VIBE CODING · LESSON 03

하네스 엔지니어링
AI가 잘 일하는 환경을 만든다

코드를 짜는 사람에서 환경을 설계하는 사람으로

하네스 · CLAUDE.md · Hook · Skill · Rule · 카파시

RECAP

우리가 이미 가진 것

1-2차시에서 배운 도구들, 기억나나요?

W

Warp

터미널+에디터+파일트리.

C

Claude Code

자연어로 시키는 AI.

P

Plan Mode

먼저 설계, 나중에 실행.

S

Skill / Hook / Plugin

매뉴얼 · 법률 · 확장팩.

M

MCP / CLI

통역사 · 심부름꾼.

이 모든 것을 하나로 묶는 개념이 있습니다.

WHY HARNESS?

"해줘" vs "환경을 만들어줘"

✕

"이거 만들어줘"

AI가 만듦 → "아닌데..." → 다시 시킴 → **시간만 날림**

✔

"환경을 깔아줄게, 그 안에서 일해"

AI가 일관되게 **좋은 결과**를 만들어낸다

> **"모델 교체로 5% 개선하는 것보다, 하네스 설계로 15% 개선하는 것이 현실적이다."**

3 LEVELS

프롬프트 → 컨텍스트 → 하네스

1

프롬프트 엔지니어링

"이렇게 말해봐"

질문을 잘 다듬으면 답이 좋아진다.

2

컨텍스트 엔지니어링

"이런 배경지식을 줘봐"

배경을 알려주면 맥락을 이해한다.

3

하네스 엔지니어링

"환경 자체를 설계하자"

작업 환경을 만들어주면 혼자서도 잘한다.

← 우리는 여기

REAL RESULTS

같은 모델인데 결과가 다르다?

L

LangChain

코딩 벤치마크 30위 → Top 5 (+14%p)
모델은 그대로. 셀프 검증 루프 + 컨텍스트 자동 수집 + 둠 루프 탐지를 추가한 것만으로 순위가 뒤집혔습니다.

O

OpenAI

5개월간 100만 줄 코드, 사람이 친 줄은 0.
엔지니어 3~7명이 한 일은 코딩이 아니라 AI가 잘 일할 환경을 설계하는 것이었습니다.

A

Anthropic

같은 모델, 같은 작업. 혼자 시키면 20분/$9에 실패.
3에이전트 하네스(계획자+생성자+평가자)를 깔면 $200이지만 완전히 작동합니다.

S

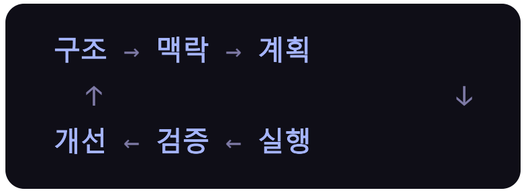
Stripe

매주 1,000개 PR을 완전 무인 자동 머지.
전문화된 소형 에이전트 + 매 스텝 검증 게이트 + 정밀 컨텍스트 관리로 가능해졌습니다.

6 AXES

하네스의 6가지 축

순환하며 점점 나아지는 구조



1

구조

뭘 깔아두는가

2

맥락

AI가 뭘 아는가

3

계획

뭘 할지 정하는가

4

실행

어떻게 시키는가

5

검증

어떻게 믿는가

6

개선

어떻게 나아지는가

폴더 구조가 품질을 결정한다

```
# 프로젝트 뼈대
my-project/
├── src/          # 비즈니스 로직
├── docs/         # 사람의 참고서
├── tests/        # 검증 인프라
├── .claude/      # AI 설정 (rules, hooks, skills)
└── CLAUDE.md     # 프로젝트 지도
```

핵심

docs/ = 사람의 참고서, **.claude/** = AI의 작업 설정.
분리해서 관리하기.

DATA **3개월 후: 구조 없음 30% vs 클린 아키텍처 85% 성공률**

CLAUDE.md — AI가 아는 것을 관리한다

```
# 상속 구조
~/.claude/CLAUDE.md        [User — 모든 프로젝트 공통]
└── my-app/
    ├── CLAUDE.md           [Project — 이 프로젝트 전용]
    └── .claude/rules/      [상황별 조건부 로드]
```

원칙

Progressive Disclosure

다 때려넣지 말고, 필요한 것만 필요할 때 보여준다.

Tip: CLAUDE.md 200줄 이상이면 rules/로 분리 검토

"해줘" 대신 "같이 계획 세우자"

!

