됩니다. 에이전트에게 "이거 기억해 둬"라고 말하면, 해당 정보가 적절한 파일에 반영됩니다.

3단계 '손': 첫 번째 스킬 만들기

집을 짓고 생명을 불어넣었으니, 이제 손을 달아 줄 차례입니다. 에이전트가 생각만 하는 것이 아니라 실제로 무언가를 수행할 수 있게 만드는 단계입니다.

스킬(Skill)은 재사용 가능한 지시문입니다. 한번 작성해 두면, 슬래시 명령이나 자연어로 언제든 호출할 수 있고, 매번 같은 프로세스를 따르기 때문에 결과의 일관성이 높아집니다. 앞서 워크플로(Workflow)라는 개념을 다뤘는데, 스킬은 워크플로의 또 다른 이름이라고 보면 됩니다. 워크플로에 도구(Tool)가 있었듯, 스킬에는 파이썬 스크립트가 있습니다. 본질은 같습니다.

첫 번째 스킬로 무엇을 만들면 좋을까요? 프로젝트 관리 도구와의 연동이 실용적인 선택입니다. 클릭업(ClickUp), 노션(Notion), 아사나(Asana) 등 자신이 쓰는 도구의 API 키를

.env

파일에 넣고, 에이전트가 해당 도구와 소통할 수 있도록 스킬을 구성하는 것입니다.

스킬을 만드는 과정을 리서치 스킬을 예로 살펴보겠습니다. 계획 모드(Plan Mode)에서 에이전트에게 이렇게 말합니다. "리서치 스킬을 만들어 줘. 퍼플렉시티(Perplexity) API를 쓸 거야.

.env

파일을 먼저 만들어 주고, 스킬이 하는 일은 단순 웹 검색이 아니라 내 사업 맥락을 반영한 깊이 있는 조사야."

에이전트는 프로젝트 구조를 탐색하고, 필요하면 보조 에이전트(Subagent)를 띄워 구조를 분석한 뒤, 계획을 세워 돌아옵니다. 그 계획에는 다음이 포함됩니다.

.env

파일 생성 (API 키 보관용)

.claude/skills/research/skill.md

작성

CLAUDE.md

업데이트 (새 스킬 등록)

〔그림 21-5〕 스킬의 기본 구조: YAML 프론트매터 + 단계별 지시문〕

스킬 파일의 구조는 명쾌합니다. 상단에 YAML(YAML Ain't Markup Language) 프론트매터가 있고, 그 아래에 마크다운으로 작성된 단계별 지시문이 따릅니다. 프론트매터에는 스킬 이름과 설명이 들어가며, 에이전트는 이 부분만 읽고 어떤 스킬을 호출할지 판단합니다. 전체 지시문은 스킬이 선택된 뒤에야 로드됩니다.

이 방식을 점진적 컨텍스트 로딩(Progressive Context Loading)이라 하며, 모델이 세는 글 조각 낭비를 막는 핵심 메커니즘입니다.

스킬이 완성되면 검증해 봐야 합니다. "리서치 스킬을 써서 포틀랜드의 아이스크림 행사를 조사해 줘"라고 입력하면, 에이전트가 스킬을 읽고, 퍼플렉시티 API를 호출하고, 결과를 정리해서

research/

폴더에 보고서로 저장합니다. 보고서에는 출처 링크까지 포함됩니다.

보조 에이전트를 활용하면 비용을 절감할 수 있습니다. 메인 에이전트는 Opus 모델을 쓰지만, 리서치를 보조 에이전트에 위임하면서 Haiku처럼 가벼운 모델을 지정할 수 있습니다. 리서치의 질은 퍼플렉시티 API가 보장하고, 정리와 요약만 가벼운 모델이 맡으니 품질 저하 없이 비용이 줄어드는 구조입니다.

보조 에이전트는

.claude/agents/

폴더에 마크다운 파일로 저장됩니다. 자체 컨텍스트 윈도(Context Window)를 갖고 독립적으로 작동합니다.

4단계 '성장': 매일 사용하며 어시스턴트를 진화시키기

4단계는 따로 설정할 것이 없습니다. 매일 사용하는 것 자체가 4단계입니다.

앞서 언급한 AI 자동화 컨설턴트의 어시스턴트가 처음부터 수십 개의 스킬과 촘촘한 컨텍스트를 갖추고 있었던 것은 아닙니다. 첫날에는 빈 폴더 하나와

CLAUDE.md

한 줄이 전부였습니다. 한 달 뒤, 그 프로젝트는 완전히 다른 모습이 되었습니다. 문서가 늘었고, 의사결정 로그가 쌓였고, 스킬이 추가되었으며, 에이전트의 응답 정밀도가 눈에 띄게 올라갔습니다.

성장의 비결은 수정 의견 순환(Feedback Cycle)에 있습니다. 스킬을 호출하고, 에이전트가 작업하는 과정을 관찰하고, "이 부분은 좋았고, 저 부분은 고쳐야 해"라고 말하면, 에이전트가 스킬 파일을 수정합니다. 처음 두세 번은 결과물이 AI가 생성한 티가 날 수 있습니다. 하지만 열 번, 스무 번 반복하면, 사용자의 취향과 업무 맥락이 스킬에 깊이 각인됩니다.

구체적인 성장 전략은 다음과 같습니다.

기존 도구를 이주시키기.

ChatGPT의 커스텀 GPT나 클로드 프로젝트에 저장해 둔 시스템 프롬프트가 있다면, 그 지시문을 가져와 스킬로 전환합니다. "이 프롬프트를 스킬로 만들어 줘"라고 에이전트에게 맡기면 됩니다.

반복 작업을 포착하기.

같은 지시를 두 번 이상 내리고 있다면, 그것은 스킬로 만들 후보입니다. 엠 대시를 쓰지 말라고 매번 말하고 있다면, 그 규칙은 스킬이나 규칙 파일에 한번 적어 두는 편이 낫습니다.

에이전트의 작업 과정을 관찰하기.

처음 몇 번은 에이전트가 스킬을 실행하는 과정을 지켜봐야 합니다. 불필요한 API 호출을 반복하고 있다면, 자주 쓰는 ID 값을 스킬 파일에 하드코딩해서 모델이 세는 글 조각과 시간을 아낄 수 있습니다. 이 관찰과 개선의 루프가 성장을 가속합니다.

새 폴더를 자유롭게 추가하기.

브랜드 에셋(Brand Asset) 폴더를 만들어 로고, 폰트, 브랜드 가이드라인을 넣어 두면, 콘텐츠 생성 스킬이 브랜드 일관성을 유지할 수 있습니다. 폴더를 추가한 뒤 에이전트에게 "brand-assets 폴더를 추가했어.

CLAUDE.md

를 업데이트해 줘"라고 말하면 됩니다.

〔그림 21-6〕 시간에 따른 어시스턴트의 진화 곡선: 1일차 vs 30일차〕

자동 메모리 기능도 활용할 수 있습니다. "나는 항상 X를 선호한다는 걸 기억해 둬"라고 말하면, 에이전트가 해당 내용을 적절한 파일에 저장하고 이후 대화에서 자동으로 반영합니다.

기존 ChatGPT·커스텀 GPT와 차별화되는 지점

ChatGPT나 클로드 웹 인터페이스에서 메모리를 저장하고, 커스텀 프롬프트를 설정하는 것만으로도 생산성은 올라갑니다. 그러나 한계가 있습니다. "이 맥락도 알아줬으면 좋겠는데"라고 느끼는 순간, 결국 추가 설명을 타이핑하고 있습니다. 50%까지는 빠르게 도달하지만, 90%에 이르기가 어렵습니다.

차이가 발생하는 지점을 살펴보겠습니다.

파일 시스템 전체에 대한 접근.

커스텀 GPT는 대화 창 안에서만 작동합니다. 그러나 코드 기반 이그제큐티브 어시스턴트는 프로젝트 폴더 안의 모든 파일을 읽고 쓸 수 있습니다.

me.md

,

work.md

,

priorities.md

, 리서치 보고서, 의사결정 로그 — 에이전트가 필요한 순간에 필요한 파일을 직접 열어봅니다. 대화창에 모든 맥락을 붙여 넣을 필요가 없습니다.

도구 실행 능력.

커스텀 GPT는 텍스트를 생성합니다. 이그제큐티브 어시스턴트는 파이썬 스크립트를 실행하고, API를 호출하고, 파일을 생성하며, 보조 에이전트를 띄웁니다. 캘린더를 확인하고 일정을 자동으로 블로킹하는 것, 클릭업에서 태스크 상태를 가져오는 것, 이 모든 동작이 대화 한 줄로 가능합니다.

컨텍스트의 영속성과 구조화.

대화가 길어지면 컨텍스트 윈도 한계에 부딪힙니다. 이그제큐티브 어시스턴트는 중요한 정보를 파일로 분리해 영구 저장합니다. 대화를 초기화해도 파일은 남아 있으므로, 다음 세션에서 곧바로 이전 맥락을 이어갈 수 있습니다. 기억을 잃지 않는 비서인 셈입니다.

성장 가능성.

커스텀 GPT의 시스템 프롬프트는 정적입니다. 수동으로 수정해야 합니다. 이그제큐티브 어시스턴트는 사용할수록 파일이 늘어나고, 스킬이 정교해지며, 의사결정 이력이 축적됩니다. 에이전트에게 "이 경험을 기록해 둬"라고 말하면, 다음에 비슷한 상황이 왔을 때 더 나은 판단을 내릴 수 있는 기반이 됩니다.

〔그림 21-7〕 기존 ChatGPT 방식과 이그제큐티브 어시스턴트 방식의 비교표〕

항목

ChatGPT / 커스텀 GPT

이그제큐티브 어시스턴트

맥락 범위

대화 창 내

프로젝트 폴더 전체

도구 실행

제한적

스크립트, API, 서브에이전트

정보 영속성

대화 종료 시 소실 가능

파일로 영구 저장

성장 방식

수동 프롬프트 수정

사용하며 자동 진화

마무리

네 개의 단계는 순서대로 밟되, 속도는 자유롭게 조절하면 됩니다. 집을 짓는 데 10분이면 충분하고, 생명을 불어넣는 인터뷰는 시간을 들여 정성껏 할수록 이후가 편해집니다. 첫 스킬은 작게 시작하되, 매일 한두 가지씩 수정 의견을 주면서 키워 나갑니다.

이 프레임워크의 진짜 힘은 3단계에서 만든 첫 번째 스킬이 매일 아침 실제로 작동하기 시작할 때 드러납니다. 캘린더를 읽고, 프로젝트 상태를 확인하고, 오늘 무엇을 해야 하는지 정리해 주는 그 스킬 — '모닝커피'라 불리는 루틴이 어떻게 하루를 바꿔 놓는지, 이어지는 이야기에서 확인해 볼 수 있습니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

22장 모닝커피 스킬과 일일 업무 자동화

전체 목차

책의 모든 장 보기

도입

아침 8시 47분. 커피 한 잔을 내리는 동안 VS Code를 열고 한 줄을 입력합니다. "오늘은 아침인 것처럼 해 줘. 모닝커피 스킬로 하루를 계획해 줘." 에이전트가 움직이기 시작합니다. 캘린더를 읽고, 프로젝트 관리 도구에 접속하고, 분기 목표 파일을 꺼내 봅니다. 커피가 다 내려질 즈음, 화면에는 오늘의 일정이 시간 단위로 배치되어 있습니다.

이것이 모닝커피 스킬입니다. 이름은 가볍지만, 그 안에서 벌어지는 일은 상당히 촘촘합니다. 캘린더, 태스크 관리, 분기 목표라는 세 개의 정보 소스를 하나로 엮어, 의사결정의 피로를 아침부터 제거해 주는 자동화 루틴입니다.

캘린더, 프로젝트 관리, 분기 목표를 통합하는 아침 브리핑

모닝커피 스킬이 호출되면 에이전트는 세 곳을 순서대로 확인합니다.

첫 번째, 캘린더.

오늘 예정된 미팅, 통화, 일정을 가져옵니다. 구글 캘린더(Google Calendar)와 연동되어 있다면, MCP(Model Context Protocol) 서버나 API를 통해 일정 데이터를 읽어 들입니다. 몇 시에 어떤 미팅이 잡혀 있는지, 빈 시간대는 어디인지 파악하는 것이 이 단계의 목적입니다.

두 번째, 프로젝트 관리 도구.

클릭업이나 노션에 접속해서 이번 주 태스크 목록과 상태를 확인합니다. 마감이 임박한 태스크, 진행 중인 태스크, 아직 시작하지 않은 태스크를 구분합니다. 에이전트는 이 정보를 자체적으로 판단하지 않습니다.

context/priorities.md

에 적힌 우선순위 기준에 따라 중요도를 매깁니다.

세 번째, 분기 목표.

context/goals.md

파일에는 이번 분기의 핵심 목표와 마일스톤이 기록되어 있습니다. 에이전트는 현재 태스크들이 이 목표에 얼마나 기여하는지를 대조합니다.

〔그림 22-1〕 모닝커피 스킬의 세 가지 정보 소스 통합 다이어그램〕

이 세 겹의 정보를 종합한 결과물이 아침 브리핑입니다. 브리핑에는 다음이 포함됩니다.

오늘의 캘린더 일정 요약

긴급 처리가 필요한 액션 아이템

진행 중인 프로젝트의 현재 상태

콘텐츠 파이프라인(영상 제작, 글 작성 등)의 진행 현황

한 가지 주목할 부분이 있습니다. 에이전트가 브리핑을 구성할 때,

CLAUDE.md

를 먼저 읽고, 거기에 명시된 경로를 따라

me.md

,

work.md

,

team.md

,

priorities.md

를 차례로 참조합니다. 프로젝트 폴더에 있는 모든 맥락 파일이 하나의 브리핑에 녹아드는 것입니다. 이것이 20장에서 만든 '집'과 '생명'이 실제로 작동하는 순간입니다.

브리핑은 단순한 나열이 아닙니다. "이번 분기 목표 달성률이 60%인데, 이 태스크를 오늘 처리하면 70%까지 끌어올릴 수 있습니다"와 같은 판단이 들어갑니다. 우선순위 파일에 적힌 기준을 기계적으로 적용한 결과이지만, 사람이 직접 계산하려면 캘린더를 열고, 프로젝트 보드를 확인하고, 목표 문서를 꺼내야 하는 — 세 개의 앱을 오가며 머릿속에서 조합해야 하는 — 작업입니다.

우선순위 기반 일정 자동 배치

브리핑에서 한 걸음 더 나아가는 기능이 일정 자동 배치입니다. 에이전트가 제안한 하루 계획을 보고 "좋아, 이대로 해 줘"라고 답하면, 에이전트가 캘린더에 시간 블록(Time Block)을 직접 잡아 줍니다.

이 기능이 왜 강력한가를 이해하려면, 의사결정 피로(Decision Fatigue)라는 개념을 떠올려야 합니다. 아침에 "다음 15분 동안 뭘 해야 하지?" "한 시간 뒤에는?" 이런 질문을 반복하는 것 자체가 에너지를 소모합니다. 모닝커피 스킬은 이 판단을 한꺼번에 처리해 버립니다.

자동 배치의 로직은 다음과 같습니다.

1. 이미 확정된 미팅이나 약속을 캘린더에서 확인합니다.

2. 빈 시간대를 추출합니다.

3.

priorities.md

의 우선순위에 따라 태스크를 시간대에 배정합니다.

4. 집중 작업이 필요한 태스크에는 넉넉한 블록을, 짧은 행정 업무에는 작은 블록을 할당합니다.

〔그림 22-2〕 우선순위 기반 일정 자동 배치 예시 화면〕

모닝커피 스킬이 제안한 일정 예시

08:30 - 09:00 이메일 및 슬랙 확인

09:00 - 10:30 〔긴급〕 웹사이트 런칭 최종 점검

10:30 - 11:00 휴식

11:00 - 12:00 클라이언트 미팅 (기 확정)

12:00 - 13:00 점심

13:00 - 14:30 영상 스크립트 초안 작성

14:30 - 15:00 팀 슬랙 체크인

15:00 - 16:30 신규 제품 리서치

16:30 - 17:00 내일 우선순위 정리

사용자가 "좋아"라고 답하면 에이전트는 구글 캘린더 API를 호출해 각 블록을 일정으로 등록합니다. 거절하거나 수정을 요청할 수도 있습니다. "오전에 영상 작업을 먼저 하고 싶어"라고 말하면, 에이전트가 블록 배치를 조정합니다.

이 과정에서 에이전트가 수행하는 것은 두 가지입니다. 하나는 정보의 종합이고, 다른 하나는 실행입니다. 정보를 종합해서 계획을 세우는 일과 그 계획을 실제 캘린더에 반영하는 일이 하나의 대화 안에서 이루어집니다. "생각"과 "행동"의 간극이 사라지는 것입니다.

팀 펄스 체크

모닝커피가 개인의 하루를 정리한다면, 펄스 체크(Pulse Check)는 팀 전체의 현황을 한눈에 보여 줍니다.

"팀 펄스 체크를 해 줘. 이번 주와 이번 분기에 우리가 궤도에 올라 있는지 확인하고 싶어." 이 한 줄이면 에이전트는 프로젝트 관리 도구에 접속해서 모든 이니셔티브(Initiative)의 진행 상황을 수집합니다. 각 태스크의 담당자, 상태, 마감일을 가져와서

team.md

의 팀원 정보와

priorities.md

의 분기 목표에 비추어 분석합니다.

〔그림 22-3〕 펄스 체크 결과 보고서 예시〕

결과 보고서에는 이런 내용이 담깁니다.

현재 진행 중인 각 이니셔티브와 완료율

일정이 지연되고 있는 태스크와 그 담당자

분기 목표 대비 전체 진행률

수동 후속 조치가 필요한 항목 목록

바쁜 날에는 중요한 후속 조치가 빠지기 쉽습니다. 영상 촬영에 집중하느라, 팀원이 기다리고 있는 승인 요청을 놓치는 식입니다. 펄스 체크 스킬은 이런 사각지대를 잡아냅니다.

펄스 체크 스킬의 내부를 잠깐 들여다보면, 최적화의 흔적을 발견할 수 있습니다. 프로젝트 관리 도구에서 데이터를 가져올 때, 에이전트가 매번 리스트 ID를 검색하면 모델이 세는 글 조각 소모가 큽니다. 그래서 자주 조회하는 리스트의 ID를 스킬 파일에 직접 기록해 둡니다. 에이전트가 탐색 없이 바로 해당 리스트에 접근할 수 있어, 속도와 비용 모두를 줄이는 방법입니다.

검색 작업 자체를 보조 에이전트에 위임하는 것도 효과적입니다. "클릭업 서처 에이전트에게 이 쿼리를 전달해서 검색 결과를 받아와"라고 스킬 파일에 명시하면, 메인 에이전트의 컨텍스트 윈도가 검색 데이터로 넘치는 것을 방지할 수 있습니다. 보조 에이전트가 원시 데이터를 걸러서 필요한 정보만 돌려주기 때문입니다.

4개의 에이전트를 동시에 실행하는 병렬 작업의 위력

이제 시야를 넓혀 보겠습니다. 모닝커피, 펄스 체크, 리서치, 시각 자료 생성 — 네 가지 작업이 있습니다. 하나씩 순서대로 실행하면 에이전트가 빠르다 해도 시간이 걸립니다. 그런데 이 네 가지를 동시에 돌리면 어떤 결과가 나오는지 확인해 보겠습니다.

VS Code에서 에이전트 탭을 네 개 열고, 각각에 하나씩 지시를 내립니다.

탭 1

: "모닝커피 스킬로 오늘 하루를 계획해 줘."

탭 2

: "보조 에이전트를 띄워서 클로드 코드의 새 음성 기능에 대해 조사하고, 링크드인 포스트와 트위터 스타일 캐러셀을 만들어 줘."

탭 3

: "팀 펄스 체크. 이번 주와 이번 분기 진행 상황을 봐 줘."

탭 4

: "이그제큐티브 어시스턴트를 갖는 게 왜 좋은지 설명하는 유튜브 영상용 시각화를 만들어 줘."

네 줄을 타이핑하는 데 30초가 걸리지 않습니다. 그리고 약 2분 뒤, 네 개의 탭 모두에서 결과가 돌아옵니다.

〔그림 22-4〕 4개 에이전트가 병렬로 작업하는 화면 레이아웃〕

첫 번째 탭에서는 오늘의 일정이 정리되어 있습니다. 캘린더 일정, 긴급 액션 아이템, 영상 파이프라인 상태까지 포함된 완전한 아침 브리핑입니다. "예"라고 답하면 캘린더에 일정이 블로킹됩니다.

두 번째 탭에서는 링크드인 포스트가 사용자의 톤으로 작성되어 있고,

projects/carousels/

폴더에 7장짜리 캐러셀 슬라이드가 저장되어 있습니다. CTA(Call To Action)까지 포함된 상태입니다.

세 번째 탭에서는 모든 이니셔티브의 현황과 후속 조치가 필요한 항목이 나열되어 있습니다.

네 번째 탭에서는 PNG 이미지 파일이

projects/visualizations/

폴더에 저장되어 있습니다. 왼쪽에는 업무에 파묻힌 사람, 오른쪽에는 어시스턴트와 함께 여유롭게 일하는 사람이 그려진 시각 자료입니다.

같은 작업을 수동으로 처리한다면 어떨까요? 캘린더를 열어 일정을 확인하고 태스크를 대조하는 데 10분. 조사하고 글을 쓰는 데 20분. 프로젝트 보드를 돌아보며 현황을 정리하는 데 15분. 시각 자료를 만드는 데 20분. 맥락 전환(Context Switching)까지 포함하면 1시간이 훌쩍 넘습니다.

병렬 실행이 가능한 이유는 각 에이전트가 독립적인 컨텍스트 윈도를 갖기 때문입니다. 하나의 에이전트가 캘린더를 읽는 동안, 다른 에이전트는 웹을 검색하고, 또 다른 에이전트는 클릭업 API를 호출합니다. 서로의 작업에 간섭하지 않습니다. 그러면서도 모두 같은 프로젝트 폴더에 접근하므로,

me.md

의 소통 스타일이나

work.md

의 사업 정보를 공유합니다. 독립성과 일관성을 동시에 확보하는 구조입니다.

이 경험을 한번 해 보면, "하나의 에이전트에게 모든 것을 시키는" 방식으로 돌아가기 어렵습니다. 마치 한 대의 컴퓨터에서 여러 작업을 동시에 처리하는 것과 비슷합니다. 품질을 희생하지 않으면서 작업량을 배수로 늘릴 수 있다는 것 — 이것을 레버리지(Leverage)라고 부릅니다.

마무리

모닝커피 스킬은 단독으로도 유용하지만, 진짜 가치는 다른 스킬들과 엮일 때 나타납니다. 아침 브리핑이 프로젝트 상태를 가져오려면 프로젝트 관리 도구와의 연동이 필요하고, 분기 목표를 참조하려면

context

폴더의 파일이 최신이어야 합니다. 어시스턴트의 정밀도는 결국 컨텍스트의 품질에 달려 있습니다.

파일을 어떤 구조로 배치하고, 정보를 어떻게 계층적으로 관리하며, 프로젝트 관리 도구와 어떻게 연결하는지 — 컨텍스트를 다루는 기술이 어시스턴트의 천장을 결정합니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

23장 맥락 관리의 기술: 어시스턴트 똑똑하게 쓰기

전체 목차

책의 모든 장 보기

도입

어시스턴트에게 "지난달 결정한 신제품 출시 일정이 언제였지?"라고 물었을 때, 에이전트가 몇 초 안에 정확한 날짜와 그때의 이유까지 찾아 준다면 — 그것은 에이전트가 똑똑해서가 아닙니다. 정보가 올바른 위치에 올바른 형식으로 저장되어 있기 때문입니다.

반대의 경우를 상상해 보겠습니다. 모든 정보가

CLAUDE.md

한 파일에 몰려 있습니다. 사용자 정보, 팀 구성, 프로젝트 현황, 의사결정 기록, 스킬 사용법까지 500줄이 넘는 문서를 에이전트가 매번 처음부터 끝까지 읽어야 합니다. 모델이 세는 글 조각이 빠르게 소모되고, 컨텍스트 윈도의 한계에 금방 도달합니다. 에이전트의 응답은 느려지고, 정밀도는 떨어집니다.

컨텍스트 관리(Context Management)는 이그제큐티브 어시스턴트의 성능을 좌우하는 보이지 않는 인프라입니다. 화려한 스킬보다 이 인프라를 얼마나 잘 설계하느냐가 어시스턴트의 천장을 결정합니다.

context, decisions, archives 폴더의 역할

프로젝트 폴더 안에는 세 개의 핵심 폴더가 있습니다. 이름만 보면 평범해 보이지만, 각각이 담당하는 시간축이 다릅니다.

context 폴더: 현재를 담는 곳.

이 폴더에는 지금 이 순간 유효한 정보가 들어갑니다.

me.md

에는 사용자의 이름, 역할, 선호 사항이,

work.md

에는 사업체의 현재 구조와 주요 제품이,

team.md

에는 함께 일하는 사람들의 이름과 역할이,

priorities.md

에는 이번 주, 이번 분기의 핵심 과제가 기록됩니다.

goals.md

에는 연간 목표와 분기별 마일스톤이 있습니다.

에이전트가 모닝커피 스킬을 실행하거나, 콘텐츠를 작성하거나, 리서치를 수행할 때, 이 폴더의 파일들이 판단 기준이 됩니다. "이번 분기 목표가 뭐였지?"라는 질문에 에이전트가 답할 수 있는 것은,

context/goals.md

를 읽으면 되기 때문입니다.

이 폴더의 파일들은 정적이지 않습니다. 분기가 바뀌면

priorities.md

가 업데이트되고, 새 팀원이 합류하면

team.md

에 추가됩니다. 에이전트에게 "priorities를 업데이트해 줘"라고 말하면, 에이전트가 직접 파일을 수정합니다.

decisions 폴더: 과거의 판단을 보존하는 곳.

중요한 의사결정이 내려질 때마다 이 폴더의

log.md

에 기록이 남습니다. 날짜, 결정 내용, 이유, 당시의 맥락이 함께 저장됩니다.

2026-03-10

결정: 서부 해안 확장 일정을 4월로 연기

이유: 현재 웹사이트 런칭에 리소스가 집중되어 있어 동시 진행 시 품질 저하 우려

맥락: 웹사이트 런칭 마감 3월 31일, 팀원 1명 추가 채용 진행 중

이 기록이 왜 중요한가? 두 달 뒤 "왜 서부 확장을 4월로 미뤘었지?"라는 질문이 떠오를 때, 에이전트가 이 로그를 찾아 답할 수 있습니다. 기억에 의존하지 않아도 됩니다. 조직이 커질수록, 과거의 결정 이유를 추적하는 일이 점점 어려워지는데, 이 폴더가 그 문제를 해결합니다.

archives 폴더: 지난 시즌의 기록을 보관하는 곳.

분기가 끝나면 해당 분기의 목표와 진행 기록이 이곳으로 옮겨집니다.

context

폴더에는 현재 분기의 정보만 남기고, 지난 분기의 데이터는 아카이브합니다. 이렇게 하면

context

폴더가 비대해지는 것을 막으면서도, 필요할 때 과거 기록을 찾아볼 수 있습니다.

〔그림 23-1〕 context, decisions, archives 폴더의 시간축 다이어그램〕

세 폴더의 관계를 요약하면 이렇습니다.

context

는 현재,

decisions

는 결정의 역사,

archives

는 과거의 스냅샷입니다. 시간이 흐르면

context

의 일부가

archives

로 이동하고, 새로운 정보가

context

에 채워집니다.

decisions

는 계속 누적됩니다.

정보의 계층적 배치

20장에서

CLAUDE.md

가 라우터 역할을 한다고 설명했습니다. 이 개념을 한 단계 더 깊이 들여다보겠습니다.

에이전트가 사용자의 메시지를 처리하는 순서는 다음과 같습니다.

1.

CLAUDE.md

를 로드합니다. (매 대화의 시작점)

2. 사용자의 요청을 분석합니다.

3. 필요한 정보가 어디에 있는지

CLAUDE.md

의 경로 안내를 따라 파악합니다.

4. 해당 파일만 선택적으로 읽습니다.

이 구조가 없다면 어떻게 될까요?

CLAUDE.md

에 모든 정보를 넣거나, 에이전트가 프로젝트 전체를 뒤져야 합니다. 전자는 모델이 세는 글 조각 낭비이고, 후자는 시간 낭비입니다.

〔그림 23-2〕 정보 계층 구조: CLAUDE.md → 라우팅 → 개별 컨텍스트 파일〕

CLAUDE.md

에는 이런 식으로 적혀 있습니다.

컨텍스트 안내

- 사용자에 대해 알아야 하면 → context/me.md

- 사업체 정보가 필요하면 → context/work.md

- 팀원 정보가 필요하면 → context/team.md

- 현재 집중 과제를 확인하려면 → context/priorities.md

- 분기 목표를 보려면 → context/goals.md

프로젝트 안내

- 웹사이트 런칭 → projects/website-launch/README.md

- 서부 확장 → projects/west-coast-expansion/README.md

- 겨울 이벤트 → projects/winter-events/README.md

이 방식의 이점은 모델이 세는 글 조각 절약에만 있지 않습니다. 정보의 단일 진실 공급원(Single Source of Truth)이 유지됩니다. 사용자 정보가

me.md

한 곳에만 있으므로, 수정할 때 한 파일만 고치면 됩니다. 만약

CLAUDE.md

와

me.md

양쪽에 사용자 정보가 중복되어 있다면, 한쪽만 수정했을 때 불일치가 발생합니다.

라우팅 구조를 설계할 때의 원칙은 명확합니다.

CLAUDE.md

에는 경로와 핵심 규칙만 둡니다.

상세 정보는 전용 파일에 분리합니다.

CLAUDE.md

의 분량을 150줄 이내로 유지합니다.

파일이 늘어나면

CLAUDE.md

에 해당 경로를 추가합니다.

스킬 파일에서도 같은 원칙이 적용됩니다. 스킬의 YAML 프론트매터에는 이름과 설명만 넣고, 에이전트가 스킬을 선택한 뒤에야 전체 지시문을 로드합니다. 참조 파일이나 스크립트는 필요한 경우에만 추가로 읽습니다. 이 3단계 로딩을 점진적 컨텍스트 로딩이라 부릅니다.

로딩 단계

읽는 범위

토큰 소모

1단계: 스킬 탐색

YAML 프론트매터 (이름, 설명)

약 100토큰

2단계: 스킬 실행

skill.md 전체

1,000~3,000토큰

3단계: 추가 참조

스크립트, 레퍼런스 파일

필요 시에만

〔그림 23-3〕 점진적 컨텍스트 로딩의 3단계〕

프로젝트 관리 도구와의 연동

context

폴더의 파일들이 아무리 잘 정리되어 있어도, 실시간 데이터가 빠져 있다면 어시스턴트의 판단은 과거에 머물게 됩니다. 팀원이 오늘 아침 태스크를 완료했는데,

priorities.md

에는 그 태스크가 여전히 "진행 중"으로 적혀 있다면 문제입니다.

이 간극을 메우는 것이 프로젝트 관리 도구와의 연동입니다. 클릭업, 노션, 아사나 같은 도구에 에이전트가 직접 접속해서 최신 데이터를 가져옵니다. 연동 방식은 크게 두 가지입니다.

API 연동.

해당 도구의 API 키를

.env

파일에 저장하고, 에이전트가 API를 호출해 데이터를 읽고 쓸 수 있도록 스킬을 구성합니다. 에이전트에게 "클릭업 연동 스킬을 만들어 줘. API 키는

.env

에 넣을 거야"라고 말하면, 에이전트가 API 문서를 조사하고 스킬을 작성합니다.

MCP 연결 서버 연동.

MCP(Model Context Protocol) 서버를 통해 연결하는 방식입니다. MCP 연결 서버가 프로젝트 관리 도구와 에이전트 사이의 다리 역할을 합니다. 클릭업용 MCP 연결 서버가 있다면 에이전트가 자연어로 "이번 주 마감인 태스크를 보여 줘"라고 요청할 수 있습니다.

〔그림 23-4〕 프로젝트 관리 도구 연동 아키텍처 다이어그램〕

연동이 완료되면, 모닝커피 스킬이나 펄스 체크 스킬이 실시간 데이터를 기반으로 작동합니다.

priorities.md

에 적힌 정적 정보와 프로젝트 관리 도구에서 가져온 동적 정보가 결합되어, 에이전트의 판단 정밀도가 올라갑니다.

연동 과정에서 주의할 점이 하나 있습니다. 에이전트가 프로젝트 관리 도구에서 데이터를 검색할 때, 매번 리스트를 탐색하고 ID를 추출하는 과정이 모델이 세는 글 조각을 크게 소모할 수 있습니다. 21장에서 언급한 것처럼, 자주 사용하는 리스트 ID를 스킬 파일에 미리 기록해 두면 이 비용을 줄일 수 있습니다.

에이전트가 작업하는 과정을 관찰하면서 반복되는 탐색 패턴을 발견했다면, 그것을 하드코딩하는 것이 실용적인 최적화입니다.

펄스 체크 스킬 내 하드코딩 예시

ClickUp 리스트 ID

- 콘텐츠 제작: list_abc123

- 웹사이트 런칭: list_def456

- 서부 확장: list_ghi789

이 ID를 사용하여 직접 접근하세요.

리스트 검색 API를 호출하지 마세요.

매일 사용할수록 정밀해지는 어시스턴트의 진화 경로

어시스턴트를 한 달 동안 매일 사용하면, 프로젝트 폴더의 모습이 크게 달라집니다. 첫날에는

context

폴더에 네 개의 파일과

CLAUDE.md

가 전부였지만, 30일 뒤에는 스킬 폴더에 여러 스킬이 추가되고, 리서치 보고서가 쌓이고, 의사결정 로그가 길어지고, 프로젝트별 폴더에 산출물이 채워져 있습니다.

이 축적이 곧 어시스턴트의 지능입니다. 에이전트 자체의 모델이 바뀌지 않아도, 참조할 수 있는 데이터가 풍부해지면서 응답의 정밀도가 올라갑니다.

진화의 경로를 시간순으로 그려 보겠습니다.

1주차: 기초 정립.

