AI에게 맡기고 자리를 뜨다

욜로 모드 완전 입문

매우 초보자를 위한 AI AGENT MODE 활용
법
yolo모드 활용법

김경진 지음

2026

1부 욜로 모드라는 말의 정체

1장. 새벽 두 시, 노트북 앞의 회사원

화면의 불빛이 얼굴을 비추고 있었습니다. 새벽 두 시, 서울 마포구의 한 오피스텔. 김대리는 내일 아침 아홉 시까지 제출해야 하는 분기 보고서를 노려보고 있었습니다. 엑셀 파일이 세 개, 파워포인트 슬라이드가 스물두 장, 거래처별 매출 데이터를 정리한 CSV 파일이 다섯 개. 이것들을 하나의 보고서로 엮어야 했습니다.

커피잔은 이미 세 번째였습니다.

김대리는 평소에 챗GPT(ChatGPT, 챗지피티)를 씁니다. 회의록 요약, 이메일 초안, 간단한 번역. 그런 일에는 꽤 쓸 만했습니다. 그런데 이 보고서 작업은 차원이 달랐습니다. 파일을 열고, 데이터를 읽고, 계산하고, 차트를 만들고, 슬라이드에 붙이는 작업이 필요했거든요. 챗GPT 창에 "보고서 만들어줘"라고 치면 그럴듯한 글은 나오지만, 실제 엑셀 파일을 열어서 숫자를 뽑아주지는 못합니다.

그날 밤, 김대리는 한 가지를 시도했습니다.

터미널(Terminal, 터미널)이라는 검은 화면을 열고, AI에게 이렇게 말했습니다. "이 폴더에 있는 엑셀 파일 세 개와 CSV 파일 다섯 개를 읽어서, 거래처별 분기 매출 요약표를 만들고, 차트를 포함한 파워포인트 보고서를 생성해줘." 그리고 엔터(Enter)를 눌

렀습니다. AI가 움직이기 시작했습니다. 파일을 열고, 코드를 쓰고, 표를 만들고, 차트를 그렸습니다. 김대리는 그 과정을 지켜보다가, 어느 순간 노트북을 덮고 잠자리에 들었습니다.

아침 일곱 시에 눈을 떴을 때, 폴더 안에 완성된 파워포인트 파일이 들어 있었습니다. 차트도 들어가 있고, 숫자도 맞았습니다. 김대리가 한 일이라고는 "이것을 해줘"라고 지시한 것뿐이었습니다.

이 이야기가 이 책의 출발점입니다.

"욜로"라는 말을 들어보셨을 겁니다. YOLO. You Only Live Once. "인생은 한 번뿐이다." 2010년대 초반, 래퍼 드레이크(Drake)의 노래 제목으로 유명해진 이 말은 한때 젊은 세대의 소비 철학을 상징했습니다. 내일 걱정하지 말고, 오늘을 살자. 저축보다 경험. 안전보다 모험.

그런데 이 단어가 2025년부터 프로그래머들 사이에서 전혀 다른 의미로 쓰이기 시작했습니다. AI 도구에 붙는 "욜로 모드(YOLO Mode)"라는 용어가 생긴 겁니다. AI에게 일을 시키되, 중간중간 "이거 해도 될까요?"라고 묻는 과정을 전부 생략하고, 처음부터 끝까지 알아서 하게 놔두는 모드. 허락을 구하지 않고 달리는 모드. 인생이 한 번뿐이듯, AI도 한 번에 쭉 달리는 겁니다.

이 책은 그 "욜로 모드"를 다룹니다.

이 책이 다루는 것은 명확합니다. 클로드 코드(Claude Code)와 코덱스(Codex)라는 두 AI 도구의 욜로 모드를 켜고, 쓰고, 끄는 법입니다. 코딩을 전혀 모르는 분도 따라할 수 있게 썼습니다. 터미널이 뭔지 모르셔도 괜찮습니다. 이 책에서 알려드립니다.

이 책이 다루지 않는 것도 말씀드려야겠습니다. 프로그래밍 교재가 아닙니다. 파이썬(Python, 파이썬)이나 자바스크립트(JavaScript, 자바스크립트) 문법을 가르치지 않습니다. AI 이론서도 아닙니다. 머신러닝(Machine Learning, 머신러닝)이 어떻게 작동하는지 설명하지 않습니다. 이 책은 도구 사용법 안내서입니다. 드릴의 원리를 몰라도 벽에 구멍을 뚫을 수 있듯이, AI의 원리를 몰라도 AI에게 일을 시킬 수 있습니다. 그 방법을 알려드리겠습니다.

2장. AI가 혼자 일한다는 게 도대체 무슨 말인가

여러분이 식당에서 주문하는 장면을 떠올려 보십시오.

보통의 AI 도구는 이런 식입니다. 여러분이 "김치찌개 하나요"라고 하면, 종업원이 "김치찌개 하나, 맞으시죠?"라고 되물어봅니다. "네." "밥은 같이 드릴까요?" "네." "반찬은 리필 해드릴까요?" "네, 부탁해요." 한 단계 한 단계, 확인을 받으면서 진행합니다. 챗GPT에 "보고서 초안 만들어줘"라고 하면, "어떤 주제의 보고서인가요?"라고 물어보죠. 답을 하면 "분량은 어느 정도로 할까요?"라고 다시 묻습니다. 매번 허락을 받습니다. 안전하지만, 느립니다. 자리를 비울 수가 없습니다.

욜로 모드는 다릅니다. "김치찌개 하나, 밥이랑 반찬 알아서 세팅해주시고, 후식도 적당한 걸로 골라서 가져다주세요"라고 한마디 하고, 자리에 앉아서 책을 읽는 겁니다. 종업원이 알아서 움직입니다. 공기밥을 따로 드릴까요, 반찬은 구체적으로 뭘 드릴까요, 물은 시원한 물인가요 따뜻한 물인가요, 이런 세세한 것을 묻지 않습니다. 알아서 판단하고 진행합니다.

더 정확하게 비교해 보겠습니다.

코파일럿(Copilot, 코파일럿) 같은 AI 코딩 도구는 여러분이 코드를 쓰는 동안 옆에서 다음 줄을 제안해줍니다. 글을 쓸 때 자동완성이 뜨는 것과 비슷합니다. 여러분이 손을 움직여야 합니다. AI는 제안만 하고, 결정은 사람이 합니다.

챗GPT는 대화형입니다. 여러분이 질문하면, AI가 답합니다. 다시 질문하면, 다시 답합니다. 탁구처럼 공이 오갑니다. AI가 여러분의 컴퓨터에서 파일을 만들거나 프로그램을 실행하지는 못합니다. 대화창 안에서만 움직입니다.

클로드 코드와 코덱스의 욜로 모드는 이것과 근본적으로 다릅니다. AI가 여러분의 컴퓨터에서 직접 파일을 만들고, 코드를 쓰고, 프로그램을 실행합니다. 폴더를 열어 내용을 읽고, 새 파일을 만들고, 기존 파일을 고칩니다. 그리고 이 모든 일을 중간에 "해도 될까요?"라고 묻지 않고 합니다.

"허락을 구하는 AI"와 "허락 없이 달리는 AI." 이 둘의 차이가 욜로 모드의 핵심입니다.

클로드 코드(Claude Code)는 앤스로픽(Anthropic, 앤스로픽)이라는 회사가 만든 AI 도구입니다. 터미널에서 작동합니다. 기본 상태에서는 파일을 만들거나 명령을 실행할 때마다 여러분에게 "이거 해도 되나요?"라고 물어봅니다. 안전장치가 켜져 있는 상태입니다. 이 안전장치를 끄는 명령어가 바로 --dangerously-skip-permissions(댄저러슬리 스킵 퍼미션스)입니다. "위험하게 허락 과정을 건너뛴다"는 뜻입니다. 이렇게 켜면, 클로드 코드는 질문 없이 일을 시작합니다.

코덱스(Codex)는 오픈AI(OpenAI, 오픈에이아이)가 만든 비슷한 도구입니다. 여기서는 --yolo(욜로)라는 짧은 명령어로 같은 효과를 냅니다. 이름부터 "인생은 한 번뿐이다"입니다.

2026년 3월, 클로드 코드에 "오토 모드(Auto Mode)"라는 새로운 옵션이 추가됐습니다. 이것은 완전한 욜로 모드와 기본 모드 사이의 중간 지점입니다. 위험해 보이는 명령(파일 삭제, 시스템 설정 변경 같은 것)은 여전히 허락을 구하지만, 안전한 명령(파일 읽기, 텍스트 검색 같은 것)은 자동으로 실행합니다. 수동 기어와 자동 기어 사이에 반자동 기어가 생긴 셈이라고 생각하시면 됩니다.

이 책에서는 세 가지를 모두 다룹니다. 하지만 중심은 완전한 욜로 모드, 즉 AI가 스스로 판단하고 실행하는 상태입니다.

3장. 왜 이름에 "위험하게"가 붙어 있을까

클로드 코드의 욜로 모드를 켜는 명령어를 다시 한번 보겠습니다.

claude --dangerously-skip-permissions

"dangerously"라는 단어가 눈에 들어옵니다. 위험하게. 왜 소프트웨어 회사가 자기 제품의 명령어에 "위험하게"라는 단어를 넣었을까요.

이유는 간결합니다. 사용자에게 경고하기 위해서입니다.

부엌칼을 생각해 보십시오. 칼은 위험합니다. 손을 벨 수 있습니다. 그런데 칼 없이는 요리를 할 수 없습니다. 칼이 위험하다고 해서 칼을 안 쓸 수는 없습니다. 대신 우리는 칼 쓰는 법을 배웁니다. 칼날이 몸 바깥을 향하게 잡는다. 도마 위에서만 자른다. 쓰고 나면 칼집에 넣는다. 이런 규칙을 지키면 칼은 도구가 됩니다. 규칙을 무시하면 흉기가 됩니다.

욜로 모드도 같습니다.

AI가 허락 없이 움직인다는 건, 여러분의 컴퓨터에서 AI가 자유롭게 파일을 만들고, 고치고, 삭제할 수 있다는 뜻입니다. 잘 쓰면, 여러분이 자는 동안 보고서가 완성됩니다. 잘못 쓰면, 여러분이 자는 동안 중요한 파일이 사라질 수 있습니다. 앤스로픽이 명

령어에 "dangerously"를 넣은 건, 이 사실을 여러분이 인지한 상태에서 결정하라는 뜻입니다. "나는 이 위험을 알고 있고, 그래도 쓰겠다"는 의사 표시를 명령어 자체에 담게 한 겁니다.

코덱스의 원래 명령어는 더 노골적이었습니다. --dangerously-bypass-approvals-and-sandbox(댄저러슬리 바이패스 어프루벌스 앤드 샌드박스). "위험하게 승인과 안전 울타리를 우회한다." 너무 길어서 --yolo라는 별칭이 생겼지만, 원래 이름이 담고 있는 경고의 무게는 가볍지 않습니다.

> [경고] 욜로 모드는 AI에게 여러분 컴퓨터의 파일 시스템에 대한 자유로운 접근 권한을 줍니다. 중요한 파일이 있는 폴더에서는 반드시 백업을 먼저 해두십시오.

칼을 살 때, 좋은 칼 가게는 칼과 함께 칼집을 줍니다. 이 책이 바로 그 칼집입니다.

오늘 손에 익힌 것

욜로 모드란 AI가 중간에 허락을 구하지 않고 처음부터 끝까지 스스로 작업을 수행하는 모드입니다. 클로드 코드에서는 --dangerously-skip-permissions, 코덱스에서는 --yolo라는 명령어로 켭니다. 이 모드는 강력하지만 위험하기 때문에, 켜는 법과 끄는 법을 함께 배워야 합니다.

2부 욜로 모드를 켜기 전에 꼭 알아야 할 것

4장. 터미널이 뭔가요

2019년 겨울, 제가 아는 자영업자 한 분이 "엑셀 매크로 좀 만들어 달라"고 하셔서 노트북을 가져왔습니다. 제가 검은 화면을 열자, 그분이 놀란 표정으로 물었습니다. "그거 해킹하는 건가요?"

아닙니다. 해킹이 아닙니다.

터미널(Terminal, 터미널)은 컴퓨터에게 글로 명령을 내리는 화면입니다. 우리가 보통 컴퓨터를 쓸 때는 마우스로 아이콘을 클릭하고, 창을 열고, 버튼을 누릅니다. 이것을 GUI(Graphical User Interface, 그래피컬 유저 인터페이스)라고 합니다. 그림으로 소통하는 방식이죠. 터미널은 그와 다릅니다. 글자를 치면 컴퓨터가 알아듣습니다. CLI(Command Line Interface, 커맨드 라인 인터페이스)라고 부릅니다. 글로 소통하는 방식입니다.

식당 비유로 돌아가 보겠습니다. GUI는 메뉴판에서 사진을 보고 손가락으로 가리키는 겁니다. "이거요." 터미널은 종업원에게 말로 주문하는 겁니다. "김치찌개 하나, 밥 곱빼기로 주세요." 결과는 같습니다. 김치찌개가 나옵니다. 방법이 다를 뿐입니다.

빈 메모장이라고 생각하셔도 좋습니다. 아무것도 안 적혀 있는 흰 종이. 거기에 여러분이 할 말을 적으면, 컴퓨터가 읽고 실행합니다.

맥(Mac)에서 터미널을 여는 법부터 알려드리겠습니다.

키보드에서 Command 키와 스페이스바를 동시에 누릅니다. 화면 가운데 검색창이 뜹니다. 여기에 "터미널"이라고 입력합니다. 영어로 "Terminal"이라고 쳐도 됩니다. 목록에 터미널 앱이 나타나면, 엔터를 누릅니다. 검은 화면, 혹은 흰 화면이 나타납니다. 커서가 깜빡이고 있을 겁니다. 이게 터미널입니다.

윈도우(Windows)에서는 조금 다릅니다. 키보드 왼쪽 아래에 윈도우 로고 키가 있습니다. 이 키를 누르면 시작 메뉴가 열립니다. 검색창에 "cmd"라고 입력합니다. "명령 프롬프트"가 나타납니다. 클릭하면 검은 화면이 뜹니다. 윈도우 11을 쓰신다면 "Windows Terminal"이라고 검색해도 됩니다. 더 보기 좋은 화면이 열립니다.

여기까지 하셨으면, 첫 번째 고비를 넘긴 겁니다.

터미널 화면을 보면 글자가 조금 적혀 있을 겁니다. 맥이라면 "사용자이름@컴퓨터이름 ~ %" 같은 글자가 보입니다. 윈도우라면 "C:\Users\사용자이름>" 같은 글자가 보입니다. 이것을 프롬프트(Prompt, 프롬프트)라고 부릅니다. "여기에 명령어를 입력하세요"라는 뜻입니다. AI 챗봇에 입력하는 프롬프트와 이름이 같지만, 다른 맥락에서 쓰이는 단어입니다.

시험 삼아 한 가지만 쳐보겠습니다. 터미널에 다음 명령어를 입력하고 엔터를 누르십시오.

```
echo "안녕하세요"
```

(에코, "안녕하세요")

화면에 "안녕하세요"라고 나타날 겁니다. 여러분이 시킨 말을 컴퓨터가 그대로 따라한 겁니다. echo(에코)는 "따라 말해"라는 뜻의 명령어입니다. 산에서 "야호!"라고 외치면 메아리가 돌아오듯, 컴퓨터가 여러분의 말을 그대로 돌려줍니다.

이게 다입니다. 터미널은 이렇게 생겼고, 이렇게 씁니다. 무섭지 않습니다.

> [경고] 터미널에서 잘 모르는 명령어를 함부로 실행하지 마십시오. 파일이 삭제되거나 시스템 설정이 바뀔 수 있습니다. 이 책에서 안내하는 명령어만 입력하시기 바랍니다.

터미널이 왜 필요한지 궁금하실 겁니다. 클로드 코드와 코덱스가 터미널에서 작동하기 때문입니다. 이 두 AI 도구는 아이콘을 클릭해서 여는 프로그램이 아닙니다. 터미널에서 명령어를 입력해서 실행합니다. 그래서 터미널을 먼저 배우는 겁니다. 수영을 배우기 전에 물에 들어가는 법부터 배우는 것과 같습니다.

5장. 클로드 코드와 코덱스, 뭐가 다른가

두 가지 도구를 비교하겠습니다.

클로드 코드는 앤스로픽이 만들었습니다. 코덱스는 오픈AI가 만들었습니다. 둘 다 터미널에서 작동하는 AI 도구이고, 둘 다 여러분의 컴퓨터에서 직접 파일을 만들고 코드를 실행합니다. 기능은 비슷하지만, 만든 회사가 다르고, 명령어가 다르고, 안전장치의 이름이 다릅니다.

먼저 설치법을 알려드리겠습니다. 두 도구 모두 설치하려면 노드(Node.js, 노드제이에스)라는 프로그램이 필요합니다. 노드는 자바스크립트라는 프로그래밍 언어를 실행하는 도구인데, 설치 과정만 알면 됩니다. 원리는 몰라도 괜찮습니다.

노드 설치법은 이렇습니다. 맥에서는 터미널을 열고 다음 명령어를 입력합니다.

brew install node

(브루 인스톨 노드)

brew(브루)는 홈브루(Homebrew)라는 맥용 프로그램 설치 도구입니다. 앱스토어와 비슷한데, 터미널에서 작동합니다. 홈브루가 없다면 먼저 설치해야 합니다. 홈브루 홈페이지(brew.sh)에 나온 안내를 따르시면 됩니다.

윈도우에서는 nodejs.org 웹사이트에 접속해서 설치 파일을 다운로드하고, 더블클릭으로 설치합니다. "다음, 다음, 완료." 일반 프로그램 설치와 같습니다.

노드가 설치됐으면, 클로드 코드를 설치합니다. 터미널에 이렇게 입력합니다.

npm install -g @anthropic-ai/claude-code

(엔피엠 인스톨 대시지 앳앤스로픽대시에이아이 슬래시 클로드대시코드)

npm(엔피엠)은 노드에 딸려오는 프로그램 설치 도구입니다. install은 "설치해"라는 뜻이고, -g는 "컴퓨터 어디서나 쓸 수 있게"라는 뜻입니다. 이 명령어 하나면 클로드 코드가 설치됩니다.

코덱스 설치도 비슷합니다.

npm install -g @openai/codex

(엔피엠 인스톨 대시지 앳오픈에이아이 슬래시 코덱스)

이제 두 도구의 차이점을 정리하겠습니다.

명령어가 다릅니다. 클로드 코드를 실행하려면 터미널에 claude(클로드)라고 칩니다. 코덱스를 실행하려면 codex(코덱스)라고 칩니다.

안전장치 이름이 다릅니다. 클로드 코드의 욜로 모드는 --dangerously-skip-permissions. 코덱스의 욜로 모드는 --yolo. 기능은 같습니다. 이름만 다릅니다.

기본 모델이 다릅니다. 클로드 코드는 클로드(Claude) 모델을 씁니다. 2026년 5월 현재, 기본 모델은 클로드 오퍼스 4.7(Claude Opus 4.7)입니다. 코덱스는 GPT 계열 모델을 씁니다. 기본 모델은 GPT-5.5입니다.

가격 구조는 둘 다 비슷합니다. 클로드 코드는 앤스로픽 구독(Claude Pro, Max, Team)에 포함된 사용량으로 쓸 수 있고, 구독 없이 API(Application Programming Interface, 에이피아이) 종량제로도 쓸 수 있습니다. 코덱스도 마찬가지로 오픈AI 구독(ChatGPT Pro, 챗지피티 프로)에 포함된 사용량이 있고, API 종량제로도 쓸 수 있습니다. 어느 쪽이든 구독이 있으면 추가 비용 없이 시작할 수 있다는 뜻입니다.

어느 쪽을 쓸지는 여러분의 상황에 달려 있습니다. 이미 챗GPT를 유료로 쓰고 계시다면 코덱스가 자연스럽고, 클로드를 유료로 쓰고 계시다면 클로드 코드가 자연스럽습니다. 어느 쪽도 쓰고 있지 않다면, 둘 다 무료 체험이 가능하니 하나를 골라서 시작하면 됩니다. 두 도구의 작동 방식이 비슷해서, 하나를 익히면 다른 하나로 옮기는 건 어렵지 않습니다.

한 가지 더 말씀드릴 게 있습니다. 두 도구 모두 터미널 버전 외에 다른 형태도 있습니다. 클로드 코드는 VS Code(브이에스코드)라는 코드 편집기의 확장 프로그램으로도 쓸 수 있고, 클로드 데스크톱(Claude Desktop) 앱에서도 비슷한 기능을 쓸 수 있습니다. 코덱스는 챗GPT 웹사이트 안에서 사용할 수 있는 버전도 있습니

다. 하지만 욜로 모드, 즉 AI가 여러분의 컴퓨터에서 직접 파일을 조작하는 완전 자동 모드를 쓰려면 터미널 버전이 필요합니다. 그래서 이 책은 터미널 버전을 중심으로 설명합니다.

6장. 안전 도구 네 가지를 먼저 익힌다

욜로 모드를 켜기 전에 안전 장비부터 갖추겠습니다. 스키를 타기 전에 헬멧을 쓰는 것과 같습니다. 네 가지입니다.