안티패턴

"해줘" → 만듦 → "아닌데" → 다시 → **시간 낭비**

✔

패턴

"같이 계획 세우자" → AI가 질문 → 사람이 답 → 합의 → 실행 → **높은 성공률**

핵심 인사이트

사람은 머릿속 전제를 다 담지 못한다. AI가 질문해야 빠진 맥락이 드러난다.

2차시 **Plan Mode — 이미 배웠다! 오늘은 더 넓은 그림에서 본다.**

혼자 vs 부하 파견 vs 팀

1

혼자 (Single)

AI 하나에게 직접 시킨다. 일상 작업의 **90%**는 이걸로 충분합니다.

2

부하 파견 (Subagent)

메인 AI가 서브에이전트에게 위임하고, 결과만 요약으로 받아 종합. **병렬 처리** + 메인 맥락 보호.

3

팀 (Team Mode)

설계자·구현자·리뷰어 등 여러 에이전트가 서로 소통하며 협업. 강력하지만 **토큰 ~7배**.

핵심

완료 기준을 정하고, 충족할 때까지 반복. 사람은 **기준만 정하면** 됩니다.

만든 AI ≠ 평가 AI

Anthropic: "자기 작업을 평가하면 mediocre해도 자신있게 칭찬한다"

해법: 만든 AI와 확인하는 AI를 분리

G

Generator (만드는 AI)

코드를 작성하고 기능을 구현합니다. 하지만 자기가 만든 결과를 스스로 평가하면 항상 "잘 했다"고 답합니다.

E

Evaluator (평가하는 AI)

만든 AI와 **다른 모델, 다른 컨텍스트**에서 결과를 평가합니다. 한 모델의 맹점을 다른 모델이 잡습니다.

핵심

검증을 코드에서 화면까지 확장하면 사람이 끼어들 일이 줄어든다.

3번 반복 → Skill, 3번 틀리면 → Rule

관측

세션 분석, AI 산출물 품질 모니터링

패턴

반복 작업 → **Skill**로 자동화
반복 실수 → **Rule**로 방지

단순화

안 쓰는 건 치운다. 모델이 좋아지면 하네스를 재평가.

좋은 신호 ✔

같은 말 안 함
실수가 규칙 됨
불필요한 것 줄어듦

나쁜 신호 ⚠

검수 시간 ↑
스킬 많은데 안 씀
CLAUDE.md 관리 안 됨

CLAUDE.MD

65줄이 세상을 바꿨다

K

카파시의 전환

안드레 카파시(전 Tesla AI 디렉터, OpenAI 창립멤버)가 2026년 1월에 자기 코딩 방식 전환을 공유했습니다. 80% 수동 → 80% 에이전트 기반으로. 에이전트를 쓰면서 자주 겪는 4가지 실패 패턴을 관찰했고, Forrest Chang이 그 관찰을 **CLAUDE.md 파일**로 변환했습니다.

☆

GitHub #1 트렌딩

28일 연속 GitHub 트렌딩 1위. **22만+ 스타**를 받으며 "AI와 일하는 방법"에 대한 업계 표준이 되었습니다.

65

단 65줄

원문은 단 65줄. 카파시의 말: "단순함이 미덕이 아니라 **단순함이 작동의 조건**이다."

카파시 CLAUDE.md 4원칙

1

Think Before Coding

가정하지 말고 물어봐. 여러 해석이 있으면 조용히 하나를 고르지 말고 전부 보여줘.

2

Simplicity First

최소 코드, 추측 없음. 요청 안 한 기능 추가 금지. 200줄이 50줄로 될 수 있으면 다시 써.

3

Surgical Changes

내가 만진 것만 바꿔. 옆에 있는 코드 개선 금지. 기존 스타일에 맞춰.

4

Goal-Driven

성공 기준을 정하고 반복. "검증 추가" 말고 "테스트 통과시켜."

카파시의 원칙을 우리 도구로 구현한다

카파시 원칙	6축	우리 도구
Think Before Coding	계획 (Planning)	Plan Mode (2차시!)
Simplicity First	구조 + 개선	CLAUDE.md 작성 (오늘!)
Surgical Changes	검증	Hook 만들기 (오늘!)
Goal-Driven	검증 + 실행	체크리스트

CHECK **작동하고 있다면: diff에 불필요한 변경이 줄고, 재작업이 줄고, 실수 전에 질문이 나온다.**

이론은 여기까지 — 이제 실제로

제가 실제로 쓰는 하네스, 사례로 보기

하네스 = AI가 매번 헤매지 않도록 **'작업 환경'을 미리 세팅**해 두는 것.
제가 세팅해 둔 걸 다음 세 가지 사례로 보여드릴게요.