인터뷰를 통해

context

파일을 채우고, 첫 번째 스킬을 만듭니다. 아직 에이전트의 응답이 일반적으로 느껴질 수 있습니다. "내 사업을 잘 모르는 신입 비서"와 대화하는 감각입니다.

2주차: 패턴 발견.

매일 사용하다 보면, 반복되는 요청이 보이기 시작합니다. "이건 스킬로 만들 수 있겠다"는 감각이 생깁니다. 기존 ChatGPT 프로젝트나 커스텀 GPT의 프롬프트를 가져와 스킬로 전환합니다. 에이전트에게 수정 의견을 주면서 스킬의 품질이 올라갑니다.

3에서 4주차: 정착.

모닝커피 스킬을 매일 돌리고, 리서치 스킬로 주 2에서 3회 조사를 수행하고, 콘텐츠 생성 스킬로 소셜 미디어 게시물을 작성합니다. 에이전트가 사업의 맥락을 깊이 이해하게 되면서, 제안의 적중률이 눈에 띄게 올라갑니다. "1개월 전과는 완전히 다른 도구를 쓰고 있는 것 같다"는 느낌이 드는 시점입니다.

〔그림 23-5〕 어시스턴트 진화의 타임라인: 1주 → 2주 → 4주〕

진화를 가속하는 몇 가지 습관이 있습니다.

에이전트에게 "기억해 둬"라고 말하기.

"나는 항상 오전에 집중 작업을 선호한다는 걸 기억해 둬." 에이전트가 이 정보를

me.md

나 적절한 파일에 저장합니다. 다음 모닝커피 스킬 실행 시, 집중 작업이 자연스럽게 오전에 배치됩니다.

스킬 실행을 관찰하고 수정 의견 주기.

처음 서너 번은 에이전트가 스킬을 실행하는 과정을 지켜봅니다. 불필요한 API 호출이 보이면 "그 과정은 생략하고 이 ID를 직접 써"라고 지시합니다. 에이전트가 스킬 파일을 수정하면, 다음 실행부터 더 효율적으로 작동합니다.

주기적으로 컨텍스트 파일을 점검하기.

에이전트는

CLAUDE.md

에 적힌 주기(주간, 월간, 분기)에 따라 파일 업데이트를 제안합니다. 주간으로는 자동 메모리 정리, 월간으로는

priorities.md

업데이트, 분기마다 목표 파일 갱신. 이 루틴을 따르면 컨텍스트가 낡지 않습니다.

.claude 폴더 구조 깊이 보기

.claude

폴더는 어시스턴트의 내부 설정을 담는 공간입니다. 프로젝트 루트에 위치하며, 크게 세 개의 하위 폴더로 구성됩니다.

.claude/

├── skills/

│ ├── morning-coffee/

│ │ └── skill.md

│ ├── research/

│ │ ├── skill.md

│ │ └── references/

│ ├── pulse-check/

│ │ └── skill.md

│ └── infographic-builder/

│ ├── skill.md

│ └── scripts/

├── agents/

│ ├── research-agent.md

│ └── clickup-searcher.md

└── rules/

└── communication-style.md

skills 폴더.

각 스킬이 독립된 하위 폴더를 갖습니다. 스킬 폴더 안에는

skill.md

가 있고, 필요에 따라

references/

나

scripts/

하위 폴더가 추가됩니다. 참조 파일이나 스크립트의 위치는 반드시

.claude/skills/

안에 있어야 하는 것은 아닙니다.

skill.md

안에서 정확한 경로를 가리키고 있으면 프로젝트 어디에든 둘 수 있습니다.

agents 폴더.

보조 에이전트의 설정 파일이 저장됩니다. 각 에이전트 파일에는 역할, 사용할 모델, 수행할 작업이 기술되어 있습니다. 메인 에이전트가 작업을 위임할 때 이 파일을 참조합니다.

rules 폴더.

소통 스타일, 포맷팅 규칙, 금지 표현 등 에이전트의 행동을 규제하는 규칙이 들어갑니다. "불릿 포인트 사용", "간결한 어조", "엠 대시 사용 금지" 같은 세부 지침들입니다.

〔그림 23-6〕 .claude 폴더의 내부 구조와 각 하위 폴더의 역할〕

이 구조에서 핵심은 분리와 참조입니다. 모든 설정을 한 파일에 넣지 않고, 용도별로 분리한 뒤, 필요한 곳에서 경로로 참조합니다. 스킬이 10개, 20개로 늘어나도 이 원칙을 지키면 관리가 어렵지 않습니다.

글로벌 스킬(Global Skill)이라는 개념도 있습니다.

.claude/skills/

안이 아니라, 사용자의 홈 디렉터리(Home Directory)에 스킬을 설치하면, 어떤 프로젝트를 열든 해당 스킬을 사용할 수 있습니다. 프론트엔드 디자인 스킬처럼 특정 프로젝트에 국한되지 않는 범용 스킬에 적합합니다.

마무리

컨텍스트 관리는 화려하지 않습니다. 폴더를 만들고, 파일을 분리하고, 경로를 정리하는 — 지루해 보이는 작업입니다. 그러나 이 인프라가 없으면 스킬은 맥락 없이 작동하고, 에이전트의 응답은 일반적인 수준에 머물며, 사용할수록 똑똑해지는 선순환이 시작되지 않습니다.

context

폴더가 현재를 담고,

decisions

가 판단의 역사를 기록하고,

archives

가 과거를 보존하며,

CLAUDE.md

가 이 모든 것의 교차로 역할을 하는 구조 — 이것이 어시스턴트를 단순한 챗봇이 아니라 사업의 두뇌로 만드는 기반입니다.

이 기반 위에서 스킬은 점점 정교해지고, 에이전트는 사용자의 판단 패턴을 학습하며, 한 달 전에는 상상할 수 없었던 수준의 자동화가 일상이 됩니다. 하지만 하나의 에이전트가 모든 것을 처리하는 데는 한계가 있습니다. 복잡한 작업을 여러 전문 에이전트에게 나누어 위임하는 구조, 즉 보조 에이전트의 세계가 다음에 펼쳐집니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

24장 보조 에이전트: 일을 나누는 기술

전체 목차

책의 모든 장 보기

메인 세션과 보조 에이전트의 관계

터미널 창 하나에 클로드 코드를 띄웠습니다. 캐러셀 스킬(Carousel Skill)을 호출하자, 화면 한쪽에 작은 에이전트 표시가 나타납니다. "캐러셀 슬라이드를 계획하겠습니다. 에이전트를 사용합니다." 메인 세션이 직접 슬라이드를 구성하는 대신, 별도의 보조 에이전트(Subagent)에게 작업을 넘긴 것입니다.

보조 에이전트가 열심히 슬라이드 구조를 짜는 동안, 메인 세션은 다른 작업을 이어갈 수 있습니다.

이 장면이 보조 에이전트의 본질을 압축하고 있습니다.

클로드 코드에서 새 세션을 열면, 우리가 대화하는 상대는 메인 세션입니다. 메인 세션은 자체 한 번에 읽는 범위를 갖고, 대화 기록을 쌓아가며, 프로젝트 전반을 관장합니다. 그런데 이 메인 세션이 모든 작업을 직접 수행해야 한다면 어떻게 될까요?

코드 리뷰, 리서치, 검증 작성, 디버깅, 문서화까지 전부 하나의 컨텍스트 안에서 처리하면, 윈도우는 빠르게 포화 상태에 이릅니다. 컨텍스트 부패(Context Rot)가 시작되고, 앞에서 나눈 대화의 맥락이 흐려집니다.

보조 에이전트는 이 문제를 해결하기 위한 구조입니다. 메인 세션이 프로젝트 리더라면, 보조 에이전트는 전문가 팀원입니다. 프로젝트 리더는 전체 그림을 관장하고, 구체적인 작업은 전문가에게 위임합니다.

〔그림 24-1〕 메인 세션과 보조 에이전트의 관계도: 메인 세션(프로젝트 리더)이 코드 리뷰어, 빌더, 디버거, 검증 러너, 아키텍트, 리서처 등 6개 보조 에이전트에게 작업을 분배하는 구조〕

보조 에이전트의 동작 방식을 좀 더 구체적으로 살펴보겠습니다.

보조 에이전트는

잠들어 있다가 깨어납니다.

평소에는 존재하지 않는 것이나 마찬가지입니다. 메인 세션이 "일어나, 이 작업을 해줘"라고 호출하는 순간, 보조 에이전트는 완전히 새로운 컨텍스트와 함께 깨어납니다. 자기만의 모델을 사용할 수 있고, 자기만의 도구 세트를 갖추고, 자기만의 목적에 집중합니다.

독립적인 컨텍스트를 갖는다는 점이 핵심입니다. 메인 세션의 대화 기록을 공유하지 않습니다. 메인 세션이 전달한 프롬프트만을 입력으로 받아, 독립적으로 작업을 수행한 뒤 결과만 돌려보냅니다.

다른 모델을 사용할 수 있다는 점도 중요합니다. 메인 세션이 오퍼스(Opus)로 돌아가고 있더라도, 보조 에이전트는 하이쿠(Haiku)나 소네트(Sonnet)로 작업할 수 있습니다. 비용과 속도를 세밀하게

조절할 수 있는 길이 열리는 셈입니다.

그리고 병렬 실행이 가능합니다. 한 데모에서 "AI 활용에 관해 중소기업용 리서치와 대기업용 리서치를 동시에 해줘"라고 요청하자, 메인 세션은 두 개의 리서치 보조 에이전트를 동시에 파견했습니다. 각 에이전트가 독립적으로 조사를 마치고, 결과를 메인 세션으로 보내왔습니다. 중소기업 AI 활용 브리프, 대기업 AI 활용 브리프, 그리고 양쪽의 교집합까지 정리되어 돌아왔습니다.

이 구조를 조직에 비유하면, 메인 세션은 계획을 세우고, 업무를 배분하고, 결과를 취합하는 역할에 집중합니다. 전문적인 작업 수행 자체는 보조 에이전트의 몫입니다. 덕분에 메인 세션의 컨텍스트는 깨끗하게 유지됩니다.

커스텀 베이커리 비유

파티를 준비한다고 상상해 보겠습니다. 정말 맛있는 컵케이크가 필요합니다.

두 가지 선택지가 있습니다. 대형 마트에 가서 대량 생산된 컵케이크를 집어 오는 방법이 있습니다. 빠르고, 싸고, 편합니다. 하지만 맛은 그저 그렇습니다. 반면 동네의 전문 베이커리를 찾아가서 맞춤 주문을 넣을 수도 있습니다. "바닐라 크림에 라벤더 향을 넣고, 위에 식용 꽃을 올려주세요." 시간은 더 걸리지만, 결과물의 품질은 비교할 수 없습니다.

〔그림 24-2〕 커스텀 베이커리 비유: 슈퍼마켓 컵케이크(범용 처리) vs 맞춤 베이커리(전문 보조 에이전트) 비교〕

보조 에이전트는 맞춤 베이커리입니다.

메인 세션이 모든 작업을 직접 처리하는 것은 대형 마트에서 컵케이크를 사 오는 것과 비슷합니다. 범용적인 역량으로 여러 작업을 한꺼번에 처리하니 빠르긴 하지만, 각 작업의 품질에는 한계가 있습니다. 한 번에 읽는 범위 안에 코드 리뷰, 리서치, 검증 작성, 디버깅이 뒤섞이면, 어느 하나에 깊이 집중하기 어렵습니다.

보조 에이전트에게 위임하면 다릅니다. "코드 리뷰 전문가" 보조 에이전트는 코드 리뷰만을 위한 시스템 프롬프트, 코드 리뷰에 최적화된 도구 세트, 코드 리뷰에 집중된 컨텍스트를 갖습니다. 맞춤 베이커리가 컵케이크 하나에 온 정성을 쏟는 것처럼, 보조 에이전트는 위임받은 작업 하나에 모든 역량을 집중합니다.

이 비유에서 한 가지 더 주목할 부분이 있습니다. 맞춤 베이커리에 주문을 넣을 때, 우리는 원하는 것을 명확하게 전달합니다. "바닐라 크림, 라벤더 향, 식용 꽃." 보조 에이전트에게 작업을 위임할 때도 마찬가지입니다. 메인 세션이 보조 에이전트에게 보내는 프롬프트가 곧 주문서입니다. 주문서가 명확할수록, 돌아오는 결과물의 품질이 높아집니다.

그리고 맞춤 베이커리는 재주문이 가능합니다. "지난번에 만들어 준 그 컵케이크, 이번에는 20개만 더 부탁합니다." 보조 에이전트도 마찬가지로, 한 번 잘 만들어 놓으면 프로젝트를 넘어 조직 전체에서 재사용할 수 있습니다.

보조 에이전트를 사용해야 하는 5가지 이유

보조 에이전트가 무엇인지, 어떤 구조로 작동하는지 이해했습니다. 그렇다면 구체적으로 어떤 상황에서 보조 에이전트가 빛을 발하는지 살펴보겠습니다.

컨텍스트를 보존하기 위해서입니다.

우리가 클로드 코드를 사용할 때 가장 신경 써야 할 자원은 한 번에 읽는 범위입니다. 컨텍스트 부패는 실재하는 위협입니다. 운영자의 역할은 한 번에 읽는 범위를 최대한 효율적으로 관리하는 것이고, 보조 에이전트는 그 핵심 수단 중 하나입니다.

대량의 데이터를 처리하고, 광범위한 리서치를 수행하고, 그 결과에서 메인 세션이 실제로 필요한 핵심 정보만 추려서 보내줍니다. 한 시연에서 보조 에이전트가 약 40,000모델이 세는 글 조각에 해당하는 작업을 수행했지만, 메인 세션은 29,000모델이 세는 글 조각만 소비한 상태로 유지되었습니다.

보조 에이전트가 처리한 40,000모델이 세는 글 조각의 내용이 메인 세션의 컨텍스트를 오염시키지 않은 것입니다.

제약을 적용하기 위해서입니다.

메인 세션은 다양한 도구를 호출할 수 있어야 합니다. 그러나 위험한 작업—깃허브 액션(GitHub Actions)이나 특정 코드 기반 작업—을 수행할 때는 사정이 다릅니다. 이런 작업을 전담하는 보조 에이전트에게 제한된 도구 세트만 부여하면, 의도치 않은 피해를 방지할 수 있습니다.

메인 세션의 강력한 권한과 보조 에이전트의 제한된 권한을 분리함으로써, 전체 워크플로우의 안전성을 높이는 전략입니다.

설정을 재사용하기 위해서입니다.

보조 에이전트는 마크다운 파일 하나로 정의됩니다. 이 파일을 다른 프로젝트에 복사하면, 동일한 보조 에이전트를 즉시 사용할 수 있습니다. 조직 차원에서 생각해 보면, 프로젝트 관리나 분기별 기획에 탁월한 보조 에이전트를 하나 만들어 놓으면, 조직 전체가 그 혜택을 누릴 수 있습니다.

스킬과 비슷한 원리이지만, 보조 에이전트는 독자적인 컨텍스트와 모델 선택이라는 차원이 추가됩니다.

전문화를 위해서입니다.

AI는 목적이 구체적이고 좁을수록 더 좋은 결과를 냅니다. 계획도 세우고, 구축도 하고, 검증도 하고, 문서화도 하고, 리서치도 하고, 리뷰도 하는 만능 에이전트 하나보다, 각 단계를 전담하는 작은 에이전트 여러 개가 낫습니다. 메인 세션은 그 결과들을 종합하면 됩니다. "작은 에이전트가 이긴다(Tiny agents win)"는 원칙은 실전에서 반복적으로 입증되고 있습니다.

비용을 절감하기 위해서입니다.

보조 에이전트마다 다른 모델을 지정할 수 있으므로, 모든 작업에 오퍼스를 사용할 필요가 없습니다. 빠른 리서치가 필요한 보조 에이전트에는 하이쿠를, 코드 생성이 필요한 보조 에이전트에는 소네트를 배정하면, 시간과 모델이 세는 글 조각 비용을 동시에 절약할 수 있습니다.

〔표 23-1〕 보조 에이전트 사용 이유 요약

이유

핵심 메커니즘

실질적 효과

컨텍스트 보존

독립 컨텍스트에서 처리 후 요약만 반환

메인 윈도우 오염 방지

제약 적용

서브에이전트별 도구 세트 제한

안전한 작업 실행

설정 재사용

마크다운 파일 기반 이식성

조직 전체 생산성 향상

전문화

목적 특화 시스템 프롬프트

작업 품질 향상

비용 절감

서브에이전트별 모델 선택

토큰 비용 최적화

위임할 때와 직접 할 때의 판단 기준

보조 에이전트의 장점을 알게 되면, 모든 작업을 위임하고 싶은 유혹이 생깁니다. 하지만 그렇게 하면 과잉 설계(Over-engineering)에 빠집니다. 위임해야 할 때와 직접 해야 할 때를 구분하는 기준이 필요합니다.

메인 세션에서 직접 처리해야 하는 경우:

대화의 왕복(Back-and-forth)이 필요한 작업입니다. "이 부분을 고쳐줘"라고 요청하고, 결과를 보고, "아니, 이쪽 방향으로"라고 수정을 요청하는 반복적인 상호작용은 메인 세션에서 직접 하는 것이 효율적입니다. 보조 에이전트에게 위임하면, 매번 새로운 컨텍스트에서 시작해야 하므로 오히려 비효

율적입니다.

빠른 수정이 필요한 작업도 마찬가지입니다. 변수명 하나를 바꾸거나, 오타를 고치거나, 간단한 로직을 수정하는 작업에 보조 에이전트를 호출하면, 보조 에이전트를 깨우고 결과를 받는 오버헤드가 작업 자체보다 큽니다.

공유된 컨텍스트가 필요한 작업, 즉 지금까지의 대화 기록을 기반으로 판단해야 하는 작업도 메인 세션이 담당해야 합니다. 보조 에이전트는 대화 기록을 물려받지 않기 때문입니다.

낮은 지연 시간(Low Latency)이 요구되는 상황에서도 메인 세션이 유리합니다. 보조 에이전트를 호출하고, 작업을 수행하고, 결과를 받아오는 과정에는 필연적으로 시간이 소요됩니다.

보조 에이전트에게 위임해야 하는 경우:

자기완결적(Self-contained)인 작업입니다. "이 코드 베이스를 분석해서 보안 취약점을 찾아줘"처럼, 입력을 받으면 독립적으로 수행한 뒤 결과를 돌려보낼 수 있는 작업은 위임에 적합합니다.

도구 제한이 필요한 작업에도 보조 에이전트가 맞습니다. 특정 보조 에이전트에게 읽기 전용 권한만 부여하거나, 특정 MCP 연결 서버만 사용하게 제한할 수 있습니다.

요약된 결과만 필요한 경우도 위임의 신호입니다. 보조 에이전트가 방대한 데이터를 처리하더라도, 메인 세션은 그 결과의 핵심 요약만 받아보면 됩니다. 이것이 컨텍스트 보존의 핵심 메커니즘입니다.

그리고 전경(Foreground)과 배경(Background) 실행의 선택지가 있습니다. 보조 에이전트를 전경에서 실행하면 메인 세션이 그 완료를 기다리고, 배경에서 실행하면 메인 세션은 다른 작업을 계속할 수 있습니다. 이 선택은 작업의 긴급도와 의존성에 따라 달라집니다.

〔그림 24-3〕 위임 판단 흐름도: "왕복 대화가 필요한가?" → 예: 메인 세션 / 아니오 → "자기완결적인가?" → 예: 보조 에이전트 / 아니오 → 메인 세션〕

한 가지 핵심적인 구분을 짚고 넘어가야 합니다. 보조 에이전트와 에이전트 팀(Agent Teams)은 이름이 비슷하고 개념이 겹치는 것처럼 보이지만, 근본적인 차이가 있습니다. 보조 에이전트는

단방향 관계

입니다. 메인 세션이 입력을 보내면, 보조 에이전트는 작업을 수행하고, 결과를 메인 세션에 돌려보냅니다. 보조 에이전트끼리는 서로 대화할 수 없습니다.

병렬로 실행되더라도, 각자 독립적으로 작업합니다.

에이전트 팀은 양방향입니다. 팀원들이 공유 태스크 리스트를 갖고, 서로 메시지를 보내고, 서로에게 작업을 할당할 수 있습니다. 그것은 완전히 다른 차원의 협업이며, 다음 장들에서 구체적으로 다루게 될 영역입니다.

지금까지 보조 에이전트가 무엇이고, 왜 필요하며, 언제 사용해야 하는지 개념적 기반을 다졌습니다. 이제 이 개념을 실제로 구현하는 방법을 손으로 짚어볼 차례입니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

25장 보조 에이전트 구축 실습

전체 목차

책의 모든 장 보기

/agents 명령어로 에이전트 생성하기

VS Code에서 클로드 코드를 열고

/agents

를 입력합니다. 그런데 화면에 안내 메시지가 뜹니다. "터미널에서 계속 진행하세요." IDE 확장 프로그램의 한계 때문에, 일부 슬래시 명령은 터미널에서만 실행됩니다. 큰 문제는 아닙니다. 터미널을 열면 명령이 바로 실행되고, 보조 에이전트 관리 화면이 나타납니다.

이 화면에서 할 수 있는 일은 명확합니다.

현재 프로젝트에 등록된 모든 에이전트를 조회할 수 있습니다. 프로젝트 에이전트와 빌트인 에이전트가 구분되어 표시됩니다. 기존 에이전트를 선택해서 편집하거나 삭제할 수 있습니다. 그리고 새 에이전트를 생성할 수 있습니다.

"Create new agent"를 선택하면, 클로드 코드가 대화형으로 에이전트 구성을 안내합니다. 여기서 두 가지 선택지가 주어집니다. 수동으로 직접 구성하는 방법과, 클로드에게 생성을 맡기는 방법입니다.

클로드에게 맡기는 방법을 선택하면, 우리가 해야 할 일은 이 에이전트가 무엇을 해야 하고, 언제 호출되어야 하는지를 자연어로 설명하는 것뿐입니다. 설명이 구체적이고 포괄적일수록 결과가 좋습니다.

프로젝트 에이전트로 만들지, 개인(글로벌) 에이전트로 만들지도 이 단계에서 결정합니다. 프로젝트 에이전트는 해당 프로젝트의

.claude/agents/

폴더에 저장되어, 그 프로젝트에서만 사용 가능합니다. 개인 에이전트는 홈 디렉토리의 글로벌 클로드 코드 설정에 저장되어, 어떤 프로젝트에서든 사용할 수 있습니다.

임시 에이전트(Temporary Agent)라는 선택지도 존재합니다. 세션 한정으로 생성했다가 사라지는 에이전트인데, 실제 활용 빈도가 낮으므로 깊이 다루지는 않겠습니다.

〔그림 25-1〕 /agents 명령어 실행 화면: 프로젝트 에이전트 목록과 빌트인 에이전트 목록이 구분되어 표시된 터미널 스크린샷〕

한 가지 실전 팁이 있습니다. 에이전트를 생성할 때 "해당 폴더가 이미 존재한다"는 오류가 발생하는 경우가 있습니다. 이 경우 기존

agents

폴더의 이름을 임시로 변경한 뒤 에이전트를 생성하고, 기존 에이전트들을 새 폴더로 옮긴 다음 이전 폴더를 삭제하면 해결됩니다. 알려진 버그이지만, 기능에는 영향이 없습니다.

YAML 메타데이터 구성

보조 에이전트의 정체는 마크다운 파일 하나입니다. 이 사실을 이해하면 개념이 한결 명료해집니다. 스킬이 마크다운 파일이듯, 보조 에이전트 역시 마크다운 파일입니다.

파일의 구조는 두 부분으로 나뉩니다. 상단의 YAML 프론트매터(Front Matter)와 하단의 본문 지시 사항입니다.

YAML 프론트매터에 들어가는 주요 필드를 살펴보겠습니다.

name: "AI Trend Hunter"

description: "AI 트렌드를 추적하고 콘텐츠 아이디어를 발굴하는 에이전트. 최신 AI 동향을 조사하고 영상 기획에 활용할 수 있는 시그널을 찾아냅니다."

tools:〔"all"〕

model: "sonnet"

memory: true

color: "cyan"

name

은 에이전트의 식별자입니다. 메인 세션이 이 이름으로 보조 에이전트를 참조합니다.

description

은 에이전트의 호출 조건을 결정하는 핵심 필드입니다. 메인 세션은 사용자의 요청을 받으면, 등록된 에이전트들의 설명문을 훑어보며 어떤 에이전트에게 위임할지 판단합니다. 스킬의 설명문이 트리거 정확도를 결정하는 것과 동일한 원리입니다. 설명문이 모호하면 잘못된 에이전트가 호출되고, 설명문이 구체적이면 정확한 위임이 이루어집니다.

tools

는 이 에이전트가 사용할 수 있는 도구 세트를 지정합니다.

"all"

로 모든 도구를 허용할 수도 있고,

"read-only"

같은 제한을 걸 수도 있습니다.

disallowed_tools

필드로 특정 도구를 명시적으로 차단하는 것도 가능합니다.

model

은 보조 에이전트가 사용할 언어 모델을 결정합니다. 메인 세션과 다른 모델을 지정할 수 있다는 점이 보조 에이전트의 핵심 강점 중 하나입니다. 빠른 탐색 작업에는 하이쿠를, 코드 생성에는 소네트를, 복잡한 추론에는 오퍼스를 배정하는 식으로 비용과 품질을 조율합니다.

memory

를 활성화하면, 보조 에이전트는 자신의 작업 이력을

agent-memory

폴더에 기록합니다. 이 부분은 뒤에서 별도로 다루겠습니다.

color

는 터미널에서 이 에이전트가 실행 중일 때 표시되는 색상입니다. 여러 보조 에이전트가 동시에 돌아갈 때 시각적으로 구분하는 데 도움이 됩니다. IDE 확장 프로그램에서는 보이지 않고, 터미널에서만 확인할 수 있습니다.

이 밖에도

permission_mode

,

max_turns

,

auto_compact

등 지원되는 프론트매터 필드가 더 있습니다.

auto_compact

의 기본값은 약 95%인데, 이 비율을 낮추면 보조 에이전트 내부의 컨텍스트 부패를 더 적극적으로 방지할 수 있습니다.

〔표 24-1〕 주요 YAML 프론트매터 필드

필드

역할

예시 값

name

에이전트 식별자

"Code Reviewer"

description

호출 조건 결정 (트리거)

"코드 품질, 보안, 유지보수성을 검토하는 전문가"

tools

사용 가능한 도구 세트

"all", "read-only", ["mcp"]

model

사용할 언어 모델

"haiku", "sonnet", "opus"

memory

작업 이력 보존 여부

true / false

color

터미널 표시 색상

"cyan", "green", "yellow"

max_turns

최대 실행 턴 수

10

auto_compact

컨텍스트 압축 임계값

50%

YAML 프론트매터 아래에는 본문이 이어집니다. 본문은 보조 에이전트가 깨어났을 때 참조하는 시스템 프롬프트입니다. 메인 세션이 "이 에이전트를 사용하겠다"고 결정한 뒤, 에이전트가 실제로 작업을 시작할 때 읽어 들이는 지시사항의 전문입니다.

AI 트렌드 헌터 에이전트 라이브 구축

개념을 손에 잡히는 실습으로 옮겨 보겠습니다. 여기서는 "AI 트렌드 헌터"라는 보조 에이전트를 처음부터 끝까지 만드는 과정을 따라갑니다.

터미널에서

/agents

를 실행하고 "Create new agent"를 선택합니다. 프로젝트 에이전트로 생성하고, 클로드에게 자동 생성을 맡깁니다.

이 에이전트가 무엇을 해야 하는지 설명합니다. "최신 AI 트렌드를 추적하고, 콘텐츠로 만들 만한 주제를 발굴하는 에이전트. 퍼플렉시티(Perplexity) API와 X(구 트위터)를 활용해 트렌드 시그널을 수집하고, 영상 기획에 활용할 수 있는 형태로 정리한다." 설명은 더 구체적일수록 좋고, 나중에 언제든 수정할 수 있으니 완벽할 필요는 없습니다.

클로드 코드가 에이전트를 생성하기 시작합니다. 이 과정에서 주목할 점이 있습니다. 클로드 코드는 에이전트 설명만으로 마크다운 파일을 만드는 것이 아니라,

현재 프로젝트의 구조와 내용을 참조하여

에이전트를 맞춤 구성합니다. 유튜브 채널을 운영하고 있고, 클로드 코드와 자동화에 관한 콘텐츠를 만드는 프로젝트라면, 에이전트의 시스템 프롬프트에 그 맥락이 반영됩니다.

생성 과정에서 몇 가지 선택을 내려야 합니다.

도구 선택:

이 에이전트가 사용할 수 있는 도구를 지정합니다. "all"을 선택하면 모든 도구를 사용할 수 있고, "read-only"와 "MCP"만 선택하면 파일 읽기와 MCP 연결 서버 호출만 허용됩니다. 리서치 에이전트에게 파일 수정 권한을 줄 필요가 있을까요? 이 질문을 스스로에게 던지는 것이 제약 적용의 시작입니다.

모델 선택:

소네트를 선택합니다. 트렌드 리서치는 방대한 정보를 빠르게 훑어야 하므로, 오퍼스의 깊은 추론보다는 소네트의 속도가 더 적합한 용도입니다.

색상 선택:

터미널에서 이 에이전트를 시각적으로 식별하기 위한 색상을 고릅니다.

메모리 설정:

"Enable"을 선택합니다. 이 설정의 의미는 다음 절에서 자세히 다룹니다.

생성이 완료되면,

.claude/agents/

폴더에 마크다운 파일이 생성됩니다. 파일을 열어보면 YAML 프론트매터에 이름, 설명, 도구, 모델, 메모리, 색상이 설정되어 있고, 본문에는 "당신은 엘리트 AI 트렌드 헌터입니다"로 시작하는 시스템 프롬프트가 작성되어 있습니다. 미션, 수색 대상, 메모리 활용법, 과거 컨텍스트 검색 방법까지 포함되어 있습니다.

〔그림 25-2〕 생성된 AI 트렌드 헌터 에이전트의 마크다운 파일 구조: YAML 프론트매터와 시스템 프롬프트 본문〕

이제 이 에이전트를 실행해 봅니다. 터미널에서 메인 세션에게 "AI 트렌드 헌터를 실행해줘"라고 요청하면, 메인 세션은 등록된 에이전트의 설명문을 참조하여 AI 트렌드 헌터를 호출합니다. 터미널에서 에이전트가 동작하는 모습이 보입니다. 웹 검색을 수행하고, 정보를 수집하고, 분석을 진행합니다.

실행 중에

Ctrl+O

를 누르면 에이전트의 사고 과정을 펼쳐 볼 수 있습니다. 메인 세션이 이 보조 에이전트에게 어떤 프롬프트를 보냈는지도 확인할 수 있습니다.

Ctrl+B

를 누르면 에이전트를 배경으로 보내서, 메인 세션과 대화를 이어가면서 보조 에이전트의 완료를 기다릴 수 있습니다.

실행이 끝나면, 결과가 메인 세션으로 돌아옵니다. 핫 시그널(Hot Signal), 웜 시그널(Warm Signal), 추천 영상 아이디어, 핵심 패턴 등이 정리된 보고서입니다.

한 가지 점검해 볼 사항이 있습니다. 생성된 에이전트의 설명문이 너무 길 수 있습니다. 설명문이 지나치게 길면 모델이 세는 글 조각을 낭비하게 됩니다. 반면 호출 정확도는 높아질 수 있으므로, 적절한 균형을 찾아야 합니다.

에이전트 빌더 스킬(Agent Builder Skill) 같은 도구를 사용해 감사(Audit)를 돌리면, 도구 제한 미설정, 최대 턴 미설정, 설명문 비대화 같은 문제를 자동으로 탐지할 수 있습니다.

에이전트 메모리

AI 트렌드 헌터를 처음 실행하기 전에

agent-memory

폴더를 열어보면, AI 트렌드 헌터용 폴더는 있지만 안에 파일이 없습니다. 메모리 파일은 에이전트를 최초 실행한 후에 자동 생성됩니다.

첫 실행이 완료되면,

agent-memory/ai-trend-hunter/

폴더 안에 메모리 파일이 생깁니다. 이 파일에는 무엇이 담겨 있을까요?

유용했던 정보 소스 목록

반복적으로 등장하는 주제 패턴

효과적이었던 영상 기획 방향

가장 최근 스캔 결과의 요약

보조 에이전트는 매번 완전히 새로운 컨텍스트에서 깨어납니다. 대화 기록이 없고, 이전 세션의 기억이 없습니다. 하지만 메모리 파일이 있으면 사정이 달라집니다. 깨어난 보조 에이전트는 메모리 파일을 읽어서, 과거의 학습 내용을 참조할 수 있습니다. 완전한 기억은 아니지만, "지난번에 이 소스가 유용했다", "이 주제는 이미 다루었다" 정도의 맥락은 유지됩니다.

〔그림 25-3〕 에이전트 메모리의 작동 흐름: 첫 실행 → 메모리 파일 자동 생성 → 이후 실행 시 메모리 파일 참조 → 작업 후 메모리 파일 업데이트〕

이 구조 덕분에 트렌드 리서치 에이전트가 2시간 전에 이미 보고한 내용을 다시 보고하는 사태를 방지할 수 있습니다. 메모리에 "최근 스캔" 정보가 남아 있으니, 중복을 피하고 새로운 시그널에 집중합니다.

메모리는 보조 에이전트가 세션을 넘어 학습을 축적해 나가는 메커니즘입니다. 사용할수록 좋아지는 에이전트를 만들고 싶다면, 메모리를 활성화하는 것이 출발점입니다.

보조 에이전트로 메인 맥락 지키기

보조 에이전트의 컨텍스트 보존 효과를 수치로 확인해 보겠습니다.

AI 트렌드 헌터 에이전트가 실행을 마쳤습니다. 이 에이전트는 웹을 검색하고, 소스를 분석하고, 시그널을 분류하고, 보고서를 작성하는 과정에서 약

40,000모델이 세는 글 조각

을 소비했습니다.

이제 메인 세션에서

/context

명령을 실행해 봅니다. 메인 세션의 현재 컨텍스트 사용량은

29,000모델이 세는 글 조각

입니다.