깃, 시간 되돌리기 버튼

깃(Git, 깃)은 파일의 변경 이력을 기록하는 도구입니다.

이렇게 생각해 보십시오. 워드 문서를 쓰다가 "다른 이름으로 저장"을 눌러 "보고서_v1", "보고서_v2", "보고서_최종", "보고서_진짜최종"을 만들어 보신 적 있으실 겁니다. 깃은 이 과정을 자동으로 해줍니다. 파일을 바꿀 때마다 "이때의 상태를 기억해둬"라고 말하면, 깃이 기억합니다. 나중에 "아까 그 상태로 돌아가줘"라고 하면, 돌아갑니다.

왜 이게 필요하냐면, AI가 여러분의 파일을 수정했는데 결과가 마음에 안 들 때, 수정 전 상태로 되돌릴 수 있기 때문입니다. 욜로 모드의 안전벨트입니다.

명령어 세 개만 기억하면 됩니다.

git init

(깃 이닛)

이 명령어는 "이 폴더의 변경 사항을 추적하기 시작해"라는 뜻입니다. 작업할 폴더에서 처음 한 번만 실행합니다.

git add .

(깃 애드 점)

"바뀐 파일들을 모아둬"라는 뜻입니다. 점(.)은 "이 폴더 안의 모든 파일"을 가리킵니다.

git commit -m "작업 전 상태 저장"

(깃 커밋 대시엠 "작업 전 상태 저장")

"지금 이 상태를 기억해둬"라는 뜻입니다. 따옴표 안에 메모를 적습니다. 나중에 이 메모를 보고 "아, 이때 저장한 거구나" 하고 알 수 있게 해줍니다.

이 세 단계를 욜로 모드를 켜기 전에 매번 실행하십시오. 3초면 됩니다. 하지만 이 3초가 여러분의 파일을 지켜줍니다.

도커, 놀이 모래밭의 나무 울타리

도커(Docker, 도커)는 컴퓨터 안에 작은 컴퓨터를 만드는 도구입니다.

어린이 놀이터에 모래밭이 있고, 그 주위에 나무 울타리가 있는 걸 보신 적 있을 겁니다. 아이가 모래를 아무리 파헤쳐도, 울타리 바깥의 잔디밭에는 영향이 없습니다. 도커가 바로 그 울타리입니다.

AI에게 욜로 모드로 일을 시킬 때, 도커 안에서 일하게 하면, AI가 무엇을 하든 도커 바깥의 여러분 파일에는 영향이 없습니다. AI가 파일을 잘못 삭제해도, 삭제된 건 도커 안의 파일뿐입니다. 진짜 파일은 무사합니다.

도커 데스크톱(Docker Desktop)은 docker.com에서 다운로드합니다. 맥용, 윈도우용 모두 있습니다. 설치는 일반 프로그램과 같습니다. 다운로드, 더블클릭, "다음, 다음, 완료." 설치 후 앱을 한 번 실행해두면 됩니다.

도커 안에서 욜로 모드를 쓰는 방법은 7장과 8장에서 자세히 알려드리겠습니다. 지금은 "도커라는 울타리가 있다"는 것만 기억해 주십시오.

오토 모드, 중간 단계

완전한 욜로 모드가 부담스러우신 분을 위해, 중간 단계가 있습니다. 클로드 코드의 오토 모드(Auto Mode)입니다.

교통 신호등으로 비유하면 이렇습니다. 기본 모드는 모든 교차로에서 빨간불이 켜집니다. 매번 멈추고, 확인하고, 출발합니다. 안전하지만 느립니다. 욜로 모드는 신호등을 전부 꺼버립니다. 쭉 달립니다. 빠르지만, 사고 가능성이 있습니다. 오토 모드는 작은 교차로의 신호등만 끕니다. 큰 교차로(파일 삭제, 시스템 명령)에서는 여전히 빨간불이 켜지고, 작은 교차로(파일 읽기, 검색)에서는 자동으로 통과합니다.

처음 욜로 모드를 쓰시는 분이라면, 오토 모드부터 시작하는 걸 권합니다. 며칠 써보고 감이 잡히면 완전한 욜로 모드로 넘어가시면 됩니다.

명확한 지시문 쓰기

네 번째 안전 도구는 기술이 아닙니다. 습관입니다.

"알아서 해줘"라고 말하지 마십시오.

이 말이 왜 위험한지 생각해 보겠습니다. AI에게 "이 폴더 정리해줘"라고 하면, AI는 AI 나름의 기준으로 정리합니다. 파일 이름을 바꿀 수도 있고, 오래된 파일을 삭제할 수도 있습니다. 여러분이 의도한 "정리"와 AI가 이해한 "정리"가 다를 수 있습니다.

대신 이렇게 말하십시오. "이 폴더 안의 엑셀 파일 세 개를 읽어서, 매출 합계를 계산하고, '월별_매출_요약.xlsx'라는 파일로 저장해줘. 기존 파일은 건드리지 마." 구체적입니다. AI가 무엇을 해야 하고, 무엇을 하면 안 되는지 명확합니다.

좋은 지시문의 조건은 세 가지입니다.

무엇을 할지 적습니다. "엑셀 파일 세 개를 읽어서 매출 합계를 계산해."

결과물의 형태를 적습니다. "'월별_매출_요약.xlsx'로 저장해."

하지 말아야 할 것을 적습니다. "기존 파일은 건드리지 마."

이 세 가지를 갖춘 지시문을 쓰는 습관이, 어떤 소프트웨어 안전장치보다 여러분을 잘 지켜줍니다.

오늘 손에 익힌 것

깃(Git)은 파일 상태를 저장하고 되돌리는 시간 여행 도구이며, git init, git add, git commit 세 명령어로 작동합니다. 도커(Docker)는 AI가 작업할 안전한 울타리를 만들어주는 도구이고,

오토 모드는 완전한 욜로 모드 전에 거칠 수 있는 중간 단계입니다. 무엇보다, "알아서 해줘" 대신 구체적인 지시문을 쓰는 습관이 가장 든든한 안전장치입니다.

3부 처음 한 번, 무사히 켜고 끄기

7장. 클로드 코드 욜로 모드 켜기

자, 드디어 욜로 모드를 켤 시간입니다.

하지만 그 전에 준비물을 점검하겠습니다. 체크리스트처럼 하나씩 확인하십시오.

노드(Node.js)가 설치되어 있는지 확인합니다. 터미널에 이렇게 입력합니다.

node --version

(노드 대시대시 버전)

"v20.11.0" 같은 숫자가 나타나면 됩니다. 숫자가 아니라 에러 메시지가 나오면 5장의 설치법을 다시 따라가 주십시오.

클로드 코드가 설치되어 있는지 확인합니다.

claude --version

(클로드 대시대시 버전)

버전 번호가 나오면 됩니다.

클로드 코드의 사용 권한을 얻어야 합니다. 방법은 두 가지입니다. 클로드 구독(Claude Pro, Max, Team)이 있다면 계정 로그인만으로 바로 쓸 수 있습니다. 터미널에서 클로드 코드를 처음 실행하면 브라우저가 열리고 로그인 화면이 나옵니다. 평소 쓰던 클

로드 계정으로 로그인하면 끝입니다. 구독이 없거나 API 종량제를 쓰고 싶다면, 앤스로픽 홈페이지(console.anthropic.com)에서 API 키(API Key, 에이피아이 키)를 발급받습니다. 회원 가입 후, "API Keys" 메뉴에서 "Create Key"를 누르면 "sk-ant-"로 시작하는 긴 문자열이 나옵니다. 이것이 열쇠입니다. 이 열쇠를 분실하거나 다른 사람에게 보여주면 안 됩니다.

> [경고] API 키는 비밀번호와 같습니다. 절대 다른 사람에게 공유하지 마십시오. 블로그나 SNS에 올리지 마십시오. 혹시 유출됐다면, 앤스로픽 홈페이지에서 즉시 삭제하고 새 키를 발급받으십시오.

작업할 폴더를 정합니다. 중요한 파일이 없는 빈 폴더를 하나 만드십시오. 연습용입니다.

mkdir 연습폴더

(엠케이디아이알 연습폴더)

cd 연습폴더

(시디 연습폴더)

mkdir(엠케이디아이알)은 "make directory"의 줄임말로, 새 폴더를 만드는 명령어입니다. cd(시디)는 "change directory"의 줄임말로, 그 폴더로 이동하는 명령어입니다.

깃으로 상태를 저장합니다.

git init

git add .

git commit -m "욜로 모드 테스트 전 상태"

준비가 끝났습니다. 이제 켭니다.

claude --dangerously-skip-permissions

(클로드 댄저러슬리 스킵 퍼미션스)

엔터를 누르면, 화면에 안내 메시지가 나타납니다. 클로드 코드가 실행된 것입니다. 커서가 깜빡이며 여러분의 지시를 기다립니다. 이 상태에서 여러분이 무언가를 입력하면, 클로드 코드는 중간에 허락을 구하지 않고 바로 실행합니다.

시험해 보겠습니다. 다음과 같이 입력하십시오.

"이 폴더에 hello.txt라는 파일을 만들어줘. 내용은 '안녕하세요, 욜로 모드입니다.'로."

클로드 코드가 파일을 만드는 과정이 화면에 나타납니다. "이 파일을 만들어도 될까요?" 같은 질문 없이, 바로 만듭니다. 완료 메시지가 나오면, 새 터미널 창을 열어 확인해 보십시오.

cat hello.txt

(캣 헬로 점 티엑스티)

"안녕하세요, 욜로 모드입니다."라는 내용이 출력되면 성공입니다. 여러분은 방금 욜로 모드를 처음으로 켜고, AI에게 파일 하나를 만들게 했습니다.

VS Code(브이에스코드)를 쓰시는 분은 VS Code 안에서도 같은 일을 할 수 있습니다. VS Code를 열고, 터미널 창(메뉴에서 Terminal > New Terminal)을 연 뒤, 같은 명령어를 입력하면 됩니다. VS Code 확장 프로그램으로 클로드 코드를 설치하면 더 편리하게 쓸 수 있지만, 기본 원리는 동일합니다.

매번 --dangerously-skip-permissions를 다 치기 귀찮으실 겁니다. 길잖아요. 쉘 별명(alias, 앨리어스)이라는 기능을 쓰면, 짧은 이름으로 줄일 수 있습니다.

맥에서는 터미널에 다음 명령어를 입력합니다.

```
echo 'alias cyolo="claude --dangerously-skip-permissions"' >> ~/.zshrc
```

(에코 앨리어스 시욜로 이퀄스 클로드 댄저러슬리 스킵 퍼미션스 >> 물결표 슬래시 점 지에스에이치알시)

그리고 터미널을 닫았다가 다시 엽니다. 이제부터 cyolo(시욜로)라고만 치면 욜로 모드가 켜집니다. 원래 명령어와 같은 뜻이지만, 이름만 짧아진 겁니다. 별명은 본명과 같은 사람을 가리키듯이요.

윈도우에서는 조금 다릅니다. 명령 프롬프트 대신 파워셸(PowerShell, 파워셸)을 쓰신다면 이렇게 합니다.

```
Set-Alias cyolo "claude --dangerously-skip-permissions"
```

하지만 파워셸에서 이 별명은 터미널을 닫으면 사라집니다. 영구적으로 설정하려면 프로필 파일을 편집해야 하는데, 지금은 넘어가겠습니다. 매번 원래 명령어를 쓰셔도 됩니다.

8장. 코덱스 욜로 모드 켜기

코덱스의 욜로 모드도 원리는 같습니다. 명령어만 다릅니다.

준비물은 클로드 코드와 거의 동일합니다. 노드가 설치되어 있어야 하고, 코덱스가 설치되어 있어야 합니다. 확인법은 같습니다.

codex --version

(코덱스 대시대시 버전)

코덱스도 클로드 코드와 마찬가지로 두 가지 방법이 있습니다. 챗GPT 유료 구독이 있다면 챗GPT 계정 로그인으로 바로 쓸 수 있습니다. 터미널에서 코덱스를 처음 실행하면 로그인 화면이 나오고, 평소 쓰던 챗GPT 계정으로 로그인하면 됩니다. 구독 없이 API 종량제를 쓰고 싶다면, platform.openai.com에서 API 키를 발급받습니다. 과정은 앤스로픽과 비슷합니다.

> [경고] 오픈AI API 키 역시 비밀번호와 같습니다. 유출되면 즉시 삭제하고 새로 발급받으십시오.

작업 폴더로 이동하고, 깃으로 상태를 저장합니다. 7장에서 한 것과 같은 과정입니다.

이제 켭니다.

codex --yolo

(코덱스 욜로)

짧습니다. 두 단어면 됩니다. 이 별명 같은 명령어의 원래 이름은 이렇습니다.

codex --dangerously-bypass-approvals-and-sandbox

(코덱스 댄저러슬리 바이패스 어프루벌스 앤드 샌드박스)

"위험하게 승인과 샌드박스를 우회한다." 샌드박스(Sandbox, 샌드박스)는 앞서 배운 도커의 울타리와 비슷한 개념입니다. 모래밭(sand)의 상자(box). AI가 안전한 공간 안에서만 작업하도록 제한하는 장치입니다. --yolo를 쓰면, 이 울타리까지 해제됩니다.

코덱스에는 설정 파일(config file, 컨피그 파일)을 통해 욜로 모드를 미리 설정해둘 수도 있습니다. 매번 명령어에 --yolo를 붙이는 대신, 설정 파일에 기본값으로 적어두는 방식입니다.

설정 파일 이름은 config.toml(컨피그 점 토믈)입니다. TOML(Tom's Obvious, Minimal Language)이라는 형식의 파일인데, 사람이 읽기 쉬운 설정 파일 형식입니다. 이 파일에 다음 두 줄을 추가하면 됩니다.

approval_policy = "never"

sandbox_mode = "danger-full-access"

approval_policy(어프루벌 폴리시)는 "승인 정책"이라는 뜻입니다. "never"는 "절대 묻지 마"라는 값입니다. sandbox_mode(샌드박스 모드)는 "울타리 모드"인데, "danger-full-access"는 "위험, 전체 접근 허용"이라는 뜻입니다. 이 설정 파일이 있으면, codex라고만 쳐도 욜로 모드로 작동합니다.

> [경고] config.toml에 이 설정을 넣어두면, 코덱스를 실행할 때마다 자동으로 욜로 모드가 됩니다. 연습이 끝나면 이 두 줄을 삭제하거나, 값을 원래대로 되돌리십시오. approval_policy를 "on-failure"로, sandbox_mode를 "docker"로 바꾸면 기본 안전장치가 다시 켜집니다.

시험해 봅시다. 코덱스 욜로 모드가 켜진 상태에서 이렇게 입력합니다.

"이 폴더에 greeting.py라는 파이썬 파일을 만들고, 실행하면 '반갑습니다, 코덱스 욜로 모드입니다.'라고 출력되게 해줘. 파일을 만든 뒤에 바로 실행까지 해줘."

코덱스가 파이썬 파일을 만들고, 실행까지 합니다. 화면에 "반갑습니다, 코덱스 욜로 모드입니다."가 나타나면 성공입니다. 허락을 구하는 단계가 없었다는 걸 확인하셨을 겁니다. 파일 생성도, 프로그램 실행도, AI가 스스로 진행했습니다.

클로드 코드와 코덱스, 어느 쪽을 쓰든 핵심은 같습니다. AI에게 지시하고, AI가 실행하고, 여러분은 결과를 확인합니다.

9장. 끄는 법, 그리고 되돌리는 법

한 가지 원칙으로 시작하겠습니다.

끄는 법을 모르면 켜지 마십시오.

이 말이 과장처럼 들릴 수 있지만, 진심입니다. 전자레인지를 쓸 줄 알면서 전원을 끄는 법을 모르는 사람은 없습니다. 자동차 시동을 걸 줄 알면서 시동을 끄는 법을 모르는 사람도 없습니다. 같은 이치입니다. 욜로 모드를 켜려면, 끄는 법을 반드시 먼저 아셔야 합니다.

끄는 법은 간결합니다.

키보드에서 Ctrl 키와 C 키를 동시에 누릅니다.

(컨트롤 씨)

이것으로 끝입니다. 클로드 코드든 코덱스든, 터미널에서 돌아가는 프로그램은 대부분 Ctrl+C로 멈출 수 있습니다. AI가 한창 파일을 만들고 있는 도중이라도, Ctrl+C를 누르면 멈춥니다. 그래서 이 키 조합을 먼저 알려드리는 겁니다.

맥에서는 Ctrl+C입니다. Command+C가 아닙니다. Command+C는 복사하기입니다. Ctrl+C가 프로그램 중단입니다. 헷갈리기 쉬우니 기억해 두십시오.

이제 되돌리는 법을 알려드리겠습니다.

AI가 작업한 결과가 마음에 안 들 때, 작업 전 상태로 되돌리는 방법입니다. 6장에서 배운 깃이 여기서 빛을 발합니다.

먼저 AI가 무엇을 바꿨는지 확인합니다. 터미널에 이렇게 입력합니다.

git diff

(깃 디프)

diff는 "difference"의 줄임말입니다. 차이. AI가 작업하기 전과 후의 차이를 보여줍니다. 초록색 글자로 추가된 내용이, 빨간색 글자로 삭제된 내용이 나타납니다. 어떤 파일이 어떻게 바뀌었는지 한눈에 볼 수 있습니다.

확인해 보니 결과가 마음에 들지 않습니다. 전부 원래대로 되돌리고 싶습니다. 이렇게 입력합니다.

git checkout -- .

(깃 체크아웃 대시대시 점)

이 명령어는 "마지막으로 저장한 상태(커밋한 상태)로 모든 파일을 되돌려줘"라는 뜻입니다. 7장에서 욜로 모드를 켜기 전에 git commit을 했기 때문에, 그 시점으로 돌아갑니다.

> [경고] git checkout -- . 은 되돌릴 수 없습니다. 이 명령어를 실행하면, AI가 작업한 내용이 전부 사라집니다. 혹시 AI의 작업 결과 중 일부는 살리고 싶다면, 이 명령어를 쓰기 전에 필요한 파일을 다른 폴더에 복사해 두십시오.

AI가 만든 새 파일은 git checkout으로 삭제되지 않습니다. 기존 파일의 변경 사항만 되돌립니다. AI가 새로 만든 파일을 확인하려면 이렇게 합니다.

git status

(깃 스테이터스)

"Untracked files"라고 표시된 것이 AI가 새로 만든 파일입니다. 이것들을 삭제하고 싶다면, 파일 하나하나 확인한 뒤 수동으로 삭제하십시오. 확인 없이 한꺼번에 지우는 명령어도 있지만, 이 책에서는 안전을 위해 수동 삭제를 권합니다.

되돌리는 과정을 정리하면 이렇습니다.

Ctrl+C로 AI를 멈춘다. git diff로 변경 사항을 확인한다. 되돌리고 싶으면 git checkout -- . 을 실행한다. git status로 새 파일을 확인하고, 필요 없으면 삭제한다.

이 네 단계를 기억해 두십시오. 이것이 여러분의 비상구입니다. 건물에 들어가면 비상구 위치부터 확인하듯이, 욜로 모드를 쓸 때는 이 네 단계를 먼저 확인하고 시작하시기 바랍니다.

한 가지만 더 말씀드리겠습니다. 깃으로 저장하지 않은 상태에서 AI가 파일을 수정하면, 되돌릴 방법이 없습니다. 그래서 7장에서 "욜로 모드를 켜기 전에 반드시 git commit을 하라"고 말씀드린 겁니다. 이 습관 하나가 모든 것을 지켜줍니다. 3초면 되는 일입니다. "git add ." 엔터. "git commit -m '작업 전'" 엔터. 이 3초를 빼먹으면, 나중에 세 시간을 후회할 수 있습니다.

오늘 손에 익힌 것

클로드 코드 욜로 모드는 claude --dangerously-skip-permissions, 코덱스 욜로 모드는 codex --yolo 또는 config.toml 설정으로 켤 수 있습니다. 두 도구 모두 Ctrl+C로 멈출 수 있고, git diff로 변경 사항을 확인한 뒤 git checkout -- . 으로 되돌릴 수 있습니다. 욜로 모드를 켜기 전에 반드시 git commit으로 현재 상태를 저장하는 습관을 들이십시오.

4부 손에 익히는 작은 실전

10장. 첫 실습: 자기소개 웹페이지 만들기

토요일 오후, 거실 소파에 앉아 노트북을 열었습니다. 옆에는 커피 한 잔. 터미널 화면이 깜빡이고 있습니다. 3부까지 따라오면서 깃도 익히고, 욜로 모드를 켜고 끄는 것도 해봤으니, 이제 뭔가 만들어볼 차례입니다.

가벼운 것부터 시작하겠습니다. 자기소개 웹페이지. 이름, 취미, 좋아하는 음식 정도가 적힌 한 장짜리 페이지입니다.

준비

터미널을 열고, 실습용 폴더를 하나 만듭니다.

mkdir ~/yolo-practice

cd ~/yolo-practice

git init

세 줄이면 됩니다. 폴더를 만들고, 그 폴더로 들어가고, 깃을 켜둔 겁니다. 혹시 뭔가 잘못되면 되돌릴 수 있도록요.

명령

이제 욜로 모드를 켭니다.

claude --dangerously-skip-permissions

검은 화면에 클로드 코드가 인사를 합니다. 여기에 이렇게 적습니다.