①

한 번 적어두기

매번 설명하기 귀찮은 규칙을 한 번만 적어두면, 모든 작업에서 자동으로 지켜진다. — 제가 직접 만든 설정

②

잘 만든 걸 가져다 쓰기

남이 잘 만들어 공개해 둔 하네스를 통째로 설치해서 쓴다. — oh-my-claudecode

③

실수에서 배우기

AI가 놓친 걸 발견하면, 그 자리에서 고쳐 **다시는 안 틀리게** 장치를 붙인다. — 직접 겪은 사례

> **다음 장부터 "아, 이게 하네스 사례구나" 하고 따라오시면 됩니다.**

"이렇게 시켰더니, 클로드가 이렇게 만들어줬어요"

클로드와 대화하면서 불편했던 부분, 반복 설명하게 되는 부분을 하나씩 다듬어 나가는 게 하네스 엔지니어링의 핵심입니다. 했던 실수, 했던 불편함을 반복하지 않게 만드는 과정이 AI와의 협업을 더욱 효과적으로 만들어줍니다.

🗣️ "기본적인 규칙은 반복 설명하지 않게 만들고 싶다."

□ 자주 쓰는 규칙(한국어로 답하기, 위험한 명령은 먼저 물어보기 등)을 딱 한 번 적어두는 '메모장'을 만들어, 무슨 작업을 하든 자동으로 먼저 읽게 했어요.

□ `~/.claude/CLAUDE.md` (전역 메모리)에 규칙을 적으면 모든 세션 시작 시 자동 로드

🗣️ "새로운 터미널 창에서 작업을 시작해도 클로드가 관련된 이전 맥락을 늘 기억하고 행동했으면 좋겠다."

□ 작업을 시작하면 이 프로젝트 요약과 '지난번에 하다 만 일'을 자동으로 펼쳐 보여주게 했어요. 처음부터 다시 설명할 필요가 없어요.

□ `SessionStart` 훅이 프로젝트 요약·미완료 작업 목록을 세션 첫 화면에 자동 출력

🗣️ "클로드가 계획의 모호함이 줄어들 때까지 집요하게 나에게 물어보면서 플랜을 구체화한 뒤에 코드 작업을 시작했으면 좋겠다."

□ 큰일은 '어떻게 진행할지' 저한테 먼저 묻게 하고, 그걸 안 물으면 다음 단계로 못 넘어가게 막아뒀어요.

□ `EnterPlanMode` 훅이 진행 방식을 묻게 하고, 안 물으면 `ExitPlanMode`를 막음(차단)

🗣️ "한 세션이 마무리되고 나면 하네스 엔지니어링을 다듬는, 그 세션의 시행착오를 복기하고 개선하는 작업까지 자동화했으면 좋겠다."

□ 일이 끝나면 '뭘 잘했고 뭘 틀렸는지' 스스로 정리해 기억에 저장하게 했어요. 그래서 다음엔 같은 실수를 피해요.

□ `Stop` 훅이 종료 직전 회고를 받아 `memory/` 파일에 적립

핵심은 '부탁'이 아니라 '자동'입니다. 한 번 세팅하면, 부탁하지 않아도 알아서 지켜져요. (수업의 6축으로 보면 — 맥락·계획·검증·개선)

잘 만든 하네스를 통째로 — oh-my-claudecode

①은 제가 직접 만든 거예요. 그런데 이런 장치들을 잘 묶어 **공개해 둔 것**도 있어요.
설치 한 줄이면, 아래 기능이 한 번에 따라옵니다. (직접 안 만들어도 돼요)

먼저 계획부터

바로 시작하지 않고, 여러 관점이 의논해 '계획'부터 세워줘요.

완성까지 알아서

아이디어만 던지면 계획·실행·점검까지 스스로 끝내줘요.

진짜 되는지 점검

결과가 실제로 작동하는지 여러 단계로 자동 확인해요.

팀처럼 협업

여러 AI가 역할을 나눠 동시에 일하게 할 수 있어요.

될 때까지 반복

"이 조건이 되면 끝"만 정해주면, 그 기준을 채울 때까지 알아서 계속 고쳐요.

```
# 터미널에 이 한 줄이면 위 기능이 전부 설치됩니다
$ npx oh-my-claudecode@latest install
```

AI는 화면을 '눈'으로 보지 못한다

부족한 점을 발견했을 때 "다음엔 잘하자"가 아니라, **그 자리에서 고쳐 다시는 안 틀리게** 만드는 것 — 이게 하네스의 핵심입니다.