이 숫자가 의미하는 바를 곱씹어 보겠습니다. 만약 보조 에이전트 없이 메인 세션에서 동일한 리서치를 직접 수행했다면, 메인 세션의 컨텍스트에 40,000모델이 세는 글 조각 이상이 추가되었을 것입니다. 기존 대화와 합산하면, 한 번에 읽는 범위의 상당 부분이 리서치 결과로 채워집니다. 이후 다른 작업—코드 리뷰, 문서 작성, 디버깅—을 할 때 사용 가능한 컨텍스트가 줄어듭니다.

보조 에이전트를 사용하면, 40,000모델이 세는 글 조각 분량의 작업이 별도의 컨텍스트에서 처리됩니다. 메인 세션에는 핫 시그널, 웜 시그널, 추천 아이디어 같은 요약만 전달됩니다. 메인 세션의 컨텍스트는 깨끗하게 보존됩니다.

〔그림 25-4〕 컨텍스트 보존 효과 비교: 직접 처리 시 메인 세션 69,000+ 모델이 세는 글 조각 vs 보조 에이전트 위임 시 메인 세션 29,000모델이 세는 글 조각〕

이것이 보조 에이전트를 사용하는 가장 실질적인 이유 중 하나입니다. 한 번에 읽는 범위는 유한한 자원이고, 보조 에이전트는 그 자원을 보호하는 방벽입니다.

클로드 코드의 빌트인 보조 에이전트

커스텀 보조 에이전트를 만들기 전에, 클로드 코드가 기본으로 탑재하고 있는 보조 에이전트들이 있습니다. 클로드 코드를 사용하면서 터미널에 "agent"라는 표시가 나타난 적이 있다면, 이 빌트인 보조 에이전트를 만난 것입니다.

탐색 에이전트(Explore)

는 코드 베이스를 검색하고 분석합니다. 하이쿠 모델로 실행되며, 읽기 전용 권한만 갖습니다. 비용이 낮고 빠른 대신, 파일을 수정하거나 코드를 생성하지는 못합니다. 코드 베이스의 구조를 파악하거나, 특정 패턴을 찾아낼 때 자동으로 호출됩니다.

계획 에이전트(Planning)

는 리서치를 수행하고 계획을 수립합니다. 부모 모델의 모델을 그대로 상속받습니다. 메인 세션이 오퍼스로 돌아가고 있다면, 계획 에이전트도 오퍼스로 실행됩니다. 읽기 전용 권한을 가지며, 계획 모드(Plan Mode)에서 이 에이전트가 호출되는 것을 자주 볼 수 있습니다.

일반 에이전트(General)

는 다단계 작업이 필요할 때 호출됩니다. 부모 모델을 상속받으며, 모든 도구를 사용할 수 있습니다. 앞의 두 에이전트와 달리 파일을 수정하고, 명령을 실행하고, 코드를 생성할 수 있습니다.

〔표 24-2〕 빌트인 보조 에이전트 비교

에이전트

모델

권한

용도

Explore

Haiku

읽기 전용

코드 베이스 검색·분석

Planning

부모 모델 상속

읽기 전용

리서치·계획 수립

General

부모 모델 상속

모든 도구

다단계 작업 수행

클로드 코드의 핵심 개발자 중 한 명이 직접 사용하는 커스텀 보조 에이전트 목록을 공개한 적이 있습니다. 구축 검증기(Build Validator), 코드 아키텍트(Code Architect), 코드 단순화기(Code Simplifier), 온콜 가이드(On-call Guide), 앱 검증기(Verify App).

클로드 코드를 만든 사람이 자신의 도구에 이렇게 다양한 보조 에이전트를 운용하고 있다는 사실 자체가, 보조 에이전트의 실용적 가치를 증명합니다.

클로드 코드 공식 문서에도 보조 에이전트 예시가 공개되어 있습니다. 코드 리뷰어(Code Reviewer), 디버거(Debugger), 데이터 과학자(Data Scientist), 데이터베이스 쿼리 검증기(Database Query Validator) 등의 전체 마크다운 구성을 확인할 수 있으니, 자신만의 보조 에이전트를 설계할 때 참고할 만합니다.

보조 에이전트를 구축하고 운용하는 실전 기술을 익혔습니다. 한 가지 확인해야 할 사실이 남아 있습니다. 보조 에이전트는 메인 세션의 지시를 받아 독립적으로 작업하고, 결과를 돌려보내는 단방향 구조입니다. 그렇다면 에이전트들이 서로 대화하고, 서로에게 작업을 할당하고, 공유된 목표를 향해 협력하는 구조는 어떤 모습일까요?

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

26장 에이전트 팀: 서로 대화하는 에이전트들

전체 목차

책의 모든 장 보기

보조 에이전트와 에이전트 팀의 핵심 차이

프롬프트 하나를 입력합니다. "가상의 AI 스타트업 랜딩 페이지를 만들어줘." 잠시 후 화면이 분할됩니다. 파란색으로 표시된 프론트엔드 개발자 에이전트, 초록색의 백엔드 개발자 에이전트, 노란색의 QA 에이전트가 동시에 깨어납니다. 세 에이전트가 각자의 영역에서 작업을 시작합니다. 그런데 보조 에이전트와는 다른 일이 벌어집니다.

프론트엔드 개발자가 백엔드 개발자에게 메시지를 보냅니다. QA 에이전트가 두 개발자에게 수정 요청을 보냅니다. 에이전트들이 서로 대화하고 있습니다.

이것이 에이전트 팀(Agent Teams)입니다.

보조 에이전트와 에이전트 팀의 차이는 소통 구조에 있습니다. 보조 에이전트는 단방향(One-way)입니다. 메인 세션이 프롬프트를 보내면, 보조 에이전트는 작업을 수행하고, 결과를 메인 세션에 반환합니다. 이 과정에서 보조 에이전트끼리는 소통할 수 없습니다. 병렬로 실행되더라도 각자 고립된 채 작업합니다.

에이전트 팀은 양방향(Two-way)입니다. 팀원들이 서로 메시지를 주고받을 수 있습니다. 서로에게 작업을 할당할 수 있습니다. 공유된 태스크 리스트를 함께 관리합니다. 메인 세션을 거치지 않고 팀원 간 직접 소통이 가능합니다.

〔그림 26-1〕 보조 에이전트 vs 에이전트 팀 소통 구조 비교: 보조 에이전트는 메인 세션과의 단방향 화살표만 존재. 에이전트 팀은 팀원 간 양방향 화살표와 공유 태스크 리스트가 존재〕

이 구조적 차이가 만들어내는 결과는 극적입니다.

보조 에이전트 구조에서 "코드를 리팩터링하고 검증를 작성해줘"라고 요청하면, 메인 세션이 리팩터 에이전트와 검증 작성 에이전트에게 각각 작업을 보냅니다. 두 에이전트는 독립적으로 작업을 마치고, 각자의 결과를 메인 세션에 보냅니다. 메인 세션이 두 결과를 종합합니다.

문제는, 리팩터 에이전트가 함수 시그니처를 변경했는데 검증 작성 에이전트는 원래 시그니처를 기준으로 검증를 작성했을 수 있다는 것입니다. 서로 대화할 수 없으니, 이런 불일치를 감지할 방법이 없습니다.

에이전트 팀 구조에서는 상황이 다릅니다. 리팩터 에이전트가 함수 시그니처를 변경하면, 검증 작성 에이전트에게 메시지를 보낼 수 있습니다. "이 함수의 시그니처가 바뀌었으니 참고해." 검증 작성 에이전트는 변경된 시그니처를 반영하여 검증를 작성합니다. QA 에이전트가 문제를 발견하면, 해당 에이전트에게 직접 수정을 요청합니다. 중간에 메인 세션을 경유할 필요가 없습니다.

에이전트 팀에는 팀 리드(Team Lead) 역할을 하는 메인 오케스트레이터가 있습니다. 프로젝트 매니저와 비슷한 역할입니다. 에이전트들을 생성하고, 태스크 리스트를 초기화하고, 전체 진행 상황을 모니터링하고, 결과의 품질을 확인합니다. 하지만 모든 소통이 이 오케스트레이터를 경유하지는 않습니다. 팀원들은 필요할 때 직접 소통합니다.

공유 태스크 리스트와 상호 작업 할당

에이전트 팀의 협업을 가능하게 하는 핵심 인프라는 공유 태스크 리스트(Shared Task List)입니다.

메인 오케스트레이터가 에이전트 팀을 생성하면, 가장 먼저 하는 일이 태스크 리스트를 구성하는 것입니다. 프론트엔드 개발, 백엔드 API 구현, 검증 작성, QA 검증 같은 항목이 나열되고, 각 항목에 담당자가 배정됩니다.

이 태스크 리스트가 "공유"된다는 점이 중요합니다. 모든 팀원이 전체 태스크 리스트를 볼 수 있습니다. 자신의 작업이 완료되면 상태를 업데이트합니다. 다른 팀원의 진행 상황을 확인할 수 있습니다. 그리고—이것이 보조 에이전트와 결정적으로 다른 부분인데—팀원이 다른 팀원에게 새로운 작업을 할당할 수 있습니다.

한 시연에서 이 메커니즘이 생생하게 드러났습니다. 프론트엔드 개발자와 백엔드 개발자가 각자의 작업을 마치고, 결과물을 QA 에이전트에게 보냈습니다. QA 에이전트가 코드를 검증한 결과, 3개의 치명적 이슈를 발견했습니다. QA 에이전트는 이 이슈들을 프론트엔드 개발자와 백엔드 개발자에게 돌려보내면서, 각각에게 수정 작업을 할당했습니다.

두 개발자가 수정을 마치고 다시 QA 에이전트에게 보냈습니다. 두 번째 검증에서 QA 에이전트가 통과 판정을 내렸습니다. 세 가지 치명적 이슈가 모두 해결되었습니다.

〔그림 26-2〕 공유 태스크 리스트 기반 협업 흐름: 프론트엔드·백엔드 작업 완료 → QA 에이전트 검증 → 3개 이슈 발견 → 각 개발자에게 수정 할당 → 재검증 → 통과〕

이 과정에서 메인 오케스트레이터는 전체 흐름을 모니터링하면서 상태 업데이트를 제공했지만, QA 에이전트가 개발자 에이전트에게 수정을 요청하는 소통은 직접 이루어졌습니다. 메인 오케스트레이터를 거치지 않은 팀원 간 직접 소통입니다.

팀원들은 메시지 전송 도구(Send Message Tool)를 사용해 서로에게 메시지를 보냅니다. 프롬프트에서 "작업이 끝나면 프론트엔드 개발자에게 메시지를 보내세요"라고 명시하면, 에이전트는 해당 팀원에게 직접 메시지를 전달합니다.

리팩터 에이전트와 검증 작성 에이전트의 협업 시나리오

구체적인 시나리오를 통해 에이전트 팀의 작동 방식을 따라가 보겠습니다.

사용자가 메인 세션에 요청합니다. "이 모듈을 리팩터링하고 검증를 추가해줘."

메인 세션은 요청을 분석합니다. 두 가지 전문 영역이 필요합니다. 리팩터링과 검증 작성. 메인 세션은 두 에이전트를 검색하고, 리팩터 에이전트와 검증 작성 에이전트를 찾아냅니다.

보조 에이전트 방식이라면

이렇게 진행됩니다. 메인 세션이 리팩터 에이전트에게 "이 모듈을 리팩터링해줘"라는 프롬프트를 보냅니다. 동시에 검증 작성 에이전트에게 "이 모듈에 대한 검증를 작성해줘"라는 프롬프트를 보냅니다. 두 에이전트가 병렬로 작업합니다. 각자의 결과가 메인 세션으로 돌아옵니다. 메인 세션이 두 결과를 종합합니다.

리팩터링된 코드에 맞게 검증를 수정해야 할 수 있습니다. 이 수정은 메인 세션이 직접 하거나, 검증 작성 에이전트를 다시 호출해야 합니다.

에이전트 팀 방식이라면

흐름이 달라집니다. 메인 오케스트레이터가 "리팩터 + 검증" 팀을 구성합니다. 공유 태스크 리스트가 만들어집니다.

리팩터 에이전트가 먼저 작업을 시작합니다. 함수 추출, 변수 이름 변경, 인터페이스 정리 등을 수행합니다. 작업이 끝나면, 검증 작성 에이전트에게 메시지를 보냅니다. "리팩터링이 완료되었습니다. 변경된 함수 시그니처는 다음과 같습니다." 검증 작성 에이전트는 이 정보를 바탕으로 검증를 작성합니다. 변경된 인터페이스를 정확히 반영한 검증가 나옵니다.

검증 실행 결과, 일부 검증가 실패했다면? 검증 작성 에이전트가 리팩터 에이전트에게 메시지를 보낼 수 있습니다. "이 함수의 반환 타입이 문서와 다릅니다. 확인해 주세요." 리팩터 에이전트가 확인하고 수정합니다. 다시 검증를 돌립니다. 이 반복이 팀 내에서 자체적으로 이루어집니다.

〔그림 26-3〕 에이전트 팀 협업 시나리오 상세 흐름: 리팩터 에이전트 작업 → 검증 에이전트에 변경 사항 전달 → 검증 작성 → 실패 시 리팩터 에이전트에 수정 의견 → 수정 → 재검증 → 통과 → 메인 오케스트레이터에 완료 보고〕

T-Mux 터미널을 사용하면, 이 과정을 시각적으로 관찰할 수 있습니다. 화면이 분할되어 각 에이전트의 작업 과정이 실시간으로 표시됩니다. 파란색 에이전트가 코드를 리팩터링하는 동안, 초록색 에이전트가 대기하고 있다가, 메시지를 수신한 순간 검증 작성을 시작하는 모습을 직접 볼 수 있습니다. 필요하다면 특정 에이전트에게 직접 메시지를 보내서 추가 지시를 할 수도 있습니다.

에이전트 팀 구성 시 고려할 아키텍처 원칙

에이전트 팀은 강력하지만, 잘못 구성하면 비용만 높고 결과는 혼란스러운 상황이 벌어질 수 있습니다. 팀을 구성할 때 지켜야 할 원칙들을 정리합니다.

역할 명확화: 각 에이전트에게 고유한 영역을 부여합니다

팀의 각 에이전트는 자신만의 파일과 자신만의 산출물을 가져야 합니다. 여러 에이전트가 같은 파일을 수정하면, 서로의 작업을 덮어쓸 위험이 있습니다. 프론트엔드 개발자는 프론트엔드 파일만, 백엔드 개발자는 백엔드 파일만 수정하도록 프롬프트에서 명확하게 지정합니다.

산출물도 구체적으로 정의해야 합니다. "좋은 코드를 작성하라"는 모호합니다. "REST API 엔드포인트를 구현하고, 각 엔드포인트에 대한 검증 파일을

/tests/

폴더에 생성하라"는 명확합니다. 에이전트에게 무엇을 만들어야 하는지, 어디에 저장해야 하는지를 분명히 지시합니다.

소통 프로토콜: 누가 누구에게, 언제 말하는지를 설계합니다

에이전트들이 자유롭게 대화할 수 있다고 해서, 소통 구조를 설계하지 않아도 된다는 뜻은 아닙니다. 프롬프트에서 소통 흐름을 명시적으로 정의하는 것이 훨씬 나은 결과를 가져옵니다.

"백엔드 개발이 완료되면, 프론트엔드 개발자에게 API 스펙을 전달하세요." "모든 개발이 완료되면, 결과물을 QA 에이전트에게 보내세요." "QA에서 이슈가 발견되면, 해당 파일의 담당 에이전트에게 수정을 요청하세요." 이렇게 명시적으로 지정하면, 에이전트들이 의존성을 이해하고 올바른 순서로 작업을 진행합니다.

수신자를 이름으로 지정하는 것도 중요합니다. "다른 에이전트에게 보내세요"는 모호합니다. "프론트엔드 개발자 에이전트에게 보내세요"는 정확합니다.

충돌 방지: 에이전트들이 서로를 방해하지 않도록 합니다

파일 소유권은 이미 언급했습니다. 그 외에 몇 가지 충돌 방지 전략이 있습니다.

권한 사전 승인:

에이전트들이 매번 권한 확인 때문에 작업을 중단하면, 전체 워크플로우가 느려집니다. 프로젝트 설정이나 내 컴퓨터 설정에서 특정 명령을 사전 승인해 두면, 중단 없이 작업이 흐릅니다. 팀원은 메인 세션의 권한을 상속받으므로, 메인 세션에서 우회 모드(Bypass Mode)를 설정하면 모든 팀원이 동일한 권한을 갖습니다.

계획 승인 모드(Plan Approval Mode):

팀원들이 작업을 시작하기 전에 계획을 수립하고, 그 계획을 메인 오케스트레이터가 승인한 뒤에만 실행하도록 설정할 수 있습니다. 초기에는 사용자가 직접 각 계획을 승인하는 방식으로 시작하고, 팀의 작동 방식에 익숙해지면 메인 세션에게 승인을 위임하는 것이 실용적입니다. 팀원 중 하나를 계획 검토 및 승인 전담으로 지정하는 구성도 가능합니다.

정상 종료(Graceful Shutdown):

작업이 끝나면, 메인 오케스트레이터가 각 팀원에게 종료 요청을 보냅니다. "작업을 저장하고 종료하세요." 팀원은 아직 진행 중인 작업이 있다면 "아직 끝나지 않았습니다"라고 응답할 수 있습니다. 모든 팀원이 완료를 확인한 뒤에 팀이 해체됩니다. 강제 종료하면 작업이 정리되지 않은 채 남을 수 있으므로, 정상 종료 절차를 거치는 것이 안전합니다.

팀 규모와 비용

에이전트 팀의 각 팀원은 독립적인 세션을 운영합니다. 3개의 에이전트가 있으면 비용이 약 3배가 됩니다. 5개라면 5배입니다.

권장하는 팀 규모는 3에서 5명입니다. 10명 이상의 큰 규모의 에이전트 스웜(Swarm)은 비용이 급격히 증가하고, 조율의 복잡도가 비례하여 높아집니다.

〔표 25-1〕 에이전트 팀 적합 여부 판단 기준

에이전트팀이 적합한 경우

에이전트팀이 과도한 경우

여러 전문 영역이 동시에 필요

순차적으로 처리해도 되는 작업

팀원 간 의존성과 소통이 필요

하나의 컨텍스트에서 처리 가능

병렬 작업 후 상호 검증이 필요

동일한 파일을 수정하는 작업

높은 품질이 요구되는 복잡한 작업

간단하고 빠른 작업

순차적으로 처리 가능한 작업이라면 보조 에이전트로 충분합니다. 에이전트 간 소통이 필요 없다면 보조 에이전트가 낫습니다. 같은 파일을 여러 에이전트가 수정해야 한다면, 에이전트 팀이든 보조 에이전트든 구조를 재설계해야 합니다.

에이전트 팀 설정 방법

에이전트 팀은 실험적 기능(Experimental Feature)으로, 기본적으로 비활성화되어 있습니다. 활성화하려면 프로젝트의

.claude/settings.local.json

에 환경 변수를 추가해야 합니다. 클로드 코드 공식 문서의 에이전트 팀 페이지에서 해당 JSON을 복사하여 설정 파일에 붙여넣으면 됩니다.

에이전트 팀을 호출하는 프롬프트의 구조는 다음과 같은 패턴을 따릅니다.

1.

목표 설정:

팀 전체가 달성해야 할 목표를 명시합니다. 팀원들은 깨어날 때 아무런 컨텍스트가 없으므로, 메인 세션이 목표를 명확히 전달해야 합니다. 2.

팀 구성:

"3명의 팀원으로 구성된 팀을 소네트 모델로 만들어줘"와 같이, 팀 규모와 모델을 지정합니다. 3.

역할 정의:

각 팀원의 역할, 담당 영역, 소통 대상을 구체적으로 기술합니다. 4.

최종 산출물 정의:

메인 세션이 팀 작업 결과로 받아볼 최종 산출물을 명시합니다.

〔그림 26-4〕 에이전트 팀 프롬프트 구조: 목표 → 팀 구성 → 역할별 지시사항(소통 대상 포함) → 최종 산출물 정의〕

T-Mux 터미널에서 에이전트 팀을 실행하면, 분할 화면으로 각 에이전트의 작업을 실시간 관찰할 수 있습니다. 잘못된 방향으로 가고 있는 에이전트를 조기에 발견하고 종료시킬 수 있어, 불필요한 비용 낭비를 방지합니다. IDE 확장 프로그램에서는 에이전트의 내부 사고 과정을 상세히 볼 수 없으므로, 복잡한 에이전트 팀 작업에는 T-Mux 환경이 더 적합합니다.

보조 에이전트는 메인 세션의 지시 아래 독립적으로 작업하는 전문가입니다. 에이전트 팀은 공유된 목표를 향해 소통하고 협력하는 팀입니다. 둘 다 강력한 도구이지만, 적용 맥락이 다릅니다. 빠르고 효율적인 위임이 필요하면 보조 에이전트를, 복잡한 상호 의존성과 높은 품질이 요구되면 에이전트 팀을 선택합니다.

이 두 구조를 상황에 맞게 조합하는 것이 클로드 코드 운영자의 역량이며, 스킬과 에이전트를 엮어 워크플로우를 설계하는 다음 단계로 나아가는 기반이 됩니다.

전체 목차

책의 모든 장 보기

27장 초급 해킹 팁 10가지

전체 목차

책의 모든 장 보기

처음 만나는 속도의 차이

터미널에 클로드 코드를 처음 열었던 날을 떠올려 보겠습니다. 커서가 깜빡이는 검은 화면 앞에서 무엇을 입력해야 할지 몰라 한참을 머뭇거렸던 그 순간. 프롬프트 한 줄을 치고, 결과물을 받아보고, 다시 고치고, 또 고치고. 몇 시간이 지나도 원하는 결과물은 나오지 않았습니다. 그런데 같은 도구를 쓰는 다른 사람은 놀라울 정도로 빠르게 결과물을 뽑아냅니다.

도구가 다른 것이 아닙니다. 몇 가지 습관이 다를 뿐입니다.

이 장에서 소개하는 열 가지 팁은 클로드 코드를 처음 접한 사람이 가장 먼저 익혀야 할 기본기입니다. 하나하나는 작은 습관이지만, 이것들이 쌓이면 작업 속도와 결과물의 품질이 눈에 띄게 달라집니다.

팁 1. /init으로 프로젝트 초기화하기

이미 파일이 들어 있는 프로젝트 폴더를 클로드 코드로 열었다고 해 보겠습니다. 가장 먼저 해야 할 일은

/init

명령어를 입력하는 것입니다.

이 명령어를 실행하면 클로드 코드가 프로젝트 전체를 훑습니다. 폴더 구조, 파일 목록, 코드 컨벤션을 파악한 뒤

claude.md

파일을 자동으로 생성합니다. 이 파일은 프로젝트의 치트시트(cheat sheet)와 같습니다. 아키텍처가 어떻게 구성되어 있는지, 핵심 파일은 무엇인지, 어떤 규칙을 따르는지가 정리되어 들어갑니다.

/init

을 실행해 두면 매 세션마다 프로젝트를 다시 설명할 필요가 없습니다. 클로드가 이미 맥락을 파악하고 있기 때문입니다.

새 프로젝트를 처음부터 시작하는 경우에는 어떨까요? 그때는 프로젝트의 목표, 기술 스택(tech stack), 주요 규칙 등을 직접 설명해 주면서

claude.md

파일을 함께 만들면 됩니다. 이 파일이 잘 갖춰져 있을수록 이후의 모든 작업이 수월해집니다.

〔그림 27-1〕

/init

실행 후 자동 생성된 claude.md 파일 예시

팁 2. 상태줄(Status Line) 설정하기

코딩에 몰두하다 보면 지금 얼마나 많은 컨텍스트(context)를 소비하고 있는지 잊기 쉽습니다. 한참 작업하다 문득 확인해 보니 컨텍스트 창이 거의 가득 차 있었던 경험, 한 번쯤 있을 것입니다.

터미널에서

/statusline

을 입력하고 어떤 정보를 보고 싶은지 알려 주면, 클로드 코드가 터미널 하단에 작은 대시보드를 만들어 줍니다. 현재 사용 중인 모델명, 컨텍스트 사용 비율, 비용 등을 실시간으로 표시할 수 있습니다.

상태줄의 진짜 가치는 컨텍스트 소진을 미리 감지할 수 있다는 점에 있습니다. 컨텍스트가 얼마 남지 않았는데 모르고 계속 작업하면, 클로드의 응답 품질이 급격히 떨어집니다. 상태줄이 있으면 그런 상황이 오기 전에 대비할 수 있습니다. 작은 스크립트 하나가 세션 전체의 품질을 좌우하는 셈입니다.

팁 3. 음성 입력 /voice 활용하기

키보드로 긴 프롬프트를 타이핑하는 것은 생각보다 시간이 많이 걸립니다. 머릿속에는 구체적인 요구사항이 있는데, 그것을 글로 옮기는 과정에서 핵심이 빠지기도 합니다.

클로드 코드에는

/voice

라는 네이티브 음성 입력 명령어가 탑재되어 있습니다. 터미널에 직접 말로 지시할 수 있습니다. "이 함수의 에러 처리를 개선해 줘"라고 입에서 나오는 대로 말하면, 그대로 프롬프트가 됩니다.

음성 입력의 장점은 속도만이 아닙니다. 말로 설명할 때는 자연스럽게 맥락과 의도가 함께 전달됩니다. 타이핑할 때 무의식적으로 줄이게 되는 설명이 음성에서는 풍부해집니다. 결과적으로 클로드가 의도를 더 정확하게 파악하게 됩니다.

별도의 음성 딕테이션(dictation) 앱을 활용하는 것도 좋은 방법입니다. 운영체제나 서드파티 앱을 통해 어디서든 음성으로 텍스트를 입력할 수 있으니, 클로드 코드뿐 아니라 다른 작업에서도 효율이 올라갑니다.

팁 4. 컨텍스트를 작게 유지하기

"많이 넣으면 많이 알겠지"라는 생각은 자연스럽지만, 클로드 코드에서는 오히려 독이 됩니다.

전체 코드베이스를 대화에 쏟아붓지 마십시오. 현재 작업에 필요한 정보만 제공해야 합니다. 큰 문제는 작은 단계로 쪼개고, 각 단계마다 해당 부분의 코드와 맥락만 전달합니다.

컨텍스트 창 안의 잡음(noise)이 줄어들수록 클로드의 성능은 올라갑니다. 이것은 사람도 마찬가지입니다. 회의실에서 열 가지 안건을 동시에 논의하면 어느 하나도 제대로 결론이 나지 않습니다. 한 번에 한 가지씩 집중해야 깊이 있는 결과가 나옵니다.

〔그림 27-2〕 컨텍스트를 작게 유지했을 때와 과도하게 채웠을 때의 응답 품질 비교

팁 5. /context로 모델이 세는 글 조각 소비 진단하기

컨텍스트를 작게 유지하라고 했는데, 대체 지금 무엇이 모델이 세는 글 조각(token)을 잡아먹고 있는지 어떻게 알 수 있을까요?

/context

명령어를 사용하면 됩니다. 이 명령어는 현재 세션에서 모델이 세는 글 조각이 어디에 얼마나 소비되고 있는지를 백분율로 보여 줍니다. 시스템 프롬프트가 차지하는 비중, 파일 내용이 차지하는 비중, MCP 연결 서버 정의가 차지하는 비중 등이 한눈에 드러납니다.

세션이 무겁게 느껴질 때, 응답이 느려지거나 품질이 떨어질 때,

/context

를 한번 실행해 보십시오. 의외의 곳에서 모델이 세는 글 조각이 대량으로 소모되고 있는 경우가 많습니다. 원인을 찾으면 구조를 재편해서 여유를 확보할 수 있습니다.

이 진단 습관은 팁 4의 "컨텍스트를 작게 유지하기"와 짝을 이룹니다. 원칙을 알더라도 현재 상태를 측정하지 못하면 실천이 어렵기 때문입니다.

팁 6. 60% 압축 규칙과 작업 간 초기화

컨텍스트 사용량이 60% 근처에 도달하면

/compact

명령어를 실행할 타이밍입니다.

이 명령어는 대화 히스토리를 압축합니다. 중요한 정보는 보존하면서 불필요한 부분을 걸러내는 방식입니다. 압축 후에도 작업을 이어갈 수 있으므로, 대화를 처음부터 다시 시작할 필요가 없습니다.

여기서 한 가지 유용한 기법이 있습니다.

/compact

을 실행할 때 보존할 내용을 명시적으로 지정할 수 있습니다. 예를 들어 "/compact, 하지만 API 연동 결정사항과 데이터베이스 스키마는 유지해 줘"라고 입력하면, 클로드가 나머지는 줄이면서 지정한 정보는 그대로 남겨 둡니다.

완전히 다른 작업으로 전환할 때는

/clear

를 사용합니다. 대화 히스토리를 깨끗이 지우고 새 세션처럼 시작하는 것입니다.

claude.md

파일과 프로젝트 파일은 그대로 남아 있으니, 맨땅에서 시작하는 것과는 다릅니다. 작업의 성격이 바뀔 때마다

/clear

로 리셋하는 습관을 들이면, 이전 작업의 잔여 맥락이 새 작업을 오염시키는 일을 막을 수 있습니다.

팁 7. 계획 모드(Plan Mode) 우선

Shift+Tab을 누르면 클로드 코드의 모드를 전환할 수 있습니다. 그중 계획 모드는 클로드가 코드를 읽고 조사할 수는 있지만, 아무것도 수정하지 않는 모드입니다.

이것이 중요한 이유가 있습니다.

계획 모드에서 클로드는 작업 계획을 세웁니다. 어떤 파일을 수정해야 하는지, 어떤 순서로 접근할 것인지, 추가로 확인해야 할 사항은 무엇인지를 정리합니다. 필요하면 명확하지 않은 부분에 대해 질문을 던지기도 합니다.

이 과정을 거친 뒤 계획 모드를 해제하고 실행을 지시하면, 클로드가 잘못된 방향으로 코드를 수정해서 되돌려야 하는 횟수가 눈에 띄게 줄어듭니다. 계획 없이 바로 코딩에 뛰어드는 것과 설계를 먼저 하고 코딩에 들어가는 것의 차이. 사람이든 AI든 원리는 같습니다.

〔그림 27-3〕 계획 모드에서 클로드가 작성한 작업 계획 예시

팁 8. 에이전트를 주니어 개발자처럼 대하기

"이 기능을 구현해"라고 직접 명령하는 것과 "사용자 이탈률을 줄이려면 어떤 접근이 좋을까?"라고 문제를 던지는 것. 이 둘의 차이는 생각보다 큽니다.

클로드에게 구체적인 명령만 내리면, 클로드는 시킨 대로만 합니다. 하지만 문제를 던지면 클로드가 스스로 추론하고, 여러 선택지를 비교하고, 왜 특정 접근을 택했는지 설명합니다.

이 과정에서 두 가지 이점이 생깁니다. 우선, 클로드가 자기 논리를 세우고 나서 코드를 작성하기 때문에 결과물의 품질이 올라갑니다. 그리고 클로드의 추론을 검토하면서, 우리가 미처 생각하지 못한 고려사항을 발견할 수도 있습니다.

주니어 개발자에게 일을 맡길 때를 떠올려 보십시오. "이렇게 해"라고만 하면 그 사람은 성장하지 못합니다. "이 문제를 어떻게 풀면 좋을까?"라고 물으면 그 사람이 생각하고, 그 생각을 검토해 줄 수 있습니다. 클로드도 마찬가지입니다. 생각할 기회를 주면 더 나은 결과를 내놓습니다.

팁 9. 질문 유도 프롬프팅(Question-Driven Prompting)

계획 모드에서 클로드가 스스로 질문을 던지기도 하지만, 이것을 좀 더 적극적으로 활용하는 방법이 있습니다.

프롬프트에 이런 지시를 추가합니다. "내가 원하는 것과 네가 해야 할 일을 95% 확신할 때까지 계속 질문해 줘." 이렇게 하면 클로드가 모호한 부분을 하나씩 짚어가며 확인합니다. 타깃 사용자가 누구인지, 에러 발생 시 어떤 행동을 기대하는지, 성능 기준은 무엇인지. 우리가 미처 명시하지 못한 요구사항이 질문을 통해 드러납니다.

이 정렬 과정이 없으면 어떻게 될까요? 클로드가 나름대로 가정을 세우고 작업을 진행합니다. 결과물을 받아 보면 의도와 다른 부분이 세 군데, 네 군데 나옵니다. 수정을 요청하고, 또 수정하고. 이 왕복이 세 번만 반복돼도 컨텍스트는 빠르게 소진됩니다.

질문 유도 프롬프팅은 이 왕복 비용을 사전에 차단합니다. 처음에 5분 더 투자해서 요구사항을 명확히 하면, 이후의 30분을 아낄 수 있습니다.

팁 10. 투두 리스트에 자가 검증 삽입하기

클로드 코드는 작업을 시작할 때 투두 리스트(to-do list)를 만듭니다. 이 리스트에 검증 단계를 직접 끼워 넣는 것이 열 번째 팁입니다.

예를 들어 웹사이트를 만드는 작업이라면, 투두 리스트를 이렇게 구성할 수 있습니다.

1. 웹사이트 레이아웃 구현

2. 웹사이트 스크린샷을 찍고 레이아웃이 의도대로 나왔는지 확인

3. 크롬 개발자 도구(Chrome DevTools)를 열어 기능 오류가 없는지 점검

4. 반응형 디자인이 모바일에서도 정상 동작하는지 확인

5. 확인 결과를 반영하여 수정

이렇게 하면 클로드가 무언가를 만들고 바로 우리에게 넘기는 것이 아니라, 스스로 만든 것을 점검하고, 문제를 발견하면 수정한 뒤에 결과를 보여줍니다.

한 가지 추가 기법이 있습니다. "다음 투두로 넘어가기 전에, 현재 항목이 95% 완성됐다고 확신할 때까지 반복해"라고 지시하는 것입니다. AI가 한 번에 완벽한 결과를 내기는 어렵지만, 자가 검증 루프를 돌면 60%짜리 결과물이 90%까지 올라옵니다. 그 차이가 쌓이면 전체 프로젝트의 완성도가 확연히 달라집니다.

〔그림 27-4〕 자가 검증 단계가 포함된 투두 리스트 구성 예시

작은 습관이 만드는 큰 차이

열 가지 팁을 하나씩 돌아보았습니다.