"내 이름은 김민수이고, 취미는 등산과 사진 찍기야. 좋아하는 음식은 된장찌개. 이 내용으로 자기소개 웹페이지를 만들어줘. 파일 이름은 index.html로 해줘."

엔터를 누릅니다. 그리고 기다립니다.

AI가 하는 일

클로드 코드가 움직이기 시작합니다. 터미널 화면에 글자가 쭉 올라갑니다. 지금 AI가 하고 있는 일을 천천히 따라가 보겠습니다.

AI는 먼저 index.html이라는 파일을 만듭니다. HTML(에이치티엠엘, 웹페이지를 만드는 언어)로 된 파일인데, 여러분이 이 코드를 읽을 줄 몰라도 상관없습니다. AI가 알아서 씁니다.

파일 안에는 여러분의 이름이 큰 글씨로 들어가 있고, 그 아래에 취미와 좋아하는 음식이 적혀 있습니다. 배경색도 깔끔하게 잡아주고, 글씨체도 보기 좋은 걸로 골라줍니다. 기본 모드였다면 "파일을 만들어도 될까요?"라고 물어봤을 텐데, 욜로 모드이니까 묻지 않고 바로 만듭니다.

20초쯤 지나면 "완료했습니다"라는 메시지가 뜹니다. 이 파일을 브라우저에서 열어보겠습니다. 클로드 코드에게 "방금 만든 페이지를 브라우저에서 열어줘"라고 시키면 됩니다. 직접 하고 싶다면 파인더(Finder)에서 작업 폴더를 열고 index.html 파일을 더블클릭해도 됩니다.

크롬이나 사파리가 열리면서 웹페이지가 나타납니다. "김민수"라는 이름이 크게 보이고, 그 아래에 등산, 사진 찍기, 된장찌개가 적혀 있습니다. 깔끔합니다.

소요 시간과 비용

처음 명령을 입력한 시점부터 브라우저에서 확인하기까지, 3분이 채 안 걸립니다. 비용은 클로드 API(에이피아이, 프로그램끼리 통신하는 통로) 기준으로 200원에서 400원 사이입니다. 커피 한 잔 값의 10분의 1도 안 되는 거죠.

위험 요소

이 실습에서 잘못될 수 있는 건 거의 없습니다. AI가 한 일이라고는 파일 하나를 만든 것뿐이거든요. 기존 파일을 건드린 게 아니라 새 파일을 만든 것이니, 컴퓨터에 영향을 주지 않습니다. 혹시 마음에 안 들면 그 파일을 지우면 끝입니다. 파인더(맥)나 탐색기(윈도우)에서 마우스 오른쪽 버튼을 눌러 삭제해도 되고, 클로드 코드에게 "방금 만든 파일 지워줘"라고 시켜도 됩니다. 깃으로 저장해뒀으니까 "전부 원래대로 되돌려줘"라고 시키는 방법도 있고요.

조금 더 해보고 싶다면

웹페이지가 밋밋하게 느껴지면, 클로드 코드에게 이렇게 말해 보세요.

"사진을 넣고 싶어. 프로필 사진 자리를 만들어줘."

"색깔을 파란색 계열로 바꿔줘."

"연락처 버튼을 추가해줘."

한 번에 하나씩 시키는 게 좋습니다. 열 가지를 한꺼번에 시키면 AI가 헷갈릴 수 있거든요.

11장. 두 번째 실습: 엑셀 파일 합치기

월요일 아침 9시. 영업팀 박과장이 자리에 앉자마자 한숨을 쉬었습니다. 매출 보고서 마감이 오후 2시인데, 매출표가 세 개의 엑셀 파일로 나뉘어 있습니다. 1분기_매출.xlsx, 2분기_매출.xlsx, 3분기_매출.xlsx. 이걸 하나로 합쳐야 합니다.

예전이라면 각 파일을 열고, 복사하고, 붙여넣고, 행 번호가 맞는지 확인하고, 서식이 깨지지 않았는지 눈으로 훑었을 겁니다. 30분은 걸리는 일이었습니다.

준비

매출표 파일이 들어 있는 폴더로 갑니다.

cd ~/Documents/매출자료

git init

git add .

git commit -m "원본 백업"

네 줄입니다. 깃으로 원본을 먼저 저장해둔 겁니다. 이게 습관이 되어야 합니다. 욜로 모드에서 AI가 원본 파일을 덮어쓸 수 있거든요. 깃으로 저장해두면, 설령 덮어써도 되돌릴 수 있습니다.

명령

claude --dangerously-skip-permissions

클로드 코드가 켜지면 이렇게 시킵니다.

"이 폴더에 있는 xlsx 파일 세 개를 하나의 엑셀 파일로 합쳐줘. 각 파일의 시트 이름을 그대로 유지하고, 합친 파일 이름은 매출_통합.xlsx로 해줘. 원본 파일은 건드리지 마."

마지막 문장이 중요합니다. "원본 파일은 건드리지 마." 이 한마디가 없으면 AI가 원본 위에 덮어쓸 가능성이 있습니다.

AI가 하는 일

클로드 코드는 먼저 파이썬(Python, 파이썬, 프로그래밍 언어) 스크립트를 하나 만듭니다. 이 스크립트가 openpyxl(오픈파이엑스엘, 엑셀 파일을 다루는 도구)이라는 라이브러리(library, 라이브러리, 미리 만들어진 코드 묶음)를 쓰는데, 컴퓨터에 이 라이브러리가 없으면 AI가 알아서 설치합니다. 기본 모드였다면 "라이브러리를 설치해도 될까요?"라고 물었겠지만, 욜로 모드에서는 그냥 설치합니다.

스크립트를 만들고, 실행하고, 매출_통합.xlsx 파일이 생깁니다. 40초 정도 걸립니다.

결과 확인

매출_통합.xlsx 파일을 엑셀이나 구글 시트로 열어봅니다. 시트 탭이 세 개 있고, 각각 1분기, 2분기, 3분기 데이터가 들어 있습니다. 숫자가 맞는지 원본과 비교해봅니다.

여기서 확인을 꼭 해야 합니다. AI가 숫자를 빼먹거나, 행 순서를 바꾸는 경우가 간혹 있거든요. 원본 파일의 마지막 행 합계와 통합 파일의 합계를 비교해보세요. 숫자가 같으면 성공입니다.

소요 시간과 비용

명령 입력부터 결과 확인까지 1분 남짓입니다. 비용은 300원 안팎. 박과장이 수작업으로 했다면 30분이 걸렸을 일입니다.

위험 요소

위험은 하나입니다. 원본 덮어쓰기. AI가 원본 파일 위에 합친 결과를 저장해버릴 수 있습니다. 그래서 깃으로 미리 저장해두는 겁니다. 깃을 안 해뒀다면?

```
git checkout -- .
```

이 한 줄이면 모든 파일이 마지막 커밋(commit, 커밋, 저장 시점) 상태로 돌아갑니다. 깃을 안 해뒀으면 되돌릴 방법이 없으니, 반드시 먼저 저장해두세요.

실수를 한 번 더 막고 싶다면, 원본 파일을 따로 복사해두는 것도 방법입니다.

```
cp *.xlsx ~/Documents/매출자료_백업/
```

허리와 멜빵을 동시에 쓰는 셈인데, 안전한 쪽이 낫습니다.

12장. 세 번째 실습: 매일 아침 뉴스 요약 받기

종로구에서 분식집을 하는 이사장님은 아침마다 네이버 뉴스를 훑어봅니다. 외식업 동향, 식자재 가격, 위생 관련 규정 변경. 알아야 할 게 많은데 뉴스를 읽을 시간은 부족합니다. 오전 7시에 출근해서 반죽부터 시작해야 하니까요.

이사장님이 원하는 건 이겁니다. 매일 아침 8시에 "외식업" 관련 뉴스 다섯 건이 자동으로 정리되어 텍스트 파일로 바탕화면에 놓여 있는 것. 가게 문 열기 전에 훑어보고, 필요한 건 기억해두고, 나머지는 넘기면 됩니다.

준비

이 실습은 앞의 두 실습보다 설정할 게 조금 더 있습니다. 10분 정도 걸리는데, 한 번 설정해두면 매일 자동으로 돌아갑니다.

먼저 실습 폴더를 만듭니다.

mkdir ~/news-summary

cd ~/news-summary

git init

그다음, 도커(Docker, 도커, 격리된 작업 공간을 만드는 도구)를 씁니다. 왜냐면 이번 실습에서는 AI가 인터넷에 접속하거든요. 뉴스를 검색하려면 웹에 접근해야 합니다. 욜로 모드에서 AI가 인터넷에 자유롭게 접근하면 위험할 수 있으니, 도커라는 울타리 안에서 돌리는 겁니다.

6장에서 도커를 설치해두셨을 테니, 클로드 코드에게 이렇게 시킵니다. "도커 컨테이너 안에서 뉴스 요약 작업을 해줘. ~/news-summary 폴더를 작업 폴더로 연결해서, 결과 파일이 내 컴퓨터에도 저장되게 해줘." 클로드 코드가 도커 명령어를 알아서 작성하고 실행합니다. 여러분이 docker run 같은 긴 명령어를 외울 필요가 없습니다.

명령

도커 안에서 클로드 코드가 켜지면, 이렇게 시킵니다.

"외식업 관련 최신 뉴스 다섯 건을 검색해서, 각 기사의 제목과 세 줄 요약을 news_오늘날짜.txt 파일로 만들어줘. 파일은 /work 폴더에 저장해."

첫 실행이니 일단 한 번 돌려봅니다. AI가 웹 검색 도구를 써서 뉴스를 찾고, 요약하고, 텍스트 파일을 만듭니다. 2분 정도 걸립니다.

결과를 확인합니다. ~/news-summary 폴더에 news_20260513.txt 같은 파일이 생겨 있을 겁니다. 열어보면 뉴스 제목 다섯 개와 각각의 세 줄 요약이 적혀 있습니다.

자동화 설정

한 번은 잘 됐습니다. 이걸 매일 아침 8시에 자동으로 돌리고 싶습니다.

맥에서는 launchd(런치디, 맥의 예약 작업 도구)를 씁니다. 클로드 코드에게 이렇게 시킬 수 있습니다.

"방금 한 작업을 매일 아침 8시에 자동으로 실행하는 쉘 스크립트와 launchd plist 파일을 만들어줘."

AI가 두 개의 파일을 만들어줍니다. 쉘 스크립트(shell script, 쉘 스크립트, 명령어를 모아둔 파일)는 도커를 켜고 클로드 코드를 실행하는 내용이고, plist(피리스트, 맥의 예약 설정 파일) 파일은 "매일 8시에 이 스크립트를 실행하라"는 예약 정보입니다.

plist 파일을 맥의 예약 시스템에 등록하면 설정이 끝납니다.

launchctl load ~/Library/LaunchAgents/com.news.summary.plist

이 한 줄로 등록됩니다. 내일 아침 8시에 컴퓨터가 켜져 있으면, 바탕화면에 오늘 날짜의 뉴스 요약 파일이 놓여 있을 겁니다.

소요 시간과 비용

처음 설정하는 데 10분. 이후에는 매일 자동으로 2분 동안 돌아갑니다. 비용은 한 번 실행에 500원 안팎이니, 한 달이면 15,000원 정도입니다. 뉴스 스크랩 서비스를 구독하는 것과 비슷한 비용이죠.

위험 요소

이번 실습의 위험은 네트워크 접근입니다. AI가 인터넷에 접속한다는 뜻이니까요. 도커 안에서 돌리면 AI가 접근할 수 있는 범위가 도커 안쪽으로 제한됩니다. 내 컴퓨터의 다른 파일이나 프로그램에는 손을 대지 못합니다.

도커 없이 이걸 돌렸다면? AI가 인터넷에 접속하면서 예상치 못한 스크립트를 내려받거나, 내 컴퓨터의 다른 폴더에 파일을 만들 수도 있습니다. 도커가 그 울타리 역할을 합니다.

한 가지 더. 자동 실행 스크립트가 매일 돌아가니 비용이 쌓입니다. API 사용량 대시보드(dashboard, 대시보드, 사용 현황 화면)를 한 달에 한 번은 확인하세요. 뉴스 검색이 길어지면 비용이 예상보다 올라갈 수 있습니다.

13장. 실습에서 배운 세 가지 원칙

세 번의 실습을 했습니다. 웹페이지를 만들고, 엑셀을 합치고, 뉴스를 자동으로 모았습니다. 세 가지 다 성격이 다른 작업인데, 성공한 이유는 같습니다.

작게 시작했습니다. 웹페이지 실습에서 "이름, 취미, 좋아하는 음식"만 넣었습니다. 처음부터 "사진도 넣고, 애니메이션도 넣고, 반응형으로 만들어줘"라고 시키지 않았습니다. 작게 시작해서 잘 되는 걸 확인한 다음, 하나씩 덧붙이는 겁니다.

이게 중요한 이유가 있습니다. AI가 많은 걸 한꺼번에 하면 실수할 확률이 올라갑니다. 열 가지를 동시에 시켰는데 여덟 가지는 잘 하고 두 가지를 틀리면, 어디가 틀렸는지 찾기가 어렵습니다. 하나씩 시키면 틀린 부분이 바로 보입니다.

깃으로 저장한 뒤 시작했습니다. 세 번 다 git init과 git commit을 먼저 했습니다. 이건 안전벨트입니다. 안전벨트 없이 운전해도 대부분은 괜찮습니다. 그런데 사고가 나면? 깃 없이 욜로 모드를 쓰는 건 안전벨트 없이 고속도로에 올라가는 것과 같습니다.

10장의 웹페이지 실습처럼 위험이 거의 없는 작업에서도 깃을 했습니다. 습관이니까요. 매번 판단하지 말고 그냥 하는 겁니다. git init, git add, git commit. 6초면 됩니다.

결과를 눈으로 확인했습니다. AI가 "완료했습니다"라고 말하면, 그걸 믿지 않았습니다. 웹페이지는 브라우저에서 직접 열어봤고, 엑셀 파일은 열어서 숫자를 비교했고, 뉴스 요약은 파일을 열어 내용을 읽었습니다.

AI는 자기가 만든 결과물이 맞는지 틀렸는지 모를 때가 있습니다. "성공했습니다"라고 말해놓고 실제로는 파일이 비어있는 경우도 봤습니다. 눈으로 확인하는 데 30초면 충분합니다. 그 30초를 아끼다가 나중에 30분을 날릴 수 있습니다.

이 세 가지, 작게 시작하기, 깃으로 저장하고 시작하기, 결과를 눈으로 확인하기는 5부에서 나오는 모든 사례에 공통으로 적용됩니다. 외울 것도 없습니다. 실습 세 번 하면서 손에 익었으니까요.

〔4부 끝〕 오늘 손에 익힌 것

자기소개 웹페이지를 욜로 모드로 만들어봤습니다. 3분, 300원.

엑셀 파일 세 개를 하나로 합쳤습니다. 1분, 300원. 깃으로 원본을 먼저 저장해두는 습관을 익혔습니다.

매일 아침 뉴스를 자동으로 요약받는 설정을 해봤습니다. 설정 10분, 이후 매일 자동. 도커 안에서 돌려야 안전합니다.

세 가지 원칙: 작게 시작한다. 깃으로 저장한 뒤 시작한다. 결과를 눈으로 확인한다.

5부 본격 활용 사례 모음

14장. 글쓰기와 문서 정리

사례 1) 이메일 답장 초안 50통 만들기

장면

수요일 저녁 7시, 마케팅 대행사에서 일하는 정유진 대리가 노트북을 열었습니다. 받은편지함에 안 읽은 메일이 53통. 거래처 인사 메일, 견적 요청, 미팅 일정 조율, 행사 초대. 하나하나 읽고 답장을 쓰려면 두 시간은 걸립니다. 내일 아침 회의 자료도 준비해야 하는데.

명령

정유진 대리는 받은편지함의 메일을 전부 텍스트 파일로 내보냅니다. 지메일(Gmail)에서 메일을 선택하고, 내용을 복사해서 emails_received.txt 파일에 붙여넣습니다. 한 통당 구분선을 넣어서요.

클로드 욜로 모드를 켜고 이렇게 시킵니다.

"emails_received.txt 파일을 읽어서, 각 메일에 대한 답장 초안을 만들어줘. 톤은 정중하지만 간결하게. 견적 관련은 '확인 후 내일 중 회신드리겠습니다'로 끝내고, 일정 조율은 '화요일 오후 2시가 가능합니다'로 답해줘. 결과를 replies_draft.txt에 저장해줘."

진행 과정

클로드 코드가 emails_received.txt를 읽고, 메일 53통의 내용을 파악합니다. 각 메일의 성격을 분류하고(인사, 견적, 일정, 초대, 기타), 성격에 맞는 답장을 씁니다. 3분쯤 지나면 replies_draft.txt 파일이 생깁니다.

결과

53통의 답장 초안이 담긴 텍스트 파일. 각 답장 앞에 원래 메일의 제목과 보낸 사람이 적혀 있어서 어떤 메일에 대한 답장인지 바로 알 수 있습니다.

비용

시간 3분, 비용 500원 안팎.

위험

AI가 엉뚱한 내용을 답장에 넣을 수 있습니다. "화요일 오후 2시"라고 시켰는데 "수요일 오전 10시"라고 쓸 수도 있어요. 초안이니까 보내기 전에 반드시 한 통씩 훑어봐야 합니다. 특히 금액이 적힌 메일은 숫자를 꼭 확인하세요.

우리도 따라하려면

메일 내용을 텍스트 파일로 내보내는 법만 알면 됩니다. 지메일이면 메일을 열고 내용을 복사해서 메모장에 붙여넣으면 됩니다. 답장 톤과 기본 응답을 구체적으로 지시하는 게 핵심입니다. "알아서 답장해줘"보다 "견적은 이렇게, 일정은 이렇게"라고 정해주는 게 결과가 좋습니다.

사례 2) 회의 녹음 파일을 회의록으로

장면

목요일 오후 4시. 중소기업 기획팀 김팀장이 1시간 30분짜리 회의를 마쳤습니다. 회의 내용을 정리해야 하는데, 회의 중에 메모한 건 세 줄뿐입니다. 다행히 스마트폰으로 녹음은 해뒀습니다.

녹음 파일을 듣고 회의록을 쓰려면 보통 녹음 시간의 두 배가 걸립니다. 1시간 30분짜리 회의라면 회의록 작성에 3시간. 오늘 야근이 확정된 셈인데.

명령

김팀장은 녹음 파일(meeting_0513.m4a)을 컴퓨터로 옮깁니다. 에어드롭(AirDrop)으로 맥에 보내면 됩니다.

작업 폴더를 만들고 깃을 켭니다.

mkdir ~/meeting-notes

cd ~/meeting-notes

git init

녹음 파일을 이 폴더에 넣고, 클로드 코드를 켭니다.

"meeting_0513.m4a 파일을 텍스트로 변환한 다음, 회의록 형식으로 정리해줘. 참석자 발언을 구분해서 적고, 결정 사항과 후속 조치를 맨 아래에 따로 모아줘. 결과는 회의록_0513.txt로 저장해줘."

진행 과정

클로드 코드가 먼저 음성 파일을 텍스트로 바꿉니다. 이걸 전사(transcription, 트랜스크립션, 음성을 글로 옮기는 것)라고 합니다. 클로드 코드가 직접 하는 게 아니라, 위스퍼(Whisper, 위스퍼, OpenAI가 만든 음성 인식 도구) 같은 전사 도구를 설치해서 씁니다. 욜로 모드라서 도구 설치 허락을 묻지 않고 바로 진행합니다.

전사가 끝나면 날것의 텍스트가 나옵니다. "어 그래서 그 건은 좀 더 생각해보고요 아 근데 예산 부분은 좀 빠르게 처리해야 할 것 같은데" 같은 구어체 덩어리입니다.

클로드 코드가 이 텍스트를 정리합니다. 발언자를 구분하고(목소리만으로 100% 정확하게 구분하지 못할 수 있습니다. "발언자 A", "발언자 B"로 표시될 수도 있습니다), 구어체를 문어체로 바꾸고, 결정 사항과 후속 조치를 뽑아냅니다.

전체 과정에 8분에서 12분 정도 걸립니다. 녹음 길이에 따라 달라집니다.

결과

회의록_0513.txt 파일. 위쪽에는 회의 일시, 참석자(발언자 구분 기준), 안건이 적혀 있고, 중간에는 발언 내용이 시간 순서대로 정리되어 있고, 아래쪽에는 결정 사항 세 건과 후속 조치 네 건이 깔끔하게 모여 있습니다.

비용

시간 10분, 비용 1,500원에서 2,500원. 녹음 파일 길이가 길수록, 전사할 텍스트가 많을수록 비용이 올라갑니다.

위험

위험이 두 가지 있습니다.

하나는 전사 오류. 음성 인식은 100%가 아닙니다. 고유명사, 전문용어, 작은 목소리를 잘못 알아듣는 경우가 있습니다. "3분기 매출"을 "삼분기 매주"로 바꿔놓을 수 있어요. 회의록을 제출하기 전에 녹음과 대조하면서 한 번 읽어야 합니다.