🗣️ 무슨 일이 있었나

AI에게 웹 화면을 만들라고 했더니, 글자가 칸 밖으로 삐져나가고 옆 칸과 겹쳤어요. AI는 코드를 '읽을' 뿐 사람처럼 화면을 '볼' 수는 없어서, 스크린샷을 줘도 이런 미세한 어긋남은 잘 못 잡아냅니다.

□ 그래서 어떻게 했나

"눈으로 말고 **숫자로 재서** 확인하자." 화면 요소가 칸을 넘쳤는지·겹쳤는지 자동으로 측정하는 작은 검사기를 만들고, AI가 화면을 만들 때마다 그 검사를 자동으로 돌리게 붙였어요.

□ `Playwright` 로 요소의 실제 크기·겹침을 숫자로 측정 + 스크린샷 찍을 때 점검을 띄우는 `PostToolUse` 훅

한 번 실수를 한 번 겪음 → 검사기를 만들어 영구 장치로

자동 화면 작업이 끝나면 자동으로 "넘침·겹침 없나?" 점검

재발 방지 다음 프로젝트에서도 같은 실수가 안 나옴

> **참고로 — 이 발표 슬라이드도 바로 그 검사기로 "글자가 칸 밖으로 안 삐져나갔는지" 숫자로 확인하고 만들었습니다. □**

HANDS-ON

harness-selfcheck.md로 내 하네스 진단하기

📥 harness-selfcheck.md 다운로드 (구글 드라이브)

1 위 링크에서 파일 다운로드 → **~/.claude/** 폴더에 저장 (전역 폴더!)

2 내 프로젝트 폴더에서 Claude Code 실행 → 플랜 모드 켜기 `/plan`

3 `~/.claude/harness-selfcheck.md 읽고 내 하네스를 진단해줘` 입력

4 AI의 질문에 답하며 함께 만들어 나가기

이런 질문들이 나옵니다

축	진단 질문	잘 되면 ✔
구조	AI가 파일 어디에 놓을지 바로 아는가?	묻지 않고 맞는 곳에 놓는다
맥락	CLAUDE.md에 기술 스택이 적혀 있는가?	같은 말 두 번 안 한다
계획	"해줘"만 하는가?	AI가 먼저 질문한다
실행	반복 작업을 자동화했는가?	사람은 시작과 확인만
검증	위험한 명령에 안전장치가 있는가?	차단 장치가 작동
개선	AI 실수를 규칙으로 만들었는가?	점점 단순해진다

지금부터 각자 진단을 시작해 봅시다!

넘어지는 자리에만 기둥을 박으세요

W1 **Week 1**

빈 CLAUDE.md 만들기 (프로젝트 이름 + 기술 스택)

W2 **Week 2**

AI가 틀린 것 1줄 추가 → "TODO 금지" 같은 규칙

W3 **Week 3**

3번 틀린 패턴 → `.claude/rules/` 에 Rule 파일로 분리

W4 **Week 4**

3번 반복한 작업 → `.claude/skills/` 에 Skill로 만들기

W5 **Week 5**

위험한 명령 → `.claude/hooks/` 에 Hook으로 차단

W6 **Week 6**

안 쓰는 거 정리. 좋은 하네스는 점점 단순해진다.

> **"처음부터 완벽하려 하지 마세요. 그게 결국 가장 빠른 길입니다."**

WRAP-UP

사람의 역할이 바뀌고 있습니다

> **"AI가 코드를 쓰는 시대, 사람의 역할은 '잘 짜기'에서 '잘 일하는 환경을 만들기'로 바뀐다."**

숙제

1. 오늘 구축한 하네스 엔지니어링을 일주일간 써보고, AI가 자주 틀리는 것 **한 가지를 개선해보기**
2. **(선택)** Hook 또는 Skill 하나 더 만들어보기
3. **(도전)** oh-my-claudecode: `npx oh-my-claudecode@latest install`

참고 자료

DoRm 블로그 (이 수업의 원본 글)
`dorm.dev/blog/harness-engineering-v2`

WalkingLabs 가이드
`walkinglabs.github.io/learn-harness-engineering/ko/`

카파시 CLAUDE.md
`github.com/forrestchang/andrej-karpathy-skills`

oh-my-claudecode
`github.com/Yeachan-Heo/oh-my-claudecode`