/init

으로 시작하고, 상태줄로 현황을 파악하고, 음성으로 빠르게 지시하고, 컨텍스트를 가볍게 유지하며, 진단하고 압축하고, 계획을 먼저 세우고, 문제를 던지고, 질문을 유도하고, 검증을 자동화하는 것. 어느 하나 복잡한 기술이 아닙니다. 그저 습관입니다.

이 기본기가 몸에 배면, 다음 단계로 넘어갈 준비가 된 것입니다. 보조 에이전트를 활용한 병렬 작업, 커스텀 스킬 파일 제작, 비용 최적화처럼 좀 더 정교한 기법들이 기다리고 있습니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

28장 중급 해킹 팁 10가지

전체 목차

책의 모든 장 보기

속도에서 전략으로

초급 팁 열 가지를 익힌 뒤, 클로드 코드를 쓰는 패턴이 바뀌기 시작합니다.

/init

은 반사적으로 실행하고, 컨텍스트를 의식하며, 계획 모드로 먼저 설계하는 습관이 붙습니다. 그런데 어느 순간 벽에 부딪힙니다. 혼자 하나의 세션에서 하나의 작업을 처리하는 방식으로는 속도에 한계가 있습니다.

중급 팁은 이 한계를 넘기 위한 전략입니다. 여러 에이전트를 동시에 움직이고, 반복 작업을 자동화하고, 비용을 최적화하는 방법을 다룹니다.

팁 11. 보조 에이전트(Subagent) 병렬 작업 위임

한 사람이 조사, 코딩, 검증를 순서대로 하는 것과, 세 사람이 각각 하나씩 동시에 하는 것. 결과가 나오는 속도는 비교할 수 없습니다.

클로드 코드에서는 복잡한 작업을 줄 때 프롬프트 안에 "보조 에이전트를 활용해"라는 지시를 포함시킬 수 있습니다. 메인 세션이 작업을 분석한 뒤, 독립적인 보조 에이전트를 여러 개 생성합니다. 각 보조 에이전트는 자기만의 컨텍스트 창을 갖고, 자기만의 모델을 사용할 수 있습니다.

한 에이전트는 리서치를, 다른 에이전트는 검증 코드 작성을, 또 다른 에이전트는 대안적 접근법 탐색을 맡습니다. 이 모든 작업이 동시에 진행됩니다. 작업이 끝나면 각 보조 에이전트가 결과를 메인 에이전트에게 보고합니다. 메인 스레드(thread)는 깔끔하게 유지되면서, 실질적으로는 개발팀 여러 명이 붙은 것과 같은 효과를 얻습니다.

〔그림 28-1〕 메인 에이전트가 보조 에이전트에게 작업을 위임하는 흐름도

팁 12. 커스텀 스킬 파일 만들기

같은 종류의 작업을 반복할 때마다 프롬프트를 다시 작성하고 있다면, 커스텀 스킬(custom skill) 파일이 필요합니다.

.claude/skills/

디렉터리 안에 마크다운 파일을 만들어 둡니다. 예를 들어

techdebt.md

라는 파일에 기술 부채(technical debt)를 점검하는 절차를 상세히 기술해 놓으면, 이후에는 "기술 부채 점검해 줘"라고 말하는 것만으로 그 워크플로가 실행됩니다.

codereview.md

에 코드 리뷰 기준을 정리해 두면, 매번 리뷰 기준을 설명할 필요 없이 일관된 품질의 리뷰가 나옵니다.

스킬 파일의 진짜 힘은 팀 공유에 있습니다. 이 파일들을 깃허브(GitHub)에 커밋하면 팀 전체가 동일한 워크플로를 사용할 수 있습니다. 사람마다 다르게 진행되던 코드 리뷰나 배포 전 점검이, 스킬 파일 하나로 표준화됩니다. 회사의 SOP(Standard Operating Procedure)를 자동화하는 것과 다르지 않습니다.

팁 13. 하이쿠(Haiku) 모델로 보조 에이전트 비용 절감

보조 에이전트를 적극적으로 활용하기 시작하면 비용이 신경 쓰입니다. 여기서 한 가지 전략이 있습니다.

보조 에이전트에게 맡기는 작업이 대량의 텍스트를 읽고 핵심만 추출하는 것이라면, 오퍼스(Opus)급 모델을 쓸 이유가 없습니다. 하이쿠처럼 가볍고 저렴한 모델을 지정하면 됩니다.

구체적인 예를 들어 보겠습니다. 수십 개의 기사를 읽고 요약해 와야 하는 리서치 작업이 있습니다. 수십만 모델이 세는 글 조각을 처리해야 합니다. 이 작업을 오퍼스로 돌리면 비용이 상당합니다. 하지만 보조 에이전트를 하이쿠로 설정하면 비용이 극적으로 줄어듭니다. 보조 에이전트가 읽고 추려낸 핵심 요약만 메인 에이전트(오퍼스)에게 전달되므로, 품질이 중요한 최종 분석 단계에는 여전히 강력한 모델이 투입됩니다.

비용이 부담된다고 보조 에이전트 활용 자체를 포기하는 것은 아깝습니다. 모델 선택을 전략적으로 하면 비용을 통제하면서도 병렬 작업의 이점을 누릴 수 있습니다.

팁 14. claude.md 지속 갱신

claude.md

파일은 살아 있는 문서입니다. 프로젝트가 진행되면서 새로운 패턴이 발견되고, 예상치 못한 함정이 드러나고, 규칙이 추가됩니다. 이런 발견이 있을 때마다

claude.md

에 반영해야 합니다.

클로드에게 직접 지시할 수 있습니다. "방금 발견한 패턴을 claude.md에 기록해 줘." 다음 세션에서 클로드는 이미 그 교훈을 알고 있게 됩니다. 같은 실수를 반복하지 않고, 프로젝트에 대한 이해가 점점 깊어집니다.

하지만 주의할 점이 있습니다.

claude.md

는 시스템 프롬프트(system prompt)처럼 작동합니다. 이 파일의 내용이 매 대화에 로드되며, 그 분량만큼 컨텍스트 창을 차지합니다. 그래서 이 파일은 150에서 200줄 이내로 유지하는 것이 좋습니다. 정보를 계속 추가만 하면 파일이 비대해지고, 정작 대화에 쓸 컨텍스트가 부족해집니다.

새 정보를 추가할 때는 오래되거나 덜 중요한 정보를 동시에 정리하십시오. 이 균형이

claude.md

의 품질을 결정합니다.

〔그림 28-2〕 claude.md 파일의 적정 분량과 구조 예시

팁 15. claude.md에서 외부 파일로 라우팅

claude.md

를 150에서 200줄로 유지하라고 했지만, 프로젝트에 필요한 정보는 그보다 훨씬 많을 수 있습니다. 스타일 가이드, 비즈니스 맥락, 참조 문서, API 명세 등. 이 모든 것을

claude.md

안에 넣으면 파일이 금방 터집니다.

해결책은 라우팅(routing)입니다.

claude.md

에는 각 정보가 어디에 있는지만 적어 둡니다. "스타일 가이드는

/docs/style-guide.md

참조", "API 명세는

/docs/api-spec.md

참조"처럼요. 클로드는 필요할 때 해당 파일을 찾아가서 읽습니다.

이 방식의 장점은 분명합니다.

claude.md

가 가벼워지므로 컨텍스트 낭비가 줄어듭니다. 동시에 클로드가 접근할 수 있는 정보의 총량은 오히려 늘어납니다. 시스템 프롬프트에 모든 프로젝트 상태를 적을 필요는 없습니다. 그 상태가 기록된 파일의 위치만 알려 주면 충분합니다.

팁 16. 잘못된 방향 즉시 탈출 (Escape)

클로드가 응답을 생성하는 도중, 방향이 틀렸다는 것이 눈에 보일 때가 있습니다. 요청하지 않은 라이브러리를 설치하려 하거나, 전혀 다른 접근법으로 코드를 작성하기 시작할 때입니다.

이때 끝까지 기다리지 마십시오. Escape 키를 누르면 생성이 즉시 멈춥니다.

잘못된 방향으로 갈수록 소비되는 모델이 세는 글 조각은 낭비입니다. 이미 쓴 모델이 세는 글 조각은 되돌릴 수 없고, 잘못된 맥락이 컨텍스트에 쌓이면 이후 응답에도 영향을 줍니다. 빨리 끊고 방향을 수정해서 다시 프롬프트를 보내는 것이 전체적으로 훨씬 경제적입니다.

"어쩌면 마지막에 바로잡을지도 몰라"라는 기대로 기다리는 것은 대부분 시간과 모델이 세는 글 조각의 이중 손실로 끝납니다. 조타는 빠를수록 좋습니다.

팁 17. 출력물에 공격적 수정 의견

클로드가 내놓은 결과물이 "그럭저럭" 수준이라면, 그냥 받아들이지 마십시오.

"이건 충분하지 않아. 완전히 다른 접근으로 다시 해 봐." "이 부분은 너무 장황해. 더 간결하고 우아한 방식으로 바꿔 줘." 이렇게 높은 기준을 명시적으로 요구하면, 클로드는 놀라울 정도로 다른 결과물을 내놓는 경우가 많습니다.

두 번째 시도에서 더 나은 결과가 나오는 이유는 분명합니다. 첫 번째 결과물에 대한 수정 의견이 "하지 말아야 할 것"을 명확히 해 주기 때문입니다. 클로드는 이제 어떤 방향이 아닌지를 알고, 다른 경로를 탐색합니다.

여기서 한 단계 더 나아갈 수 있습니다. 개선된 결과물이 나오면, 그 개선 포인트를

claude.md

나 스킬 파일에 기록하도록 지시하십시오. "이 실수를 다시 반복하지 않도록 업데이트해 줘." 같은 종류의 실수가 반복되는 것을 구조적으로 차단할 수 있습니다.

팁 18. /rewind로 빠르게 되돌리기

Escape로 생성을 중단하는 것은 아직 진행 중인 응답을 멈추는 것입니다. 하지만 이미 완료된 응답이 문제였다면? 이미 클로드가 파일을 수정하고 대화가 진행된 뒤에 "아, 세 단계 전이 더 나았는데"라고 깨달았다면?

/rewind

명령어가 이 상황을 해결합니다. 대화의 이전 지점으로 되돌아갈 수 있습니다. 세션을 처음부터 다시 시작할 필요가 없습니다. 잘못 틀어진 지점까지만 되감고, 거기서 다른 방향으로 진행하면 됩니다.

/rewind

는 Escape와 함께 사용하면 강력한 조합이 됩니다. 진행 중인 잘못은 Escape로 멈추고, 이미 지나간 잘못은

/rewind

로 되돌립니다. 이 두 가지 수단이 있으면 클로드 코드 세션에서 돌이킬 수 없는 실수란 사실상 없습니다.

팁 19. 훅(Hooks) 알림 설정

클로드 코드 세션 여러 개를 동시에 돌리기 시작하면 새로운 문제가 생깁니다. 어느 세션이 끝났는지, 어느 세션이 내 입력을 기다리고 있는지 파악하기 어려워집니다.

/hooks

명령어를 사용하면 알림 훅을 설정할 수 있습니다. 자연어로 지시할 수도 있습니다. "이 세션이 끝나면 알림 소리를 울려 줘." 클로드 코드가 작업을 완료했을 때 시스템 알림이 울리도록 설정하는 것입니다.

이렇게 해 두면 작업이 끝날 때까지 터미널을 쳐다보고 있을 필요가 없습니다. 다른 작업을 하다가 알림이 울리면 돌아가서 결과를 확인하고, 다음 지시를 내리면 됩니다.

15개의 세션을 동시에 돌리고 있다고 상상해 보십시오. 알림 없이는 어느 것이 끝났는지 확인하려고 터미널을 하나하나 전환해 봐야 합니다. 알림이 있으면 "딩" 소리가 나는 곳에만 가면 됩니다. 사소해 보이지만, 다중 세션 운영에서는 필수적인 설정입니다.

〔그림 28-3〕 훅 설정 후 알림이 동작하는 화면 예시

팁 20. 스크린샷 활용 시각적 자가 점검

클로드는 볼 수 있습니다. 텍스트만 처리하는 것이 아니라, 이미지를 분석할 수 있다는 의미입니다.

이 능력을 가장 효과적으로 활용하는 방법은 시각적 자가 점검 루프(self-check loop)를 만드는 것입니다. 웹사이트를 만드는 작업이라면, 이런 흐름을 구성합니다.

1. 웹사이트 디자인 구현

2. 스크린샷 촬영

3. 스크린샷을 분석하여 레이아웃 문제 식별

4. 문제 수정

5. 다시 스크린샷 촬영 및 분석

이 루프를 세 번 정도 반복한 뒤에 첫 번째 버전(V1)을 보여주도록 설정하면, 자가 점검 없이 바로 나온 V1과는 완성도 차이가 확연합니다.

스크린샷 활용은 웹 디자인에만 국한되지 않습니다. 에러 메시지 화면을 찍어서 보여줄 수도 있고, 참고하고 싶은 다른 사이트의 디자인을 찍어서 영감의 소스로 제공할 수도 있습니다. 텍스트로 설명하기 어려운 시각적 정보를 전달하는 데 스크린샷은 가장 효율적인 수단입니다.

〔그림 28-4〕 스크린샷 기반 자가 점검 루프 다이어그램

기본기 위에 전략을 얹다

중급 팁 열 가지를 살펴보았습니다. 이 팁들의 공통점은 "혼자서 하나씩"이라는 한계를 넘는 전략이라는 것입니다. 보조 에이전트로 병렬화하고, 스킬 파일로 자동화하고, 모델 선택으로 비용을 최적화하고, 훅으로 여러 세션을 관리하고, 수정 의견과 되감기로 품질을 끌어올립니다.

이 전략들이 손에 익으면, 한 걸음 더 나아갈 준비가 됩니다. 브라우저를 자동화하고, MCP 연결 서버를 연결하고, 에이전트가 스스로 에이전트를 만들게 하는 고급 기법들이 그 다음 단계에 있습니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

29장 고급 해킹 팁 7가지

전체 목차

책의 모든 장 보기

한계를 밀어붙이는 사람들

터미널 다섯 개가 동시에 열려 있습니다. 왼쪽 모니터에서는 세 개의 클로드 코드 세션이 각각 다른 기능을 만들고 있고, 오른쪽 모니터에서는 브라우저가 자동으로 열리면서 방금 생성된 웹 앱의 폼을 채우고 버튼을 누릅니다. 알림음이 울릴 때마다 완료된 세션을 확인하고, 다음 작업을 할당합니다.

공상 과학이 아닙니다. 클로드 코드의 고급 기능을 조합하면 실제로 가능한 작업 방식입니다. 이 장에서 다루는 일곱 가지 팁은 클로드 코드를 극한까지 활용하려는 사람들을 위한 것입니다.

팁 21. Chrome DevTools로 기능 검증

스크린샷은 시각적 문제를 잡아냅니다. 하지만 버튼을 눌렀을 때 실제로 동작하는지, 폼 제출이 정상적으로 처리되는지는 스크린샷만으로 알 수 없습니다.

클로드 코드는 크롬 개발자 도구(Chrome DevTools)를 직접 활용할 수 있습니다. 브라우저를 열고, 앱과 상호작용하고, 콘솔 에러를 확인하고, 네트워크 요청이 정상적으로 오가는지 점검합니다.

이것은 스크린샷 루프의 확장입니다. 스크린샷이 "눈으로 보는 검사"라면, DevTools 활용은 "손으로 만져보는 검사"입니다. 프론트엔드 작업에서 이 두 가지를 조합하면, 클로드가 만든 결과물의 시각적 완성도와 기능적 안정성을 동시에 확보할 수 있습니다.

한 가지 더. DevTools 검증은 명시적 API가 없는 서비스와 상호작용해야 할 때도 유용합니다. 로그인된 상태에서 브라우저를 통해 데이터를 읽어오거나, 양식을 채우거나, 버튼을 클릭하는 작업을 클로드에게 위임할 수 있습니다.

〔그림 29-1〕 클로드 코드가 Chrome DevTools를 통해 앱 기능을 검증하는 화면 예시

팁 22. MCP 연결 서버 체이닝(Chaining)

MCP(Model Context Protocol) 서버는 클로드 코드에 외부 기능을 연결하는 강력한 수단입니다. 노션(Notion) MCP를 연결하면 노션 데이터베이스를 읽고 쓸 수 있고, 깃허브 MCP를 연결하면 풀 리퀘스트(Pull Request)를 관리할 수 있습니다.

그런데 MCP 연결 서버를 하나만 쓸 이유는 없습니다. 여러 MCP 연결 서버를 동시에 연결하면, 클로드가 서로 다른 서비스를 넘나들며 작업합니다. 노션에서 기획 문서를 읽고, 그 내용을 바탕으로 깃허브에 이슈를 생성하고, 구글 캘린더에 마감일을 등록하는 흐름이 하나의 세션 안에서 이루어집니다.

다만 주의할 점이 있습니다. MCP 연결 서버마다 도구 정의(tool definition)가 컨텍스트 창에 로드됩니다. 서버를 많이 연결할수록 기본 모델이 세는 글 조각 소비가 늘어납니다. 현재 작업에 필요한 서

버만 활성화하고, 쓰지 않는 서버는 비활성화하는 관리가 필요합니다.

앞서 팁 5에서 배운

/context

명령어로 각 MCP 연결 서버가 얼마나 모델이 세는 글 조각을 잡아먹고 있는지 확인할 수 있습니다.

모든 MCP를 연결해 놓는 것이 능사가 아닙니다. 프로젝트에서 실제로 한두 가지 기능만 필요하다면, MCP 대신 직접 API 엔드포인트(endpoint)를 하드코딩하는 것이 모델이 세는 글 조각 효율 면에서 나을 수 있습니다. 상황에 맞는 판단이 필요합니다.

〔그림 29-2〕 복수의 MCP 연결 서버가 체이닝되어 작동하는 구조도

팁 23. 에이전트가 에이전트를 만들게 하기

보조 에이전트는 메인 에이전트가 작업을 위임하는 대상입니다. 그런데 한 단계 더 나아갈 수 있습니다. 클로드에게 "이 작업을 수행할 에이전트를 설계해 줘"라고 지시하는 것입니다.

이것은 작업을 위임하는 것이 아니라, 작업을 수행할 체계 자체를 설계하게 하는 것입니다. 클로드가 에이전트의 역할 분담, 사용할 모델, 실행 순서, 검증 방법을 정하고, 그 설계에 따라 에이전트들을 생성합니다.

에이전트 팀(Agent Teams) 기능을 사용하면 이 접근이 더 효과적입니다. 일반 보조 에이전트는 서로 통신할 수 없습니다. 각자 독립적으로 작업하고 결과를 메인에 보고할 뿐입니다. 에이전트 팀은 다릅니다. 팀 내의 에이전트들이 작업 목록을 공유하고, 서로 소통하며, 상대방의 완료 여부를 인지합니다. 다른 에이전트에게 작업을 할당할 수도 있습니다.

비용은 더 들고 실행 시간도 깁니다. 하지만 규모가 큰 프로젝트에서 훨씬 응집력 있는 결과물이 나옵니다. 에이전트들이 각각 다른 곳을 바라보고 만든 코드 조각들을 억지로 합치는 것과, 에이전트들이 서로 조율하면서 만든 코드를 통합하는 것은 결과의 일관성이 다릅니다.

팁 24. 여러 클로드 코드 세션 동시 운영

한 프로젝트 폴더에서 터미널 여러 개를 열고 각각에서 클로드 코드를 실행하면, 같은 파일을 동시에 수정하다 충돌이 발생할 위험이 있습니다.

깃 워크트리(Git Worktree)와 결합하면 이 문제가 해결됩니다. 각 세션이 별도의 브랜치, 별도의 디렉터리에서 작업하므로 서로 간섭하지 않습니다. 클로드 코드에서는

--worktree

플래그를 사용해 간단히 워크트리를 생성할 수 있습니다.

claude --worktree feature-login

claude --worktree bugfix-payment

claude --worktree refactor-api

이렇게 하면 세 개의 독립적인 작업 공간이 만들어지고, 각각에서 클로드 코드 세션이 돌아갑니다. 네 개, 다섯 개를 동시에 돌릴 수도 있습니다. 작업이 끝나면 각 브랜치를 메인에 병합(merge)하면 됩니다.

이 방식을 극한으로 밀어붙이는 사람도 있습니다. 클로드 코드의 핵심 개발자 중 한 명은 터미널에서 항상 다섯 개의 에이전트를 돌리고, 웹에서 추가로 열 개 정도를 운영한다고 합니다.

〔그림 29-3〕 여러 워크트리에서 클로드 코드 세션을 동시 운영하는 화면 구성 예시

팁 25. 커스텀 슬래시 명령어 만들기

클로드 코드의 기본 슬래시 명령어(

/init

,

/compact

,

/rewind

등) 외에도 나만의 명령어를 만들 수 있습니다.

스킬 파일과 비슷하지만, 슬래시 명령어는 더 직접적인 호출 방식을 제공합니다. 자주 수행하는 워크플로를 하나의 명령어로 묶어서, 한 줄 입력으로 복잡한 절차를 시작할 수 있습니다.

예를 들어

/deploy-check

라는 커스텀 명령어를 만들어 둔다고 합시다. 이 명령어 하나로 검증 실행, 린트(lint) 검사, 구축 확인, 스테이징 환경 배포까지 순차적으로 진행되도록 구성할 수 있습니다. 매번 "검증 돌리고, 린트 확인하고, 구축해 보고..."라고 타이핑하는 대신

/deploy-check

한 줄이면 됩니다.

팀 전체가 사용하는 커스텀 명령어를 만들면 워크플로의 표준화에 기여합니다. 신규 팀원이 합류해도 "이 명령어들 사용하면 돼"라고 안내하면, 숙련된 팀원과 동일한 품질의 작업 흐름을 바로 시작할 수 있습니다.

팁 26. 브라우저 자동화

크롬 DevTools를 이용한 기능 검증에서 한 걸음 더 나아가면, 본격적인 브라우저 자동화 영역에 들어섭니다.

클로드 코드가 브라우저를 열고, 웹사이트를 탐색하고, 양식을 채우고, 버튼을 클릭하고, 데이터를 읽어옵니다. 명시적 API가 제공되지 않는 서비스에서 데이터를 수집하거나, 반복적인 웹 작업을 자동화하는 데 활용할 수 있습니다.

참고하고 싶은 웹사이트의 디자인을 가져오는 것도 가능합니다. 마음에 드는 사이트의 스크린샷을 찍거나 HTML 스타일을 추출해서 클로드에게 전달하면, 그 디자인 패턴을 재현합니다. 물론 그대로 복사하는 것이 아니라, 영감의 소스로 활용하고 자신만의 요소를 더하는 것이 바람직합니다.

브라우저 자동화의 범위는 넓습니다. 다만 이미 로그인된 상태에서 탐색, 클릭, 입력 등의 작업을 수행하는 형태가 가장 안정적입니다. 자동화 범위를 정할 때는 어디까지를 클로드에게 맡기고 어디부터를 사람이 개입할지 미리 정해 두는 것이 좋습니다.

〔그림 29-4〕 브라우저 자동화를 통한 웹 데이터 수집 흐름도

팁 27. 보안 주의사항: API 키 관리

고급 기능을 적극 활용할수록 보안의 중요성도 커집니다. MCP 연결 서버 여러 개를 연결하고, 브라우저 자동화를 하고, 다중 세션을 돌리면, 그만큼 API 키와 인증 정보가 여러 곳에 노출될 수 있습니다.

몇 가지 원칙을 지켜야 합니다.

API 키는 코드에 직접 넣지 않습니다. 환경 변수(environment variable)나

.env

파일을 통해 관리합니다.

.env

파일은 반드시

.gitignore

에 추가해서 깃허브에 올라가지 않도록 합니다. 한 번이라도 커밋 히스토리에 들어간 키는 이미 노출된 것으로 간주하고 즉시 재발급해야 합니다.

클로드 코드 세션에서도 마찬가지입니다. API 키가 포함된 응답을 클로드가 생성했다면, 그 내용이 컨텍스트에 남습니다. 가능한 한 키 값 자체가 대화에 등장하지 않도록 구조를 설계해야 합니다.

VS Code 확장 프로그램(extension)이나 MCP 연결 서버를 설치할 때도 보안 검증이 필요합니다. 개발자의 신원, 아웃바운드 네트워크 호출 여부, 데이터 수집 여부, 셸 명령어 실행 가능 여부 등을 사전에 확인하십시오. 확인 없이 설치했다가 크리덴셜(credential)이 외부로 유출되면 돌이킬 수 없습니다.

〔그림 29-5〕 API 키 관리 체크리스트

도구의 깊이를 아는 사람

고급 팁 일곱 가지는 클로드 코드라는 도구의 깊이를 보여줍니다. 브라우저를 제어하고, 에이전트가 에이전트를 설계하게 하고, 수십 개의 세션을 동시에 관리하면서도 보안을 놓지 않는 것. 이것이 도구를 제대로 활용하는 수준입니다.

이런 기법들을 안전하게 사용하려면, 깃 워크트리의 작동 원리와 클로드 코드의 권한 설정 체계를 이해해야 합니다. 병렬 작업의 안전망이 되는 워크트리, 그리고 자율성과 통제 사이의 균형을 잡아주는 권한 설정이 그 기반입니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

30장 깃 워크트리 — 안전한 병렬 개발의 비결

전체 목차

책의 모든 장 보기

긴급 전화 한 통

새로운 결제 기능을 만들고 있습니다. 코드를 절반쯤 작성했는데, 슬랙 알림이 울립니다. 운영 서버에서 로그인 버그가 발생했다는 긴급 보고입니다. 지금 당장 고쳐야 합니다.

문제는, 지금 작업 중인 코드가 아직 완성되지 않았다는 것입니다. 커밋하기에는 미완성이고, 그렇다고 변경사항을 버릴 수도 없습니다. 긴급 수정을 위해 브랜치를 전환하려면 현재 작업을 어딘가에 임시 저장해야 합니다.

git stash

로 넣어 두고, 브랜치를 바꾸고, 수정하고, 다시 원래 브랜치로 돌아와서 스태시를 꺼내고... 복잡합니다. 실수가 끼어들 여지가 곳곳에 있습니다.

깃 워크트리(Git Worktree)는 이 상황을 근본적으로 해결합니다.

워크트리란 무엇인가

깃(Git)의 기본 구조에서 하나의 저장소(repository)는 하나의 작업 디렉터리와 연결됩니다. 그 디렉터리 안에서 한 번에 하나의 브랜치만 체크아웃할 수 있습니다. 다른 브랜치에서 작업하려면 현재 브랜치의 변경사항을 정리한 뒤 브랜치를 전환해야 합니다.

워크트리는 이 제약을 깨뜨립니다. 하나의 저장소에서 여러 브랜치를 동시에 체크아웃할 수 있습니다. 각 브랜치가 별도의 폴더에 존재합니다.

비유하자면 이렇습니다. 기존 방식은 책상이 하나뿐인 사무실입니다. 프로젝트 A의 서류를 펼쳐 놓고 작업하다가 프로젝트 B를 해야 하면, A의 서류를 전부 치우고 B의 서류를 꺼내야 합니다. 워크트리는 책상을 여러 개 놓는 것입니다. A 서류는 첫 번째 책상에 그대로 펼쳐 두고, 두 번째 책상에서 B 작업을 합니다. 각 책상은 독립적이지만, 같은 사무실(저장소) 안에 있습니다.

〔그림 30-1〕 기존 단일 작업 디렉터리 vs. 워크트리 복수 디렉터리 구조 비교

워크트리의 기본 사용법

워크트리를 생성하는 깃 명령어는 간결합니다.

git worktree add ../hotfix-login hotfix/login-bug

이 명령어는 현재 저장소에서

hotfix/login-bug

라는 브랜치를 체크아웃한 새 작업 디렉터리를

../hotfix-login

경로에 만듭니다. 기존 작업 디렉터리는 건드리지 않습니다.

이제 두 개의 폴더가 있습니다. 원래 폴더에는 개발 중인 결제 기능 코드가 그대로 남아 있고, 새 폴더에는 긴급 수정용 브랜치가 깨끗하게 체크아웃되어 있습니다. 두 폴더를 오가며 작업할 수 있고, 서로의 파일에 영향을 주지 않습니다.

워크트리 목록을 확인하려면:

git worktree list

작업이 끝난 워크트리를 제거하려면:

git worktree remove ../hotfix-login

실제로 폴더를 통째로 복사하는 것이 아니기 때문에, 디스크 사용량도 효율적입니다. 깃의 내부 데이터(

.git

디렉터리)는 공유되고, 각 워크트리에는 해당 브랜치의 파일만 체크아웃됩니다.

〔그림 30-2〕 git worktree 명령어 실행 후 생성된 디렉터리 구조

클로드 코드의 네이티브 워크트리 지원

클로드 코드는 깃 워크트리를 네이티브로 지원합니다. 깃 명령어를 직접 입력하지 않아도 됩니다.

claude --worktree feature-payment

이 한 줄이면 클로드 코드가 자동으로 워크트리를 생성하고, 새 브랜치를 만들고, 해당 워크트리에서 클로드 코드 세션을 시작합니다. 깃 워크트리의 복잡한 명령어 체계를 몰라도 워크트리의 이점을 그대로 누릴 수 있습니다.

물론 세부적인 제어가 필요하다면 깃 명령어를 직접 사용할 수도 있습니다. 클로드 코드의 네이티브 지원은 편의를 위한 것이지, 깃의 기능을 제한하는 것이 아닙니다. 두 방식을 상황에 따라 혼용하면 됩니다.

다중 세션 운영: 실전 시나리오

워크트리의 진가는 클로드 코드와 결합했을 때 드러납니다. 각 워크트리에서 독립적인 클로드 코드 세션을 실행할 수 있기 때문입니다.

구체적인 시나리오를 그려 보겠습니다.

상황:

전자상거래 플랫폼을 개발 중입니다. 세 가지 작업이 동시에 필요합니다.

새로운 위시리스트 기능 개발

결제 페이지의 UI 리뉴얼

프로덕션에서 발생한 장바구니 버그 긴급 수정

실행:

터미널 A에서:

claude --worktree feature-wishlist

"위시리스트 기능을 구현해 줘. 사용자가 상품을 추가하고 제거할 수 있어야 하고, 목록을 공유할 수 있어야 해."

터미널 B에서:

claude --worktree redesign-checkout

"결제 페이지 UI를 모바일 우선으로 재설계해 줘. 현재 디자인의 스크린샷을 참고해."

터미널 C에서:

claude --worktree hotfix-cart

"장바구니에서 수량 변경 시 총액이 갱신되지 않는 버그를 수정해 줘."

세 세션이 동시에 작동합니다. 각 세션은 자기 워크트리 안에서만 파일을 수정하므로 충돌이 발생하지 않습니다. 긴급 수정이 먼저 끝나면 해당 브랜치를 메인에 병합하고, 나머지 기능은 계속 개발을 진행합니다.

〔그림 30-3〕 세 개의 워크트리에서 동시에 진행되는 클로드 코드 세션 구성도

기능 개발과 긴급 수정의 동시 처리

위 시나리오에서 핵심은 긴급 수정이 기존 개발 작업을 방해하지 않는다는 것입니다.

워크트리가 없었다면 어떻게 됐을까요? 위시리스트 기능을 반쯤 만들다가 작업을 멈추고, 변경사항을 스태시(stash)하거나 미완성 커밋을 만들고, 브랜치를 전환하고, 버그를 고치고, 다시 원래 브랜치로 돌아와서 스태시를 복원해야 합니다. 이 과정에서 스태시 충돌이 날 수도 있고, 미완성 커밋이 히스토리를 더럽힐 수도 있습니다.

워크트리에서는 이런 번거로움이 없습니다. 위시리스트 작업은 그 자리에 그대로 있습니다. 새 터미널을 열고, 새 워크트리에서 버그를 고칩니다. 끝나면 워크트리를 제거합니다. 위시리스트 작업으로

돌아갈 때 복원할 것도, 전환할 것도 없습니다. 원래 있던 터미널에서 계속하면 됩니다.

이 안전성은 클로드 코드를 여러 세션 돌릴 때 더욱 중요해집니다. 사람이 직접 코드를 작성하면 실수를 바로 인지할 수 있지만, AI 에이전트가 코드를 작성할 때는 서로의 작업을 덮어쓰는 사고가 은밀하게 벌어질 수 있습니다. 워크트리는 이 사고를 구조적으로 불가능하게 만듭니다.

워크트리 관리의 실용적 조언

워크트리를 많이 만들다 보면 관리가 필요합니다. 몇 가지 실용적인 팁을 정리합니다.

명명 규칙을 정하십시오.

feature-

,

hotfix-

,

refactor-

같은 접두사를 일관되게 사용하면, 어떤 워크트리가 어떤 목적인지 한눈에 파악됩니다.

작업이 끝난 워크트리는 즉시 제거하십시오.

방치하면 어떤 워크트리가 활성 상태인지 혼란이 생깁니다. 브랜치를 메인에 병합한 뒤에는

git worktree remove

로 깔끔하게 정리합니다.

워크트리 디렉터리의 위치를 일관되게 잡으십시오.

프로젝트 폴더와 같은 수준의 상위 디렉터리에 두거나, 프로젝트 내 특정 폴더를 지정하는 등 규칙을 만들어 두면, 워크트리를 찾으러 헤매는 일이 없어집니다.

〔그림 30-4〕 워크트리 관리 모범 사례 요약표

병렬 개발의 안전망

워크트리는 화려한 기술이 아닙니다. 깃의 기본 기능을 확장한 것에 불과합니다. 하지만 이 확장이 만들어내는 차이는 큽니다.

하나의 브랜치에서 하나의 작업만 할 수 있다는 제약이 사라지면, 작업 방식이 근본적으로 바뀝니다. 긴급 수정 때문에 기능 개발을 중단하지 않아도 됩니다. 여러 에이전트가 같은 프로젝트에서 동시에 일하면서도 서로 방해하지 않습니다. 실험적인 시도를 별도의 워크트리에서 마음껏 해 볼 수 있습니다.