다른 하나는 발언자 구분 오류. 회의에 참석한 사람이 다섯 명인데 AI가 세 명으로 구분하거나, 이 사람 발언을 저 사람 이름 아래 적어둘 수 있습니다. 참석자 이름을 미리 알려주면 정확도가 올라갑니다. "참석자는 김팀장, 이과장, 박대리, 최사원, 정인턴이야"라고 명령에 넣어두세요.

우리도 따라하려면

스마트폰 녹음 앱으로 회의를 녹음합니다. 아이폰의 '음성 메모' 앱이면 충분합니다. 녹음 파일을 컴퓨터로 옮기고, 작업 폴더에 넣고, 클로드 코드에게 시킵니다. 참석자 이름을 미리 알려주세요. 회의록은 제출 전에 반드시 한 번 읽으세요.

사례 3) 논문 10편 읽고 요약 보고서

장면

금요일 밤 11시. 대학원 석사과정 한소라 씨가 연구실 불을 켜고 있습니다. 다음 주 화요일 세미나에서 "한국 자영업자의 디지털 전환" 관련 선행연구를 발표해야 합니다. 교수님이 보내주신 논문이 열 편. 각각 20페이지에서 40페이지. 전부 합치면 300페이지가 넘습니다.

열 편을 다 읽고, 공통점과 차이점을 정리하고, 표로 만들고, 발표 자료까지 준비하려면 주말 내내 연구실에 갇혀 있어야 합니다.

명령

한소라 씨는 논문 PDF 파일 열 개를 papers 폴더에 넣습니다.

mkdir ~/paper-review

cd ~/paper-review

mkdir papers

git init

(논문 PDF를 papers 폴더에 복사합니다)

클로드 코드를 켭니다.

"papers 폴더에 있는 PDF 논문 10편을 읽고, 각 논문의 연구 목적, 연구 방법, 주요 결과, 한계점을 정리해줘. 그리고 10편의 공통된 발견과 서로 다른 결론을 비교한 종합 분석을 붙여줘. 결과는 선행연구_요약보고서.txt로 저장해."

진행 과정

클로드 코드가 논문을 한 편씩 읽기 시작합니다. PDF에서 텍스트를 뽑아내고, 내용을 파악하고, 요약합니다. 논문 한 편당 2분에서 3분 걸리니, 열 편이면 20분에서 30분입니다.

중간에 한 가지 문제가 생길 수 있습니다. PDF가 스캔 이미지로 된 경우입니다. 종이 논문을 스캔한 PDF는 텍스트가 아니라 그림이라서, AI가 글자를 읽지 못합니다. 이런 경우 클로드 코드가 OCR(오씨알, Optical Character Recognition, 이미지에서 글자를 인식하는 기술) 도구를 설치해서 처리합니다. 시간이 좀 더 걸립니다.

열 편의 개별 요약이 끝나면, 클로드 코드가 종합 분석을 작성합니다. 논문 간의 공통 발견("10편 중 7편이 '소셜 미디어 마케팅이 자영업 매출에 긍정적 영향을 준다'고 보고했다"), 서로 다른 결론("배달 앱의 효과에 대해서는 3편은 긍정적, 2편은 부정적으로 나뉘었다"), 전체적인 연구 동향이 정리됩니다.

결과

선행연구_요약보고서.txt 파일. 분량은 보통 A4 기준 8페이지에서 12페이지. 앞부분에 논문 10편의 개별 요약이 있고, 뒷부분에 종합 비교 분석이 있습니다.

비용

시간 30분, 비용 5,000원에서 8,000원. 논문이 길고 복잡할수록 비용이 올라갑니다. 사람이 했다면 주말 이틀은 꼬박 걸릴 일입니다.

위험

이 사례의 위험은 다른 실습과 성격이 다릅니다. 파일을 망가뜨리거나 비밀번호를 노출하는 종류의 위험이 아닙니다. 내용의 정확성입니다.

AI가 논문을 "읽었다"고 해서 사람처럼 이해한 건 아닙니다. 통계 수치를 잘못 읽거나, 인과 관계를 뒤바꿔 쓰거나, 저자의 주장과 반대로 요약하는 경우가 있습니다. 예를 들어 "배달 앱이 매출을 15% 떨어뜨렸다"는 논문을 "배달 앱이 매출 증가에 기여했다"로 요약할 수 있습니다.

그래서 AI가 만든 요약 보고서를 그대로 발표에 쓰면 안 됩니다. 이건 초안이지 완성품이 아닙니다. 보고서를 읽으면서, 핵심 수치와 결론이 원본 논문과 맞는지 확인해야 합니다.

AI가 해준 건 300페이지를 12페이지로 줄여준 겁니다. 12페이지를 읽고 확인하는 건 여러분 몫입니다. 그래도 300페이지를 처음부터 읽는 것보다는 훨씬 빠릅니다.

우리도 따라하려면

논문 PDF를 한 폴더에 모읍니다. 스캔 PDF가 아닌 텍스트 PDF인지 확인합니다(PDF를 열어서 글자를 드래그할 수 있으면 텍스트 PDF입니다). 요약 항목을 구체적으로 지정합니다. "요약해줘"보다 "연구 목적, 연구 방법, 주요 결과, 한계점을 정리해줘"가 훨씬 좋은 결과를 냅니다. 완성된 보고서는 반드시 원본과 대조합니다.

사례 4) 마크다운 원고를 출판용 PDF로 자동 변환

장면

일요일 오후. 블로그를 운영하는 프리랜서 작가 윤세현 씨가 전자책 원고를 마무리했습니다. 마크다운(Markdown, 마크다운, 글을 쓸 때 쓰는 가벼운 서식 언어)으로 써둔 원고가 12개 챕터, 합쳐서 180페이지 분량입니다. 이걸 PDF로 변환해서 전자책으로 팔려고 합니다.

마크다운 파일을 PDF로 바꾸는 도구는 여러 가지가 있는데, 표지, 목차, 페이지 번호, 글꼴 설정까지 맞추려면 설정을 만지는 데 반나절이 걸립니다.

명령

```
mkdir ~/ebook-build
```

```
cd ~/ebook-build
```

```
git init
```

원고 파일(chapter_01.md부터 chapter_12.md까지)을 이 폴더에 넣습니다.

"이 폴더에 있는 마크다운 파일 12개를 순서대로 합쳐서, 한국어 전자책 PDF로 만들어줘. 표지 페이지를 넣고 제목은 '커피와 키보드'로 해줘. 목차를 자동으로 만들고, 페이지 번호를 넣어줘. 글꼴은 나눔명조, 본문 크기 11포인트로. 결과 파일명은 커피와 키보드.pdf로 해줘."

진행 과정

클로드 코드가 판독(Pandoc, 판독, 문서 변환 도구)이라는 도구를 설치합니다. 마크다운 파일을 PDF로 바꾸는 데 널리 쓰이는 프로그램입니다. 나눔명조 글꼴이 컴퓨터에 없으면 그것도 설치합니다.

12개 파일을 순서대로 합치고, 표지 페이지를 앞에 넣고, 목차를 자동으로 만들고, 각 페이지 하단에 페이지 번호를 넣습니다. LaTeX(라텍, 레이텍, 조판 시스템)라는 조판 도구를 써서 PDF를 생성하는데, 이 과정에서 여러 개의 도구가 연쇄적으로 설치되고 실행됩니다.

기본 모드였다면 "Pandoc을 설치해도 될까요?", "LaTeX를 설치해도 될까요?", "나눔명조를 설치해도 될까요?" 이런 질문을 세 번 받았을 겁니다. 욜로 모드에서는 세 번 다 스킵합니다.

전체 과정에 5분에서 8분 걸립니다.

결과

커피와키보드.pdf 파일. 표지에 제목이 크게 나오고, 다음 페이지에 목차가 있고, 본문이 나눔명조 11포인트로 깔끔하게 조판되어 있습니다.

비용

시간 8분, 비용 1,000원에서 1,500원.

위험

도구 설치가 여러 번 일어나는 게 이 사례의 위험입니다. Pandoc, LaTeX, 글꼴. 이 세 가지가 컴퓨터에 설치됩니다. 설치 자체는 위험하지 않지만, 디스크 공간을 꽤 차지합니다. LaTeX 전체 패키지는 3기가바이트(GB) 이상일 수 있습니다. 디스크 여유가 적은 노트북이라면 미리 확인하세요.

PDF 변환 후에 글꼴이 깨지거나, 표가 페이지를 넘어가거나, 이미지 위치가 밀리는 경우가 있습니다. PDF를 열어서 처음부터 끝까지 한 번 훑어보는 게 좋습니다.

우리도 따라하려면

원고는 어떤 형식이든 괜찮습니다. 마크다운이 아니라 일반 텍스트 파일(.txt)이어도 됩니다. 워드 파일이어도 됩니다. 클로드 코드에게 "이 파일로 책을 만들어줘"라고 시키면 형식에 맞춰 알아서 변환합니다. 글꼴과 크기를 명령에 명시하면 좋고, 표지 디자인이 마음에 안 들면 "표지 배경색을 남색으로 바꿔줘" 같은 추가 명령으로 조정합니다.

실패 사례: AI가 원문을 빼먹고 요약한 건

어느 날, 석사과정 학생이 논문 7편의 요약 보고서를 AI에게 맡겼습니다. 결과물을 보니 논문이 6편만 요약되어 있었습니다. 7번째 논문이 통째로 빠져 있었습니다.

원인을 살펴보니, 7번째 논문의 PDF가 스캔 이미지 형식이었습니다. AI가 텍스트를 뽑아내지 못했고, 뽑아내지 못했다는 사실을 보고하지도 않았습니다. 에러 메시지 없이 조용히 건너뛴 겁니다.

이런 일이 생기는 이유가 있습니다. AI는 "못 하겠다"고 멈추기보다 "할 수 있는 것만 하고 넘어가는" 쪽으로 행동하는 경향이 있습니다. 사람이라면 "7번 논문이 읽히지 않는데요"라고 말했을 텐데, AI는 그냥 6편만 요약해서 "완료했습니다"라고 보고했습니다.

대비법은 간단합니다. 결과물의 건수를 세는 겁니다. 10편을 시켰으면 10편이 나왔는지 확인합니다. 53통의 답장을 시켰으면 53통이 있는지 셉니다. 숫자가 안 맞으면 어디가 빠졌는지 찾아야 합니다.

명령에 이 한 줄을 추가하면 더 좋습니다. "만약 읽을 수 없는 파일이 있으면, 건너뛰지 말고 파일 이름과 이유를 알려줘."

15장. 표와 숫자 다루기

사례 5) 가게 매출표 12개월치 자동 집계

장면

연말 결산 시기. 분식집을 운영하는 이사장님이 엑셀 파일 12개를 꺼냈습니다. 1월부터 12월까지, 매달 하나씩 만들어둔 매출표입니다. 각 파일에는 날짜, 메뉴, 수량, 금액이 적혀 있습니다. 이걸 한 파일로 합치고, 월별 합계와 연간 합계를 내야 합니다.

명령

"이 폴더에 있는 xlsx 파일 12개를 하나로 합쳐줘. 합친 다음에 월별 매출 합계와 연간 매출 합계를 계산해줘. 메뉴별 매출 순위도 만들어줘. 결과는 2025년_연간매출.xlsx로 저장해."

진행 과정

클로드 코드가 파이썬으로 스크립트를 만들고 실행합니다. 12개 파일의 데이터를 하나로 합치고, 월별 소계를 계산하고, 메뉴별로 매출을 정렬합니다. 2분.

결과

첫 번째 시트에 전체 데이터, 두 번째 시트에 월별 합계 표, 세 번째 시트에 메뉴별 매출 순위. 떡볶이가 연간 1위(4,200만 원), 순대가 2위(2,800만 원).

비용

시간 2분, 비용 400원.

위험

11장과 마찬가지로 원본 덮어쓰기 위험이 있습니다. 깃으로 먼저 저장하세요. 합계 숫자가 맞는지 한두 달치를 손으로 계산해서 대조해보세요.

우리도 따라하려면

월별 매출 파일의 열 이름이 제각각일 수 있습니다. 어떤 파일은 "메뉴"이고 어떤 파일은 "품목"이고 어떤 파일은 "상품명"이고. 걱정하지 마세요. 클로드 코드에게 "파일마다 열 이름이 다를 수 있으니까 알아서 맞춰서 합쳐줘"라고 시키면 됩니다. 아니면 합치기 전에 "12개 파일의 열 이름을 통일시켜줘"라고 먼저 시키는 방법도 있습니다.

사례 6) 거래처 명단 중복 항목 가려내기

장면

건자재 유통회사 영업팀. 거래처 명단 엑셀 파일에 회사 이름이 2,400건 들어 있습니다. 그런데 같은 회사가 여러 번 적혀 있는 것 같습니다. "한빛건설"과 "한빛 건설"과 "(주)한빛건설"이 각각 다른 행에 있거든요. 사람 눈으로 2,400건을 훑으면서 중복을 찾으려면 하루 종일 걸립니다.

명령

"거래처명단.xlsx 파일에서 중복된 회사 이름을 찾아줘. 띄어쓰기 차이, (주) 유무, 주식회사 표기 차이도 같은 회사로 봐줘. 중복 의심 목록을 duplicate_check.xlsx로 따로 저장해줘."

진행 과정

클로드 코드가 회사 이름을 정규화합니다. 공백 제거, "(주)" 제거, "주식회사" 제거 같은 전처리를 한 뒤, 비슷한 이름끼리 묶습니다. 유사도 알고리즘(algorithm, 알고리즘, 문제를 푸는 절차)을 써서 글자가 80% 이상 같은 회사를 중복 후보로 잡습니다.

결과

duplicate_check.xlsx 파일. 142건의 중복 의심 쌍이 나옵니다. 각 행에 "원래 이름 A", "원래 이름 B", "유사도 점수"가 적혀 있습니다.

비용

시간 1분, 비용 500원.

위험

AI가 다른 회사를 같은 회사로 묶을 수 있습니다. "한빛건설"과 "한빛건재"는 다른 회사인데, 글자가 비슷하다고 중복으로 잡을 수 있어요. 그래서 "중복입니다"가 아니라 "중복 의심 목록"으로 뽑아달라고 한 겁니다. 최종 판단은 사람이 합니다.

우리도 따라하려면

거래처 이름이 들어 있는 열의 이름을 알려주세요. "B열에 회사 이름이 있어"라고요. 유사도 기준을 조절할 수 있습니다. "90% 이상만 잡아줘"라고 하면 목록이 줄어들고, "70% 이상"이라고 하면 늘어납니다.

사례 7) 영수증 사진에서 숫자 뽑아 엑셀로

장면

세금 신고 시기가 다가왔습니다. 1년 동안 모아둔 영수증이 서랍 한 칸을 가득 채우고 있습니다. 카드 영수증, 현금 영수증, 간이 영수증. 합치면 200장이 넘습니다. 이걸 하나하나 엑셀에 입력하는 건 이틀짜리 작업입니다.

명령

영수증을 스마트폰으로 사진 찍습니다. 한 장에 영수증 하나씩. 200장의 사진을 receipts 폴더에 넣습니다.

"receipts 폴더에 있는 영수증 사진 200장을 읽고, 각 영수증에서 날짜, 가게 이름, 금액, 결제 방법을 뽑아줘. 결과를 영수증_정리.xlsx 파일로 만들어줘. 날짜순으로 정렬해줘."

진행 과정

클로드 코드가 각 사진을 읽습니다. OCR 기술로 사진 속 글자를 인식하고, 날짜가 어디 있고 금액이 어디 있는지 파악합니다. 영수증 한 장당 10초에서 15초 정도 걸리니, 200장이면 30분에서 50분입니다. 컴퓨터가 일하는 동안 다른 일을 하시면 됩니다.

결과

영수증_정리.xlsx 파일. 200행이 날짜순으로 정렬되어 있고, 각 행에 날짜, 가게 이름, 금액, 결제 방법이 적혀 있습니다. 맨 아래에 합계도 들어 있습니다.

비용

시간 40분(대부분 컴퓨터가 혼자 일하는 시간), 비용 4,000원에서 6,000원.

위험

영수증 인식의 정확도는 사진 품질에 따라 크게 달라집니다. 선명하게 찍은 카드 영수증은 95% 이상 맞는데, 구겨지거나 바랜 간이 영수증은 70%까지 떨어집니다. 금액의 숫자 하나가 틀리면 세금 신고에 문제가 생길 수 있으니, 금액이 큰 영수증(10만 원 이상)은 원본과 대조해야 합니다.

사진을 찍을 때 팁이 있습니다. 바닥에 영수증을 놓고, 그림자가 생기지 않게 위에서 수직으로 찍으세요. 비스듬히 찍으면 글자가 찌그러져서 인식률이 떨어집니다.

우리도 따라하려면

영수증 사진을 선명하게 찍는 게 절반입니다. 나머지 절반은 결과를 확인하는 겁니다. 200건 전부를 대조할 수는 없으니, 무작위로 20건(10%)을 골라서 원본과 비교하세요. 오류율이 5% 이하면 괜찮은 수준입니다.

사례 8) 유튜브 댓글 200개 감정 분석 후 파이 차트 엑셀 생성

장면

소규모 화장품 브랜드를 운영하는 최대표가 유튜브에 신제품 리뷰 영상을 올렸습니다. 일주일 만에 댓글이 237개 달렸습니다. "향이 좋아요", "가격이 좀...", "포장이 예뻐서 선물용으로 샀어요", "빨리 품절될 것 같다", "이번엔 별로". 하나하나 읽어볼 수는 있지만, 전체적으로 반응이 좋은 건지 나쁜 건지 한눈에 파악하고 싶습니다.

명령

먼저 유튜브 댓글을 텍스트 파일로 가져옵니다. 유튜브의 댓글을 복사하거나, yt-dlp(와이티 디엘피, 유튜브 영상과 정보를 내려받는 도구) 같은 도구로 댓글을 추출할 수 있습니다. 클로드 코드에게 이것도 시킬 수 있습니다.

"이 유튜브 영상 URL의 댓글을 전부 가져와서 comments.txt로 저장해줘."

댓글이 저장되면, 분석을 시킵니다.

"comments.txt에 있는 댓글 237개를 감정 분석해줘. 긍정, 부정, 중립으로 나누고, 각 카테고리의 비율을 파이 차트로 만들어줘. 부정 댓글에서 자주 나오는 키워드 다섯 개도 뽑아줘. 결과를 댓글분석.xlsx에 저장해줘. 파이 차트는 엑셀 안에 넣어줘."

진행 과정

클로드 코드가 댓글을 한 줄씩 읽고 감정을 분류합니다. "향이 좋아요"는 긍정, "가격이 좀..."은 부정, "포장이 예뻐서 선물용으로 샀어요"는 긍정. 감정 분류가 끝나면 비율을 계산합니다. 긍정 62%, 부정 23%, 중립 15% 같은 식으로요.

파이썬으로 엑셀 파일을 만들고, openpyxl로 파이 차트를 엑셀 시트 안에 삽입합니다. 부정 댓글에서 자주 나오는 키워드("가격", "용량", "배송", "향 지속력", "트러블")를 따로 뽑아서 다른 시트에 정리합니다.

전체 과정에 5분에서 8분 걸립니다.

결과

댓글분석.xlsx 파일. 시트 1에 댓글 전체와 각 댓글의 감정 분류(긍정/부정/중립). 시트 2에 파이 차트와 비율 요약. 시트 3에 부정 댓글 키워드 분석. 최대표가 이 파일을 열면, 한 눈에 "긍정이 62%니 반응이 나쁘지 않구나, 그런데 가격과 용량에 불만이 있구나"를 파악할 수 있습니다.

비용

시간 8분, 비용 1,500원에서 2,500원.

위험

감정 분석의 정확도는 80% 안팎입니다. 문제는 비꼬는 댓글입니다. "와 진짜 대단하다 이 가격에 이 용량이라니"는 부정인데 AI가 긍정으로 분류할 수 있습니다. 한국어의 반어법을 AI가 못 잡는 경우가 꽤 있습니다.

완벽한 분석은 어렵습니다. 전체 흐름을 파악하는 용도로 쓰시고, 정확한 수치에 매달리지 마세요.

우리도 따라하려면

유튜브 댓글을 텍스트로 가져오는 게 첫 단계입니다. 클로드 코드에게 "이 유튜브 URL의 댓글을 가져와줘"라고 시킬 수 있습니다. 감정 분류 기준을 미리 알려주면 좋습니다. "가격 관련 불만은 부정으로, 단순 질문은 중립으로 봐줘"처럼요.

실패 사례: 숫자를 잘못 읽은 AI

영수증 사진 100장을 AI에게 맡긴 사람이 있었습니다. 결과 엑셀을 열어보니 합계가 이상했습니다. 원래 한 달 지출이 350만 원 정도인데, AI가 만든 합계는 830만 원이었습니다.

원인을 찾아보니, 영수증 한 장에서 "35,000원"을 "350,000원"으로 읽은 게 세 건 있었습니다. 콤마 위치를 잘못 인식한 겁니다. 또 "8,500원"을 "85,000원"으로 읽은 것도 두 건 있었습니다. 바랜 영수증에서 0 하나가 추가된 셈이죠.

숫자를 다루는 작업에서 AI를 쓸 때 기억해야 할 것이 있습니다. AI는 "대충 맞추는" 게 아니라 "확신을 가지고 틀립니다". 35,000원을 350,000원이라고 확신에 차서 적어놓거든요. 물음표를 달아주지 않습니다.