이렇게 자유롭게 여러 세션을 운영하려면, 각 세션에 어느 정도의 권한을 줄 것인지에 대한 판단도 필요합니다. 에이전트가 파일을 자유롭게 수정하게 할 것인지, 아니면 매번 승인을 받게 할 것인지. 자율성과 통제 사이의 균형을 잡는 방법을 살펴볼 차례입니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

31장 권한 설정과 보안 실무

전체 목차

책의 모든 장 보기

"위험하게 권한 건너뛰기"라는 이름의 유혹

유튜브에서 클로드 코드 튜토리얼을 검색하면, 상당수 영상에서

--dangerously-skip-permissions

라는 플래그를 사용합니다. 이 플래그를 붙이면 클로드가 파일 수정, 명령어 실행, 패키지 설치 등 모든 작업을 승인 없이 자동으로 수행합니다. 화면에서 보기에는 깔끔합니다. 작업이 끊기지 않고 쭉 이어지니까요.

그런데 이 플래그의 이름에는 "dangerously(위험하게)"라는 단어가 붙어 있습니다. 이유가 있습니다. 클로드가 실수로 중요한 파일을 삭제하거나, 의도치 않은 시스템 명령어를 실행하거나, 운영 서버에 영향을 미치는 작업을 승인 없이 수행할 수 있다는 뜻입니다.

빠르게 작업하고 싶은 마음은 이해합니다. 하지만 속도와 안전 사이에서 균형을 잡는 더 나은 방법이 있습니다.

4단계 자율성 모델

클로드 코드의 권한 설정은 네 단계로 구성됩니다. 각 단계는 에이전트에게 부여하는 자율성의 정도를 나타냅니다.

1단계: Plan (계획)

가장 제한적인 모드입니다. 클로드가 파일을 읽고 분석할 수 있지만, 어떤 것도 수정하지 않습니다. 코드를 작성하지 않고, 명령어를 실행하지 않습니다. 계획만 세우고 보고합니다.

새 프로젝트의 구조를 파악하거나, 복잡한 문제의 접근 전략을 논의할 때 적합합니다.

2단계: Ask Before (사전 승인)

클로드가 작업을 수행할 수 있지만, 매 단계마다 사용자의 승인을 받습니다. "이 파일을 수정해도 될까요?", "이 명령어를 실행해도 될까요?" 하나하나 물어봅니다.

안전하지만 느립니다. 사용자가 계속 화면 앞에 앉아서 승인 버튼을 눌러야 하기 때문입니다. 중요한 시스템에서 민감한 작업을 할 때 선택하는 모드입니다.

3단계: Auto Edit (자동 편집)

이 단계에서는 파일 편집은 자동으로 수행하되, 시스템 명령어 실행은 여전히 승인을 요구합니다. 코드 작성과 수정은 자유롭게 하되,

rm

,

npm publish

,

git push

같은 명령어는 사람이 확인한 뒤에 실행됩니다.

대부분의 일상적 개발 작업에 적합한 균형점입니다.

4단계: Bypass (우회)

모든 제한이 해제됩니다.

--dangerously-skip-permissions

에 해당하는 단계입니다. 클로드가 모든 작업을 자동으로 수행합니다.

이 모드는 사용자가 화면 앞에서 실시간으로 감독할 때만 사용해야 합니다. 자리를 비운 상태에서 바이패스 모드로 에이전트를 돌려 놓는 것은 위험합니다.

〔그림 31-1〕 4단계 자율성 모델 다이어그램 — Plan, Ask Before, Auto Edit, Bypass

허용 목록과 거부 목록으로 세밀하게 제어하기

4단계 모델보다 정밀한 제어가 필요할 때는 허용 목록(allow list)과 거부 목록(deny list)을 활용합니다.

이 방식은 "전부 허용하거나 전부 차단하는" 이분법이 아니라, "안전하다고 확인된 명령어만 허용하고, 위험한 명령어는 명시적으로 차단하는" 세밀한 설정입니다.

허용 목록에

git commit

,

npm test

,

npm run build

같은 안전한 명령어를 넣고, 거부 목록에

rm -rf

,

git push --force

,

DROP TABLE

같은 파괴적 명령어를 넣습니다.

여기서 중요한 규칙이 하나 있습니다. 거부 목록이 허용 목록보다 우선합니다. 허용 목록에

rm

이 포함되어 있더라도 거부 목록에도

rm

이 있으면, 거부됩니다. 안전장치가 항상 이깁니다.

이렇게 설정하면

--dangerously-skip-permissions

와 거의 같은 속도를 얻으면서도, 치명적인 명령어는 차단됩니다. 속도를 포기하지 않고 안전을 확보하는 현실적인 방법입니다.

{

"permissions": {

"allow": [

"git commit",

"git add",

"npm test",

"npm run build",

"npm run dev"

],

"deny": [

"rm -rf",

"git push --force",

"git reset --hard",

"DROP",

"DELETE FROM"

]

}

〔그림 31-2〕 허용 목록과 거부 목록 설정 예시 화면

VPS 호스팅과 원격 클로드 코드 운영

노트북을 닫으면 클로드 코드 세션이 끊깁니다. 장시간 작업이나 상시 모니터링이 필요한 경우, 이것은 제약이 됩니다.

VPS(Virtual Private Server, 가상 사설 서버)에 클로드 코드를 설치하면 이 문제가 해결됩니다. 서버는 24시간 돌아가므로, 노트북을 닫아도 세션이 유지됩니다. SSH(Secure Shell)로 원격 접속하면 언제 어디서든 세션을 이어갈 수 있습니다.

텔레그램(Telegram) 연동을 설정해 두면, 폰에서 클로드 코드에 지시를 내릴 수도 있습니다. 출근길 지하철에서 "어젯밤 돌려 놓은 검증 결과 어때?"라고 물으면 답이 옵니다.

클로드 코드에는 원격 제어(remote control) 기능도 있습니다. 내 컴퓨터 컴퓨터에서 실행 중인 세션을 브라우저나 폰에서 조종할 수 있습니다. 코드는 내 컴퓨터 머신을 떠나지 않으면서, 제어만 원격으로 합니다. 사무실 컴퓨터에서 작업을 시작하고, 커피를 마시러 나가면서 폰으로 방향을 잡아줄 수 있습니다.

장기 실행 세션을 운영할 때 권한 설정은 더 신중해야 합니다. 사용자가 자리를 비운 상태에서 에이전트가 돌아가므로, 바이패스 모드는 적합하지 않습니다. 허용 목록과 거부 목록을 세밀하게 구성하고, 필요한 경우 승인이 필요한 작업에서는 에이전트가 대기하도록 설정하는 것이 안전합니다.

〔그림 31-3〕 VPS에서 클로드 코드를 원격 운영하는 구조도

팀 환경에서의 보안

혼자 사용할 때와 팀에서 사용할 때는 보안의 차원이 다릅니다. 한 사람의 실수가 팀 전체에 영향을 미칠 수 있기 때문입니다.

API 키 관리

팀 프로젝트에서 API 키를 코드에 직접 넣는 것은 절대 해서는 안 됩니다. 한 명이 키를 하드코딩하고 커밋하면, 그 키는 깃 히스토리에 영원히 남습니다. 비공개 저장소라 해도 안전하지 않습니다.

API 키는 환경 변수로 관리합니다.

.env

파일에 키를 저장하고, 이 파일은

.gitignore

에 반드시 추가합니다. 팀원 각자가 자신의

.env

파일을 내 컴퓨터에서 관리하고, 키 값 자체는 안전한 채널(비밀번호 관리자, 암호화된 메시지 등)로 공유합니다.

settings.local.json 활용

클로드 코드의 설정 파일에는 두 가지 유형이 있습니다. 프로젝트 전체에 적용되는 공유 설정과 개인 환경에만 적용되는 내 컴퓨터 설정입니다.

settings.local.json

은 개인 설정 파일입니다. 이 파일에는 개인의 권한 설정, 환경별 경로, 개인 MCP 연결 서버 구성 등 자기 환경에만 해당하는 내용을 넣습니다. 이 파일은 깃에 커밋하지 않습니다.

프로젝트 공통 설정은 별도의 공유 설정 파일에 넣고 커밋합니다. 팀원 모두가 동일한 기본 규칙 하에서 작업하되, 개인별 차이는

settings.local.json

에서 처리하는 구조입니다.

최소 권한 원칙

팀 환경에서 권한 설정의 기본 원칙은 간결합니다. 각 역할에 필요한 최소한의 권한만 부여합니다.

프론트엔드 개발자의 클로드 코드 세션이 데이터베이스 삭제 명령어를 실행할 수 있어야 할 이유는 없습니다. 배포 담당자가 아닌 사람의 세션에서 프로덕션 배포 명령어가 허용될 이유도 없습니다. 역할별로 허용 목록과 거부 목록을 다르게 구성하면, 실수의 피해 범위를 구조적으로 제한할 수 있습니다.

〔그림 31-4〕 팀 환경에서의 권한 설정 구조 — 공유 설정과 개인 설정의 분리

보안은 속도의 적이 아닙니다

권한 설정과 보안 관리에 시간을 쓰는 것이 번거롭게 느껴질 수 있습니다. "그냥

--dangerously-skip-permissions

쓰면 안 되나?"라는 생각이 듭니다. 실제로 개인 프로젝트에서 간단한 실험을 할 때는 그렇게 해도 큰 문제가 없을 수 있습니다.

하지만 프로젝트가 커지고 팀원이 늘고 운영 서버가 관여하기 시작하면, 보안 설정 없이 달리는 것은 안전벨트 없이 고속도로를 달리는 것과 같습니다. 사고가 나기 전에는 불편함만 느끼지만, 사고가 나면 그때는 이미 늦습니다.

허용 목록과 거부 목록을 한 번 세밀하게 구성해 놓으면, 이후에는 바이패스 모드와 거의 같은 속도로 작업하면서도 치명적 실수는 차단됩니다. 처음에 한 시간 투자해서 이후의 모든 세션을 안전하게 만드는 것. 이보다 효율적인 시간 투자는 드뭅니다.

클로드 코드를 진지하게 활용하려는 사람이라면, 권한 설정은 선택이 아니라 기본입니다. 도구의 힘이 강할수록, 그 힘을 통제하는 체계도 강해야 합니다.

기술적 역량과 보안 의식을 갖추었다면, 이제 이 능력을 실제 사업으로 전환하는 단계가 남아 있습니다. 클로드 코드를 다루는 기술만으로는 충분하지 않습니다. 그 기술을 가진 사람이 시장에서 어떻게 첫 클라이언트를 확보하고, 가격을 책정하고, 지속 가능한 비즈니스를 만들 수 있는지—이 질문에 답하는 것이 다음 여정의 출발점입니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

32장 임포스터 신드롬을 넘어서

전체 목차

책의 모든 장 보기

세 가지 심리적 허들: 임포스터 신드롬, 가격 결정, 거절 공포

화면 앞에 앉아 있습니다. 첫 번째 잠재 클라이언트에게 보낼 메시지 창이 열려 있습니다. 커서가 깜빡입니다. 손가락은 키보드 위에 올려져 있지만 움직이지 않습니다. 머릿속에서 목소리가 들립니다. "내가 이걸 할 자격이 있나?" 이 순간을 경험하지 않은 AI 자동화 실무자는 거의 없습니다.

이것이 임포스터 신드롬(Imposter Syndrome)입니다. 자신의 능력을 실제보다 낮게 평가하고, 곧 들통날 것이라는 두려움에 사로잡히는 심리 현상입니다. 미국의 한 AI 자동화 컨설턴트도 이 감정을 고백했습니다. "내가 정말 돈을 받아도 되는 걸까?" "사기꾼처럼 느껴지면 어쩌지?" "아직 준비가 안 됐는데." 이런 생각이 머리를 가득 채웠다고 합니다.

흥미로운 점은 이 감정이 완전히 사라지는 시점이 오지 않는다는 것입니다. 수십 개의 프로젝트를 납품한 베테랑 컨설턴트도 새로운 산업의 클라이언트를 만날 때면 비슷한 불안을 느낍니다. 차이가 있다면, 베테랑은 그 감정을 인식하고 다루는 방법을 알고 있다는 것입니다. 초보자는 그 감정에 압도되어 행동을 멈춥니다.

임포스터 신드롬 옆에는 두 번째 허들이 서 있습니다. 가격 결정의 공포입니다.

사람들은 이 단계에서 너무 이른 시점에 막힙니다. 월 1,000만 원짜리 리테이너(Retainer)를 어떻게 제안하느냐는 질문에 빠져드는 것입니다. 하지만 아직 한 번도 가치를 전달해 본 적 없는 사람이 가격표부터 고민하는 것은 순서가 뒤바뀐 일입니다. 신뢰가 리테이너보다 먼저 와야 합니다.

처음 만난 사람에게 추천서를 부탁하는 것이 어색한 것처럼, 한 번도 함께 일해 보지 않은 클라이언트에게 월정액 계약을 제안하는 것은 자연스럽지 않습니다.

지금 집중해야 할 것은 가격표가 아니라 발을 들여놓는 것입니다. 진짜 문제를 해결하고, 그 결과를 눈에 보이게 보여 주는 것. 신뢰가 쌓이면 리테이너 대화는 자연스럽게 시작됩니다.

세 번째 허들은 거절에 대한 공포입니다. 이 두려움은 사람들이 인정하는 것보다 훨씬 깊이 뿌리박혀 있습니다. 새로운 영역에 발을 디딜 때, 무시당하는 경험은 피할 수 없습니다. 메시지에 답장이 오지 않습니다. "지금은 괜찮습니다"라는 정중한 거절이 돌아옵니다. 어떤 사람은 아예 읽지도 않습니다.

이 세 가지 허들—임포스터 신드롬, 가격 결정의 막막함, 거절의 두려움—은 서로 얽혀 있습니다. 자신을 사기꾼처럼 느끼니까 가격을 정하지 못하고, 가격을 정하지 못하니까 제안을 보내지 않고, 제안을 보내지 않으니까 거절조차 경험하지 못합니다. 그리고 아무 경험도 쌓이지 않으니까 임포스터 신드롬은 더 강해집니다.

〔그림 32-1〕 세 가지 심리적 허들의 악순환 구조 다이어그램〕

이 악순환을 끊는 방법은 생각보다 소박합니다. 완벽한 준비를 기다리는 대신, 작은 행동 하나를 실행하는 것입니다.

"저도 아직 배우는 중입니다"라는 정직함의 힘

미국의 한 AI 자동화 커뮤니티에서 첫 클라이언트를 5일 만에 확보한 실무자가 있었습니다. 그가 처음부터 전문가의 어투를 사용했을까요? 아닙니다. 그는 이렇게 말했습니다.

"저는 이 분야에 뛰어든 지 얼마 되지 않았습니다. 하지만 완전히 빠져 있습니다. 많이 배우고 있고, 많이 만들고 있습니다. 당신의 이 문제를 도와드리고 싶습니다."

이 메시지에는 과장이 없습니다. "업계 최고"라거나 "혁신적 솔루션"이라는 표현도 없습니다. 대신 세 가지가 있습니다. 정직한 현재 위치, 열정의 증거, 구체적 도움의 의지.

이런 접근이 효과적인 이유는 인간의 기본적인 신뢰 메커니즘과 관련이 있습니다. 사람은 완벽한 세일즈 피치보다 진정성 있는 대화에 마음을 엽니다. "저는 전문가입니다"라고 선언하는 것보다 "저도 배우는 중이지만 이 문제에는 자신 있습니다"라고 말하는 편이 거부감이 적습니다. 왜냐하면 후자는 과장할 동기가 없어 보이기 때문입니다.

과잉 약속(Over-promise)을 하지 않는 것이 핵심입니다. 임포스터 신드롬이 가장 위험해지는 순간은 그 불안을 보상하려고 능력 이상의 것을 약속할 때입니다. "어떤 것이든 자동화할 수 있습니다"라고 말하는 순간, 돌아올 길이 없는 다리를 건너게 됩니다. 그 대신 "이 부분은 자동화할 수 있고, 저 부분은 먼저 검증가 필요합니다"라고 경계를 짓는 것이 신뢰를 쌓는 방법입니다.

정직함은 약점이 아닙니다. 초기 단계에서 클라이언트의 리스크를 낮추는 전략입니다. 클라이언트 입장에서 생각해 보겠습니다. 이제 막 시작한 사람이 무료로 또는 낮은 비용으로 문제를 풀어 주겠다고 합니다. 실패하더라도 잃을 것이 거의 없습니다. 반면 성공하면 예상치 못한 가치를 얻게 됩니다. 이것은 클라이언트에게 위험 부담이 극히 낮은 제안입니다.

여기서 한 가지 구별해야 할 것이 있습니다. 정직함과 자기 비하는 다릅니다. "저는 아무것도 모릅니다"는 자기 비하입니다. "이 분야를 열심히 파고 있고 이런 것들을 만들어 봤습니다"는 정직한 자기 소개입니다. 전자는 클라이언트에게 불안을 주고, 후자는 가능성을 봅니다.

〔그림 32-2〕 정직한 자기 소개 메시지의 구조 예시〕

무료 파일럿으로 경험과 신뢰를 동시에 얻기

"무료로 일하면 내 가치를 떨어뜨리는 것 아닌가?" 이 질문은 거의 모든 초보 컨설턴트가 던지는 질문입니다. 답은 맥락에 따라 달라집니다.

경험이 전혀 없는 상태에서 무료 파일럿(Free Pilot)은 투자입니다. 비용을 지불하는 것은 자신의 시간이고, 얻는 것은 실전 경험, 포트폴리오, 추천 가능성입니다. 이미 여러 건의 프로젝트를 납품한 뒤에도 계속 무료로 일한다면 그때는 가치를 떨어뜨리는 것이 맞습니다. 하지만 첫 한두 건에서는 이야기가 다릅니다.

무료 파일럿의 핵심 원칙은 범위를 작게 잡는 것입니다. 전체 업무 프로세스를 자동화하겠다는 제안이 아니라, 한 가지 반복 작업을 자동화하겠다는 제안이어야 합니다. 구체적으로 이렇게 말할 수 있습니다.

"이 부분의 반복 작업을 자동화하는 작은 워크플로우를 무료로 만들어 드리겠습니다. 제 목표는 이 워크플로우가 실제로 시간을 절약해 준다는 것을 증명하는 것입니다. 대신 솔직한 수정 의견을 부탁드리겠습니다."

이 제안에는 네 가지 요소가 들어 있습니다.

범위 한정

: "이 부분"이라는 특정 영역 지정

결과 초점

: 시간 절약이라는 측정 가능한 성과

교환 조건

: 무료이지만 수정 의견이라는 대가 존재

종료 지점

: 하나의 작은 워크플로우라는 명확한 완료 기준

무료 파일럿이 성공하면 두 가지가 동시에 일어납니다. 자신에게는 "나도 실제 문제를 풀 수 있다"는 증거가 생깁니다. 클라이언트에게는 "이 사람이 만든 것이 실제로 작동한다"는 증거가 생깁니다. 임포스터 신드롬은 경험 앞에서 힘을 잃습니다. 실제 결과라는 데이터가 쌓이면, "내가 자격이 있나?"라는 질문은 "다음에는 어떤 문제를 풀까?"로 바뀝니다.

여기서 중요한 것은 과잉 제공(Over-deliver)의 자세입니다. 무료라고 해서 대충 만들면 오히려 역효과입니다. 이 파일럿은 실무자의 명함입니다. 작지만 깔끔하게, 제대로 작동하는 것을 보여 주어야 합니다. 클라이언트가 "무료치고 괜찮네"가 아니라 "이걸 무료로 해 줬다고?"라고 반응하는 것이 목표입니다.

〔그림 32-3〕 무료 파일럿의 범위 설정과 기대 결과 매트릭스〕

거절을 수정 의견으로 전환하는 사고방식

열 통의 메시지를 보냈습니다. 세 명은 답장이 없습니다. 두 명은 "지금은 괜찮습니다"라고 했습니다. 한 명은 읽기만 했습니다. 네 명과는 대화가 시작되었습니다. 이것이 현실적인 초기 반응률입니다. 여기서 질문이 갈립니다. 답장 없는 여섯 건에 좌절할 것인가, 아니면 그 여섯 건에서 정보를 추출할 것인가.

거절을 수정 의견으로 바꾸는 첫 번째 단계는 질문을 바꾸는 것입니다. "왜 나를 거절했을까?"가 아니라 "내 메시지에서 무엇이 부족했을까?"로 전환합니다. 전자는 자존감의 문제가 됩니다. 후자는 기

술의 문제가 됩니다. 기술은 개선할 수 있습니다.

구체적으로 점검할 항목이 있습니다.

메시지가 명확했는가? 받는 사람이 10초 안에 핵심을 파악할 수 있었는가?

메시지가 너무 길지 않았는가? 바쁜 사업주가 읽을 분량이었는가?

상대방이 관심을 가질 이유가 메시지에 담겨 있었는가?

구체적인 문제를 언급했는가, 아니면 추상적인 가능성만 이야기했는가?

이 점검을 거치면 다음 메시지는 반드시 달라집니다. 미국의 한 AI 자동화 커뮤니티에서 관찰된 패턴이 있습니다. 성과를 내는 사람과 그렇지 못한 사람의 차이는 재능이나 기술 수준이 아니었습니다. 차이는 거절 이후의 행동에 있었습니다. 성과를 내는 사람은 거절당한 뒤 무엇인가를 고쳤습니다. 메시지의 길이를 줄이거나, 대상을 바꾸거나, 제안의 구체성을 높였습니다.

성과를 내지 못하는 사람은 같은 메시지를 같은 방식으로 반복해서 보냈습니다.

거절은 데이터입니다. 데이터는 축적될수록 가치가 올라갑니다.

다섯 번 거절당하면 다섯 가지 개선 포인트가 생깁니다. 열 번 거절당하면 메시지의 톤, 길이, 대상, 제안 구조에 대한 패턴이 보이기 시작합니다. 이 패턴을 인식하는 순간, 거절은 더 이상 실패의 증거가 아니라 성공을 향한 반복 실험(Iteration)의 일부가 됩니다.

한 가지 실용적인 방법이 있습니다. 거절 로그를 만드는 것입니다. 간단한 스프레드시트에 날짜, 대상, 보낸 메시지 요약, 반응, 추정되는 이유를 기록합니다. 열 건쯤 쌓이면 패턴이 드러납니다. "이 유형의 메시지에는 답장이 오지 않는다", "이 업종에서는 반응이 좋다", "구체적인 문제를 언급했을 때 응답률이 올라간다" 같은 인사이트가 나옵니다.

〔그림 32-4〕 거절 로그 스프레드시트 양식 예시〕

세 가지 심리적 허들은 사라지지 않습니다. 그러나 행동 패턴을 바꾸면 허들의 높이가 달라집니다. 임포스터 신드롬은 경험이 쌓이면서 낮아지고, 가격 결정은 가치를 입증한 뒤에 쉬워지며, 거절은 수정 의견 시스템의 일부로 편입됩니다.

이 심리적 기반 위에서 구체적인 행동 계획이 비로소 의미를 갖습니다. 내면의 준비가 끝났다면, 이제 밖으로 나갈 차례입니다. 7일이라는 시간 안에 첫 클라이언트를 확보하는 프레임워크가 그 출발점이 됩니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

33장 7일 프레임워크

전체 목차

책의 모든 장 보기

Day 1: 느슨한 방향 설정과 신뢰 지도 작성

월요일 아침, 노트북을 열었습니다. 브라우저에는 링크드인, 이메일, 메모장이 나란히 열려 있습니다. "오늘부터 AI 자동화 사업을 시작한다." 그런데 무엇부터 시작해야 하는지가 문제입니다.

대부분의 사람이 이 지점에서 범하는 실수가 있습니다. 초(超)세밀한 니치(Niche)를 먼저 정하려는 것입니다. "조지아주에 있는 치과 클리닉 중 환자 온보딩 프로세스에 시간을 낭비하고 있는 곳을 타깃으로 하겠다"처럼 구체적으로 시작하면 오히려 발이 묶입니다. 아직 어떤 시장이 반응하는지, 어떤 문제가 자주 등장하는지 데이터가 없기 때문입니다.

첫날의 방향 설정은 작업 가설(Working Hypothesis)에 가깝습니다. 평생의 결정이 아닙니다. 이렇게 충분합니다. "저는 중소기업이 반복적이고 지루한 업무를 AI로 자동화하도록 돕습니다." 이 한 문장이면 대화를 시작할 수 있습니다.

이 느슨한 방향 위에 몇 가지 예시 문제를 준비합니다. 리드 후속 연락(Lead Follow-up) 자동화, 고객 접수 양식 처리, CRM 간 데이터 동기화 같은 것들입니다. 아직 이것들에 전념할 필요는 없습니다. 대화에서 어떤 문제가 공명을 일으키는지 관찰하기 위한 시험 카드에 불과합니다.

첫날의 두 번째 과제는 신뢰 지도(Trust Map)를 그리는 것입니다. 구글 시트를 하나 엽니다. 행에는 사람 이름, 열에는 다섯 가지 정보를 채웁니다.

이름

업종/역할

친밀도

잠재 클라이언트 가능성

소개 가능성

인사이트 제공 가능성

김OO

부동산 중개업

높음

중간

높음

이OO

마케팅 팀장

중간

높음

중간

높음

〔그림 33-1〕 신뢰 지도 스프레드시트 예시〕

목표는 20명을 채우는 것입니다. 사업을 운영하는 친구나 가족, 직장 동료나 상사, 커뮤니티나 슬랙 그룹에서 아는 사람, 그리고 2촌 관계—친구의 친구—까지 포함합니다. 이 목록을 실제로 적어 보면 놀라운 일이 일어납니다. "나는 아는 사람이 없다"고 생각했던 것이 착각이었음을 깨닫게 됩니다. 대부분의 사람은 자신의 네트워크를 실제보다 작게 인식합니다.

이 지도가 중요한 이유는 시작 지점이 따뜻한 관계(Warm Outreach)여야 하기 때문입니다. 통계적으로 따뜻한 접촉과 소개를 통한 연락은 차가운 접촉(Cold Outreach)보다 전환율이 월등히 높습니다. 이미 어느 정도 신뢰가 존재하고, 그것이 친구의 친구로부터 빌려온 신뢰라 해도 아무 관계 없는 이메일보다는 강력합니다. 속도도 빠릅니다. 어색함도 덜합니다.

큰 그림은 이렇습니다. 따뜻한 관계에서 시작하고, 넓은 범위를 유지하고, 대화하며 패턴을 발견하고, 나중에 니치를 좁힙니다. 차가운 접촉은 확장(Scale) 준비가 되었을 때 사용하는 것이지, 첫 클라이언트를 잡기 위한 도구가 아닙니다.

Day 2에서 3: 5에서 10회 워밍업 대화

둘째 날과 셋째 날의 목표는 판매가 아닙니다. 대화입니다. 5회에서 10회 사이의 가벼운 대화를 나누는 것이 전부입니다.

이 대화의 성격을 명확히 구분해야 합니다. 영업 전화가 아닙니다. 피칭(Pitching)도 아닙니다. 호기심 많은 사업가가 질문하는 자리입니다. 다음과 같은 메시지로 시작할 수 있습니다.

"안녕하세요, 저는 기업의 반복 업무를 AI로 자동화하는 사업을 시작하려고 합니다. 무언가를 팔려는 것이 아닙니다. 일상 업무에서 수동적이고 번거로운 부분이 어디인지 몇 가지 여쭤봐도 될까요?"

이 메시지에서 빠져서는 안 되는 문구가 있습니다. "무언가를 팔려는 것이 아닙니다." 이 한 문장이 상대방의 방어벽을 내립니다. 사람은 누군가가 무언가를 팔려 한다고 느끼는 순간 경계합니다. 그 경계를 먼저 해제하는 것이 대화의 시작입니다.

대화가 시작되면 핵심 인사이트를 기록합니다. 메모 앱이든, 노트 앱이든, 종이 노트든 상관없습니다. 중요한 것은 기록하는 행위 자체입니다. 나중에 이 기록을 다시 볼 것이기 때문입니다. 기록할 항목은 다음과 같습니다.

그들이 반복적이라고 표현한 업무는 무엇인가?

그 업무에 얼마나 많은 시간을 쓰고 있는가?

그 문제를 설명할 때 어떤 단어를 사용했는가?

이미 시도해 본 해결책이 있는가?

네 번째 항목이 의외로 중요합니다. 상대방이 문제를 묘사할 때 쓰는 표현은 나중에 자신의 마케팅 언어가 됩니다. 기술자의 용어가 아니라 사업주의 용어로 말해야 공감이 일어납니다.

만약 정말로 네트워크에 사업주가 한 명도 없다면 어떻게 해야 하는지 살펴보겠습니다. 그래도 차가운 접촉으로 뛰어들 필요는 없습니다. 대신 이렇게 물어봅니다.

"혹시 주변에 이런 것이 도움이 될 만한 분이 계실까요?"

이 한 문장의 위력은 생각보다 큽니다. 친구에게 무언가를 팔려는 것이 아니라, 누군가를 소개해 달라고 부탁하는 것이기 때문에 어색함이 줄어듭니다. 그리고 소개를 통해 만나는 사람에게는 "OO님이 소개해 주셔서 연락드렸습니다"라는 한 마디가 즉각적인 신뢰를 만듭니다.

〔그림 33-2〕 워밍업 대화 인사이트 기록 양식〕

Day 4에서 5: 소규모 파일럿 제안

대화 기록을 펼쳐 놓습니다. 다섯 건에서 열 건의 대화 내용이 정리되어 있습니다. 이 가운데 가장 선명한 고통점(Pain Point)을 가진 사람을 고릅니다. 빈도가 높고, 시간을 많이 잡아먹고, 반복적인 업무. 이 세 가지 조건이 겹치는 문제가 이상적인 파일럿 대상입니다.

선택이 끝나면 연락합니다.

"안녕하세요, 지난번 대화에서 말씀하신 〔구체적 문제〕를 해결하는 작은 자동화를 무료 파일럿으로 만들어 드리고 싶습니다. 제 목표는 이 워크플로우가 실제로 시간을 절약해 준다는 것을 증명하는 것입니다. 대신 솔직한 수정 의견을 부탁드립니다. 이것이 저에게는 배움이 됩니다."

이 제안이 효과적인 이유는 위험 부담이 거의 없기 때문입니다. 클라이언트 입장에서는 비용이 들지 않습니다. 실패하더라도 잃을 것이 없습니다. 성공하면 시간을 절약할 수 있습니다. 그리고 수정 의견이라는 대가가 있어서 자선 행위가 아닌 동등한 교환으로 느껴집니다.

제안할 때 반드시 지켜야 할 원칙이 있습니다. 범위를 극도로 작게 잡는 것입니다. "전체 영업 프로세스를 자동화하겠습니다"가 아니라 "리드가 접수되면 자동으로 분류하고 담당자에게 알림을 보내는 워크플로우를 만들겠습니다"처럼 한 가지 동작에 집중합니다. 작게 시작해야 완성할 수 있고, 완성해야 증명할 수 있습니다.

Day 5에서 6: 최소 기능 자동화 구축

파일럿 제안이 받아들여졌습니다. 이제 만들 차례입니다.

이 단계에서 빠지기 쉬운 함정이 있습니다. 기술적으로 인상적인 것을 만들려는 충동입니다. API 호출을 여러 겹으로 쌓고, 복잡한 분기 로직을 넣고, 최신 모델을 적용하고 싶은 마음이 들 수 있습니다. 그러나 목표는 인상이 아니라 결과입니다.

최소 기능 자동화, 즉 MVP(Minimum Viable Product)의 기준은 하나입니다. "이 워크플로우가 돌아간 뒤에 클라이언트가 시간을 절약했는가?" 이 질문에 "예"라고 답할 수 있으면 충분합니다.

구축하면서 주의를 기울여야 할 것이 있습니다. 클라이언트가 문제를 설명할 때 사용한 표현에 귀를 기울이는 것입니다. 그들이 "리드 관리"라 하지 않고 "매일 쏟아지는 문의 정리"라고 했다면, 컨설턴트도 보고할 때 같은 표현을 사용하는 것이 좋습니다. 이 언어 맞춤은 이후 제안서나 영업 메시지에서 강력한 무기가 됩니다.

〔그림 33-3〕 MVP 워크플로우 구축의 체크리스트〕

구축 과정은 복잡할 필요가 없습니다.

1. 문제를 한 문장으로 정의합니다.

2. 입력과 출력을 명확히 합니다.

3. 입력에서 출력으로 가는 최단 경로를 설계합니다.

4. 만들고 검증합니다.

5. 클라이언트에게 결과를 보여 줍니다.

이것이 전부입니다. 복잡하게 만드는 것은 나중에 확장 단계에서 할 일입니다.

Day 7: 유지보수와 확장 제안

일곱째 날이 왔습니다. 파일럿은 작동하고 있습니다. 이제 강하게 밀어붙일 시간일까요? 아닙니다.

이 날의 목표는 클로징(Closing)이 아닙니다. 파일럿 결과를 바탕으로 다음 단계가 무엇인지 함께 결정하는 것입니다. 강압적인 분위기는 지금까지 쌓은 신뢰를 한 순간에 무너뜨릴 수 있습니다.

대화의 구조는 간결합니다. 두 가지 선택지를 제시합니다.

선택지 1: 유지보수.

이미 만든 자동화가 계속 작동하도록 관리합니다. 어딘가가 고장 나면 수정하고, 도구나 프로세스가 바뀌면 맞춤 조정합니다. "앞으로 몇 달간 이 자동화가 계속 잘 돌아가도록 관리하겠습니다. 무언가 고장 나거나 변경이 필요하면 바로 대응하겠습니다."

선택지 2: 확장.

파일럿 위에 한두 가지 기능을 추가합니다. 구축하는 동안 발견한 개선 가능성을 활용하는 것입니다. "파일럿을 만들면서 이 부분을 추가하면 결과가 더 안정적일 것 같다는 아이디어가 있었습니다. 범위를 정해서 제안서를 보내 드릴까요?"

이 두 가지를 압박 없이 제시하면, 클라이언트는 자연스럽게 자신에게 맞는 방향을 선택합니다.

여기서 한 가지 고급 기술이 있습니다. 구축 과정에서 관련된 다른 업무를 발견했다면, 자연스럽게 질문합니다. "워크플로우를 만들면서 이 프로세스도 눈에 들어왔는데, 이 부분도 자동화하면 도움이 될까요?" 이 질문은 판매가 아니라 협업의 언어입니다. 상대방은 컨설턴트가 자신의 사업에 관심을 갖고 있다고 느끼게 됩니다.

파일럿이 잘되었지만 클라이언트가 지금 당장 추가 작업을 원하지 않는 경우도 있습니다. 그것은 전혀 문제가 아닙니다. 이때의 전략은 다음 절에서 설명합니다.

테스티모니얼과 소개 요청의 순서

순서가 있습니다. 이 순서를 어기면 신뢰가 깨집니다.

유지보수 또는 확장 논의가 먼저입니다. 관계를 심화하는 것이 우선입니다. 그다음이 테스티모니얼(Testimonial)입니다. 그리고 마지막이 소개 요청(Referral)입니다.

클라이언트가 파일럿에 만족했지만 추가 작업을 원하지 않는다면, 이렇게 요청합니다.

"워크플로우가 어떻게 도움이 되었는지 짧은 영상으로 말씀해 주실 수 있을까요? 1에서 2분이면 충분합니다."

영상 추천사는 텍스트보다 강력합니다. 실제 사업주가 카메라 앞에서 직접 경험을 이야기하면, 그 어떤 마케팅 카피보다 설득력이 있습니다.

추천사를 받은 뒤에, 그리고 오직 자연스러운 흐름일 때만, 소개를 언급합니다.

"혹시 비슷한 어려움을 겪고 있는 분이 주변에 계시다면, 소개해 주시면 감사하겠습니다."

이 순서가 왜 중요한지 생각해 보겠습니다. 아직 아무것도 해 주지 않은 상태에서 소개를 부탁하면 어색합니다. 무언가를 해 줬지만 결과가 좋지 않았는데 추천사를 요청하면 더 어색합니다. 순서는 논리를 따릅니다. 가치 전달 → 관계 심화 → 증거 확보 → 네트워크 확장.

파일럿이 실패했다면 어떻게 해야 하는지 살펴보겠습니다. 무언가를 팔려고 하지 않습니다. 추천사를 요청하지 않습니다. 수정 의견을 받고, 배운 것을 정리하고, 다음 사이클을 시작합니다.

〔그림 33-4〕 테스티모니얼→소개 요청의 흐름도〕

7일 사이클 반복 → 콜드 아웃리치 확장

7일 프레임워크는 일회성 이벤트가 아닙니다. 반복하는 루프(Loop)입니다.

첫 번째 사이클에서는 메시지가 서투릅니다. 어떤 문제에 어떤 해결책이 맞는지 감이 없습니다. 두 번째 사이클에서는 메시지가 짧아지고 구체적이 됩니다. 어떤 대화가 에너지를 얻고 어떤 대화가 방향을 잃는지 느끼기 시작합니다. 세 번째 사이클쯤 되면 자신만의 언어가 생깁니다. 고객의 표현을 빌려 쓰는 법을 알게 되고, 제안의 톤이 자연스러워집니다.

각 사이클이 제공하는 것은 데이터입니다. 자신감도 올라가지만, 진짜 가치는 데이터에 있습니다. 어떤 업종이 반응하는지, 어떤 문제가 자주 등장하는지, 어떤 메시지 구조가 대화를 여는지에 대한 패턴이 축적됩니다.

이 패턴이 충분히 쌓였을 때—보통 두세 번의 사이클을 돈 뒤—비로소 콜드 아웃리치로 확장하는 것이 의미를 갖습니다. 왜냐하면 이 시점에서 이 시점에서 세 가지가 생겼기 때문입니다.

1.

증거

: 실제로 결과를 낸 파일럿 또는 프로젝트

2.

언어

: 시장이 반응하는 메시지 구조

3.

자신감

: 실전에서 검증된 역량

이 세 가지 없이 콜드 아웃리치를 하면, 아무 맥락 없는 이메일을 하루에 수백 통 보내는 사람과 다를 바가 없습니다. 증거도 없고, 신뢰도 없고, 실질적 연결고리도 없는 상태에서 보내는 메시지는 수신자에게 잡음(Noise)입니다.

〔그림 33-5〕 7일 사이클 반복과 콜드 아웃리치 확장 로드맵〕

미국의 한 AI 자동화 커뮤니티에서 첫 클라이언트를 확보한 실무자가 따른 경로가 정확히 이것이었습니다. 따뜻한 관계에서 시작하고, 성과 중심으로 접근하고, 신뢰를 얻고, 가치를 전달하고, 반복했습니다. 그리고 두세 번의 사이클을 돈 뒤에야 잠재 고객 목록을 구축하고 같은 프레임워크를 차가운 접촉에 적용했습니다. 그때쯤에는 추측과 불안으로 가득 찬 행동이 아니라, 작동하는 시스템을 운영하는 느낌이었습니다.

첫 클라이언트를 확보하는 것은 전체 사업의 시작일 뿐입니다. 이 관계를 지속하고 확장하려면 가격이라는 민감한 주제를 다루는 법을 알아야 합니다. 시간 단위로 청구할 것인가, 가치 단위로 청구할 것인가—이 선택이 사업의 궤도를 결정합니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

34장 가치 기반 가격 책정

전체 목차

책의 모든 장 보기

시간 기반 가격의 함정

견적서를 써야 할 때가 왔습니다. 빈 문서에 커서가 깜빡입니다. 머릿속으로 계산이 시작됩니다. "이 워크플로우를 만드는 데 대략 열다섯 시간이 걸리겠지. 시간당 5만 원이면... 75만 원?" 이렇게 가격을 정하면 두 가지 문제가 동시에 발생합니다.

첫 번째 문제는 실력이 올라갈수록 불리해진다는 것입니다. 처음에는 열다섯 시간이 걸리던 워크플로우가 경험이 쌓이면 다섯 시간이면 끝납니다. 같은 품질, 같은 결과물인데 청구할 수 있는 금액은 3분의 1로 줄어듭니다. 실력 향상이 수입 감소로 이어지는 구조입니다. 이것은 분명히 잘못된 인센티브입니다.

두 번째 문제는 클라이언트와의 관계가 감시 구조로 변한다는 것입니다. 시간 기반 청구에서 클라이언트는 자연스럽게 "정말 그만큼 걸렸나?"라는 의문을 품게 됩니다. 프리랜서처럼 보이기 시작합니다. 파트너가 아니라 시간을 파는 사람으로 포지셔닝되는 것입니다.

기업은 컨설턴트의 시간을 사는 것이 아닙니다. 기업이 구매하는 것은 결과입니다. AI 워크플로우를 구축할 때 컨설턴트가 기업에 전달하는 것은 보통 세 가지 중 하나입니다. 비용 절감, 시간 절약, 인적 오류 감소. 이것이 진짜 가치이고, 가격은 이 가치에 연결되어야 합니다.

이것을 가치 기반 가격 책정(Value-Based Pricing)이라고 부릅니다. 투입이 아니라 산출에 가격표를 붙이는 방식입니다.

가치 기반 3축: 비용 절감, 시간 절약, 오류 감소

가치 기반 가격 책정의 근거는 세 개의 축 위에 서 있습니다.

비용 절감

가장 직관적인 축입니다. 워크플로우가 도입되기 전에 해당 업무에 얼마가 들었는지 계산합니다. 워크플로우 도입 후에 얼마로 줄어드는지 비교합니다. 그 차이가 절감 금액입니다.

시간 절약

직원이 반복 업무에 소비하는 시간을 화폐 가치로 환산합니다. 한 직원이 하루 한 시간을 특정 업무에 쓰고, 그 직원의 시급이 5만 원이라면, 연간 약 1,200만 원에 해당하는 시간을 그 업무에 투입하고 있는 셈입니다. 워크플로우가 이 시간을 회수해 줍니다.

오류 감소

정량화하기 가장 어렵지만 영향력은 큰 축입니다. 수동 데이터 입력에서 발생하는 오류, 잘못된 태깅, 누락된 후속 조치—이런 것들은 비용으로 드러나기까지 시간이 걸립니다. 그러나 한 번 드러나면 피

해가 큽니다. 잘못된 고객에게 잘못된 메시지를 보내거나, 중요한 리드를 놓치거나, 데이터가 꼬여서 전체 보고서를 다시 만들어야 하는 상황을 생각해 보겠습니다.

〔그림 34-1〕 가치 기반 가격 책정의 3축 다이어그램〕

이 세 축에서 한 가지 중요한 특성이 있습니다. 같은 워크플로우라도 기업마다 가치가 다르다는 것입니다. 사막에서 5km를 뛴 뒤 마시는 물 한 병과 에어컨이 켜진 사무실에서 마시는 물 한 병은 같은 제품이지만 가치가 다릅니다.

리드 분류 자동화를 주 5건의 리드를 받는 스타트업에 제안하는 것과 주 200건을 받는 중견 기업에 제안하는 것은 같은 기술이지만 가치가 완전히 다릅니다. 가격 책정은 이 차이를 반영해야 합니다.

ROI 계산법

가치를 숫자로 바꾸는 구체적인 방법을 알아보겠습니다. 핵심은 디스커버리(Discovery) 단계에서 확보한 정보입니다. 가격을 정하기 전에 현재의 수동 프로세스를 처음부터 끝까지 파악해야 합니다.

파악해야 할 항목은 다음과 같습니다.

이 프로세스는 얼마나 자주 발생하는가? (매일, 매주, 월 단위)

무엇이 이 프로세스를 촉발하는가?

누가 관여하는가? 핵심 이해관계자는 누구인가?

한 번 수행하는 데 얼마나 걸리는가?

관여하는 직원의 시간당 가치는 얼마인가? (급여, 인원 수, 소프트웨어 비용 등)

이 정보를 확보한 뒤에 수동 버전과 자동화된 버전을 나란히 비교합니다.

실전 계산 사례

미국의 한 AI 자동화 컨설턴트가 실제로 작업한 사례입니다. 클라이언트의 인바운드 영업 프로세스를 자동화하는 프로젝트였습니다.

현재 상황:

웹사이트 폼을 통해 주당 20건의 리드 유입

각 리드를 연락·적격 판정·양성·미팅 예약하는 데 직원 1시간 소요

해당 직원의 시급: 약 5만 원(40달러 기준)

비용 계산:

주당 비용: 20건 × 5만 원 = 100만 원

월간 비용: 약 400만 원

연간 비용: 약 4,800만 원

이 프로세스에 연간 4,800만 원의 인건비가 투입되고 있었습니다. 워크플로우가 이 업무를 자동화하면, 4,800만 원의 절감 효과가 발생합니다.

여기서 경험 법칙(Rule of Thumb)이 등장합니다. 클라이언트가 첫해에 투자 대비 10배의 수익(10x ROI)을 볼 수 있도록 가격을 설정하는 것입니다. 4,800만 원의 10%인 480만 원이 이 워크플로우의 출발 가격이 됩니다.

이 계산을 클라이언트 앞에서 그대로 걸어가면 됩니다. "이 프로세스에 연간 이만큼의 비용이 들고 있습니다. 자동화하면 이만큼을 절약하게 됩니다. 투자 금액은 연간 절감액의 10% 수준입니다." 이 설명은 30초면 끝나지만, 그 30초가 신뢰를 결정합니다.

〔그림 34-2〕 ROI 계산 워크시트 예시〕

여기에 기회비용(Opportunity Cost)을 더하면 가치가 더 커집니다. 직원의 하루 한 시간이 해방된다는 것은 비용을 절감하는 것만이 아닙니다. 그 직원이 더 높은 가치의 업무—고가치 고객 대응, 신규 채용 지원, 시스템 개선—로 시간을 전환할 수 있다는 뜻입니다. ROI는 시간이 지남에 따라 복리로 증가하는 경향이 있습니다.

첫 달에 100만 원 절감이 둘째 달에는 150만 원, 셋째 달에는 200만 원이 될 수 있습니다. 가치가 커지면 가격 결정력도 커집니다.

다양한 가격 모델 실험

미국의 한 AI 자동화 컨설턴트는 거의 모든 가격 모델을 시도해 봤다고 고백합니다. 고정 비용으로 JSON 파일을 납품한 적도 있고, 시간당 청구를 한 적도 있습니다. 시간 묶음을 판매하기도 했고, 범위가 정해지지 않은 월정액 리테이너를 운영하기도 했습니다. 범위가 엄격하게 정해진 리테이너도 있었습니다.

이 다양한 실험 끝에 도달한 결론은 두 가지 모델의 조합이 가장 잘 작동한다는 것입니다.

모델 1: 가치 기반 프로젝트 가격

시작점으로 추천되는 모델입니다. 앞서 설명한 ROI 계산에 근거하여 프로젝트 단위로 가격을 정합니다. 발을 들여놓기에 적합합니다. 계산이 명쾌하고 신뢰를 구축합니다.

모델 2: 월정액 리테이너

한두 건의 프로젝트를 납품하고 신뢰가 쌓이면, 장기 계약으로 전환합니다. 리테이너는 클라이언트에게 예측 가능한 비용과 우선 접근권을, 컨설턴트에게는 안정적 수입과 장기 파트너십을 제공합니다.

리테이너의 구조에는 여러 방식이 있습니다. 시간 기반, 산출물 기반, 혼합형. 하지만 권장되는 것은 마일스톤(Milestone) 기반 리테이너입니다. 시간 기반은 프리랜서처럼 보이게 만들고, 마일스톤 기반은 컨설턴트이자 AI 파트너로 포지셔닝합니다.

리테이너 단계에서 가치 기반 가격 책정을 유지하는 방법은 관점의 전환에 있습니다. 개별 프로젝트에서 최대 이익을 뽑는 것이 아니라, 신뢰를 쌓고 성과를 축적하면서 리테이너 금액을 점진적으로 올리는 것입니다. 마치 최고 AI 책임자(Chief AI Officer) 역할로 조직에 깊이 들어가는 것과 비슷합니다.

〔그림 34-3〕 프로젝트 가격 → 리테이너 전환 흐름도〕

리테이너 가격 산정에는 마진(Margin) 보호가 중요합니다. 이 클라이언트를 서비스하는 데 월간 얼마의 비용이 드는지 계산합니다. 자신의 시간, 개발자가 필요한 경우 그 비용, 시스템 유지 비용 등을 포함합니다. 서비스업에서 목표 마진은 50%에서 70% 사이입니다. 50%가 기본선이고, 70%면 확장 여력이 충분합니다.

예를 들어 월 비용이 500만 원이라면 리테이너는 1,000만 원 수준을 목표로 합니다. 50% 마진입니다. 이 여유가 인력을 충원하거나 범위를 확장할 공간을 만들어 줍니다.

반복 수익 모델: 유지보수, 최적화, 확장

리테이너가 아니더라도 반복 수익을 만들 수 있는 소규모 서비스가 있습니다.

유지보수 비용:

API 변경, 모델 업데이트, 버그 수정 등을 다루는 월정액. 시스템당 월 20만 원에서 150만 원 수준입니다.

최적화 및 모니터링:

주간 또는 월간으로 시스템 로그를 검토하고, 프롬프트를 조율하고, 품질을 점검하는 서비스입니다. AI 중심 시스템에서는 새로운 모델 출시나 데이터 변화에 따른 지속적 개선이 가치를 만듭니다.

확장 프로젝트:

거의 모든 프로젝트에서 클라이언트와 함께 새로운 아이디어, 버전 2 업그레이드, 기능 백로그를 발견하게 됩니다. 이것이 일회성 프로젝트를 장기 파트너십으로 전환하는 자연스러운 경로입니다.

이 서비스들의 가격 기준은 세 가지가 있습니다. 정액 월정액, 시간 묶음(월 5에서 20시간), 또는 원래 프로젝트 비용의 10에서 25%입니다.

업계 표준 부재가 기회인 이유

AI 자동화 서비스의 가격에 관해 한 가지 솔직한 사실이 있습니다. 업계 표준이 아직 없습니다. 아무도 "이것이 올바른 가격 책정 방법"이라고 합의하지 않았습니다. 모두가 아직 실험 중입니다.

이것은 위협이 아니라 기회입니다. 표준이 없다는 것은 스스로 기준을 세울 수 있다는 뜻이기도 합니다. 클라이언트 역시 이 서비스에 얼마를 지불해야 하는지 모릅니다. 그래서 가격을 제시할 때 핵심은 단순히 숫자를 말하는 것이 아니라, 그 숫자에 도달한 과정을 보여 주는 것입니다.

"이 금액이 어떻게 나왔는지 설명해 주실 수 있나요?" 이 질문에 자신 있게 답할 수 있어야 합니다. ROI 계산을 걸어가며 보여 주면, 가격은 비용이 아니라 투자처럼 느껴지기 시작합니다. 기업은 자체

적으로 상환되는 투자에 기꺼이 지갑을 엽니다.

가격 제시 전에 반드시 해야 할 것이 있습니다. 변환(Transformation)을 먼저 그려 주는 것입니다. 숫자를 말하기 전에 클라이언트가 "이 시스템이 가동되면 우리 팀의 하루가 어떻게 바뀔까?"를 상상하게 만들어야 합니다. 어떤 문제가 사라지고, 어떤 것이 쉬워지는지. 그 그림이 머릿속에 그려진 뒤에 가격을 말하면 맥락이 완전히 달라집니다.

〔그림 34-4〕 가격 제시 전 변환 설명 → 가격 제시 → ROI 설명의 순서 도식〕

제안서에 포함되어야 할 항목도 명확합니다. 설정, 호스팅, 검증 및 품질 검수, 최적화, 클라이언트의 참여 범위, 문서화, 교육, 그리고 해당되는 경우 유지보수. 이 항목들이 명시되면 클라이언트는 가격이 어디에서 오는지 이해하고, 단순 지출이 아니라 투명한 투자로 받아들입니다.

마지막으로, 가격 협상에서 지켜야 할 원칙이 있습니다. 가격을 깎지 말고 범위를 조정하는 것입니다. 클라이언트가 예산이 부족하다면 가격을 할인하는 대신 기능을 줄이거나, 단계를 나누거나, 복잡도를 낮춥니다. 가치에 대한 할인은 자신의 전문성에 대한 할인입니다. 범위 조정은 비즈니스 판단입니다.

가격이 정해졌고 프로젝트가 시작되었습니다. 그런데 구축한 워크플로우를 클라이언트에게 넘기는 과정은 생각보다 정교한 작업입니다. 검증, 품질 보증, 인수인계—이 단계들이 프로젝트의 성패와 장기 관계의 성격을 결정합니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

35장 워크플로우 검증, QA, 핸드오버

전체 목차

책의 모든 장 보기

배포 전 품질 보증의 원칙

워크플로우가 완성되었습니다. 모든 노드가 연결되어 있고, 검증 데이터 한두 건을 넣어 보니 결과가 나옵니다. "잘 되네!" 이 순간의 안도감은 위험합니다. 검증 한두 건으로 검증된 워크플로우를 프로덕션(Production)에 올리는 것은 새 차를 주차장에서만 몰아 보고 고속도로에 나가는 것과 비슷합니다.

배포 전 품질 보증, 즉 QA(Quality Assurance)는 워크플로우가 실전에서 망가지지 않도록 보호하는 과정입니다. 이 과정을 생략하면 컨설턴트의 평판이 직접적으로 타격을 받습니다. 버그가 있는 시스템을 넘기면 클라이언트의 신뢰가 무너지고, 한 번 무너진 신뢰를 복구하는 데는 처음 쌓는 것보다 몇 배의 노력이 듭니다.

QA의 시작은 클라이언트와 함께 검증 데이터를 계획하는 것입니다. 임의로 만든 가짜 데이터가 아니라, 실제 사용과 유사한 샘플 데이터를 요청합니다. 이메일, 고객 문의 티켓, CRM 레코드, 통화 기록—워크플로우가 프로덕션에서 처리할 데이터와 같은 형태의 자료입니다. 개인정보 보호가 필요하면 익명 처리를 해도 됩니다.

데이터를 확보하면 성공 기준을 합의합니다. 좋은 출력이 무엇이고, 절대 발생하면 안 되는 것이 무엇인지를 정합니다.

"실제 데이터로 시스템을 돌려 보겠습니다. 그 과정에서 시스템이 정확히 어떻게 동작하는지 확인하실 수 있습니다."

이 한 마디가 클라이언트에게 전문성의 인상을 줍니다. 대부분의 사람은 "다 됐습니다, 써 보세요"라고만 말합니다. QA 과정을 먼저 안내하는 것은 차별화입니다.

스트레스 검증: 실패를 설계하는 사고방식

개발자처럼 각 노드를 하나씩 점검하는 것에서 벗어나야 합니다. 엔지니어처럼 실패를 계획하는 사고방식으로 전환합니다.

자동화에는—AI가 포함되어 있으면 더욱—예측 불가능한 상황이 반드시 발생합니다. 프로덕션에 들어가면 실제 사용자와 실제 데이터가 상상하지 못한 엣지 케이스(Edge Case)를 만들어 냅니다. 검증 단계에서 해야 할 일은 스스로에게 이렇게 질문하는 것입니다.

"이 필드에 아무 데이터도 들어오지 않으면?"

"중복된 데이터가 들어오면?"

"예상과 전혀 다른 형식의 입력이 들어오면?"

"API 응답이 지연되거나 실패하면?"

"사용자가 완전히 예상 밖의 행동을 하면?"

모든 가능한 문제를 제거하는 것은 불가능합니다. 목표는 다릅니다. 문제가 발생했을 때 조용하고 안전하게 실패하도록 만드는 것입니다. 타임아웃(Timeout)을 설정하여 무한 대기를 방지하고, 오류 발생 시 담당자에게 알림을 보내는 에러 핸들링(Error Handling)을 넣고, 실패 기록을 스프레드시트에 자동으로 쌓아 패턴을 추적할 수 있게 합니다.

〔그림 35-1〕 스트레스 검증 시나리오 매트릭스〕

이 기반 위에서 블랙박스 검증(Black Box Testing)를 수행합니다. 워크플로우를 하나의 상자로 보고, 가능한 한 많은 샘플 입력—한두 건이 아니라 수십, 가능하다면 수백 건—을 넣습니다. 각 입력에 대해 세 가지를 기록합니다. 무엇이 들어갔는가, 중간에 무슨 일이 일어났는가, 최종 출력은 무엇인가. 그리고 이 출력을 클라이언트와 합의한 성공 기준과 비교합니다.

실패한 건, 이상한 결과가 나온 건, 경계선에 걸린 건을 표시합니다. 이것이 내부 QA 패스(Internal QA Pass)입니다. 클라이언트가 시스템을 만지기 전에 가능한 한 많은 문제를 잡는 것이 목표입니다. 내부 QA는 최소 며칠간 진행하는 것이 좋습니다.

AI 품질 검수의 추가 레이어

워크플로우에 AI가 포함되어 있다면 추가 검수 항목이 있습니다. AI 노드가 실행되는 것 자체가 아니라 출력의 품질을 점검해야 합니다.

관련성과 정확성:

AI의 응답이 실제 요청에 대한 답변인가? 정보가 정확한가?

톤과 안전성:

부적절하거나 브랜드에 맞지 않는 표현은 없는가? 시스템 프롬프트나 내부 정보가 노출되지는 않는가?

일관성:

같은 입력 10건을 넣었을 때 대략 같은 품질의 답변이 나오는가?

뒷면에서는 A/B 검증와 평가를 수행할 수 있습니다. 서로 다른 프롬프트와 모델을 같은 데이터셋에 적용하고, 어느 조합이 최상의 결과를 내는지 추적합니다.

"실제 데이터로 여러 프롬프트와 모델을 검증했고, 품질과 일관성이 가장 높은 조합을 선택했습니다. 평가 데이터를 보여 드리겠습니다."

이런 설명은 클라이언트에게 단순한 빌더가 아니라 시스템 엔지니어라는 인식을 심어 줍니다.

로그 기록의 중요성

로그(Log)는 QA의 증거입니다. 워크플로우의 실행 이력을 구글 시트 등에 저장하여 입력, 출력, 도구 호출, 오류, 모델이 세는 글 조각 사용량을 추적합니다. 이 로그를 통해 반복되는 실패 유형, 빈번한 불량 입력, 프롬프트나 모델 선택의 약점을 식별할 수 있습니다.

로그는 클라이언트와 대화할 때 근거 자료가 됩니다. "이것을 검증했고, 이런 결과가 나왔으며, 이 때문에 이런 개선을 했습니다"라고 데이터로 보여 줄 수 있습니다.

〔그림 35-2〕 QA 로그 스프레드시트 구조 예시〕

클라이언트 인계: 프로페셔널한 핸드오버의 기술

내부 QA를 마치면 클라이언트 대면 QA(Client-Facing QA)로 넘어갑니다. 클라이언트에게 시스템을 검증할 수 있는 간결한 인터페이스를 제공합니다. 챗봇이라면 채팅창, 데이터 처리 워크플로우라면 입력 폼 같은 것입니다. 핵심은 내부 구조를 보여 주지 않는 것입니다. 클라이언트가 봐야 하는 것은 결과입니다.

수정 의견을 요청합니다. 출력의 정확도, 톤, 형식에 대한 의견을 받습니다. 이 시점에서 모든 것이 잘 진행되었다면 수정 사항은 프롬프트 조율(Prompt Tuning)이나 모델 미세 조정 수준에 그칠 것입니다. 라운드를 거치며 수정 의견을 반영하고, 다시 내부 QA를 하고, 다시 클라이언트 확인을 받습니다.

마무리 단계에서는 짧은 업데이트 영상을 녹화합니다. 한두 건의 전체 실행을 보여 주면서—입력에서 워크플로우 처리, 최종 출력까지—로그를 가리키며 시스템이 실제 데이터를 어떻게 처리했는지 설명합니다.

QA가 완료되면 인계(Handover) 프로세스에 들어갑니다. 인계의 형태는 두 가지 요소에 따라 달라집니다.

구축 환경:

클라이언트의 환경에서 직접 구축했다면 인계는 상대적으로 간결합니다. 이미 모든 것이 그들의 인프라 안에 있기 때문입니다. 반면 컨설턴트의 환경에서 구축했다면 자격 증명 이전, 연결 재설정 등의 추가 작업이 필요합니다.

프로젝트 종료 여부:

이 프로젝트가 끝인지, 아니면 추가 작업이나 유지보수 계약이 예정되어 있는지에 따라 인계의 완결성이 달라집니다. 함께 계속 일할 예정이라면 검증 인프라를 유지할 수 있습니다. 떠날 예정이라면 인계가 최종적이고 완전해야 합니다.

인계 시 필수 체크리스트

워크플로우 복제:

프로덕션용과 백업/검증용을 분리합니다. 소프트웨어 팀이 개발 환경과 운영 환경을 나누는 것과 같은 원리입니다. 변경이나 업데이트가 필요하면 검증 버전에서 먼저 확인한 뒤 프로덕션에 반영합니다.

백업:

워크플로우를 깃허브(GitHub), 구글 드라이브 등에 내보내어 이전 버전으로 되돌릴 수 있도록 합니다. 자동 백업 프로세스를 구축하면 더 좋습니다.

워크플로우 정리:

각 단계에 명확한 이름을 붙이고, 스티키 노트로 로직을 설명합니다. 나중에 다른 사람이 보더라도 구조를 즉시 파악할 수 있어야 합니다.

민감 정보 제거:

워크플로우 안에 평문으로 남아 있는 API 키나 모델이 세는 글 조각이 없는지 최종 점검합니다.

영상 가이드:

1에서 2분짜리 화면 녹화로 시스템 작동 방식, 설정 방법, 업데이트 시 확인할 사항을 안내합니다.

문서화:

해당 워크플로우가 어떤 논리로 설계되었는지 설명하는 문서를 남깁니다. 컨설턴트가 떠난 뒤 클라이언트 팀의 누군가가, 또는 새로운 개발자가 인수할 때 이 문서가 가이드가 됩니다.

〔그림 35-3〕 핸드오버 체크리스트 전체 도식〕

유지보수 계약으로 장기 관계 전환

인계가 끝나면 프로젝트를 마무리하는 단계입니다. 이 과정에서 두 가지를 해결합니다.

청구 마무리:

처음에 합의한 작업 범위(Scope of Work)를 다시 꺼내 놓고, 각 항목이 완료되었음을 함께 확인합니다. 클라이언트가 프로젝트 완료를 확정하면 최종 청구서를 보냅니다. "합의한 워크플로우가 구축, 검증, 문서화, 인수 완료되었습니다. 최종 청구서를 발행합니다."

유지보수 계약 논의:

리테이너는 프로젝트 비용과 별도의 계약입니다. 버그 수정, 소규모 조정, 의존성 변경, 모니터링, 기본 보안 점검을 포함합니다. 새로운 기능, 새로운 워크플로우, 큰 규모의 범위 변경은 포함하지 않습니다. 이것들은 별도 프로젝트입니다.

서비스 수준 기대치(SLA)를 간략히 설정합니다. 긴급 장애는 몇 시간 이내 대응, 일반 요청은 며칠 이내 처리. 복잡할 필요 없지만 기대치는 명확해야 합니다.

소유권과 지식재산권(IP)도 정리합니다. 대부분의 컨설팅 계약에서 납품물은 대금 지급 후 클라이언트 소유가 됩니다. 다만 컨설턴트의 재사용 가능한 패턴, 범용 도구, 기본 템플릿에 대한 권리는 보호해야 합니다.

퇴출 절차(Exit Process)도 정의합니다. 클라이언트가 서비스를 종료하고자 할 때 무엇을 인계하는지, 무엇이 포함이고 무엇이 유료인지를 미리 정합니다.

"납품물은 대금 지급 후 클라이언트의 사업에서 자유롭게 사용하실 수 있습니다. 나중에 다른 파트너로 전환하실 경우, 체계적인 인계 프로세스를 통해 모든 것을 넘겨 드리겠습니다."

〔그림 35-4〕 프로젝트 완료 → 유지보수 계약 → 확장 프로젝트 전환 흐름〕

초보자에게 가장 중요한 원칙은 이것입니다. 비공식적으로 일하는 것을 중단하는 것. 범위, 완료 정의, 대금, 유지보수, 서비스 수준, 버그와 변경의 구분, 소유권, 퇴출 조건을 서면 계약에 넣습니다. 양쪽이 무엇을 사고 무엇이 이후에 일어나는지를 알 때 프로젝트는 원활하게 흘러가고 오해는 줄어듭니다.

이 전체 프로세스—구축, 검증, 인계, 유지보수—를 전문적으로 수행하면, 일회성 프로젝트가 장기 관계로 전환됩니다. 관계가 깊어질수록 클라이언트의 비즈니스를 누구보다 잘 이해하게 되고, 그 이해가 경쟁 우위가 됩니다. 대체하기 어려운 파트너가 되는 것입니다.

따뜻한 관계에서 시작한 파트너십이 안정되면, 시야를 넓혀야 할 때가 옵니다. 따뜻한 네트워크 밖의 시장으로 나아가는 방법, 즉 콜드 아웃리치를 전략적으로 활용하는 시점과 방법이 다음 과제입니다.

전체 목차

책의 모든 장 보기

36장 콜드 아웃리치에서 장기 파트너십으로

전체 목차

책의 모든 장 보기

워밍업 후 콜드 아웃리치 시점

어느 날 스프레드시트를 열어 봅니다. 무료 파일럿 두 건이 성공적으로 마무리되었습니다. 한 건은 유지보수 계약으로 전환되었고, 다른 한 건에서는 추천사 영상을 받았습니다. 따뜻한 네트워크에서 만날 수 있는 잠재 클라이언트 목록이 바닥을 보이기 시작합니다.

이 시점에서 질문이 떠오릅니다. "이제 콜드 아웃리치(Cold Outreach)를 시작해도 되나?"

답은 준비 상태에 달려 있습니다. 콜드 아웃리치는 확장(Scale)의 도구입니다. 탐색(Exploration)의 도구가 아닙니다. 첫 클라이언트를 찾기 위해 사용하는 것이 아니라, 작동하는 시스템을 더 넓은 시장에 적용하기 위해 사용합니다.

콜드 아웃리치를 시작하기 전에 갖추어야 할 세 가지가 있습니다.

증거.

실제 프로젝트에서 나온 결과물이 있어야 합니다. "저는 AI 자동화를 합니다"가 아니라 "제가 만든 워크플로우로 A 기업은 주당 20시간을 절약했습니다"라고 말할 수 있어야 합니다. 숫자가 있는 문장은 숫자가 없는 문장보다 무게가 다릅니다.

언어.

기술자의 용어가 아니라 사업주의 용어로 말하는 능력이 필요합니다. 워밍업 대화에서 수집한 표현—"매일 반복되는 문의 정리", "엑셀에 수동으로 입력하는 시간"—이 자신의 어휘가 되어 있어야 합니다.

자신감.

막연한 자기 확신이 아니라, 실전에서 검증된 역량에서 나오는 자신감입니다. "이 유형의 문제는 풀어 본 적이 있고, 결과를 냈다"는 경험이 뒷받침하는 자세.

이 세 가지 없이 콜드 아웃리치를 하면 결과가 참담합니다. 하루에 수백 통의 이메일을 보내지만, 아무 맥락 없는 메시지를 아무 관계 없는 사람에게 보내는 것입니다. 증거도 없고, 신뢰도 없고, 연결고리도 없는 상태의 이메일은 수신자에게 잡음일 뿐입니다. 상품(Commodity)이 되어 버립니다. 상품이 되면 가격 경쟁에 빠지고, 가격 경쟁에서는 항상 더 싸게 제안하는 누군가가 있습니다.

증거 기반 접근: 콜드 메시지의 구조

콜드 아웃리치의 성공률을 결정하는 것은 발송 숫자가 아니라 메시지의 구조입니다.

미국의 한 AI 자동화 컨설턴트가 커뮤니티에서 관찰한 패턴이 있습니다. 사업 초기에 대량 콜드 이메일에 매달리던 실무자들의 공통점입니다. 링크드인(LinkedIn)을 웹 정보 수집하고, 이메일을 추출하고, AI로 일반적인 카피를 작성하고, 발송 도구에 넣어 하루 400에서 500통을 뿌렸습니다. 서류상으로는 생산적으로 보입니다. 그러나 현실은 다릅니다.

증거도 없고, 포지셔닝도 없고, 영혼도 없는 메시지를 보내는 것은 또 하나의 AI 템플릿 판매자에 불과합니다.

증거 기반 콜드 메시지는 구조가 다릅니다. 핵심 원칙은 "내가 무엇을 파는가"가 아니라 "내가 무엇을 해냈는가"로 시작하는 것입니다.

효과적인 콜드 메시지에는 네 가지 요소가 들어갑니다.