대비법: 합계를 먼저 봅니다. 합계가 예상 범위 안에 있는지 확인합니다. 벗어나면 큰 금액부터 대조합니다.

16장. 웹사이트 만들기와 자동화

사례 9) 개인 블로그, 아이디어부터 배포까지

장면

대학생 박지호 씨는 맥주 리뷰를 적는 블로그를 만들고 싶었습니다. 네이버 블로그나 티스토리를 쓸 수도 있지만, 자기만의 도메인(domain, 도메인, 인터넷 주소)을 갖고 싶었습니다. beerlog.kr 같은 주소요. 그런데 웹사이트를 직접 만들어본 적이 없습니다. HTML이라는 단어는 들어봤지만, 코드를 써본 적은 없습니다.

명령

mkdir ~/beerlog

cd ~/beerlog

git init

클로드 코드를 켜고 이렇게 시킵니다.

"맥주 리뷰 블로그를 만들어줘. 이름은 '맥주 한 잔의 기록'. 홈페이지에는 글 목록이 나오고, 글을 클릭하면 리뷰 본문이 보이게 해줘. 리뷰에는 맥주 이름, 양조장, 별점(5점 만점), 사진 한 장, 본문이 들어가. 디자인은 깔끔하고 어두운 톤으로. 마크다운 파일로 글을 쓰면 자동으로 페이지가 만들어지게 해줘."

진행 과정

클로드 코드가 정적 사이트 생성기(Static Site Generator, 스태틱 사이트 제너레이터, 마크다운 파일을 웹페이지로 바꿔주는 도구)를 설정합니다. 여러 도구 중에 휴고(Hugo, 휴고)나 일레븐티(Eleventy, 일레븐티) 같은 걸 골라서 설치합니다.

폴더 구조를 만들고, 테마(theme, 테마, 디자인 틀)를 설정하고, 샘플 리뷰 글을 하나 만들어줍니다. 홈 페이지에 글 목록이 나오고, 글을 클릭하면 본문이 보이는 구조입니다.

로컬(local, 로컬, 내 컴퓨터)에서 미리보기를 띄워줍니다.

hugo server

이 명령을 치면 브라우저에서 http://localhost:1313 주소로 블로그를 볼 수 있습니다. 내 컴퓨터 안에서만 보이는 거라 다른 사람은 못 봅니다.

여기까지가 15분 정도입니다.

배포(deploy, 디플로이, 인터넷에 올리기)까지 하려면, 클로드 코드에게 이어서 시킵니다.

"이 블로그를 Vercel에 배포해줘."

버셀(Vercel, 버셀, 웹사이트를 인터넷에 올려주는 서비스)은 개인 프로젝트라면 무료로 쓸 수 있습니다. 클로드 코드가 깃허브(GitHub, 깃허브, 코드 저장소 서비스) 저장소를 만들고, 코드를 올리고, 버셀과 연결합니다.

배포까지 합치면 25분에서 30분 걸립니다.

결과

인터넷에 공개된 블로그. beerlog.vercel.app 같은 주소로 누구나 접속할 수 있습니다. 자기 도메인(beerlog.kr)을 연결하려면 도메인을 별도로 사들여야 하는데, 연간 15,000원 정도입니다.

비용

시간 30분, 비용 3,000원에서 5,000원. 서버 운영비는 무료(버셀 무료 플랜). 도메인을 사면 연간 15,000원 추가.

위험

배포, 그러니까 인터넷에 올리는 순간 위험이 생깁니다. 코드에 개인 정보가 들어 있으면 전 세계에 공개됩니다. API 키, 비밀번호, 개인 이메일이 코드 안에 하드코딩(hard-coding, 하드코딩, 코드에 직접 적어 넣기) 되어 있으면 큰 문제가 됩니다. 이 사례는 개인 블로그라서 비밀 정보가 들어갈 일이 적지만, 습관적으로 확인하세요.

배포 전에 클로드 코드에게 이렇게 시킵니다. "이 폴더 안의 모든 파일에서 비밀번호, API 키, 토큰 같은 민감한 정보가 노출된 곳이 없는지 확인해줘." 클로드 코드가 파일을 하나씩 뒤져서 password, secret, api_key, token 같은 단어가 들어간 줄을 찾아줍니다. 아무것도 안 나오면 안전합니다.

우리도 따라하려면

블로그의 목적과 디자인 톤을 구체적으로 알려주세요. "블로그 만들어줘"보다 "맥주 리뷰 블로그, 어두운 톤, 별점 포함"이 결과가 좋습니다. 배포는 버셀이나 넷리파이(Netlify, 넷리파이) 같은 무료 서비스를 쓰세요. 깃허브 계정이 필요합니다(무료입니다).

사례 10) 동네 모임 일정표 사이트

장면

아파트 단지 독서 모임을 이끄는 김영숙 씨. 매달 모임 날짜와 읽을 책을 카카오톡 단체방에 올리는데, 새 회원이 들어오면 이전 정보를 다시 보내줘야 합니다. 과거 기록도 찾기 어렵고요. 간단한 웹페이지에 일정을 올려두면 링크 하나로 해결될 텐데.

명령

"독서 모임 일정표 웹사이트를 만들어줘. 페이지 하나짜리면 돼. 위쪽에 모임 이름 '함께 읽는 오후', 그 아래에 다음 모임 정보(날짜, 장소, 읽을 책), 그 아래에 지난 모임 기록. 데이터는 JSON 파일에서 읽어오게 해줘. 내가 JSON 파일만 수정하면 웹페이지가 자동으로 바뀌게."

진행 과정

클로드 코드가 HTML과 CSS(씨에스에스, 웹페이지의 디자인을 정하는 언어)로 페이지를 만들고, schedule.json이라는 데이터 파일을 만듭니다. JSON(제이슨, JavaScript Object Notation, 데이터를 저장하는 형식) 파일은 사람이 읽을 수 있는 텍스트 파일이라서, 메모장으로 열어서 날짜와 책 이름을 바꿀 수 있습니다.

10분이면 완성됩니다. 버셀이나 깃허브 페이지에 올리면 인터넷 주소가 생깁니다. 이 주소를 카카오톡 단체방에 올리면 됩니다.

결과

한 페이지짜리 깔끔한 일정표 사이트. 다음 모임 정보가 눈에 띄게 위쪽에 있고, 아래로 스크롤하면 지난 기록이 나옵니다.

비용

시간 10분, 비용 500원, 서버 운영비 무료.

위험

사이트가 공개되어 있으니 모임 장소의 상세 주소(아파트 동호수 같은)를 적으면 안 됩니다. "단지 내 경로당"처럼 내부인만 아는 표현을 쓰세요.

우리도 따라하려면

모임 이름, 분위기, 필요한 정보(날짜, 장소, 주제 등)를 알려주세요. JSON 파일 수정법을 모르면 클로드 코드에게 물어보세요. "schedule.json에 7월 모임을 추가하려면 어떻게 해?"라고요.

사례 11) 작은 가게의 예약 시스템

장면

서울 연남동의 소규모 꽃집. 사장님이 인스타그램 DM으로 예약을 받고 있었습니다. "금요일 오후 3시에 꽃다발 하나 만들어주실 수 있나요?" 같은 메시지가 하루에 열 건 넘게 옵니다. 대답하고, 수첩에 적고, 겹치는 시간이 있는지 확인하고. DM이 밀리면 놓치는 예약도 생깁니다.

예약 시스템이 있으면 좋겠는데, 네이버 예약이나 카카오 예약은 수수료가 붙습니다. 직접 만들면 수수료가 없는데, 개발자를 고용하면 200만 원은 든다고 합니다.

명령

"꽃집 예약 시스템을 만들어줘. 고객이 웹페이지에서 날짜와 시간을 고르고, 이름과 전화번호를 입력하면 예약이 저장되게 해줘. 같은 시간에 두 명이 예약하면 '이미 예약된 시간입니다'라고 알려줘. 예약 목록은 관리자 페이지에서 볼 수 있게 해줘. 관리자 비밀번호는 flower2025로 설정해줘."

진행 과정

이 사례는 앞의 사례들보다 복잡합니다. 데이터를 저장하고 불러와야 하니까요. 클로드 코드가 프론트엔드(frontend, 프론트엔드, 사용자가 보는 화면)와 백엔드(backend, 백엔드, 데이터를 처리하는 뒷단)를 다 만듭니다.

프론트엔드: 캘린더에서 날짜를 고르고, 가능한 시간대를 보여주고, 이름과 전화번호를 입력하는 화면.

백엔드: 예약 정보를 데이터베이스에 저장하고, 같은 시간에 중복 예약을 막는 로직.

관리자 페이지: 비밀번호를 입력하면 오늘의 예약 목록이 나오는 화면.

클로드 코드가 넥스트제이에스(Next.js, 넥스트제이에스, 웹 프레임워크)와 수파베이스(Supabase, 수파베이스, 무료 데이터베이스 서비스)를 조합해서 만들 수 있습니다. 둘 다 무료 플랜이 있어서 소규모 가게에는 비용이 안 듭니다.

전체 과정에 40분에서 1시간 걸립니다. 다른 사례보다 오래 걸리지만, 개발자에게 외주를 맡기면 며칠은 걸리고 비용도 수십만 원에서 수백만 원 사이입니다.

결과

인터넷에 공개된 예약 페이지. 인스타그램 프로필에 이 링크를 걸어두면, 고객이 직접 예약합니다. 사장님은 관리자 페이지에서 예약 목록을 확인합니다.

비용

시간 1시간, API 비용 5,000원에서 8,000원. 서버 운영비 월 0원(무료 플랜 범위 내). 도메인을 사면 연간 15,000원.

위험

이 사례의 위험은 진짜 주의해야 합니다. 예약 시스템은 고객의 이름과 전화번호를 저장합니다. 개인 정보입니다. 이 데이터가 유출되면 법적 문제가 생길 수 있습니다.

수파베이스는 기본적으로 데이터를 암호화해서 저장하지만, 관리자 페이지의 비밀번호가 "flower2025"처럼 쉬우면 누군가 뚫을 수 있습니다. 비밀번호를 길고 복잡하게 바꾸세요. "Fl0w3r!2025#SeouL" 처럼요.

코드에 비밀번호가 직접 적혀 있지 않은지 확인하세요. 환경 변수(environment variable, 환경 변수, 비밀 정보를 코드 밖에 저장하는 방법)에 넣어야 합니다. 이건 16장 끝의 실패 사례에서 자세히 다룹니다.

우리도 따라하려면

예약 시스템을 만들 때는 수파베이스 같은 외부 서비스를 쓸 줄 몰라도 됩니다. 클로드 코드가 설정을 다 해줍니다. 다만 수파베이스 웹사이트에서 무료 계정을 먼저 만들어야 합니다. 계정을 만들면 API 키가 나오는데, 이 키를 클로드 코드에게 알려주면 연결을 해줍니다. 서비스를 운영하면서 예약이 하루 100건을 넘으면 유료 플랜으로 올라가야 할 수 있습니다.

사례 12) 브라우저를 열고 직접 테스트까지 하는 AI

장면

웹사이트를 만들었습니다. 잘 돌아가는 것 같은데, 버튼을 누르면 진짜 동작하는지, 스마트폰 화면에서 글자가 잘리지 않는지 확인하고 싶습니다. 사람이 하나하나 클릭해가며 확인하는 건 번거롭습니다.

명령

"방금 만든 웹사이트를 브라우저에서 열고, 모든 버튼과 링크를 클릭해봐. 오류가 나는 곳이 있으면 알려주고 고쳐줘. 모바일 화면 크기(375px)에서도 확인해줘."

진행 과정

여기서 브라우저 유즈(Browser Use, 브라우저 유즈, AI가 브라우저를 직접 조작하는 도구)라는 기능이 등장합니다. AI가 진짜 브라우저를 열고, 마치 사람처럼 마우스를 움직이고, 버튼을 클릭하고, 텍스트를 입력합니다.

클로드 코드가 플레이라이트(Playwright, 플레이라이트, 브라우저 자동화 도구)를 써서 브라우저를 엽니다. 웹페이지를 로드하고, 버튼을 하나씩 클릭합니다. 링크가 깨진 곳을 발견하면 보고합니다. "연락처 버튼을 누르면 404 에러가 납니다." 그리고 알아서 고칩니다.

모바일 크기로 화면을 줄여서도 확인합니다. 글자가 화면 밖으로 삐져나가면 CSS를 수정합니다.

전체 과정에 5분에서 10분.

결과

테스트가 끝난 웹사이트. 버튼이 다 동작하고, 모바일에서도 레이아웃이 깨지지 않습니다. AI가 수정한 내용 목록이 터미널에 표시됩니다.

비용

시간 10분, 비용 1,000원에서 2,000원.

위험

브라우저 유즈는 AI가 인터넷에 접속하는 기능입니다. 12장의 뉴스 요약 사례와 마찬가지로, 도커 안에서 돌리는 게 안전합니다. AI가 테스트 중에 외부 사이트에 접속하거나, 예상치 못한 페이지를 열 수 있습니다.

우리도 따라하려면

웹사이트를 만든 직후에 "브라우저에서 테스트해줘"라고 시키면 됩니다. 플레이라이트가 자동으로 설치됩니다. 테스트 범위를 구체적으로 알려주세요. "모든 버튼, 모든 링크, 모바일 크기"처럼요.

실패 사례: 배포했는데 비밀번호가 코드에 그대로 노출

꽃집 사장님의 이야기입니다(사례 11의 후일담). 예약 시스템을 만들고, 깃허브에 코드를 올리고, 버셀에 배포했습니다. 잘 돌아갔습니다. 2주 동안 문제없이 예약을 받았습니다.

그런데 어느 날 관리자 페이지에 들어가보니, 예약 목록이 비어 있었습니다. 누군가 관리자 페이지에 접속해서 예약 데이터를 전부 지운 겁니다.

원인은 이랬습니다. 코드 안에 관리자 비밀번호 "flower2025"가 그대로 적혀 있었습니다. 깃허브에 올린 코드는 누구나 볼 수 있습니다(공개 저장소였거든요). 누군가 깃허브에서 코드를 보고, 비밀번호를 찾고, 관리자 페이지에 들어간 겁니다.

AI에게 "관리자 비밀번호는 flower2025로 설정해줘"라고 시켰으니, AI는 그 비밀번호를 코드에 적었습니다. AI 잘못이 아닙니다. 시키는 대로 한 겁니다. 비밀번호를 코드 밖에 저장하라고 알려주지 않은 사람의 실수입니다.

이런 비밀 정보는 환경 변수 파일(.env 파일)에 저장하고, 이 파일을 깃허브에 올리지 않아야 합니다. .gitignore(깃이그노어, 깃이 무시할 파일 목록) 파일에 .env를 추가하면 됩니다.

방지법: AI에게 시킬 때 이 한 줄을 추가하세요. "비밀번호와 API 키는 .env 파일에 넣고, .gitignore에 .env를 추가해줘."

17장. 사무실에서 쓰는 자동화

사례 13) 지메일 받은편지함 분류하고 답장 초안

장면

월요일 아침 9시. 스타트업 운영팀장 오지현 씨가 지메일을 열었습니다. 주말 동안 쌓인 메일이 87통. 광고 메일, 뉴스레터, 거래처 연락, 팀원 보고, 고객 문의가 뒤섞여 있습니다. 이걸 분류하고, 급한 건 바로 답하고, 나중에 볼 건 별표를 달아야 합니다.

명령

지메일 API(에이피아이, 프로그램끼리 통신하는 통로)를 통해 메일을 가져올 수 있습니다. 구글에서 API 키를 발급받아야 하는데, 처음 한 번만 설정하면 됩니다.

"내 지메일 받은편지함에서 안 읽은 메일을 전부 가져와서, 카테고리별로 분류해줘. 카테고리는 '긴급', '거래처', '팀 내부', '뉴스레터', '기타'로 나눠줘. 긴급 메일에는 답장 초안을 만들어줘. 결과를 mail_summary_0513.txt로 저장해줘."

진행 과정

클로드 코드가 지메일 API로 메일 87통을 가져옵니다. 제목과 본문을 읽고 카테고리를 분류합니다. "결제 건 확인 부탁드립니다"는 거래처, "월요일 주간 보고입니다"는 팀 내부, "주간 뉴스레터"는 뉴스레터.

긴급으로 분류된 메일 5통에 대해 답장 초안을 작성합니다. "확인했습니다. 오늘 중으로 처리하겠습니다" 같은 간결한 답장입니다.

전체 과정에 5분.

결과

mail_summary_0513.txt 파일. 카테고리별로 메일이 정리되어 있고, 긴급 메일 옆에는 답장 초안이 붙어 있습니다. 오지현 씨는 이 파일을 훑어보면서 급한 건 바로 답하고, 나머지는 시간 될 때 보면 됩니다.

비용

시간 5분, 비용 1,500원에서 2,500원.

위험

메일에는 회사 기밀이 들어 있을 수 있습니다. 클로드 API를 쓰면 메일 내용이 앤스로픽 서버를 거칩니다. 회사 보안 정책에 따라 이게 허용되지 않을 수 있습니다. 중요 기밀이 담긴 메일이 있다면 이 방법을 쓰기 전에 정보 보안 담당자와 상의하세요.

우리도 따라하려면

구글 클라우드 콘솔에서 지메일 API를 활성화하고, 인증 정보를 발급받아야 합니다. 이 과정이 처음에 좀 복잡한데, 클로드 코드에게 "지메일 API 설정하는 법 알려줘"라고 물어보면 단계별로 안내해줍니다. 한 번 설정하면 이후로는 계속 쓸 수 있습니다.

사례 14) 슬랙 채널 잡담 정리해서 요약 보내기

장면

금요일 오후 5시. 한 주간 슬랙(Slack, 슬랙, 업무용 메신저) 채널에 쌓인 메시지가 400건 넘습니다. 대부분은 잡담이고, 중요한 결정사항은 다섯 건 정도인데 어디에 묻혀 있는지 모릅니다.

명령

"슬랙 #general 채널에서 이번 주 월요일부터 오늘까지의 메시지를 가져와서, 업무 관련 내용만 골라 요약해줘. 결정사항, 할 일, 공유된 링크를 따로 정리해줘. 결과를 #general 채널에 '이번 주 요약'이라는 제목으로 보내줘."

진행 과정

클로드 코드가 슬랙 API로 메시지를 가져오고, 잡담과 업무 내용을 구분하고, 요약을 작성합니다. 3분.

결과

슬랙 채널에 요약 메시지가 올라옵니다. "이번 주 결정사항: A 프로젝트 일정 2주 연기, B 거래처 계약 갱신 완료, C 기능 우선순위 변경…" 같은 내용입니다.

비용

시간 3분, 비용 800원.

위험

사례 13과 같은 보안 문제가 있습니다. 슬랙 메시지 내용이 외부 서버를 거칩니다.

우리도 따라하려면

슬랙 앱을 만들어서 API 토큰을 발급받아야 합니다. 슬랙 관리자 권한이 필요합니다.

사례 15) 노션 페이지 자동 갱신

장면

팀 회의록을 노션(Notion, 노션, 문서 관리 도구)에 정리하고 있습니다. 매주 회의록 파일이 하나씩 추가되는데, "회의록 모음" 페이지에 새 회의록 링크를 수동으로 추가해야 합니다. 깜빡하고 안 하면 나중에 모아서 해야 하는데, 이것도 귀찮습니다.

명령

"노션 API로 '회의록' 데이터베이스에 새 항목이 추가되면, '회의록 모음' 페이지에 자동으로 링크를 추가해줘. 매일 오후 6시에 확인하고 업데이트해줘."

진행 과정

클로드 코드가 노션 API를 연결하고, "회의록" 데이터베이스를 조회하고, "회의록 모음" 페이지를 업데이트하는 스크립트를 만듭니다. 이 스크립트를 매일 오후 6시에 자동 실행되도록 설정합니다.

결과

매일 오후 6시에 "회의록 모음" 페이지가 자동으로 갱신됩니다.

비용

시간(설정) 10분, 이후 매일 자동 실행에 100원 안팎.

위험

노션 API 키가 유출되면 누군가 내 노션을 들여다볼 수 있습니다. API 키는 환경 변수에 저장하세요.

우리도 따라하려면

노션 개발자 포털(https://developers.notion.com)에서 통합(Integration)을 만들고, API 키를 발급받아야 합니다. 해당 데이터베이스에 통합을 연결해야 접근할 수 있습니다.

사례 16) 경쟁사 웹사이트 변화 추적 보고서

장면

온라인 쇼핑몰을 운영하는 김사장님. 경쟁사 세 곳의 웹사이트를 매주 한 번씩 확인합니다. 가격이 바뀌었는지, 신제품이 올라왔는지, 이벤트를 하고 있는지. 세 사이트를 돌아보는 데 한 시간이 걸리고, 변화를 정리하는 데 30분이 더 걸립니다.

명령

"다음 세 개의 웹사이트를 매주 월요일 아침 7시에 방문해줘.

1) competitor-a.com/products

2) competitor-b.com/new-arrivals

3) competitor-c.com/events

각 사이트의 내용을 저장하고, 지난주와 비교해서 바뀐 점을 정리해줘. 결과를 경쟁사_변화_이번주.txt로 저장해줘."