1.

맥락 연결:

상대방의 상황에 대한 한 문장. "귀사의 〔구체적 프로세스〕를 보고 연락드렸습니다."

2.

증거 제시:

비슷한 문제를 풀었던 경험. "비슷한 업무를 자동화해서 〔구체적 수치〕의 결과를 낸 적이 있습니다."

3.

가치 가설:

상대방에게 적용했을 때의 예상 효과. "〔이런 효과〕가 가능할 것으로 보입니다."

4.

낮은 장벽의 다음 단계:

부담 없는 제안. "15분 통화로 가능성을 함께 살펴볼 수 있을까요?"

이 네 요소를 5에서 6문장 안에 담습니다. 길어지면 읽히지 않습니다. 바쁜 사업주가 이메일을 스캔하는 시간은 몇 초입니다.

〔그림 36-1〕 증거 기반 콜드 메시지의 구조 다이어그램〕

콜드 메시지에서 가격을 먼저 이야기하면 안 됩니다. 가격은 대화가 시작된 뒤에 디스커버리(Discovery)를 통해 다루는 주제입니다. 메시지의 목표는 판매가 아니라 대화의 시작입니다.

커뮤니티 성과 공유 → 인바운드 전환

콜드 아웃리치만이 확장의 유일한 경로는 아닙니다. 오히려 더 강력한 경로가 있습니다. 인바운드(Inbound)입니다.

인바운드란 잠재 클라이언트가 먼저 찾아오는 것을 말합니다. 이것이 발생하는 메커니즘은 생각보다 간결합니다.

미국의 한 AI 자동화 커뮤니티에서 일어난 사례를 보겠습니다. 한 실무자가 첫 클라이언트를 확보하고 5일 만에 파일럿을 성공적으로 마무리했습니다. 그는 그 경험을 커뮤니티에 공유했습니다. "이런 문제를 이렇게 풀었고, 이런 결과가 나왔습니다." 구체적인 과정과 숫자를 담은 글이었습니다.

그 글을 본 다른 커뮤니티 멤버가 연락해 왔습니다. 자신의 클라이언트를 위해 일할 수 있는 개발자를 찾고 있었고, 이 사람의 실제 성과를 보고 신뢰가 생긴 것입니다. 이것이 두 번째 클라이언트가 되었습니다. 콜드 이메일을 보내지 않았습니다. 소개를 부탁하지도 않았습니다. 성과를 공유했을 뿐인데 기회가 찾아왔습니다.

이 패턴은 우연이 아닙니다. 성과 공유는 세 가지 효과를 동시에 발생시킵니다.

신뢰 축적:

구체적인 숫자와 과정이 담긴 공유는 자기 홍보(Self-promotion)가 아닌 사례 연구(Case Study)로 읽힙니다. 읽는 사람은 "이 사람이 진짜로 했구나"라고 생각합니다.

검색 가능성:

온라인 커뮤니티, 블로그, 소셜 미디어에 공유한 콘텐츠는 검색 엔진과 플랫폼 알고리즘에 의해 노출됩니다. 자고 있는 동안에도 누군가가 "AI 자동화 리드 처리"를 검색하고 그 글을 발견할 수 있습니다.

네트워크 효과:

한 사람이 본 글을 다른 사람에게 전달합니다. "이 사람 괜찮은 것 같은데 한번 보세요." 이런 전달은 직접 보낸 콜드 메시지보다 수신자에게 훨씬 강한 인상을 줍니다.

성과 공유의 형식은 다양합니다. 커뮤니티 포럼 글, 링크드인 포스트, 블로그 글, 짧은 영상. 형식보다 중요한 것은 내용의 구체성입니다. "클라이언트에게 자동화를 만들어 줬더니 좋았습니다"는 효과가 없습니다. "인바운드 리드 처리 프로세스를 자동화했더니 직원 한 명이 주당 10시간을 절약했고, 리드 응답 시간이 24시간에서 5분으로 줄었습니다"는 효과가 있습니다.

〔그림 36-2〕 성과 공유 → 인바운드 전환 경로 다이어그램〕

AI 사고 파트너 포지셔닝

콜드 아웃리치든 인바운드든, 장기 파트너십으로 가는 길에서 포지셔닝이 결정적 역할을 합니다.

포지셔닝(Positioning)의 차이를 살펴보겠습니다.

포지션 A:

"저는 AI 워크플로우를 만드는 사람입니다."

포지션 B:

"저는 기업이 AI를 활용해 운영을 개선하도록 돕는 파트너입니다."

A는 기술을 팝니다. B는 성과를 팝니다. A에게 클라이언트는 "이거 만들어 주세요"라고 말합니다. B에게 클라이언트는 "어떻게 하면 좋겠습니까?"라고 묻습니다. A는 대체 가능합니다. B는 대체하기 어렵습니다.

AI 사고 파트너(AI Thought Partner)가 되는 것은 기술 수준의 문제만이 아닙니다. 관점의 문제입니다. 기업은 지금 AI가 필요하다는 것을 알고 있습니다. 그러나 AI가 자신들의 운영에 어디에 어떻게 들어맞는지는 모릅니다. 이 간극을 메워 줄 수 있는 사람이 필요합니다.

이미 한 클라이언트의 생태계 안에 깊이 들어가 있다면—시스템을 구축했고, 데이터를 이해하고, 프로세스를 알고 있다면—그 클라이언트가 새로운 AI 도입 결정을 할 때 가장 먼저 연락하는 사람이 될 수 있습니다. 이것이 사고 파트너의 위치입니다.

이 포지션에 도달하려면 기술 구현 너머를 봐야 합니다. 감사(Audit), 교육(Enablement), 전략 자문이 그 영역입니다. 기업이 AI로 무엇이 가능한지 모를 때, 그 가능성을 보여 주는 사람. 새로운 모델이 출시되었을 때, 기존 시스템에 미치는 영향을 분석해 주는 사람. 이런 역할은 워크플로우 하나를 만들어 주는 것보다 가치가 큽니다.

〔그림 36-3〕 템플릿 판매자 vs. AI 사고 파트너 포지셔닝 비교〕

장기 파트너십의 구조는 이렇게 형성됩니다. 첫 프로젝트로 발을 들여놓고, 유지보수로 관계를 지속하고, 확장 프로젝트로 가치를 증명하며, 리테이너로 안정적 파트너십을 맺습니다. 이 과정에서 클라이언트의 비즈니스에 대한 컨설턴트의 이해는 점점 깊어지고, 다른 사람이 대체하기 어려운 위치에 서게 됩니다.

관계가 깊어질수록 클라이언트는 기술 제공자가 아니라 사업 파트너로 인식하기 시작합니다. 그리고 그 클라이언트가 같은 업계의 다른 사업주에게 컨설턴트를 소개할 때, 소개의 무게는 직접 보낸 어떤 콜드 메시지보다 무겁습니다.

이 모든 과정이 추상적으로 느껴질 수 있습니다. 이론이 아니라 실제로 이 경로를 걸어간 사람의 이야기를 구체적으로 따라가 보면 그림이 훨씬 선명해집니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

37장 실전 사례 연구

전체 목차

책의 모든 장 보기

대량 콜드 이메일의 실패

미국에서 AI 자동화 사업을 시작하려던 한 실무자가 있었습니다. 그는 열정이 넘쳤고, 행동력도 있었습니다. 문제는 방향이었습니다.

그가 처음 한 일은 대부분의 초보자가 하는 것과 정확히 같았습니다. 링크드인에서 프로필을 웹 정보 수집하고, 이메일 주소를 추출하고, AI를 이용해 일반적인 메시지를 작성했습니다. 대량 발송 도구에 이 모든 것을 넣고 하루에 400에서 500통의 이메일을 보냈습니다. 서류상으로 보면 엄청난 활동량입니다. 스프레드시트에는 발송 숫자가 빠르게 올라갔습니다.

그런데 응답은 오지 않았습니다.

이유는 명확합니다. 수신자 입장에서 그의 메시지는 매일 받는 수십 통의 영업 메일 중 하나였을 뿐입니다. 보낸 사람이 누구인지 모릅니다. 어떤 결과를 냈는지도 모릅니다. AI 템플릿을 커스터마이즈해 주겠다는 제안은 다른 열 명도 하고 있었습니다. 차별점이 없었습니다. 그는 상품(Commodity)이었습니다. 상품이 되면 가격으로 경쟁해야 하고, 가격 경쟁에서는 더 싸게 해 주겠다는 누군가가 항상 나타납니다.

이 상태에서 더 많은 이메일을 보내는 것은 구멍 뚫린 양동이에 물을 더 붓는 것과 같습니다. 양이 아니라 구조가 문제였습니다.

〔그림 37-1〕 대량 콜드 이메일의 전환율 구조 분석〕

마인드셋의 전환: 워크플로우 판매자에서 성과 파트너로

전환점이 된 것은 도구의 변경이 아니었습니다. 사고방식의 변화였습니다.

이 실무자가 AI 자동화 커뮤니티에 합류한 뒤 받은 수정 의견의 핵심은 이것이었습니다. 워크플로우를 파는 사람처럼 생각하는 것을 멈추고, 성과를 이끄는 파트너처럼 생각하기 시작하라는 것.

구체적으로 언어가 바뀌었습니다.

이전:

"이런 자동화를 만들었습니다. 이 템플릿을 커스터마이즈해 드릴 수 있습니다."

이후:

"귀사의 〔구체적 업무〕에서 이런 결과를 만들어 드릴 수 있습니다."

전자는 자신이 만든 것을 설명합니다. 후자는 상대방이 얻을 것을 설명합니다. 이 한 가지 전환이 모든 것을 바꿨습니다. 같은 기술, 같은 도구, 같은 사람. 달라진 것은 접근 방식뿐입니다.

그리고 그는 차가운 접촉을 멈추고 따뜻한 접촉으로 돌아갔습니다. 자신이 아는 사람, 아는 사람의 아는 사람에게 먼저 손을 뻗었습니다. "무언가를 팔려는 것이 아닙니다. 일상 업무에서 반복적이고 번거로운 부분이 어디인지 여쭤봐도 될까요?" 이 질문으로 대화를 시작했습니다.

5일 만에 첫 클라이언트

따뜻한 네트워크에서 대화를 나누면서 그는 한 가지 뚜렷한 고통점을 가진 사업주를 만났습니다. 반복적이고 시간이 많이 드는 수동 작업이었습니다.

그는 이렇게 제안했습니다.

"이 부분을 자동화하는 작은 워크플로우를 무료로 만들어 드리겠습니다. 실제로 시간이 절약되는지 함께 확인해 보죠. 대신 솔직한 수정 의견을 부탁드립니다."

무료라는 제안은 상대방의 리스크를 제거했습니다. 솔직한 수정 의견이라는 대가는 자선이 아닌 교환의 구조를 만들었습니다. 사업주는 동의했습니다.

그 다음의 며칠은 집중적인 구축 기간이었습니다. 복잡한 시스템을 만든 것이 아닙니다. 작지만 확실하게 작동하는 하나의 자동화를 만들었습니다. API 호출을 겹겹이 쌓은 인상적인 아키텍처가 아니라, "이 업무에 매일 한 시간 쓰던 것을 이제 5분이면 됩니다"라는 결과가 목표였습니다.

파일럿이 작동하기 시작했을 때, 사업주의 반응은 가격표 없이도 가치를 증명했습니다. 시간이 절약되었고, 그 절약은 측정 가능했습니다.

첫 연락부터 파일럿 완성까지 5일. 대량 콜드 이메일로 수개월을 보냈던 것과 비교하면 극적인 차이입니다. 차이를 만든 것은 기술이 아니라 접근법이었습니다.

〔그림 37-2〕 콜드 이메일 접근법 vs. 워밍업 접근법의 타임라인 비교〕

커뮤니티에 성과를 공유하다

파일럿이 성공한 뒤, 이 실무자는 자신의 경험을 AI 자동화 커뮤니티에 공유했습니다. 구체적인 내용이 담긴 글이었습니다. 어떤 문제를 풀었는지, 어떤 접근을 했는지, 결과가 어떠했는지. 추상적인 자기 홍보가 아니라, 다른 사람이 따라 할 수 있는 실전 기록이었습니다.

이 공유는 두 가지 효과를 낳았습니다.

첫 번째는 커뮤니티 내에서의 신뢰 축적이었습니다. 구체적인 숫자와 과정이 담긴 사례 공유는 이론적인 조언보다 훨씬 무게가 있습니다. "이 사람은 말만 하는 것이 아니라 실제로 했구나"라는 인식이 형성됩니다.

두 번째가 더 중요합니다. 예상치 못한 기회가 찾아왔습니다.

두 번째 클라이언트는 인바운드로

커뮤니티에 올린 성과 공유 글을 본 다른 멤버가 연락해 왔습니다. 이 멤버는 자신의 클라이언트를 위해 워크플로우를 구축할 수 있는 사람을 찾고 있었습니다. 실무자의 글을 읽고, 실제 결과를 확인하고, "이 사람이라면 믿을 수 있겠다"고 판단한 것입니다.

이것이 두 번째 클라이언트였습니다. 콜드 이메일을 보내지 않았습니다. 영업 전화를 하지 않았습니다. 성과를 공유했을 뿐인데 기회가 걸어왔습니다.

여기서 주목할 것은 이 기회가 순전한 행운이 아니라는 점입니다. 성과 공유는 일종의 수동형 마케팅(Passive Marketing)입니다. 한 번 게시하면 여러 사람이 여러 시점에 볼 수 있습니다. 그 중 한 명이라도 비슷한 필요를 가지고 있으면 연결이 발생합니다. 콜드 이메일이 일대일 접촉이라면, 커뮤니티 공유는 일대다(多) 접촉입니다. 효율이 구조적으로 다릅니다.

〔그림 37-3〕 성과 공유 → 인바운드 기회 발생의 메커니즘〕

신뢰 → 리테이너 → 소개 사이클

두 번째 클라이언트와의 작업이 시작되면서 패턴이 더욱 명확해졌습니다. 이 실무자가 따른 경로를 복기해 보겠습니다.

1.

따뜻한 접촉으로 시작.

아는 사람에게 질문하고, 고통점을 파악했습니다. 2.

성과 중심 제안.

워크플로우가 아니라 결과를 제안했습니다. 3.

무료 파일럿으로 신뢰 구축.

리스크 없는 작은 프로젝트로 증명했습니다. 4.

가치 전달.

측정 가능한 시간 절약이라는 결과를 보여 줬습니다. 5.

관계 심화.

유지보수 또는 확장 제안으로 관계를 지속했습니다. 6.

증거 확보.

추천사와 사례를 축적했습니다. 7.

성과 공유.

커뮤니티에 경험을 게시했습니다. 8.

인바운드 기회.

공유를 본 누군가가 먼저 연락해 왔습니다.

이 여덟 단계는 순환합니다. 두 번째 클라이언트에서 다시 가치를 전달하고, 신뢰를 쌓고, 추천사를 얻고, 경험을 공유하면 세 번째 기회가 열립니다. 각 순환마다 증거의 양이 늘어나고, 메시지의 정교함이 올라가고, 자신감이 커집니다.

이 사이클이 두세 번 돌면 리테이너 대화가 자연스러워집니다. 이미 가치를 반복적으로 증명했기 때문입니다. 클라이언트는 월정액을 지불하는 것이 새로운 파트너를 찾는 것보다 효율적이라는 것을 경험으로 알게 됩니다.

리테이너가 안정되면 소개(Referral)도 자연스러워집니다. 만족한 클라이언트는 같은 업계의 지인에게 추천합니다. 이 추천은 직접 보낸 어떤 콜드 메시지보다 전환율이 높습니다. 왜냐하면 추천에는 이미 검증된 신뢰가 내장되어 있기 때문입니다.

〔그림 37-4〕 신뢰 → 리테이너 → 소개 → 새 클라이언트 순환 다이어그램〕

이 실무자의 이야기에서 가장 중요한 교훈은 무엇일까요? 더 좋은 도구나 더 많은 자동화가 아니었습니다. 마인드셋의 전환이었습니다. 워크플로우를 파는 사람에서 성과를 이끄는 파트너로의 전환. 대량 발송에서 개인적 연결로의 전환. 즉각적 수익 추구에서 신뢰 축적으로의 우선순위 전환.

이 전환이 수개월간 성과 없던 콜드 아웃리치를 5일 만의 첫 클라이언트로, 그리고 인바운드로 찾아온 두 번째 클라이언트로 바꿔 놓았습니다.

돈이 되는 구조 만들기와 비즈니스 전략에 관한 구체적인 기술과 사례를 살펴보았습니다. 이제 시야를 넓혀 이 책 전체에서 다룬 것들을 하나의 통합된 시스템으로 정리할 차례입니다. 개별 기술과 프레임워크를 어떻게 연결하여 지속 가능한 실무 체계로 만들 수 있는지, 그 구조를 그려 보겠습니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

38장 배운 것을 하나의 시스템으로 묶기

전체 목차

책의 모든 장 보기

흩어진 퍼즐을 하나의 그림으로

책상 위에 수십 장의 메모가 놓여 있다고 상상해 보겠습니다. 한 장에는 WAT 프레임워크가 적혀 있습니다. 다른 장에는 한 번에 읽는 범위 관리 전략이. 또 다른 장에는 7일 프레임워크, 가치 기반 가격 책정, MCP 연결 서버 설정 방법이 적혀 있습니다. 각각은 의미가 있지만 따로 놓여 있으면 그저 메모 더미입니다.

이 장의 목적은 그 메모들을 하나의 지도 위에 올려놓는 것입니다. 개별 기술이 아니라 시스템으로 작동하게 만드는 것. 에이전트형 AI 실무자가 프로젝트를 구상하고, 구축하고, 납품하고, 확장하는 전체 흐름을 하나의 경로로 연결하는 작업입니다.

WAT 프레임워크 복습: 모든 프로젝트의 시작점

이 책의 초반에 등장한 WAT 프레임워크—워크플로우(Workflow), 에이전트(Agent), 도구(Tool)—를 다시 꺼내 보겠습니다. 이 세 요소는 개별 개념이 아니라 프로젝트를 설계할 때 반복적으로 사용하는 렌즈입니다.

새로운 프로젝트를 맡았을 때 물어야 할 질문은 세 가지입니다.

워크플로우:

이 문제를 풀기 위한 전체 흐름은 어떤 모양인가? 시작점은 어디이고, 끝점은 어디인가? 어떤 단계를 거치는가?

에이전트:

이 흐름 안에서 AI가 판단해야 하는 지점은 어디인가? 어떤 단계에서 에이전트가 개입하고, 어떤 단계는 규칙 기반으로 처리하는 것이 나은가?

도구:

에이전트가 일을 하려면 어떤 도구에 접근해야 하는가? MCP 연결 서버로 연결해야 할 외부 서비스는 무엇인가? 파일 시스템, API, 데이터베이스 중 어떤 것이 필요한가?

이 세 가지 렌즈를 통해 어떤 프로젝트든 분해할 수 있습니다. 웹사이트 클론이든, 고객 지원 자동화든, 채용 데이터 웹 정보 수집이든 구조는 같습니다. 워크플로우를 그리고, 에이전트의 역할을 정의하고, 도구를 연결합니다.

〔그림 38-1〕 WAT 프레임워크와 프로젝트 설계 흐름도〕

이 프레임워크의 힘은 반복 가능성에 있습니다. 한 번 익히면 산업이 바뀌어도, 문제가 바뀌어도, 도구가 바뀌어도 적용할 수 있습니다. 기술은 진화하지만 프레임워크는 안정적입니다.

EA 로드맵 점검: 현재 위치 파악

이 책에서 다룬 경로를 하나의 로드맵으로 시각화해 보겠습니다. 에이전트형 AI 실무자(EA, Agentic AI Practitioner)의 성장 단계입니다.

1단계: 환경 구축과 기본 이해

터미널 설정, 클로드 코드 설치, 기본 명령어 학습. 한 번에 읽는 범위의 개념과 모델이 세는 글 조각 관리 전략. CLAUDE.md 파일 작성. 이 단계는 도구를 손에 쥐는 것에 해당합니다.

2단계: 첫 번째 결과물 생성

웹사이트 클론, 간단한 웹 정보 수집 워크플로우, 문서 자동화. 클로드 코드에 지시를 내리고 결과를 받는 경험. 계획 모드에서 복잡한 작업을 단계별로 분해하는 연습. 이 단계에서 "에이전트가 실제로 일을 한다"는 감각이 형성됩니다.

3단계: 복잡한 워크플로우 구축

MCP 연결 서버 연결, 검색 결합 생성 시스템 구축, 보조 에이전트 활용, 멀티스텝 파이프라인 설계. 하나의 에이전트가 아니라 여러 에이전트가 협업하는 구조를 만드는 단계. 에러 핸들링, 로그 기록, 컨텍스트 관리가 중요해지는 시점입니다.

4단계: 실전 납품과 비즈니스 구축

클라이언트 확보, 가격 책정, QA, 핸드오버. 기술적 역량 위에 비즈니스 역량을 쌓는 단계. 무료 파일럿에서 유료 프로젝트로, 일회성에서 리테이너로, 따뜻한 접촉에서 콜드 아웃리치로 확장하는 경로.

〔그림 38-2〕 EA 성장 로드맵 4단계 다이어그램〕

지금 독자가 어디에 있든, 이 로드맵 위에서 자신의 위치를 확인하는 것이 중요합니다. 1단계에 있다면 무리하게 4단계의 고민을 할 필요가 없습니다. 3단계에 있다면 기술을 더 쌓는 것보다 4단계로 넘어가는 행동이 더 효과적일 수 있습니다.

각 단계를 건너뛸 수는 없습니다. 하지만 각 단계에서 보내는 시간은 단축할 수 있습니다. 이 책에서 제시한 프레임워크, 체크리스트, 사례들이 그 단축의 도구입니다.

워크플로우 포트폴리오 정리

실무자로서의 가치를 증명하는 가장 직접적인 방법은 포트폴리오입니다. 그런데 AI 자동화 실무자의 포트폴리오는 디자이너의 포트폴리오와 다릅니다. 시각적 결과물이 아니라 문제 해결의 기록이 핵심입니다.

효과적인 포트폴리오 항목은 네 가지 요소로 구성됩니다.

문제 정의:

어떤 비즈니스 문제가 있었는가? 누가 어떤 고통을 겪고 있었는가?

접근 방법:

WAT 프레임워크 관점에서 워크플로우를 어떻게 설계했는가? 어떤 도구와 에이전트 구조를 사용했는가?

결과:

정량적 성과는 무엇인가? 시간 절약, 비용 절감, 오류 감소 등의 숫자.

배운 점:

프로젝트에서 무엇을 개선했고, 다음번에는 어떻게 달리 하겠는가?

이 네 가지가 갖추어진 사례 하나가 "저는 AI 자동화를 합니다"라는 자기소개보다 백 배 강력합니다.

포트폴리오를 정리하는 실용적인 방법은 간결한 노션(Notion) 페이지나 구글 독스(Google Docs) 문서입니다. 화려할 필요가 없습니다. 명확하면 됩니다. 각 프로젝트를 위의 네 가지 틀로 정리하고, 가능하다면 짧은 영상 데모를 첨부합니다.

무료 파일럿도 포트폴리오에 포함됩니다. 무료였다는 것은 가격 문제이지 성과의 문제가 아닙니다. 무료 파일럿에서 클라이언트의 시간을 주당 5시간 절약했다면, 그것은 유효한 성과입니다.

〔그림 38-3〕 워크플로우 포트폴리오 항목 구성 예시〕

지속적 학습 리소스와 실무 체계

AI의 세계는 빠르게 변합니다. 새로운 모델이 출시되고, 도구가 업데이트되고, 가능성의 경계가 넓어집니다. 이 속도를 따라가는 것은 불가능합니다. 모든 것을 알려고 하면 아무것도 깊이 알 수 없습니다.

대신 효과적인 학습 체계를 갖추는 것이 중요합니다.

핵심 기술 갱신:

자신이 주로 사용하는 도구—클로드 코드, 주요 MCP 연결 서버, 배포 플랫폼—의 업데이트를 추적합니다. 모든 AI 뉴스를 따라갈 필요는 없지만, 자신의 실무에 직접 영향을 미치는 변화는 놓치지 않아야 합니다.

커뮤니티 참여:

같은 분야의 실무자들과 교류하는 것은 정보 수집과 동기 부여를 동시에 해결합니다. 온라인 커뮤니티, 오프라인 모임, 분기별 이벤트 등 형태는 다양합니다. 중요한 것은 혼자 고립되지 않는 것입니다.

실험 시간 확보:

클라이언트 작업 외에 개인 프로젝트를 위한 시간을 의도적으로 확보합니다. 새로운 모델을 검증하거나, 새로운 워크플로우 패턴을 시도하거나, 완전히 다른 산업의 문제에 자신의 기술을 적용해 보는 것. 이 실험 시간이 역량의 폭을 넓히고, 클라이언트에게 새로운 가능성을 제안할 수 있는 원천이 됩니다.

기록 습관:

프로젝트마다 배운 점, 실패한 시도, 성공한 패턴을 기록합니다. 이 기록은 나중에 포트폴리오의 재료가 되고, 비슷한 프로젝트를 만났을 때의 참고 자료가 되며, 성장의 궤적을 보여 주는 증거가 됩니다.

〔그림 38-4〕 지속적 학습 체계의 4가지 요소 다이어그램〕

이 모든 것—WAT 프레임워크, EA 로드맵, 포트폴리오, 학습 체계—은 하나의 시스템으로 연결됩니다. 프레임워크가 프로젝트를 구조화하고, 로드맵이 성장 방향을 잡아 주고, 포트폴리오가 성과를 축적하고, 학습 체계가 진화를 보장합니다.

이 시스템은 독자의 것입니다. 이 책이 제공한 것은 초기 설계도이지, 완성된 건물이 아닙니다. 실전에서 부딪히면서 수정하고, 자신의 맥락에 맞게 변형하고, 경험이 쌓이면서 더 정교하게 다듬어 나가야 합니다.

개별 기술과 프레임워크를 하나의 시스템으로 엮는 작업이 끝났습니다. 그런데 시스템을 가진 사람이 바라보아야 할 더 넓은 풍경이 남아 있습니다. 에이전트형 AI가 빠르게 진화하는 이 시대에, 독자가 서야 할 자리는 어디인가라는 질문입니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

39장 에이전트형 AI 시대, 당신의 자리

전체 목차

책의 모든 장 보기

빌더에서 아키텍트로

지금 이 문장을 읽고 있는 독자는 이 책의 마지막 장에 와 있습니다. 수백 페이지의 여정이 여기서 하나의 질문으로 수렴합니다. "이제 무엇을 할 것인가?"

한 가지 장면을 떠올려 보겠습니다. 이 책의 초반에 독자는 터미널 창 앞에서 처음으로 클로드 코드를 실행했습니다. 커서가 깜빡이고, 무엇을 입력해야 할지 몰라 잠시 멈칫했습니다. 그때와 지금 사이에는 거리가 생겼습니다. 한 번에 읽는 범위를 관리하는 법을 알게 되었고, CLAUDE.md 파일로 에이전트에게 직무 기술서를 쓰는 법을 배웠습니다.

MCP 연결 서버로 외부 도구를 연결하고, 검색 결합 생성 시스템을 구축하고, 보조 에이전트를 조율하는 방법을 익혔습니다. 클라이언트를 확보하고, 가격을 책정하고, 워크플로우를 납품하는 과정까지 경험했거나 경험할 준비가 되었습니다.

그런데 이 모든 기술의 아래에 더 본질적인 전환이 있습니다. 빌더(Builder)에서 아키텍트(Architect)로의 전환입니다.

빌더는 만듭니다. 아키텍트는 설계합니다. 빌더는 도구를 사용합니다. 아키텍트는 도구가 어떻게 조합되어야 하는지를 결정합니다. 빌더는 하나의 워크플로우를 완성합니다. 아키텍트는 여러 워크플로우가 기업의 운영 체계 안에서 어떻게 맞물려야 하는지를 구상합니다.

이 책의 전반부에서 독자는 빌더였습니다. 웹사이트를 클론하고, 데이터를 웹 정보 수집하고, 문서를 자동 생성했습니다. 후반부로 오면서 아키텍트의 사고방식이 조금씩 스며들기 시작했습니다. WAT 프레임워크로 문제를 분해하고, 에이전트 팀의 역할을 배분하고, 클라이언트의 비즈니스 전체를 조망하며 자동화의 우선순위를 정하는 것. 이것이 아키텍트의 일입니다.

앞으로의 여정에서 독자의 가치는 코드를 작성하는 능력보다 설계하는 능력에서 나올 것입니다. AI 모델은 점점 강해지고, 코드 생성은 점점 쉬워집니다. 그러나 "이 기업에 무엇이 필요한가?"를 파악하고, "어떤 순서로, 어떤 구조로 해결할 것인가?"를 결정하는 일은 여전히 인간의 몫입니다.

〔그림 39-1〕 빌더 역할에서 아키텍트 역할로의 전환 다이어그램〕

한국 시장의 기회

이 책에서 다룬 사례와 전략은 미국 시장을 배경으로 합니다. 그러나 독자가 서 있는 곳은 한국입니다. 그리고 한국 시장에는 고유한 기회가 있습니다.

한국 기업의 디지털 인프라는 세계적인 수준입니다. 인터넷 보급률, 모바일 기기 사용률, 클라우드 도입률 모두 높습니다. 그런데 이 인프라 위에서 AI 자동화를 실무에 적용하는 속도는 인프라 수준에 비해 느립니다. 많은 기업이 AI를 도입하고 싶다는 의사를 밝히지만, 실제로 운영 프로세스에 AI 워크

플로우를 삽입한 사례는 아직 제한적입니다.

이 간극이 기회입니다.

한국의 중소기업과 중견기업에는 자동화할 수 있는 반복 업무가 풍부합니다. 고객 문의 분류, 견적서 작성 보조, 내부 보고서 자동 생성, 데이터 입력과 검증, 일정 관리와 알림. 이런 업무들은 미국 시장에서 자동화되고 있는 것과 본질적으로 같습니다. 기술은 국경을 넘지만, 그 기술을 현지 비즈니스 맥락에 맞게 적용하는 것은 현장에 있는 사람만이 할 수 있습니다.

한국어로 소통하고, 한국 기업의 업무 문화를 이해하며, 한국 시장의 규제와 관행에 익숙한 AI 자동화 실무자는 아직 많지 않습니다. 이것은 수요와 공급의 불균형입니다. 수요는 빠르게 증가하고 있지만 공급은 아직 형성 단계에 있습니다.

여기서 이 책에서 배운 접근법이 그대로 적용됩니다. 따뜻한 네트워크에서 시작하고, 사업주의 언어로 대화하고, 작은 파일럿으로 증명하고, 성과를 공유하며 신뢰를 축적합니다. 시장이 한국이라는 사실이 프레임워크를 바꾸지 않습니다. 대화의 언어와 비즈니스 맥락이 달라질 뿐입니다.

〔그림 39-2〕 한국 시장의 AI 자동화 기회 영역 맵〕

변하지 않는 것: 문제 정의 능력

기술은 빠르게 변합니다. 오늘의 최신 모델이 6개월 뒤에는 구식이 될 수 있습니다. 새로운 도구가 등장하고, 기존 도구가 사라지기도 합니다. 이 변화의 속도 앞에서 불안을 느끼는 것은 자연스럽습니다. "지금 배운 것이 내년에도 유효할까?"

이 질문에 대한 답은 독자가 무엇을 배웠느냐에 따라 달라집니다. 특정 도구의 버튼 위치를 배웠다면 그것은 금방 쓸모없어질 수 있습니다. 하지만 이 책에서 강조한 것은 버튼 위치가 아닙니다.

문제를 정의하는 능력. 이것은 변하지 않습니다.

"이 기업에서 가장 반복적이고 시간을 잡아먹는 업무는 무엇인가?" "그 업무를 자동화하면 어떤 가치가 발생하는가?" "자동화의 범위를 어디까지 잡아야 실현 가능하고 의미 있는가?" 이 질문들은 AI 모델이 GPT-3에서 GPT-5로 바뀌어도, 클로드 코드가 버전 10이 되어도 동일하게 유효합니다.

도구는 바뀝니다. 에이전트의 능력은 확장됩니다. 그러나 "무엇을 풀어야 하는가?"라는 질문을 던지는 것은 도구의 역할이 아닙니다. 그것은 비즈니스를 이해하고, 사람과 대화하고, 고통점을 포착하는 인간의 역할입니다. 독자가 이 능력을 갖추고 있다면, 도구가 무엇이든 가치를 만들어 낼 수 있습니다.

WAT 프레임워크가 도구에 독립적인 이유도 여기에 있습니다. 워크플로우, 에이전트, 도구라는 세 가지 렌즈는 특정 소프트웨어에 묶여 있지 않습니다. 클로드 코드 대신 다른 에이전트 도구가 등장하더라도, 워크플로우를 설계하고 에이전트의 역할을 정의하고 도구를 연결하는 구조적 사고는 그대로 전이됩니다.

디스커버리 과정에서 클라이언트의 프로세스를 처음부터 끝까지 파악하는 능력, ROI를 계산하여 가격을 정당화하는 능력, QA를 통해 품질을 보증하는 능력, 핸드오버를 전문적으로 수행하는 능력—이

것들은 기술 스택과 무관한 역량입니다. 시대가 바뀌어도 가치를 잃지 않습니다.

첫 워크플로우를 만들 시간

이제 정말로 마지막입니다.

이 책의 38개 장을 읽는 것과 첫 번째 워크플로우를 만드는 것 사이에는 하나의 간극만 있습니다. 행동입니다.

지식은 충분합니다. 완벽하지 않아도 됩니다. 모든 것을 기억하지 않아도 됩니다. 막히면 다시 펼쳐 보면 됩니다. 중요한 것은 시작하는 것입니다.

터미널을 열고, 클로드 코드를 실행하고, 자신의 일상에서 반복되는 한 가지 업무를 자동화해 보는 것입니다. 자신을 위한 워크플로우부터 시작해도 좋습니다. 매일 아침 뉴스를 정리하는 워크플로우, 이메일을 분류하는 워크플로우, 회의록을 요약하는 워크플로우. 작고 구체적이고 자신에게 직접적으로 유용한 것.