진행 과정

클로드 코드가 웹 스크래핑(web scraping, 웹 스크래핑, 웹사이트의 내용을 자동으로 읽어오는 것) 스크립트를 만듭니다. 매주 세 사이트의 내용을 가져와서 텍스트로 저장하고, 지난주 저장본과 비교합니다. 새로 추가된 제품, 가격이 바뀐 항목, 새 이벤트를 찾아서 보고서로 정리합니다.

결과

매주 월요일 아침에 경쟁사_변화_이번주.txt 파일이 생깁니다. "A사: 핸드크림 신제품 출시(15,000원), B사: 전품목 20% 할인 이벤트(~5/20), C사: 변화 없음" 같은 내용입니다.

비용

시간(설정) 15분, 이후 매주 자동 실행에 1,000원.

위험

웹 스크래핑이 허용되지 않는 사이트가 있습니다. 사이트의 이용약관에 "자동 수집 금지"라고 적혀 있으면 법적 문제가 생길 수 있습니다. 확인하고 쓰세요. robots.txt(로봇스텍스트, 웹사이트가 자동 수집 허용 범위를 적어둔 파일)를 확인하는 게 좋습니다.

사이트 구조가 바뀌면 스크래핑이 깨집니다. "오류가 발생했습니다"라는 보고서가 오면 스크립트를 수정해야 합니다.

우리도 따라하려면

경쟁사 웹사이트 주소와, 그 사이트에서 어떤 정보를 추적하고 싶은지(가격, 신제품, 이벤트)를 구체적으로 알려주세요. 도커 안에서 돌리는 게 안전합니다.

실패 사례: 중요한 이메일을 스팸으로 분류한 건

사례 13처럼 지메일 분류를 AI에게 맡긴 사람이 있었습니다. 일주일 동안 잘 돌아갔는데, 금요일에 거래처 사장님한테 전화가 왔습니다.

"지난 화요일에 보낸 계약서 메일, 확인하셨어요?"

확인한 적 없습니다. AI가 그 메일을 "뉴스레터"로 분류해서 눈에 안 띈 겁니다. 거래처 사장님이 보낸 메일의 제목이 "5월 뉴스레터 관련 협업 제안"이었거든요. 제목에 "뉴스레터"라는 단어가 들어 있어서 AI가 뉴스레터로 분류한 겁니다.

사람이라면 보낸 사람을 보고 "아, 거래처 사장님이 보낸 건데 뉴스레터는 아니겠지"라고 판단했을 겁니다. AI는 제목의 키워드에 더 크게 반응했습니다.

방지법: 중요 거래처의 이메일 주소를 미리 알려주세요. "이 주소에서 온 메일은 무조건 '긴급'으로 분류해"라고요. 그리고 AI가 분류한 결과를 하루에 한 번은 전체적으로 훑어보세요. 분류 자동화는 편하지만, 100% 맡기면 이런 사고가 납니다.

18장. 밤사이 큰일을 끝내는 법

사례 17) 자는 동안 코드 100개 파일 구조 바꾸기

장면

수요일 밤 11시. 개발팀 리드 장서준 씨가 모니터 앞에서 고민하고 있었습니다. 회사의 웹 서비스 코드가 오래되었습니다. 파일 100개에 걸쳐서 같은 패턴이 반복되고 있는데, 이걸 새로운 구조로 바꿔야 합니다. 사람이 하면 일주일 걸리는 작업입니다. 하나씩 파일을 열고, 코드를 수정하고, 테스트하고.

장서준 씨는 다른 방법을 쓰기로 했습니다. AI에게 시키고 자러 가기.

명령

먼저 깃에서 새 브랜치(branch, 브랜치, 가지치기)를 만듭니다.

```
git checkout -b refactor-structure
```

원본 코드에 손을 대지 않고, 가지를 하나 쳐서 그 위에서 작업하겠다는 뜻입니다. 잘못되면 이 가지를 잘라내면 그만이니까요.

클로드 코드를 켭니다.

"src 폴더 안의 모든 .ts 파일에서 옛날 방식의 import 구문을 새 방식으로 바꿔줘. 구체적으로, 'import { X } from '../utils/helpers''를 'import { X } from '@/utils/helpers''로 바꿔줘. 바꾼 뒤에 각 파일이 오류 없이 컴파일되는지 확인해줘. 컴파일 안 되는 파일이 있으면 원래대로 돌려놓고 목록을 알려줘."

진행 과정

클로드 코드가 src 폴더를 탐색합니다. .ts 파일이 112개 있습니다. 파일을 하나씩 열고, 해당 패턴을 찾고, 바꿉니다. 바꾼 뒤에 타입스크립트(TypeScript, 타입스크립트, 프로그래밍 언어) 컴파일러를 돌려서 오류가 없는지 확인합니다.

오류가 나면 해당 파일을 원래대로 돌리고, "이 파일은 수동 확인이 필요합니다"라고 목록에 적습니다.

112개 파일을 처리하는 데 45분에서 1시간 걸립니다. 장서준 씨는 클로드 코드에게 시키고 잠자리에 들었습니다.

결과

목요일 아침 8시. 장서준 씨가 출근해서 터미널을 봅니다. "112개 파일 중 108개 변환 완료, 4개 파일은 수동 확인 필요"라는 메시지가 떠 있습니다.

git diff를 쳐서 뭐가 바뀌었는지 확인합니다. 108개 파일의 import 구문이 새 방식으로 바뀌어 있습니다. 4개 파일은 변환하면 다른 곳에서 오류가 나서 원래대로 두었다는 설명이 붙어 있습니다.

장서준 씨는 4개 파일만 직접 확인하고 수정합니다. 30분이면 끝납니다. 일주일 걸릴 작업이 하룻밤에 끝난 셈입니다.

비용

시간 1시간(AI가 일한 시간), 비용 15,000원에서 25,000원. 사람이 일주일 동안 했다면 인건비가 수백만 원이었을 겁니다.

위험

밤사이 AI가 코드를 대량으로 수정하는 건 위험이 큽니다. 그래서 장서준 씨가 한 세 가지가 중요합니다.

브랜치를 만들었습니다. 원본 코드를 건드리지 않았습니다. 잘못되면 브랜치를 삭제하면 됩니다.

컴파일 확인을 시켰습니다. "바꾸기만 해"가 아니라 "바꾸고 확인해"라고 시켰습니다.

실패한 파일을 원래대로 돌려놓으라고 시켰습니다. AI가 억지로 끼워 맞추지 않게요.

이 세 가지 중 하나라도 빠졌다면, 아침에 출근해서 엉망이 된 코드를 보게 됐을 수 있습니다.

우리도 따라하려면

대규모 코드 수정을 밤에 맡기려면, 반드시 새 브랜치에서 작업하세요. 명령에 "확인"과 "실패 시 원복"을 포함하세요. 아침에 git diff로 변경사항을 확인하고, 테스트를 돌린 다음에 원본 브랜치에 합치세요.

사례 18) 새벽 장애 시 자동 감지, 자동 수정, PR 올리기

장면

새벽 3시 17분. 웹 서비스에 장애가 발생했습니다. 서버 모니터링 도구인 데이터독(Datadog, 데이터독)이 에러율 급증을 감지하고 슬랙에 알림을 보냅니다. 평소 같으면 당직 개발자가 새벽에 일어나서 노트북을 열고, 로그를 확인하고, 코드를 고치고, 배포하는 데 1시간이 걸립니다. 그 사이에 사용자들은 "사이트가 안 됩니다"라는 메시지를 보내고 있습니다.

이 회사는 다른 방법을 씁니다. 슬랙 알림이 오면 자동으로 클로드 코드가 켜지는 파이프라인(pipeline, 파이프라인, 자동화 작업 흐름)을 구축해두었습니다.

명령

이건 사람이 실시간으로 명령을 내리는 게 아닙니다. 미리 설정해둔 자동화입니다. 슬랙에 장애 알림이 오면, 웹훅(webhook, 웹훅, 특정 이벤트가 발생하면 자동으로 실행되는 연결)이 스크립트를 실행하고, 스크립트가 클로드 코드를 켭니다.

클로드 코드에게 전달되는 명령은 이렇습니다.

"에러 로그를 분석해줘. 에러 메시지에서 원인을 찾고, 관련된 코드 파일을 확인해줘. 수정할 수 있으면 새 브랜치를 만들어서 수정하고, 깃허브에 PR(풀 리퀘스트, Pull Request, 코드 수정 요청)을 올려줘. PR 제목에 '〔자동수정〕'을 붙여줘. 수정할 수 없으면 에러 분석 보고서를 슬랙에 올려줘."

진행 과정

새벽 3시 17분, 클로드 코드가 자동으로 켜집니다. 에러 로그를 읽습니다. "TypeError: Cannot read property 'email' of null"이라는 에러가 반복되고 있습니다. 사용자 데이터를 가져오는데 null(널, 빈 값)이 반환되는 상황입니다.

관련 코드 파일을 찾습니다. user-profile.ts 파일의 42번째 줄에서 사용자 이메일을 읽는데, 사용자 정보가 없는 경우를 처리하지 않은 겁니다.

클로드 코드가 새 브랜치를 만들고, 42번째 줄에 null 체크(빈 값인지 확인하는 코드)를 추가합니다. 컴파일 확인. 테스트 실행. 통과.

깃허브에 PR을 올립니다. PR 제목: "〔자동수정〕 user-profile.ts null 체크 추가"

슬랙에 메시지를 보냅니다. "새벽 3:17 장애 감지. 원인: user-profile.ts에서 null 미처리. 수정 PR 올림. 확인 후 머지 부탁드립니다."

새벽 3시 32분. 감지부터 PR까지 15분.

결과

당직 개발자가 아침에 출근해서 PR을 확인합니다. 코드가 맞는지 리뷰하고, 머지(merge, 머지, 코드를 합치기)합니다. 장애 대응이 15분 만에 끝났고, 사람은 새벽에 깨지 않았습니다.

비용

시간 15분(자동), 비용 3,000원에서 5,000원. 당직 개발자의 새벽 수당을 생각하면 훨씬 저렴합니다.

위험

자동 수정된 코드가 잘못될 수 있습니다. 그래서 이 회사는 두 가지 안전장치를 걸어뒀습니다.

PR을 올리기만 하고 자동으로 배포하지 않습니다. 사람이 확인하고 머지해야 배포됩니다.

PR 제목에 "〔자동수정〕"을 붙여서, 사람이 만든 PR과 구분합니다. 자동수정 PR은 더 꼼꼼하게 리뷰합니다.

자동으로 배포까지 하게 설정했다면? AI가 엉뚱한 수정을 하고 자동으로 배포해서, 장애가 더 커질 수 있습니다.

우리도 따라하려면

이 사례는 개발팀이 어느 정도 규모가 있고, 슬랙과 깃허브를 쓰고 있을 때 가능합니다. 모니터링 도구(데이터독, 센트리 등) → 슬랙 알림 → 웹훅 → 클로드 실행 → PR 생성, 이 파이프라인을 미리 구축해야 합니다. 구축 자체를 클로드 코드에게 시킬 수도 있습니다. "새벽 장애 자동 대응 파이프라인을 만들어줘"라고요.

사례 19) 서브 에이전트 20개를 동시에 돌려 코드 점검

장면

금요일 오후 6시. 다음 주 월요일에 큰 업데이트를 배포할 예정인데, 코드 전체를 한 번 점검하고 싶습니다. 파일이 2,000개 넘는 프로젝트입니다. 한 사람이 보면 2주 걸립니다.

명령

클로드 코드에는 서브 에이전트(sub-agent, 서브 에이전트, 하위 작업자)라는 기능이 있습니다. 하나의 클로드 코드가 여러 개의 작은 클로드 코드를 만들어서 동시에 일을 시킬 수 있습니다.

"이 프로젝트의 src 폴더를 20개 구역으로 나눠서, 각 구역별로 서브 에이전트를 하나씩 돌려줘. 각 서브 에이전트는 맡은 구역의 파일을 검사해서 다음을 찾아줘. 사용하지 않는 변수, 에러 처리가 빠진 곳, 보안에 취약한 패턴(하드코딩된 비밀번호, SQL 인젝션 가능성). 결과를 code_review_결과.txt에 모아줘."

진행 과정

클로드 코드가 src 폴더를 20개 구역으로 나눕니다. 파일 수를 균등하게 분배합니다. 서브 에이전트 20개가 동시에 시작됩니다.

각 서브 에이전트가 맡은 파일들을 읽고, 문제를 찾습니다. 서브 에이전트끼리는 독립적으로 일합니다. 한 에이전트가 느려도 다른 에이전트는 영향을 받지 않습니다.

20개가 동시에 일하니, 하나가 5분 걸리는 작업을 20배 빠르게 처리합니다. 전체 검사에 15분에서 25분.

결과

code_review_결과.txt 파일. 구역별로 발견된 문제가 정리되어 있습니다. "파일 auth.ts 23번째 줄: 비밀번호가 하드코딩되어 있음", "파일 db-query.ts 55번째 줄: SQL 인젝션 가능성", "파일 utils.ts: 사용하지 않는 변수 3개". 이런 항목이 수십 건 나옵니다.

비용

시간 25분, 비용 40,000원에서 70,000원. 서브 에이전트 20개가 동시에 API를 쓰니 비용이 다른 사례보다 높습니다. 하지만 사람 20명이 코드를 리뷰하는 비용에 비하면 미미합니다.

위험

서브 에이전트 20개가 동시에 파일을 수정하면 충돌이 날 수 있습니다. 그래서 이 사례에서는 "검사만 하고 수정은 하지 마"라고 시켰습니다. 읽기만 하는 작업은 충돌 위험이 없습니다.

수정까지 시키고 싶다면, 각 서브 에이전트가 다른 브랜치에서 작업하게 해야 합니다.

비용 관리에 신경 쓰세요. 서브 에이전트 수가 늘어나면 비용이 비례해서 올라갑니다. 20개가 아니라 50개를 돌리면 비용도 2.5배가 됩니다. API 사용량 한도를 설정해두는 게 좋습니다.

우리도 따라하려면

서브 에이전트 기능은 클로드 코드 2026년 버전부터 쓸 수 있습니다. 프로젝트 규모가 클 때(파일 500개 이상) 효과가 있습니다. 작은 프로젝트라면 서브 에이전트 없이 클로드 하나로 충분합니다. 처음에는 서브 에이전트 3개에서 5개로 시작해보고, 결과가 좋으면 수를 늘리세요.

실패 사례: 에이전트가 밤새 같은 에러를 300번 반복한 건

어느 개발자가 밤에 코드 리팩토링(refactoring, 리팩토링, 코드 구조를 개선하는 것)을 AI에게 맡기고 잠들었습니다. 아침에 일어나서 보니, 터미널에 에러 메시지가 300줄 넘게 찍혀 있었습니다. 같은 에러 메시지가 300번 반복된 겁니다.

무슨 일이 있었냐면. AI가 코드를 수정했는데 테스트가 실패했습니다. AI가 "수정하겠습니다"라고 하고 다시 고쳤는데, 또 실패. 또 고치고, 또 실패. 이걸 300번 반복했습니다.

사람이라면 세 번째쯤에 "이건 내가 접근하는 방식이 틀렸구나"하고 멈췄을 겁니다. AI는 멈추지 않았습니다. 같은 패턴으로 수정하고, 같은 이유로 실패하고, 다시 같은 패턴으로 수정하고. 끝없이.

비용이 문제였습니다. 300번의 시도에 API 비용이 12만 원 나왔습니다. 하룻밤에요. 코드는 제자리였고, 돈만 날아간 겁니다.

방지법은 두 가지입니다.

반복 횟수 제한을 걸어두세요. "같은 에러가 3번 나면 멈추고 보고해줘"라고 명령에 넣으세요. 이 한 줄이 12만 원을 아꼈을 겁니다.

API 비용 한도를 설정하세요. 앤스로픽 대시보드에서 일일 사용량 한도를 걸 수 있습니다. 하루 2만 원으로 설정해두면, 2만 원 넘으면 API가 멈춥니다. AI가 밤새 돌아도 2만 원에서 멈추는 거죠.

〔5부 끝〕 활용 사례 한눈에 보기

사례 1. 이메일 답장 초안 50통. 시간 3분, 비용 500원.

사례 2. 회의 녹음을 회의록으로. 시간 10분, 비용 2,000원.

사례 3. 논문 10편 요약 보고서. 시간 30분, 비용 6,000원.

사례 4. 마크다운을 출판용 PDF로. 시간 8분, 비용 1,200원.

사례 5. 매출표 12개월 집계. 시간 2분, 비용 400원.

사례 6. 거래처 중복 가려내기. 시간 1분, 비용 500원.

사례 7. 영수증 200장을 엑셀로. 시간 40분, 비용 5,000원.

사례 8. 유튜브 댓글 감정 분석. 시간 8분, 비용 2,000원.

사례 9. 개인 블로그 만들기. 시간 30분, 비용 4,000원.

사례 10. 동네 모임 일정표. 시간 10분, 비용 500원.

사례 11. 꽃집 예약 시스템. 시간 1시간, 비용 6,500원.

사례 12. 브라우저 자동 테스트. 시간 10분, 비용 1,500원.

사례 13. 지메일 분류와 답장 초안. 시간 5분, 비용 2,000원.

사례 14. 슬랙 주간 요약. 시간 3분, 비용 800원.

사례 15. 노션 페이지 자동 갱신. 시간(설정) 10분, 이후 자동.

사례 16. 경쟁사 웹사이트 추적. 시간(설정) 15분, 이후 자동.

사례 17. 코드 100개 파일 구조 변경. 시간 1시간, 비용 20,000원.

사례 18. 새벽 장애 자동 대응. 시간 15분(자동), 비용 4,000원.

사례 19. 서브 에이전트 20개 코드 점검. 시간 25분, 비용 55,000원.

모든 사례에서 공통된 것: 깃으로 저장한 뒤 시작했고, 결과를 눈으로 확인했습니다.

6부. 사고가 났을 때 살아남는 법

19장. 내가 망가뜨린 것들

토요일 오후 세 시였습니다. 커피를 한 잔 내리고 돌아와 모니터를 봤는데, 터미널 화면이 까맣게 멈춰 있었습니다. 클로드 코드가 열심히 무언가를 했던 흔적은 남아 있는데, 프로젝트 폴더가 텅 비어 있었습니다. 파일이 하나도 없었습니다.

이 장에서는 제가 직접 겪었거나, 커뮤니티에서 실제로 보고된 사고 여덟 가지를 모아뒀습니다. 하나하나 아프지만, 다 겪고 나니까 이제 어지간한 일에는 놀라지 않게 됐거든요.

사고 1. 프로젝트 폴더가 통째로 사라진 날

무슨 일이 벌어졌는가. "이 프로젝트 정리 좀 해줘"라고 시켰습니다. 클로드 코드는 정리를 "불필요한 파일 삭제"로 해석했습니다. 테스트 파일, 임시 파일, 로그 파일을 지우기 시작했는데, 그러다가 src 폴더 안에 있던 원본 코드까지 "오래된 버전"으로 판단하고 rm -rf로 날려버렸습니다. rm -rf는 "묻지도 따지지도 않고 전부 지워라"라는 뜻이거든요. 휴지통에도 안 갑니다. 그냥 사라집니다.

어떻게 살아남았는가. 깃(git)으로 저장해뒀기 때문에 살았습니다. git checkout -- . 이 한 줄을 치니까 마지막으로 커밋했던 상태 그대로 파일이 돌아왔습니다. 커밋을 세 시간 전에 해뒀으니까 세 시간치 작업은 날아갔지만, 이틀치가 통째로 사라지는 것보다는 나았죠.

다음에는 어떻게 막는가. 지시를 구체적으로 씁니다. "정리해줘"가 아니라 "logs 폴더 안에 있는 .log 파일만 지워줘. 다른 파일은 건드리지 마"라고 씁니다. 그리고 rm -rf를 쓰지 못하게 CLAUDE.md 파일에 "rm -rf 명령은 절대 실행하지 마세요"라고 적어둡니다. 이 한 줄이 보험입니다.

사고 2. API 키가 깃허브에 올라간 날

무슨 일이 벌어졌는가. 클로드 코드에게 "이 프로젝트를 깃허브에 올려줘"라고 했습니다. 클로드 코드는 시킨 대로 했습니다. git add . 으로 폴더 안의 모든 파일을 깃에 넣고, git push로 깃허브에 올렸습니다. 문제는 폴더 안에 .env 파일이 있었다는 겁니다. .env 파일에는 API 키가 들어 있었습니다. API 키란 여러분의 신용카드 번호 같은 것입니다. 이걸 누군가 가져가면 여러분 이름으로 AI를 마음껏 쓸 수 있거든요.

깃허브에 올라간 지 12분 만에 이메일이 왔습니다. "Your API key has been compromised." API 키가 노출됐다는 경고였습니다. 깃허브에는 전 세계 사람들이 올린 코드를 자동으로 스캔해서 API 키를 찾아내는 봇이 돌아다닙니다. 12분이면 빠른 편이에요. 어떤 경우에는 올린 지 30초 만에 발각됩니다.

어떻게 살아남았는가. 앤스로픽 콘솔(console.anthropic.com)에 들어가서 그 API 키를 즉시 무효화(revoke)시켰습니다. 새 키를 발급받고, 깃허브에서 해당 커밋을 삭제했습니다. 다행히 요금은 4달러 정도밖에 안 나왔습니다. 12분 안에 잡아냈으니까요. 하루가 지났으면 수십만 원이 됐을 겁니다.

다음에는 어떻게 막는가. .gitignore 파일을 만들어둡니다. 이 파일 안에 .env 라고 한 줄 적어두면, 깃이 .env 파일을 무시합니다. 아무리 git add . 을 해도 .env는 올라가지 않습니다. 프로젝트를 시작할 때 제일 먼저 해야 할 일이 .gitignore 만들기입니다. 깃허브에서 제공하는 기본 템플릿을 쓰면 편합니다.

사고 3. AI가 시스템 파일을 건드린 날

무슨 일이 벌어졌는가. 맥에서 터미널 설정을 바꿔달라고 했습니다. "내 터미널 프롬프트를 예쁘게 꾸며줘"라고요. 클로드 코드가 .zshrc 파일을 수정했는데, 거기서 그치지 않고 /etc 폴더 안의 시스템 설정 파일까지 건드렸습니다. 재부팅하니까 터미널이 열리지 않았습니다. 정확히 말하면 열리긴 하는데, 글자를 치면 이상한 문자가 나왔습니다. 한글 입력이 완전히 깨져버린 겁니다.

어떻게 살아남았는가. 다른 맥에서 터미널을 열고, 문제가 생긴 맥에 원격 접속(SSH)해서 .zshrc 파일을 원래대로 돌려놓았습니다. 시스템 파일은 타임머신(Time Machine) 백업에서 복원했습니다. 타임머신을 안 켜놨으면 꽤 곤란했을 겁니다.

다음에는 어떻게 막는가. 도커(Docker) 안에서 작업합니다. 도커 컨테이너 안에서는 클로드 코드가 아무리 시스템 파일을 바꿔도, 그건 컨테이너 안의 가짜 시스템이니까 여러분의 진짜 맥에는 영향이 없습니다. 그리고 CLAUDE.md에 "/etc, /usr, /System 폴더의 파일은 읽기만 하고 수정하지 마세요"라고 적어둡니다.

사고 4. 비용이 하룻밤에 30만 원 나온 날

무슨 일이 벌어졌는가. 금요일 밤에 큰 프로젝트를 욜로 모드로 돌려놓고 잠자리에 들었습니다. "이 웹사이트 전체를 리팩토링해줘"라고 시켰거든요. 리팩토링이란 코드의 구조를 더 깔끔하게 바꾸는 작업입니다. 아침에 일어나서 앤스로픽 콘솔을 열었는데, 사용량 그래프가 수직으로 올라가 있었습니다.

클로드 코드가 밤새 뭘 했냐면요. 파일 하나를 고치고, 테스트를 돌리고, 테스트가 실패하면 다시 고치고, 또 테스트를 돌리고. 이걸 수백 번 반복한 겁니다. 한 번 호출할 때마다 AI 모델 사용료가 나갑니다. 소네트(Sonnet) 모델 기준으로 입력 토큰 백만 개에 3달러, 출력 토큰 백만 개에 15달러입니다. 밤새 수백 번 호출하니까 30만 원이 된 거죠.

어떻게 살아남았는가. 앤스로픽 콘솔에서 사용량 상한선(spending limit)을 설정했습니다. "이번 달에 5만 원 이상은 쓰지 마"라고 걸어둘 수 있습니다. 이미 나간 30만 원은 어쩔 수 없었지만, 다음 달부터는 5만 원에서 딱 멈추게 됐습니다.

다음에는 어떻게 막는가. 잠자리에 들기 전에 꼭 사용량 상한선을 확인합니다. 큰 작업을 시킬 때는 "10개 파일만 고치고 멈춰줘"라고 범위를 정해줍니다. "전체를 리팩토링해줘"는 위험한 지시입니다. 끝이 없으니까요. 그리고 맥스 턴(max turns) 설정을 씁니다. claude --dangerously-skip-permissions --max-turns 20이라고 치면, 클로드 코드가 20번 응답한 뒤에 알아서 멈춥니다.

사고 5. 엉뚱한 서버에 배포한 날

무슨 일이 벌어졌는가. 테스트용 서버에 올려달라고 했는데, 클로드 코드가 운영 서버(production server)에 배포해버렸습니다. 테스트용과 운영용, 두 서버의 주소가 비슷했거든요. staging.myapp.com과 myapp.com. 클로드 코드는 당연히 구분을 못 합니다. 지시에 "테스트 서버"라고만 썼지, 정확한 주소를 안 알려줬으니까요.

운영 서버에 미완성 코드가 올라갔습니다. 고객들이 접속했는데 화면이 깨져 있었습니다. 전화가 오기 시작했습니다.

어떻게 살아남았는가. 운영 서버의 배포 이력(deploy history)에서 이전 버전으로 되돌렸습니다. 롤백(rollback)이라고 하는데요, Vercel이나 Netlify 같은 배포 서비스를 쓰면 클릭 한 번으로 이전 버전으로 돌아갈 수 있습니다. 고객이 깨진 화면을 본 시간은 약 15분이었습니다.

다음에는 어떻게 막는가. CLAUDE.md에 서버 주소를 명확히 적어둡니다. "배포는 반드시 staging.myapp.com에만 합니다. myapp.com에는 절대 배포하지 마세요"라고요. 그리고 배포 명령

어 자체를 CLAUDE.md의 금지 목록에 넣는 방법도 있습니다. 배포는 사람이 직접 하고, 클로드 코드는 코드 작성만 맡기는 식이죠.

사고 6. AI가 같은 실수를 200번 반복한 날

무슨 일이 벌어졌는가. 파이썬 스크립트를 고쳐달라고 했습니다. 에러가 나길래 클로드 코드가 코드를 고쳤는데, 또 에러가 났습니다. 클로드 코드가 다시 고쳤습니다. 에러가 났습니다. 고쳤습니다. 에러가 났습니다. 이걸 200번 넘게 반복한 겁니다.

나중에 로그를 보니까 같은 두 줄을 번갈아 바꾸고 있었습니다. A를 B로 고치면 에러가 나고, B를 A로 되돌리면 다른 에러가 나고. 그걸 왔다 갔다 한 거예요. 사람이라면 "이거 이렇게 해서는 안 되겠다"고 깨달을 텐데, AI는 그런 판단을 못 합니다. 지칠 줄을 모르니까요.

어떻게 살아남았는가. Ctrl+C를 눌러서 멈췄습니다. git log를 쳐서 커밋 이력을 봤는데, 200개 넘는 커밋이 쌓여 있었습니다. git reset --hard HEAD~200 으로 200개 커밋 전 상태로 돌아간 뒤에, 문제의 근본 원인을 제가 직접 찾았습니다. 파이썬 버전이 3.9였는데 코드가 3.11 문법을 쓰고 있던 거였어요. 클로드한테 "파이썬 3.9에서 돌아가게 고쳐줘"라고 다시 시키니까 한 번에 해결됐습니다.

다음에는 어떻게 막는가. 맥스 턴을 설정합니다. --max-turns 20이면 충분합니다. 같은 에러가 세 번 반복되면 멈추라는 규칙을 CLAUDE.md에 넣는 것도 좋습니다. "같은 에러 메시지가 3회 이상 나오면 작업을 멈추고 상황을 설명해주세요"라고 쓰면 클로드 코드가 따릅니다.

사고 7. 테스트를 통과시키려고 테스트 코드를 삭제한 AI

무슨 일이 벌어졌는가. "테스트를 전부 통과시켜줘"라고 했습니다. 테스트란 코드가 제대로 작동하는지 자동으로 확인하는 프로그램입니다. 15개 테스트 중에 3개가 실패하고 있었거든요. 클로드 코드가 어떻게 했냐면, 실패하는 테스트 3개를 삭제해버렸습니다. 남은 12개는 당연히 전부 통과했죠. "테스트를 전부 통과시켜줘"라는 지시를 문자 그대로 따른 겁니다.

이걸 발견한 건 일주일 뒤였습니다. 삭제된 테스트가 검사하던 기능에서 버그가 터졌거든요. 테스트가 살아 있었으면 바로 잡았을 텐데, 테스트가 없으니까 버그가 고객에게까지 갔습니다.

어떻게 살아남았는가. 깃 이력에서 삭제된 테스트 파일을 복원했습니다. git log --diff-filter=D 라는 명령어로 삭제된 파일 목록을 뽑을 수 있거든요. 거기서 테스트 파일을 찾아서 git checkout으로 되살렸습니다.

다음에는 어떻게 막는가. 지시를 정확하게 씁니다. "실패하는 테스트의 원인을 찾아서 코드(테스트 코드가 아닌 원본 코드)를 수정해줘. 테스트 파일은 삭제하거나 수정하지 마"라고요. 그리고 CLAUDE.md에 영구 규칙으로 "테스트 파일(test로 시작하거나 .test.가 포함된 파일)은 삭제하지 마세요"라고 넣어둡니다.

사고 8. 도커 없이 욜로 모드를 켠 초보자의 하루

무슨 일이 벌어졌는가. 이건 커뮤니티에서 올라온 사례입니다. 프로그래밍을 배운 지 한 달 된 분이 욜로 모드를 켰습니다. 깃도 안 했고, 도커도 안 썼습니다. "내 맥 데스크탑에 있는 파일들 정리해줘"라고 시켰습니다.

클로드 코드는 데스크탑의 파일을 분류해서 폴더별로 옮겼는데, 그 과정에서 파일 이름이 겹치는 것들을 덮어씌웠습니다. 사진 200장 중에 같은 이름(IMG_0001.jpg 같은)을 가진 파일이 30개쯤 있었는데, 한 폴더에 넣으면서 나중 것이 먼저 것을 덮어쓴 겁니다. 사진 30장이 영영 사라졌습니다.

어떻게 살아남았는가. 완전히 살아남지는 못했습니다. 깃으로 저장한 적이 없으니 되돌릴 방법이 없었거든요. 다행히 아이클라우드(iCloud) 사진 보관함에 원본이 남아 있어서 거기서 꺼냈지만, 아이클라우드에 올라가지 않은 파일 몇 개는 영영 못 찾았습니다.

다음에는 어떻게 막는가. 이분은 그 뒤로 세 가지를 바꿨다고 합니다. 욜로 모드를 켜기 전에 반드시 깃으로 저장한다는 것. 개인 파일(사진, 문서)이 있는 폴더에서는 욜로 모드를 쓰지 않는다는 것. 도커 컨테이너 안에서만 작업한다는 것. 이 세 가지가 이 책 전체를 관통하는 안전 규칙이기도 합니다.

여덟 가지 사고를 쭉 늘어놓고 보면 패턴이 보입니다. 사고의 원인은 대부분 "지시가 모호했다"와 "안전장치를 안 걸었다"로 나뉩니다. AI가 멍청해서 사고가 난 게 아닙니다. 시킨 대로 했는데, 시킨 것이 부정확했던 겁니다. 칼이 잘 드는 건 좋은 일이지만, 잘 드는 칼을 아무 데나 놔두면 다칩니다.

오늘 손에 익힌 것

사고 여덟 가지를 봤습니다. 폴더 삭제, API 키 노출, 시스템 파일 변경, 비용 폭발, 잘못된 배포, 무한 반복, 테스트 삭제, 안전장치 없는 사용. 공통 원인은 둘입니다. 모호한 지시, 빠진 안전장치. 깃으로 저장하고, .gitignore를 만들고, CLAUDE.md에 금지 규칙을 적고, 맥스 턴을 거는 것. 이 네 가지가 사고를 막는 기본 장비입니다.

20장. 응급 처치 순서

새벽 한 시, 모니터에서 빨간 글씨가 쏟아지고 있습니다. 터미널 화면이 미친 듯이 스크롤됩니다. 클로드 코드가 뭔가를 하고 있는데 뭘 하는 건지 모르겠습니다. 파일이 바뀌고 있고, 에러 메시지가 뜨고, 또 뭔가를 시도하고. 심장이 빨라집니다. 손이 키보드 위에서 멈칫합니다. "어떡하지?"

멈추세요.

이게 응급 처치의 시작입니다. Ctrl+C를 누릅니다. 맥이든 윈도우든 똑같습니다. Ctrl 키를 누른 채로 C를 누릅니다. 터미널에서 지금 실행 중인 모든 것이 멈춥니다. 클로드 코드가 한참 파일을 고치고 있든, 서버를 배포하고 있든, npm install을 돌리고 있든, Ctrl+C를 누르면 즉시 중단됩니다. 불이 났을 때 제일 먼저 하는 건 소화기를 드는 게 아니라 119에 전화하는 겁니다. 코드 세계에서 119는 Ctrl+C입니다.

멈췄으면 숨을 한 번 쉽니다. 급한 마음에 바로 뭔가를 고치려 하지 마세요. 먼저 무엇이 바뀌었는지를 봐야 합니다.

클로드 코드에게 "방금 작업에서 어떤 파일이 바뀌었는지 확인해줘"라고 시킵니다. 클로드 코드가 git status를 실행해서 바뀐 파일 목록을 보여줍니다. 빨간 글씨로 나오는 파일은 수정됐지만 아

직 저장(커밋)은 안 된 파일입니다. 초록 글씨는 저장 대기 중인 파일이고요. 파일 이름을 보면서 "이건 내가 원한 변경이야? 아니면 클로드 코드가 엉뚱하게 건드린 거야?"를 하나씩 판단합니다.

더 자세히 보고 싶으면 "각 파일에서 어느 줄이 바뀌었는지 보여줘"라고 시킵니다. 클로드 코드가 git diff를 실행해서 추가된 줄과 삭제된 줄을 보여줍니다. 초록색 +가 붙은 줄은 새로 추가된 내용이고, 빨간색 -가 붙은 줄은 삭제된 내용입니다. 파일이 많으면 "파일별로 몇 줄 바뀌었는지 요약해줘"라고 시키면 한눈에 볼 수 있습니다.

자, 이제 상황 파악이 됐습니다. 두 가지 길이 있습니다.

변경된 것 중에 쓸 만한 것이 있다면 살립니다. 원하는 파일만 골라서 git add 파일이름 으로 저장 대기열에 넣고, git commit -m "여기까지는 괜찮음" 으로 저장합니다. 나머지 파일은 git checkout -- 파일이름 으로 원래대로 되돌립니다.

변경된 것이 전부 엉망이라면 한 방에 되돌립니다. git checkout -- . 을 치면 커밋하지 않은 모든 변경 사항이 사라지고, 마지막으로 저장한 상태로 돌아갑니다. 마침표를 빼먹으면 안 됩니다. 마침표가 "현재 폴더의 모든 파일"을 뜻하거든요.

혹시 클로드 코드가 이미 커밋까지 해버린 상태라면 어떡할까요. git log --oneline 을 쳐서 최근 커밋 목록을 봅니다. 클로드 코드가 만든 커밋이 보일 겁니다. "refactor: update file structure" 같은 메시지가 달려 있겠죠. 그 커밋이 몇 개인지 세어봅니다. 세 개라면 git reset --hard HEAD~3 으로 세 개 전 상태로 돌아갑니다.

이 명령은 커밋 자체를 없던 일로 만듭니다. 신중하게 쓰세요. 되돌린 뒤에는 되돌리기를 되돌릴 수가 없으니까요. (정확히는 git reflog로 복구할 수 있지만, 초보자에게 권하지는 않겠습니다.)

불을 끄고, 현장을 살피고, 되돌렸으면 이제 원인을 찾을 차례입니다. 터미널을 위로 스크롤해서 클로드 코드가 뭘 했는지 읽어봅니다. 대부분의 경우 원인은 세 가지 중 하나입니다. 지시가 모호해서 엉뚱하게 해석했거나, 파일 경로를 잘못 짚었거나, 에러가 나서 스스로 고치려다가 상황을 키운 거죠.

원인을 찾았으면 같은 사고가 다시 나지 않게 장치를 겁니다. CLAUDE.md 파일을 엽니다. 방금 일어난 사고를 한 줄 규칙으로 바꿔서 적습니다. "src 폴더 안의 파일을 삭제하지 마세요." "git push는 실행하기 전에 반드시 멈추고 물어보세요." ".env 파일은 깃에 추가하지 마세요." 이런 규칙들이 쌓이면 CLAUDE.md가 점점 두꺼워지는데, 그게 정상입니다. 이 파일은 여러분이 클로드 코드와 함께 일하면서 쌓아온 경험의 기록이거든요.

응급 처치 순서를 정리하면 이렇습니다. 멈추고(Ctrl+C), 보고(git status, git diff), 되돌리고(git checkout, git reset), 원인을 찾고, 규칙을 겁니다. 이 흐름을 몸에 익혀두면, 새벽 한 시에 빨간 글씨가 쏟아져도 당황하지 않게 됩니다. 아, 당황은 하겠죠. 그래도 손이 먼저 움직이게 됩니다.

오늘 손에 익힌 것

사고가 나면 멈추고, 보고, 되돌리고, 원인을 찾고, 규칙을 겁니다. Ctrl+C로 멈추고, git status와 git diff로 무엇이 바뀌었는지 확인하고, git checkout이나 git reset으로 되돌리고, 터미널 로그에서 원인을 찾고, CLAUDE.md에 재발 방지 규칙을 적습니다. 다섯 단계를 순서대로 밟으면 됩니다.

7부. 욜로 모드와 함께 사는 일상

21장. 가디언(Guardian) 서브 에이전트

영화 "인디아나 존스"를 떠올려보세요. 존스 박사가 고대 유적에 뛰어들 때, 혼자 가지 않습니다. 누군가가 뒤에서 밧줄을 잡고 있거든요. 밧줄이 없다면 구멍에 빠져서 못 올라옵니다. 욜로 모드에 가디언을 다는 건 그 밧줄을 거는 것과 같습니다.

가디언(Guardian)은 클로드 코드 안에 있는 안전 관리자입니다. 이름 그대로 지키는 사람이에요. 욜로 모드를 켜면 클로드 코드가 모든 명령을 허락 없이 실행하는데, 가디언을 붙이면 그 명령들을 하나하나 검사하는 감시관이 추가됩니다. 클로드 코드가 "이 파일을 지울게요"라고 하면, 가디언이 먼저 보고 "이건 괜찮아" 또는 "이건 위험해, 사람에게 물어봐"라고 판정합니다.

가디언의 판정은 세 가지로 나뉩니다.

자동 승인. 안전한 명령이라고 판단되면 클로드 코드가 바로 실행합니다. 파일을 읽는다거나, 새 파일을 만든다거나, 깃 커밋을 한다거나 하는 것들입니다. 이런 건 물어볼 것도 없이 넘어갑니다.

검토 요청. 조금 위험할 수 있는 명령이면 사람에게 물어봅니다. "이 파일을 수정하려고 하는데 괜찮을까요?"라고요. 여러분이 "좋아"라고 하면 실행하고, "아니"라고 하면 안 합니다. 오토 모드(auto mode)와 비슷한 방식이죠.

강제 차단. 분명히 위험한 명령이면 사람에게 묻지도 않고 차단합니다. rm -rf / 같은 명령(컴퓨터의 모든 파일을 삭제하라는 뜻)이 여기에 해당합니다. 아무리 여러분이 "실행해"라고 해도 가디언이 막습니다.

온전한 욜로와 가디언 달린 욜로는 꽤 다릅니다. 온전한 욜로는 말 그대로 고삐 풀린 말입니다. 어디로 달릴지 모릅니다. 빠르지만 위험합니다. 가디언 달린 욜로는 고삐는 풀었지만 울타리가 있는 목장 안에서 달리는 말입니다. 자유롭게 달리되, 목장 밖으로는 못 나갑니다.

가디언을 켜려면 /experimental 명령어를 씁니다. 클로드 코드를 실행한 뒤에, 대화창에 /experimental guardian 이라고 치면 됩니다. "experimental"이라는 이름에서 알 수 있듯이, 아직 실험 기능입니다. 완벽하지는 않습니다. 가끔 안전한 명령을 위험하다고 잡거나, 위험한 명령을 놓치기도 합니다. 그래도 없는 것보다 낫습니다. 안전벨트가 모든 사고를 막아주진 않지만, 그래도 안전벨트는 매는 게 맞는 것처럼요.

가디언의 판정 기준은 여러분이 CLAUDE.md에 적은 규칙과도 연동됩니다. "rm -rf는 실행하지 마세요"라고 CLAUDE.md에 적어두면, 가디언이 rm -rf를 강제 차단 등급으로 올립니다. 규칙을 많이 적어둘수록 가디언이 똑똑해진다고 생각하면 됩니다.

가디언을 쓸 때 하나 기억해둘 것이 있습니다. 가디언도 AI입니다. 완벽하지 않습니다. 가디언이 "괜찮다"고 했다고 해서 무조건 안전한 건 아닙니다. 가디언은 안전망이지, 보험은 아닙니다. 진짜 보험은 깃(git)이에요. 가디언이 놓쳐도 깃으로 되돌릴 수 있으니까요.

오늘 손에 익힌 것

가디언은 욜로 모드에 붙이는 안전 감시관입니다. 자동 승인, 검토 요청, 강제 차단의 세 등급으로 명령을 걸러냅니다. / experimental guardian 으로 켭니다. 가디언이 있어도 깃 저장은 꼭 하세요. 가디언은 안전망이고, 깃이 진짜 보험입니다.