이 첫 번째 워크플로우가 완성되는 순간, 이 책에서 읽은 모든 것이 체감으로 전환됩니다. 한 번에 읽는 범위가 실제로 어떻게 소진되는지 느끼게 됩니다. 에이전트에게 명확한 지시를 내리는 것이 왜 중요한지 몸으로 알게 됩니다. 그리고 작은 자동화 하나가 시간을 되돌려 주는 경험이 다음 워크플로우로 이어지는 동력이 됩니다.

두 번째 워크플로우는 다른 사람을 위한 것일 수 있습니다. 친구의 사업에서 반복 업무를 하나 자동화해 주는 무료 파일럿. 세 번째는 그 파일럿의 성공에 기반한 유료 프로젝트가 될 수 있습니다. 네 번째부터는 패턴이 보이기 시작합니다.

에이전트형 AI의 시대가 열리고 있습니다. 이 시대에 필요한 사람은 모든 것을 아는 사람이 아닙니다. 문제를 정의하고, 해결책을 설계하고, 결과를 납품할 수 있는 사람입니다. 기술은 도구입니다. 독자의 판단, 공감, 실행력이 그 도구에 방향을 부여합니다.

〔그림 39-3〕 첫 워크플로우에서 비즈니스까지의 경로 요약〕

이 책이 건네고 싶었던 것은 기술 매뉴얼이 아닙니다. 새로운 종류의 일을 시작할 수 있다는 확신입니다. 도구는 준비되어 있습니다. 프레임워크는 제시되었습니다. 사례는 공유되었습니다.

남은 것은 독자의 첫 번째 엔터(Enter) 키입니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

40장 클로드 코드 v2.1.86 완전 레퍼런스

전체 목차

책의 모든 장 보기

도입

앞 장에서는 클로드 코드로 무엇을 할 수 있는지를 열 가지 사례로 풀어 보았습니다. 이번 장에서는 도구 자체를 체계적으로 정리합니다. 키보드 단축키부터 시작해서 명령줄 방식 명령어와 플래그, 세션 안에서 쓰는 슬래시 명령어, 스킬과 에이전트 생태계, 그리고 이 모든 것이 만나는 고수 워크플로우까지 순서대로 훑어봅니다. 기준 버전은 2026년 4월 현재의 v2.1.86입니다.

한 번 훑어 읽으면 전체 구조가 머리에 들어옵니다. 책상 옆에 펴두고 필요할 때마다 뒤지셔도 됩니다. 매일 쓰는 기능은 열 개를 넘지 않지만, 나머지가 있다는 것을 알아두면 막히는 순간에 꺼낼 카드가 생깁니다.

Claude Code 한눈 요약

클로드 코드의 구성은 크게 여섯 덩어리입니다. 키보드, MCP, 슬래시 명령어, 메모리, 워크플로우, 스킬/에이전트, 명령줄 방식. 이 중에 처음 입문하는 분이 외울 것은 딱 세 가지입니다.

Ctrl+C로 생성을 멈추고, Ctrl+L로 화면을 정리하고, /clear로 대화를 초기화한다.

이 세 가지만 있으면 일단 시작할 수 있습니다. 나머지는 쓰다가 필요해지는 순간에 하나씩 찾아서 익히면 됩니다.

여섯 영역 각각의 핵심만 한 줄씩 정리합니다.

키보드: Ctrl+C 멈춤, Ctrl+L 화면 정리, Ctrl+D 종료, Shift+Tab 모드 전환, Alt+T 심층추론, 특수 접두사는 슬래시(/), 느낌표(!), 골뱅이(@) 세 가지.

MCP: 외부 도구 연결 규격. --transport로 http, stdio, sse 세 가지 연결 방식을 지원하고, 범위는 Local, Project, User 세 층으로 관리합니다. /mcp 명령어가 관리 UI이고, Elicitation(질의응답 방식 도구 호출)도 지원합니다.

슬래시: 세션 안에서 치는 명령어. /clear, /compact, /resume으로 대화를 관리하고, /plan, /voice, /loop로 모드를 바꾸고, /init으로 프로젝트를 초기화하고, /memory로 지침서를 편집합니다. /btw와 /rc, /schedule은 v2.1 계열에서 새로 들어왔거나 강화된 명령어입니다.

메모리: CLAUDE.md가 프로젝트 지침의 본체, rules/*.md가 세부 규칙, @path 형식으로 파일을 불러오고, memory/ 폴더에는 약 25KB 분량의 자동 메모리가 저장됩니다.

워크플로우: Plan Mode로 먼저 계획을 세우고, ultrathink로 최강 사고 모드를 한 번 쓰고, /batch로 워크트리를 병렬 돌리고, /compact로 컨텍스트를 압축합니다.

설정 파일: settings.json이 동작 설정의 본체, ANTHROPIC_API_KEY 환경변수는 필수. managed-settings.d/ 폴더를 두면 관리자가 배포한 설정이 적용되고, hooks는 조건부로 실행되는 트리거입니다.

스킬/에이전트: .claude/skills/ 안에 SKILL.md로 커스텀 스킬을 만듭니다. 내장 에이전트는 Explore, Plan, General, Bash 네 종이고, initialPrompt와 SendMessage는 v2.1 계열의 신규 기능입니다.

명령줄 방식: claude -p로 비대화형 실행, claude -c로 마지막 대화 이어서, --bare는 최소 실행, --channels는 권한 릴레이, --permission-mode plan은 계획 모드 강제입니다.

이 전체를 한 장면으로 그리면, 터미널에 자리 잡은 한 명의 조수가 단축키로 움직이고, 슬래시 명령어로 일을 바꾸고, MCP로 외부 도구를 부르고, 스킬로 전문 지식을 꺼내쓰고, 명령줄 방식로 자동화 파이프라인에 연결되는 구조입니다. 각 기능은 서로 겹치지 않고 역할이 분명합니다. 이제 하나씩 자세히 봅니다.

키보드 단축키

클로드 코드는 터미널 안에서 돌아가는 도구라서, 키보드 단축키가 마우스를 대체합니다. 익혀두면 손이 키보드를 떠나지 않은 채 세션을 지휘할 수 있습니다.

일반 제어는 Ctrl 조합 다섯 개로 정리됩니다. Ctrl+C는 생성을 멈춥니다. 클로드가 엉뚱한 방향으로 코드를 짜기 시작하거나, 파일을 잘못 읽고 있거나, 응답이 길어지는데 이미 필요한 답을 얻었을 때 바로 누릅니다. 누르자마자 중단되고, 모델이 세는 글 조각 낭비가 그 지점에서 끊깁니다. Ctrl+D는 세션을 종료합니다. Ctrl+L은 화면을 깨끗이 정리합니다.

스크롤을 내리며 이전 출력을 넘기느니 한 번 지우고 새로 시작하는 편이 빠릅니다. Ctrl+O는 대화 기록 뷰어를 엽니다. 이전에 나눈 대화를 다시 보고 싶을 때, 또는 클로드가 어느 단계에서 어떤 결정을 했는지 복기할 때 씁니다.

v2.1 계열에서는 Ctrl+O가 일반 상세 전사(verbose transcript) 토글로 역할이 좁혀졌고, 포커스 뷰는 별도로 /focus 명령어에 넘어갔습니다. Ctrl+B는 백그라운드 실행입니다. 긴 작업을 뒤에서 돌리고 다른 일을 이어갈 때 유용합니다.

모드 전환은 Shift+Tab과 Alt 조합 세 개로 나뉩니다. Shift+Tab은 일반 → 자동승인 → 계획 모드를 순환합니다. 코드를 빠르게 짜야 할 때는 자동승인으로 두고, 설계가 중요한 작업 앞에서는 계획 모드로 먼저 전환한 뒤 클로드에게 계획서만 쓰게 합니다. Alt+P는 모델 변경입니다.

Haiku와 Sonnet, Opus 사이를 즉시 전환할 수 있어, 가벼운 질문은 Haiku로 비용을 아끼고 복잡한 리팩터링은 Opus로 올립니다. Alt+T는 심층추론(Thinking) 토글입니다. 복잡한 판단이 필요한 순간 켰다가, 단순한 파일 수정에는 꺼둡니다. Alt+O는 빠른응답 모드를 켜고 끕니다.

특수 접두사는 세 개뿐입니다. 슬래시(/)는 내장된 슬래시 명령어를 호출합니다. 느낌표(!)는 터미널 명령을 직접 실행합니다. 코드에서 "!ls -la"를 입력하면 파일 목록이 바로 나옵니다. 골뱅이(@)는 파일을 불러와 대화에 포함시킵니다. "@src/auth.py 이 파일을 리팩터링해줘"처럼 씁니다. 이 세 접두사가 거의 모든 고급 상호작용의 입구입니다.

여러 줄 입력은 백슬래시+Enter, 또는 Ctrl+J입니다. 긴 지시를 입력할 때 Enter를 누르면 바로 전송되는 기본 동작을 피할 수 있습니다.

세션 선택기(Session Selector)에는 별도의 키가 배정됩니다. 위아래 방향키로 목록을 이동하고, P로 미리보기, R로 이름 변경, 슬래시로 검색, B로 현재 브랜치만 필터링합니다. 이 화면은 claude --resume을 실행하거나 /resume을 쳤을 때 뜹니다.

대화 기록 뷰어에서는 Ctrl+O로 뷰어를 열고, 슬래시로 검색(v2.1 계열에서 새로 추가), N이나 Shift+N으로 검색 결과 사이를 이동, Ctrl+E로 전체 펼치기, Q나 Esc로 닫기입니다.

단축키는 외우려고 하면 끝이 없어 보이지만, 실제로는 Ctrl+C, Ctrl+L, Shift+Tab, Alt+T 네 개가 전체의 90%입니다. 나머지는 막히는 순간에 떠올려서 쓰시면 됩니다.

명령줄 방식 명령어와 플래그

터미널에서 "claude"라고 치면 대화 모드가 시작됩니다. 하지만 이 명령어 뒤에 무엇을 붙이느냐에 따라 완전히 다른 도구가 됩니다.

핵심 명령어는 다섯 개입니다.

claude # 대화 모드 시작

claude "질문" # 바로 질문하며 시작

claude -p "질문" # 비대화형 실행 (자동화용)

claude -c # 마지막 대화 이어서

claude -r "세션명" # 이름으로 세션 열기

claude update # 최신 버전으로 업데이트

이 중에서 일상적으로 쓰는 것은 "claude"와 "claude -c" 두 가지입니다. 자동화 파이프라인을 만들 때는 "claude -p"가 중심이 됩니다. -p는 print의 약자로, 대화형 UI를 띄우지 않고 결과만 출력하고 종료합니다. 쉘 스크립트에 끼워 넣거나, GitHub Actions 같은 CI에서 부를 때 필수입니다.

주요 플래그는 두 묶음으로 나뉩니다.

기본 플래그는 여덟 개 정도가 실무에서 자주 쓰입니다.

--model # 모델 지정 (haiku/sonnet/opus)

-w # Git 워크트리 생성

-n, --name # 세션 이름 지정

--add-dir # 작업 폴더 추가

--agent # 에이전트 지정 (Explore/Plan/General/Bash)

--allowedTools # 도구 사전 승인

--output-format json|stream # 출력 형식 (자동화용)

--json-schema # 구조화 출력 스키마

--max-turns # 턴 수 제한

--max-budget-usd # 비용 상한 ($)

--allowedTools는 자동화 스크립트에서 중요한 플래그입니다. 인터랙티브 세션에서는 도구를 쓸 때마다 사용자에게 승인을 묻지만, 스크립트 실행 중에는 그럴 사람이 없습니다. 이 플래그에 "Read,Edit,Bash" 같이 나열해두면 해당 도구들에 대해서는 묻지 않고 바로 실행합니다. --max-turns와 --max-budget-usd는 안전장치입니다. 클로드가 무한 루프에 빠지거나 예상보다 많은 토큰을 쓰는 것을 원천 차단합니다.

고급 플래그는 v2.1 계열에서 대거 추가되었습니다.

--console # Anthropic 콘솔 인증

--verbose # 상세 로그

--bare # 최소 실행 (훅/LSP/스킬 없음, NEW)

--channels # 권한 릴레이 (NEW, 리서치 프리뷰)

--remote # 웹 세션으로 실행

--effort # low/mid/high/max/auto

--permission-mode plan|default # 권한 모드

--dangerously-skip-permissions # 권한 질문 생략 (주의)

--chrome # Chrome 연동

--bare는 2026년 3월 v2.1.81에서 추가된 플래그로, 스크립트 환경에 최적화된 최소 실행 모드입니다. 훅, LSP, 플러그인 동기화, 스킬 디렉터리 탐색을 모두 건너뛰기 때문에 시동 시간이 빠릅니다. 대신 OAuth와 키체인 인증은 비활성화되고, ANTHROPIC_API_KEY 환경변수나 --settings를 통한 apiKeyHelper가 반드시 필요합니다. 자동 메모리도 완전히 꺼집니다. CI 환경에서 "팀원 컴퓨터에 깔린 훅이 내 빌드에 끼어드는" 문제를 막는 용도로 씁니다.

--channels는 같은 시기에 리서치 프리뷰로 들어왔습니다. 채널 서버가 권한 요청을 사용자 휴대폰으로 전달할 수 있게 해주는 기능입니다. 서버에서 자동화가 돌다가 민감한 작업이 필요해지면, 개발자의 폰에서 승인 알림이 뜨고 거기서 결정을 내립니다. 노트북을 닫고 외출한 상태에서도 장시간 실행 세션을 관리할 수 있게 됩니다.

--dangerously-skip-permissions는 이름 그대로 위험한 플래그입니다. 모든 권한 질문을 건너뛰고 클로드가 원하는 대로 실행합니다. 도커 컨테이너나 VM 같은 샌드박스 안에서만 쓰는 것이 원칙이고, 로컬 맥에 직접 실행하면 rm -rf 같은 명령도 막을 방법이 없어집니다.

--effort는 클로드의 사고 노력 수준을 조절합니다. 기본값 auto는 작업 복잡도를 보고 스스로 결정하지만, "low"를 주면 빠르고 저렴하게, "max"를 주면 가장 느리지만 가장 깊게 생각합니다. 비용 민감한 배치 작업에는 low, 중요한 리팩터링에는 high나 max를 지정합니다.

실전 조합 예시 세 가지를 보면 감이 잡힙니다.

1. 오류 분석: 빠른 모델 + JSON 출력

claude -p "오류 분석해줘" --model claude-haiku-4-5 --output-format json

2. 계획 모드로 src 폴더 분석

claude --permission-mode plan --add-dir ./src

3. 로그 파일을 파이프로 넘겨 원인 분석, 1달러 한도

cat error.log | claude -p "원인 찾아줘" --max-budget-usd 1

세 번째 예시는 특히 재밌습니다. 유닉스 파이프 한 줄로 로그 분석 결과를 받아볼 수 있고, 1달러 안에서 답을 못 내면 그냥 멈춥니다. 이런 조합이 가능하다는 것이 명령줄 방식 도구의 위력입니다.

슬래시 명령어 완전 가이드

세션 안에서 슬래시(/)를 치면 내장 명령어 목록이 뜹니다. 슬래시만 쳐도 자동완성 목록이 내려오므로, 이름을 외우지 못해도 찾을 수 있습니다. 전체 명령어는 네 범주로 나뉩니다.

세션 관리는 여덟 개 명령어로 구성됩니다.

/clear # 대화 초기화 (컨텍스트 비우기)

/compact # 대화 압축 (컨텍스트 60% 이상 찼을 때)

/resume # 이전 세션 이어하기

/rename # 세션 이름 지정

/branch # 대화 분기

/cost # 모델이 세는 글 조각 비용 확인

/rewind # 이전 대화 지점으로 되돌리기

/export # 내보내기

/clear는 작업이 바뀔 때 씁니다. 회의록을 정리하다가 이제 코드 리뷰를 시작한다면, 이전 대화 내용이 남아 있으면 방해가 됩니다. /clear로 초기화하고 새로 시작합니다. /compact는 한 세션에서 오래 작업할 때 쓰는 압축 기능입니다. 컨텍스트가 가득 찰 즈음에 자동으로 작동하기도 하지만, 명시적으로 불러서 "이 정보만 남기고 압축해줘"라고 지시할 수 있습니다.

/rewind는 잘못된 방향으로 빠진 대화를 이전 지점으로 되돌립니다. /branch는 하나의 대화에서 여러 갈래로 나눠 실험할 때 유용합니다. "이 방식으로 짜봤다가 마음에 안 들면 원래대로 돌아가는" 실험적 탐색에 씁니다.

설정 관련 명령어는 여덟 개입니다.

/config # 설정 화면 열기

/model # 모델 전환

/fast # 빠른모드 ON/OFF

/effort # 품질(노력) 조절

/theme # 테마 변경

/permissions # 권한 관리

/keybindings # 단축키 설정

/effort는 명령줄 방식의 --effort와 같은 역할을 세션 안에서 하는 명령어입니다. 현재 작업의 난이도에 맞춰 노력 수준을 바꿉니다. /permissions는 권한 관리 UI입니다. 어떤 도구는 허용하고 어떤 명령은 항상 물어보게 할지, 어떤 명령은 절대 허용하지 않을지를 세밀하게 설정합니다.

"git commit은 자동 허용, rm -rf는 거부, 네트워크 요청은 매번 물어보기" 같은 규칙을 이곳에서 만듭니다.

도구 관리 명령어는 여덟 개입니다.

/init # CLAUDE.md 생성

/memory # 지침서 편집

/mcp # MCP 연결 서버 관리

/hooks # 훅 설정

/skills # 스킬 목록 보기

/agents # 에이전트 관리

/add-dir # 작업 폴더 추가

/init은 새 프로젝트를 시작할 때 첫 번째로 치는 명령어입니다. 클로드가 현재 폴더의 구조를 분석해서 CLAUDE.md 초안을 자동으로 생성합니다. 이 파일이 프로젝트의 지침서가 되고, 이후 세션에서 매번 자동으로 읽힙니다. /memory는 이 지침서를 편집하는 인터페이스입니다.

"이 프로젝트에서는 Python 3.12를 쓴다", "PEP 8을 엄격히 지킨다", "검증 커버리지 80% 이상 유지" 같은 규칙을 여기에 적어둡니다.

/mcp는 MCP 연결 서버 관리 화면입니다. 어떤 서버가 연결되어 있고, 각 서버가 어떤 도구를 제공하는지 한눈에 보여줍니다. 문제가 생긴 서버를 재시작하거나, 새로 추가하거나, 잠시 비활성화하는 작업을 이곳에서 합니다. /hooks는 훅 설정 UI입니다. 특정 이벤트(파일 수정 전, 도구 사용 전, 세션 종료 시 등)에 자동으로 실행할 쉘 명령을 걸어둡니다.

특수 명령어 범주에는 비교적 새로 들어온 것들이 많습니다.

/btw # 컨텍스트 소모 없이 질문하기

/plan # 계획 모드

/loop # 반복 예약 실행

/voice # 음성 모드

/doctor # 진단

/remote-control # 원격 제어 (NEW)

/schedule # 클라우드 예약 (Routines)

/security-review # 보안 검토

/btw는 "by the way"의 약자로, 본 작업의 컨텍스트를 소모하지 않고 짧은 질문을 던질 때 씁니다. "참, PostgreSQL과 MySQL 중 이 경우에 뭐가 나아?"처럼 옆길 질문을 해야 할 때, /btw로 물으면 현재 진행 중인 코드 작업 맥락이 보존됩니다. /plan은 Shift+Tab으로 들어가는 계획 모드와 같은 기능입니다.

/loop은 세션 안에서 반복 작업을 예약합니다. "/loop 5m CI 상태 확인"이라고 치면 5분마다 CI 상태를 확인하고 바뀐 것이 있으면 알려줍니다. /voice는 음성 모드입니다. 스페이스바를 누른 채 말하면 자동으로 텍스트로 변환되어 명령이 됩니다. 20개 언어를 지원하고 한국어도 포함됩니다.

/doctor는 환경 진단 명령어입니다. 노드 버전이 맞는지, API 키가 설정되었는지, MCP 연결 서버가 응답하는지, 필요한 권한이 있는지를 한꺼번에 확인해줍니다. 처음 설치하고 뭔가 안 될 때 제일 먼저 쳐보는 명령어입니다.

/schedule은 2026년 4월 14일 출시된 Routines를 제어합니다. 내 컴퓨터 /loop이 세션 안에서만 도는 것과 달리, /schedule로 만든 작업은 Anthropic 클라우드에서 돌기 때문에 노트북을 닫아도 실행됩니다. 매일 아침 7시 브리핑, 매주 월요일 9시 주간 보고, 매달 1일 비용 정산 같은 루틴을 여기에 올립니다.

/security-review는 코드베이스 전체를 훑어 보안 취약점을 찾는 명령어입니다. OWASP Top 10 기준으로 SQL 인젝션, XSS, 하드코딩된 비밀 같은 문제를 색출합니다. 릴리즈 전에 한 번 돌리는 습관을 들이면 큰 사고를 예방할 수 있습니다.

스킬과 에이전트 생태계

슬래시 명령어가 내장 기능이라면, 스킬은 사용자가 직접 만드는 기능입니다. 같은 지시를 반복해서 내릴 바에는 아예 스킬로 저장해두고 한 번에 호출하는 편이 낫습니다. 스킬과 슬래시 명령어는 v2.1 계열에서 통합되어 같은 엔진으로 작동합니다. .claude/commands/deploy.md 파일과 .claude/skills/deploy/SKILL.md 파일 둘 다 /deploy 명령어를 만듭니다.

내장 스킬부터 봅니다. 클로드 코드 설치 직후부터 쓸 수 있는 다섯 개가 있습니다.

/simplify # 코드 리뷰 (AI 3개 병렬)

/batch # 대량 변경 (5에서 30개 워크트리)

/debug # 로그 분석

/loop # 반복 예약 실행

/claude-api # Claude API/SDK 참조 로드

/simplify는 코드 리뷰에 특화된 스킬입니다. 세 개의 에이전트가 병렬로 돌면서 코드를 각기 다른 관점(품질, 보안, 가독성)에서 검토합니다. /batch는 워크트리 기능과 결합되어, 5에서 30개의 독립된 작업 공간에서 동시에 변경 작업을 수행합니다. 라이브러리 전체에 타입 힌트를 달거나, 여러 파일에서 일관된 리팩터링을 할 때 씁니다.

/debug는 로그 파일을 통째로 읽고 오류 원인을 찾아냅니다. /claude-api는 Claude API와 SDK 공식 문서를 컨텍스트로 끌어와 API 관련 작업을 할 때 참고하게 만듭니다.

스킬 저장 위치는 두 곳입니다.

.claude/skills/〈이름〉/ # 프로젝트 스킬 (팀 공유)

~/.claude/skills/〈이름〉/ # 개인 스킬 (모든 프로젝트에 적용)

프로젝트 폴더에 둔 스킬은 git에 커밋되어 팀 전체가 공유합니다. 홈 디렉터리에 둔 스킬은 해당 사용자만 쓰는 전역 스킬입니다. 회사 공통 배포 규칙은 프로젝트 스킬로, 개인 메모 정리 루틴은 개인 스킬로 관리하는 구분이 자연스럽습니다.

SKILL.md 파일의 구조는 YAML 프론트매터와 마크다운 본문으로 나뉩니다. 프론트매터의 필드를 하나씩 봅니다.

name: deploy

description: 프로덕션 배포. 사용자가 배포를 요청하거나 deploy를 언급할 때 사용.

allowed-tools: [Bash, Edit]

model: claude-haiku-4-5

effort: low

paths: [src/**]

context: fork

disable-model-invocation: true

name은 스킬의 식별자입니다. 소문자와 하이픈만 쓸 수 있고 최대 64자입니다. description은 클로드가 언제 이 스킬을 불러올지 판단하는 근거가 됩니다. 가장 중요한 필드입니다. "무엇을 하고, 언제 써야 하는지"를 구체적으로 적습니다. "코드를 도와줍니다" 같은 모호한 설명은 스킬이 발동되지 않거나 엉뚱한 곳에서 발동됩니다.

allowed-tools는 이 스킬이 실행될 때 클로드가 쓸 수 있는 도구를 제한합니다. 읽기 전용 스킬에는 [Read, Grep, Glob]을, 배포 스킬에는 [Bash, Edit]을 지정합니다. 이 제한이 안전장치 역할을 합니다. model은 스킬 실행에 쓸 모델을 강제합니다. 복잡한 분석이면 claude-opus-4-6을, 단순 반복이면 claude-haiku-4-5를 지정합니다.

effort는 v2.1 계열에서 스킬에 추가된 필드입니다. low, med, high, max 중 하나를 지정하면, 이 스킬이 발동될 때 클로드의 사고 노력 수준이 그 값으로 설정됩니다. paths는 이 스킬이 어떤 파일 패턴에 대해 활성화될지를 정합니다. [src/**]로 지정하면 src 폴더 안 파일을 다룰 때만 이 스킬이 후보에 올라옵니다.

context: fork는 중요한 옵션입니다. 이 스킬을 보조 에이전트의 독립 컨텍스트에서 실행하라는 뜻입니다. 대량의 탐색 결과가 메인 대화로 쏟아져 들어오는 것을 막아줍니다. 책 한 권 분량의 코드베이스를 훑어야 하는 스킬은 대부분 context: fork로 돌리는 편이 안전합니다.

disable-model-invocation: true는 이 스킬이 클로드에 의해 자동 호출되지 못하게 막습니다. 사용자가 명시적으로 /deploy라고 쳤을 때만 실행됩니다. 배포, 커밋, 외부 메시지 발송처럼 결과가 되돌리기 어려운 작업에는 반드시 걸어둬야 할 안전장치입니다.

본문의 마크다운에서는 특수 문법 세 가지가 쓰입니다.

$ARGUMENTS # 사용자가 스킬 호출 시 넘긴 인수

${CLAUDE_SKILL_DIR} # 스킬 폴더 경로

!

cmd

동적 컨텍스트 주입 (쉘 명령 실행 후 결과를 문맥에 포함)

$ARGUMENTS를 쓰면 스킬이 함수처럼 동작합니다. /analyze-file src/auth.js라고 치면 src/auth.js가 $ARGUMENTS에 담겨 스킬 본문에 삽입됩니다. ${CLAUDE_SKILL_DIR}은 스킬 폴더의 절대 경로를 돌려줍니다. 이 경로를 이용해 스킬과 함께 배포된 스크립트를 실행합니다.

!

date

처럼 느낌표+백틱 구문은 쉘 명령을 스킬 로드 시점에 실행해 결과를 컨텍스트에 주입합니다. 현재 날짜, 브랜치 이름, 최근 커밋 해시 같은 동적 정보를 스킬에 넣을 때 씁니다.

내장 에이전트는 네 종류입니다.

Explore # 빠른 탐색 전용 (Haiku 기반, 읽기 전용 도구에 최적화)

Plan # 계획 모드 조사용

General # 범용 (전체 도구 접근)

Bash # 터미널 독립 실행

Explore는 가볍고 빠른 에이전트입니다. 코드베이스 구조 파악, "이 함수가 어디서 호출되는지" 같은 조사에 씁니다. Haiku 모델을 쓰기 때문에 비용이 적고 속도가 빠릅니다. Plan은 Plan Mode 안에서 활동합니다. General은 모든 도구에 접근 가능한 범용 에이전트입니다. Bash는 터미널 작업을 독립된 프로세스에서 실행하는 에이전트로, 긴 구축나 검증를 백그라운드로 돌릴 때 유용합니다.

에이전트 프론트매터는 스킬보다 옵션이 많습니다.

permissionMode: default/plan/bypass

isolation: worktree # 독립 워크트리에서 실행

memory: user|project # 기억 저장 범위

background: true # 백그라운드 실행

maxTurns: 10 # 행동 횟수 제한

initialPrompt: ...

자동 첫 질문 (NEW)

SendMessage # 에이전트 재개 (resume 대체, NEW)

initialPrompt는 v2.1 계열의 신기능입니다. 에이전트가 생성되자마자 자동으로 이 프롬프트를 실행합니다. "현재 PR의 변경사항을 요약해라" 같은 초기 지시를 걸어두면, 사용자가 세션에 들어오자마자 결과가 준비되어 있습니다. SendMessage도 신규 기능으로, 기존의 resume을 대체합니다. 일시 중단된 에이전트에게 새 메시지를 보내 작업을 재개시킵니다.

고수 워크플로우

앞의 다섯 섹션이 부품 설명이었다면, 이번 섹션은 부품을 조립해서 쓰는 방법입니다. 여덟 개의 핵심 워크플로우가 서로 맞물려 클로드 코드의 진짜 속도를 만듭니다.

계획 모드 우선. 큰 규모의 변경 전에 반드시 통과해야 할 관문입니다. Shift+Tab으로 계획 모드에 진입하면 클로드는 코드를 한 줄도 수정하지 않고 계획서만 작성합니다. 사용자는 이 계획을 검토하고 승인한 뒤에야 실제 수정이 진행됩니다. "데이터베이스 스키마를 바꾸자"는 지시에 클로드가 곧장 코드를 수정하기 시작하면 사고가 큽니다.

계획 모드에서 영향 범위를 미리 보고, 위험한 부분을 지적하고 넘어가는 과정이 있어야 합니다.

생각과 노력량 조절. Alt+T로 심층추론을 켜고 끕니다. 단순한 파일 조작에는 꺼두고, 아키텍처 결정이나 미묘한 버그 추적에는 켭니다. 한 번만 최강 모드로 돌리고 싶으면 입력창에 "ultrathink"라는 단어를 타이핑합니다. 그 턴 한 번만 최고 사고 수준으로 동작합니다. /effort 명령어를 쓰면 low, med, high, max 네 단계로 전체 세션의 기본 노력 수준을 바꿀 수 있습니다.

Git 워크트리. 개발자가 여러 브랜치를 동시에 건드릴 때 겹치지 않는 작업 공간을 주는 기능입니다. claude 명령에 -w 플래그를 붙이면 독립된 워크트리에서 세션이 시작됩니다. /batch로 5에서 30개 작업을 병렬로 돌릴 때, 각 작업이 자기 워크트리에서 자기 변경만 다루므로 충돌이 없습니다. 라이브러리 전체에 걸친 일괄 수정, 여러 PR을 동시에 진행하는 작업에서 위력이 큽니다.

claude -w feature-login # 워크트리 생성 후 시작

/batch 5 "각 워커에 할당된 모듈 리팩터링"

음성 모드. /voice를 치면 음성 입력 모드로 들어갑니다. 스페이스바를 누른 채 말하면 자동으로 텍스트로 변환됩니다. 한국어를 포함한 20개 언어를 지원합니다. 긴 지시를 타이핑하기 귀찮을 때, 산책하며 구상한 설계를 쏟아내고 싶을 때 유용합니다.

컨텍스트 관리. /context를 치면 현재 대화가 컨텍스트 창의 몇 퍼센트를 차지하는지 보여줍니다. 60에서 70%에 도달하면 /compact로 압축할 때입니다. 95%에 도달하면 자동으로 압축되지만, 그 전에 명시적으로 압축하면 어떤 정보를 남길지 지시할 수 있어 정확도가 높습니다. Opus 4.6은 Max 구독에서 1M 모델이 세는 글 조각 컨텍스트를 지원합니다.

수십만 줄의 코드베이스를 한 번에 올려놓고 분석할 수 있는 분량입니다.

/context # 현재 사용량 확인

/compact "핵심 결정 사항과 현재 작업 파일만 남겨줘"

세션 활용. claude -c는 마지막 대화를 이어서 시작합니다. 퇴근할 때 세션을 그냥 종료해도, 다음 날 아침 -c 한 번이면 어제 작업이 그대로 이어집니다. claude -r "세션명"은 이름으로 저장해둔 세션을 불러옵니다. 여러 프로젝트를 오가며 일할 때 세션에 이름을 붙여두면 혼란이 줄어듭니다. /btw는 현재 컨텍스트를 건드리지 않고 짧은 질문을 던지는 용도입니다.

자동화와 SDK. claude -p "질문"이 자동화의 입구입니다. 대화형 UI를 띄우지 않고 결과만 반환합니다. --output-format json을 붙이면 JSON으로 결과가 돌아오므로, 다른 프로그램에서 파싱하기 쉽습니다. 유닉스 파이프와도 잘 맞습니다.

tail -200 app.log | claude -p "Slack에 이상 징후 알림"

git diff main --name-only | claude -p "변경 파일 보안 검토"

예약과 원격 제어. /loop 30m은 30분마다 지시를 반복 실행합니다. 세션이 열려 있는 동안만 돌아갑니다. /schedule은 Routines를 만들어 Anthropic 클라우드에서 실행합니다. 노트북이 꺼져 있어도 돌아가는 진짜 자동화입니다. /rc는 claude.ai 웹과 세션을 연결해 모바일에서도 제어할 수 있게 해줍니다. --remote 플래그는 아예 웹 세션 안에서 작업이 실행되게 만듭니다.

이 여섯 섹션이 v2.1.86 기준 클로드 코드의 전체 그림입니다. 매일 쓰는 기능은 Ctrl+C, Ctrl+L, /clear, Shift+Tab, claude -c 다섯 가지로 줄어들고, 자동화 파이프라인을 구축할 때는 claude -p와 --output-format, MCP 연결 기능, Routines가 중심이 됩니다.

그 사이의 수십 가지 명령어와 플래그는, 특정 순간에 필요해지는 순간 이 장으로 돌아와 찾으시면 됩니다.

버전은 빠르게 바뀝니다. 2026년 3월에 --bare와 --channels가, 4월에 Routines가 들어왔듯이, 매달 새 기능이 추가됩니다. claude update를 가끔 돌려서 최신 버전을 유지하고, /doctor로 환경 상태를 확인하는 습관만 있으면 뒤처지지 않습니다.

이 책에서 다루지 못한 새 기능은 Claude Code 공식 체인지로그 (code.claude.com/docs/en/changelog)에서 바로 확인할 수 있습니다.

전체 목차

책의 모든 장 보기

인공지능 전문가 김경진 변호사

이 책이 잠시라도 당신 곁에 머물렀다면, 다음 이야기가 세상에 나올 수 있도록 후원해 주세요.

이 책을 잘 읽으셨으면 그리고 새로운 가치있는 지식을 얻으셨다고 판단되시면
농협 302-1096-0948-81 (예금주 김경진) 에 자발적 후원 부탁드립니다.