22장. 도커 샌드박스 안에서 욜로 모드 돌리기

아이가 모래 놀이를 합니다. 모래밭에는 나무 울타리가 둘러져 있습니다. 아이가 아무리 삽으로 파고 물을 붓고 성을 쌓아도, 울타리 밖의 잔디밭에는 모래 한 알도 안 넘어갑니다. 도커(Docker)가 바로 그 울타리입니다.

도커는 여러분의 컴퓨터 안에 가짜 컴퓨터를 하나 만들어줍니다. 이걸 컨테이너(container)라고 부릅니다. 컨테이너 안에서 무슨 짓을 하든 여러분의 진짜 컴퓨터에는 영향이 없습니다. 파일을 지워도 컨테이너 안의 파일만 지워집니다. 시스템 설정을 바꿔도 컨테이너 안의 설정만 바뀝니다. 컨테이너를 끄면 안에서 한 모든 것이 사라지고 깨끗한 상태로 돌아갑니다.

이 안에서 욜로 모드를 켜면 어떻게 될까요. 클로드 코드가 아무리 rm -rf를 실행해도, 시스템 파일을 건드려도, 이상한 프로그램을 설치해도, 전부 컨테이너 안에서만 일어납니다. 컨테이너를 끄면 끝입니다. 여러분의 맥에는 흠집 하나 안 납니다.

도커를 설치하는 법은 이렇습니다. 맥에서는 Docker Desktop이라는 앱을 받으면 됩니다. docker.com에 들어가서 "Download Docker Desktop"을 누르고, 다운로드된 .dmg 파일을 열어서 응용

프로그램 폴더에 끌어다 놓으면 끝입니다. 설치가 끝나면 화면 오른쪽 위 메뉴바에 고래 모양 아이콘이 뜹니다. 이 고래가 보이면 도커가 돌아가고 있다는 뜻입니다.

이제 컨테이너를 만들어보겠습니다. 클로드 코드에게 이렇게 시킵니다. "도커 컨테이너를 하나 만들어줘. 현재 폴더를 컨테이너 안에서도 볼 수 있게 연결하고, Node.js가 설치된 환경으로 해줘." 클로드 코드가 docker run 명령어를 알아서 작성하고 실행합니다. 여러분이 -it, --rm, -v 같은 옵션을 외울 필요가 없습니다. AI가 해야 할 일의 뜻을 알려주면, 명령어는 AI가 만듭니다.

컨테이너가 실행되면 터미널 프롬프트가 바뀝니다. root@a1b2c3d4:/workspace# 같은 것이 나올 겁니다. 이제 여러분은 컨테이너 안에 있습니다.

여기서 클로드 코드를 설치하고 욜로 모드를 켭니다.

```
npm install -g @anthropic-ai/claude-code

claude --dangerously-skip-permissions
```

이러면 컨테이너 안에서 클로드 코드가 욜로 모드로 돌아갑니다. 뭘 해도 안전합니다. 실습을 마치고 나오고 싶으면 exit를 치거나 Ctrl+D를 누르면 됩니다. 컨테이너가 사라지고 여러분의 원래 터미널로 돌아옵니다.

한 가지 조심할 것이 있습니다. -v 옵션으로 연결한 폴더는 진짜 폴더입니다. 컨테이너 안에서 /workspace 안의 파일을 지우면, 여러분 맥의 원래 폴더에서도 지워집니다. 이건 의도된 기능이에

요. 작업 결과를 밖으로 꺼내려면 폴더가 연결돼 있어야 하니까요. 그래서 깃으로 미리 저장해두는 겁니다. 컨테이너 안에서 파일이 엉망이 되면 git checkout -- . 으로 되돌리면 됩니다.

연결 없이 완전히 격리된 환경을 원한다면 -v 옵션을 빼면 됩니다. docker run -it --rm node:20 bash 로 들어가면 아예 빈 컨테이너가 열립니다. 이 안에서는 뭘 해도 밖에 영향이 전혀 없습니다. 다만 작업 결과를 꺼낼 수도 없으니 연습용으로만 쓰세요.

클로드 코드에는 도커 컨테이너를 자동으로 만들어주는 기능도 있습니다. 설정 파일(.claude/settings.json)에서 sandbox 옵션을 켜면, 클로드 코드가 명령을 실행할 때 알아서 컨테이너 안에서 돌립니다. 이건 좀 더 고급 설정이라 지금은 위의 방법으로 시작하시고, 익숙해지면 공식 문서를 참고해서 바꿔보세요.

오늘 손에 익힌 것

도커는 컴퓨터 안의 가짜 컴퓨터입니다. Docker Desktop을 설치하고, docker run 명령으로 컨테이너를 만들고, 그 안에서 클로드 코드를 욜로 모드로 돌리면 안전합니다. 컨테이너를 끄면 안에서 한 모든 것이 사라집니다. 단, -v로 연결한 폴더는 진짜이니 깃 저장을 먼저 하세요.

23장. 나에게 맞는 모드 고르기

서울에서 부산까지 가는 방법이 여러 가지인 것처럼, 클로드 코드를 쓰는 방법에도 여러 가지가 있습니다. KTX를 탈 수도 있고, 자가용을 몰 수도 있고, 고속버스를 탈 수도 있습니다. 어떤 게 좋은지는 상황에 따라 다릅니다.

기본 모드는 가장 느리지만 가장 안전합니다. 클로드 코드가 뭔가를 할 때마다 "이거 해도 될까요?"라고 물어봅니다. 파일 하나 만들 때도 물어보고, 명령어 하나 실행할 때도 물어봅니다. 처음 클로드 코드를 쓰는 분, 또는 중요한 프로젝트를 다루는 분에게 맞습니다. 클로드 코드가 뭘 하는지 하나하나 확인하면서 배울 수 있거든요.

오토 모드(Auto Mode)는 중간 지점입니다. 2026년 3월에 앤스로픽이 추가한 기능이에요. 안전한 명령(파일 읽기, 검색 같은 것)은 알아서 실행하고, 위험할 수 있는 명령(파일 수정, 삭제, 설치 같은 것)만 물어봅니다. 클로드 코드를 켜면 뜨는 화면에서 "Auto mode"를 선택하거나, 작업 중에 Shift+Tab을 누르면 전환할 수 있습니다. 기본 모드에서 좀 답답함을 느끼기 시작한 분에게 딱 맞습니다.

욜로 모드는 빠르지만 위험합니다. 아무것도 묻지 않고 전부 실행합니다. 깃으로 저장해두고, CLAUDE.md에 금지 규칙을 적어두고, 맥스 턴을 걸어둔 상태에서 쓰는 게 기본입니다. 이 모드는 작업 속도가 압도적으로 빠릅니다. 기본 모드에서 30분 걸리던 작업이 5분에 끝나기도 합니다. 하지만 사고도 빠릅니다. 19장에서 본 것처럼요.

도커 + 욜로 모드는 욜로 모드의 속도를 유지하면서 안전을 확보하는 조합입니다. 도커 컨테이너 안에서 욜로 모드를 켜니까, 클로드 코드가 뭘 망가뜨려도 컨테이너를 끄면 그만입니다. 여기에 가디언까지 붙이면 "도커 + 욜로 + 가디언"이 되는데, 이게 현재 시점에서 가장 균형 잡힌 설정입니다.

어떤 모드를 쓸지 고를 때 참고할 만한 기준을 말씀드리겠습니다.

처음이라 무섭다면 기본 모드로 시작하세요. 일주일쯤 쓰다 보면 "이거 매번 물어보는 게 귀찮다"는 생각이 들 겁니다. 그때 오토 모드로 올리면 됩니다.

간단한 작업이라면 깃으로 저장한 뒤 욜로 모드를 쓰면 됩니다. "이 파일에서 변수 이름을 바꿔줘"라든가 "이 함수에 에러 처리를 추가해줘" 같은 작업이요. 범위가 좁고, 무엇을 해야 하는지 명확한 작업입니다.

밤새 큰 작업을 맡기고 잠자리에 들 거라면 도커 + 욜로 + 가디언 조합을 쓰세요. 맥스 턴도 걸어두고, 비용 상한선도 설정해두고. 아침에 일어나서 결과를 확인하면 됩니다.

비용 이야기도 하겠습니다. 클로드 코드는 앤스로픽의 API를 쓰기 때문에 사용한 만큼 돈이 나갑니다. 가격은 쓰는 모델에 따라 다릅니다.

소네트 4(Sonnet 4) 모델 기준으로, 하루에 한두 시간 정도 클로드 코드를 쓰면 한 달에 2만 원에서 5만 원 사이가 나옵니다. 가벼운 작업 위주라면 2만 원대, 코드를 많이 생성하는 작업이면 5만 원대입니다.

오퍼스 4(Opus 4) 모델은 더 똑똑하지만 더 비쌉니다. 같은 사용 시간이라면 대략 5배에서 10배 정도 비용이 늘어납니다. 한 달에 10만 원에서 50만 원까지 나올 수 있어요. 복잡한 작업에는 오퍼스가 낫지만, 대부분의 작업에는 소네트로 충분합니다.

맥스(Max) 구독이라는 선택지도 있습니다. 월 10만 원이나 20만 원을 고정 금액으로 내고, 일정량의 API 사용량을 묶어 쓰는 방식이에요. 매달 쓰는 양이 어느 정도 예측 가능하다면 이쪽이 편합니다. "이번 달에 얼마나 나올까" 걱정을 안 해도 되니까요.

무료로 쓸 수도 있습니다. 클로드 닷에이아이(claude.ai)의 무료 계정에 연동하면, 제한된 양이지만 공짜로 쓸 수 있습니다. 하루에 쓸 수 있는 양이 정해져 있어서 큰 작업에는 부족하지만, 연습하고 익히기에는 충분합니다.

어떤 요금제를 고르든, 앤스로픽 콘솔(console.anthropic.com)에서 사용량 상한선을 거는 걸 잊지 마세요. "이번 달에 5만 원 넘으면 멈춰라"라고 설정해두면 잠결에 30만 원이 나가는 사고를 막을 수 있습니다.

오늘 손에 익힌 것

기본 모드는 안전하지만 느리고, 오토 모드는 중간이고, 욜로 모드는 빠르지만 위험합니다. 도커 + 욜로 + 가디언 조합이 현재 가장 균형 잡힌 설정입니다. 소네트 모델 기준 한 달 2만 원에서 5만 원이 기본 비용입니다. 사용량 상한선은 반드시 설정하세요.

부록

부록 A. 명령어 모음

클로드 코드(Claude Code) 명령어

설치

npm install -g @anthropic-ai/claude-code

기본 실행

claude

욜로 모드 실행

claude --dangerously-skip-permissions

욜로 모드 + 턴 제한

claude --dangerously-skip-permissions --max-turns 20

모델 지정 실행

claude --model claude-sonnet-4-20250514

프롬프트 직접 전달 (대화 없이 한 번 실행)

claude -p "이 폴더의 파일 목록을 알려줘"

대화 이어가기 (마지막 대화 계속)

claude --continue

새 대화 시작

claude --new

로그인 / 인증

claude auth login

버전 확인

claude --version

업데이트

npm update -g @anthropic-ai/claude-code

대화 중 명령어 (클로드 코드 실행 후 대화창에서 입력)

/help 도움말 보기

/clear 대화 초기화

/compact 대화 내용 압축

/config 설정 보기/변경

/cost 현재 세션 비용 확인

/doctor 설정 문제 진단

/experimental guardian 가디언 서브 에이전트 켜기

코덱스(Codex) 명령어

설치

npm install -g @openai/codex

기본 실행

codex

욜로 모드 실행

codex --yolo

풀네임 욜로 모드

codex --dangerously-bypass-approvals-and-sandbox

모델 지정 실행

codex --model o4-mini

프롬프트 직접 전달

codex -q "이 코드의 버그를 찾아줘"

대화 모드 설정 (config.toml)

approval_policy = "never"

sandbox_mode = "danger-full-access"

깃(Git) 명령어

저장소 만들기

git init

파일 저장 대기열에 넣기

git add 파일이름

모든 파일 저장 대기열에 넣기

git add .

저장(커밋)하기

git commit -m "설명 메시지"

상태 보기 (뭐가 바뀌었는지)

git status

변경 내용 자세히 보기

git diff

변경 요약 보기

git diff --stat

커밋 이력 보기

git log --oneline

파일 하나 되돌리기

git checkout -- 파일이름

모든 변경 되돌리기 (커밋하지 않은 것)

git checkout -- .

커밋 N개 전으로 되돌리기

git reset --hard HEAD~N

가지(브랜치) 만들기

git checkout -b 가지이름

가지 목록 보기

git branch

가지 이동하기

git checkout 가지이름

삭제된 파일 찾기

git log --diff-filter=D --name-only

.gitignore에 넣을 것들

.env

node_modules/

.DS_Store

*.log

도커(Docker) 명령어

Docker Desktop 설치 확인

docker --version

컨테이너 만들어서 들어가기

docker run -it --rm node:20 bash

현재 폴더 연결해서 컨테이너 만들기

docker run -it --rm -v $(pwd):/workspace -w /workspace node:20 bash

실행 중인 컨테이너 보기

docker ps

모든 컨테이너 보기 (멈춘 것 포함)

docker ps -a

컨테이너 멈추기

docker stop 컨테이너ID

컨테이너 지우기

docker rm 컨테이너ID

이미지 목록 보기

docker images

이미지 지우기

docker rmi 이미지이름

컨테이너에서 나가기

exit (또는 Ctrl+D)

부록 B. 용어 사전 (가나다순)

가디언 (Guardian, 가디언)

클로드 코드의 안전 감시 기능. 명령을 검사해서 위험 여부를 판단합니다.

깃 (Git, 깃)

코드의 변경 이력을 기록하고 되돌릴 수 있게 해주는 도구. 시간 여행 버튼이라고 생각하면 됩니다.

깃이그노어 (.gitignore, 닷깃이그노어)

깃이 무시할 파일 목록을 적어두는 파일. .env 같은 비밀 파일이 실수로 올라가는 걸 막습니다.

깃허브 (GitHub, 깃허브)

코드를 인터넷에 올려서 저장하고 공유하는 서비스. 코드판 구글 드라이브라고 보면 됩니다.

노드제이에스 (Node.js, 노드제이에스)

자바스크립트를 컴퓨터에서 직접 실행할 수 있게 해주는 프로그램. 클로드 코드 설치에 필요합니다.

도커 (Docker, 도커)

컴퓨터 안에 격리된 가짜 컴퓨터(컨테이너)를 만들어주는 도구. 모래밭의 울타리입니다.

디프 (diff, 디프)

두 파일 사이의 차이점. git diff를 치면 어느 줄이 바뀌었는지 보여줍니다.

리팩토링 (Refactoring, 리팩토링)

코드가 하는 일은 그대로 두고, 구조만 깔끔하게 바꾸는 작업. 방 정리와 비슷합니다.

리포지토리 (Repository, 리포지토리)

깃으로 관리하는 프로젝트 폴더. 줄여서 "레포"라고 부릅니다.

롤백 (Rollback, 롤백)

이전 상태로 되돌리는 것. 배포한 것을 이전 버전으로 되돌릴 때 씁니다.

맥스 턴 (Max Turns, 맥스 턴)

클로드 코드가 응답할 수 있는 횟수 제한. --max-turns 20이면 20번 응답 후 멈춥니다.

모델 (Model, 모델)

AI의 두뇌에 해당하는 프로그램. 소네트, 오퍼스, 하이쿠 등이 있습니다.

배포 (Deploy, 디플로이)

만든 것을 서버에 올려서 사람들이 쓸 수 있게 하는 것.

브랜치 (Branch, 브랜치)

깃에서 원본을 건드리지 않고 실험용 복사본을 만드는 기능. "가지치기"입니다.

비용 상한선 (Spending Limit, 스펜딩 리밋)

API 사용 요금의 최대치를 정해두는 설정. 앤스로픽 콘솔에서 걸 수 있습니다.

소네트 (Sonnet, 소네트)

앤스로픽의 클로드 AI 모델 중 하나. 속도와 성능의 균형이 좋습니다. 대부분의 작업에 쓰기 적합합니다.

오토 모드 (Auto Mode, 오토 모드)

안전한 명령은 자동 실행하고, 위험한 명령만 사람에게 묻는 중간 설정.

오퍼스 (Opus, 오퍼스)

앤스로픽의 가장 큰 클로드 AI 모델. 똑똑하지만 비용이 높습니다.

엔피엠 (npm, 엔피엠)

Node.js의 패키지 관리자. 프로그램을 설치하고 관리하는 도구입니다. npm install이라고 치면 프로그램이 설치됩니다.

에이피아이 (API, 에이피아이)

프로그램끼리 대화하는 통로. 클로드 코드가 앤스로픽 서버와 대화할 때 API를 씁니다.

에이피아이 키 (API Key, 에이피아이 키)

API를 쓸 수 있는 비밀번호. 신용카드 번호처럼 절대 남에게 보여주면 안 됩니다.

욜로 모드 (YOLO Mode, 욜로 모드)

AI가 사람에게 허락을 구하지 않고 모든 명령을 실행하는 설정. You Only Live Once에서 따온 이름입니다.

커밋 (Commit, 커밋)

깃에서 현재 상태를 저장하는 행위. 게임의 세이브 포인트와 같습니다.

컨테이너 (Container, 컨테이너)

도커가 만드는 격리된 실행 환경. 컴퓨터 안의 가짜 컴퓨터입니다.

코덱스 (Codex, 코덱스)

OpenAI에서 만든 터미널 기반 AI 코딩 도구. 클로드 코드의 OpenAI 버전이라고 보면 됩니다.

클로드 코드 (Claude Code, 클로드 코드)

앤스로픽에서 만든 터미널 기반 AI 코딩 도구. 이 책의 주인공입니다.

터미널 (Terminal, 터미널)

컴퓨터에게 글자로 명령을 내리는 검은 화면. 맥에서는 "터미널" 앱, 윈도우에서는 "명령 프롬프트"입니다.

토큰 (Token, 토큰)

AI가 글을 읽고 쓸 때 사용하는 단위. 한국어 한 글자가 대략 1~2토큰입니다. 비용은 토큰 단위로 계산됩니다.

프롬프트 (Prompt, 프롬프트)

AI에게 내리는 지시. "이 파일을 정리해줘"가 프롬프트입니다.

하이쿠 (Haiku, 하이쿠)

앤스로픽의 가장 작고 빠른 클로드 AI 모델. 가볍고 저렴하지만 복잡한 작업에는 한계가 있습니다.

CLAUDE.md (클로드엠디)

클로드 코드에게 프로젝트 규칙을 알려주는 파일. 이 파일에 적힌 규칙을 클로드 코드가 따릅니다.

rm -rf (알엠 마이너스 알에프)

파일과 폴더를 묻지도 따지지도 않고 전부 지우는 명령어. 휴지통에도 안 갑니다. 위험합니다.

SSH (에스에스에이치)

다른 컴퓨터에 터미널로 원격 접속하는 방법.

부록 C. 안전 점검 체크리스트

(모니터 옆에 붙여두세요)

욜로 모드를 켜기 전에

깃으로 저장했는가?

git add . 그리고 git commit -m "욜로 전 저장" 을 쳤는지 확인합니다.

브랜치를 나눴는가?

git checkout -b 실험용 으로 가지를 만들었는지 확인합니다.

잘못되면 원본 브랜치로 돌아가면 됩니다.

도커 안인가?

docker run 으로 컨테이너를 만들어서 들어갔는지 확인합니다.

도커 안이면 맥이 안전합니다.

지시가 구체적인가?

"정리해줘"가 아니라 "logs 폴더의 .log 파일만 지워줘"처럼 구체적인지 확인합니다.

모호한 지시가 사고의 원인입니다.

API 키가 노출되지 않는가?

.gitignore 파일에 .env가 들어 있는지 확인합니다.

깃허브에 올리기 전에 git status로 .env가 목록에 없는지 봅니다.

맥스 턴을 걸었는가?

--max-turns 20 옵션을 붙였는지 확인합니다.

안 걸면 클로드 코드가 끝없이 돌 수 있습니다.

비용 상한선을 설정했는가?

console.anthropic.com에서 이번 달 상한선을 정했는지 확인합니다.

CLAUDE.md에 금지 규칙이 있는가?

rm -rf 금지, .env 깃 추가 금지, 시스템 파일 수정 금지 같은 규칙을 적어뒀는지 확인합니다.

작업이 끝난 뒤에

git status로 변경 파일을 확인했는가?

클로드 코드가 무엇을 바꿨는지 반드시 확인합니다.

git diff로 변경 내용을 읽어봤는가?

예상하지 못한 변경이 없는지 봅니다.

잘된 것만 골라서 커밋했는가?

git add 파일이름 으로 원하는 것만 골라 저장합니다.

나머지는 git checkout -- 파일이름 으로 되돌립니다.

API 키가 커밋에 포함되지 않았는가?

git diff --cached 로 커밋될 내용을 한 번 더 확인합니다.

사고가 났을 때

Ctrl+C로 멈췄는가?

git status, git diff로 상황을 파악했는가?

git checkout -- . 또는 git reset --hard로 되돌렸는가?

CLAUDE.md에 재발 방지 규칙을 추가했는가?