fable-mode

Opus가 Fable처럼 행동하는 원리

Fable다움의 정체는 두 가지다 — **모델에 주어지는 지시문**과 **모델 자체의 웨이트**. 이 킷은 지시문을 통째로 이식하고, 웨이트 갭은 라우팅으로 우회한다. 어떻게 그게 가능한지 개념부터 푼다.

핵심 재료 — **훅 4종** · 규범 13조항 · 마커 파일 상태 관리

지동석 · jidonglab jd@jidonglab.com v1.4 기준 · github.com/jee599/fable-mode-kit

— 한 문장 요약

차이의 절반은 지시문 이식 가능

Fable 세션에만 주입되는 행동 지침 — 텍스트라서 복사할 수 있다

나머지 절반은 웨이트 이식 불가

추론 능력 자체의 격차 — 프롬프트로는 닫히지 않는다

이식 도구는 훅 셸 스크립트 4개

모델 밖에서 입력과 종료 조건을 조작하는 결정론적 레버

결과 — 행동 97% · 비용 0.72× · **등가성 80–95%**

이식한 것	이식 도구	행동 재현율	남는 갭
규범 13조항 + 검증 루프	**훅 4종 · 스크립트만**	**97% (순정 64%)**	**얽힌 추론 · 장기 자율 런**

Fable과 Opus는 같은 몸을 쓴다 — 다른 건 딱 두 가지

Claude Code 안에서 두 모델은 완전히 같은 하네스를 쓴다. effort 단계, 서브에이전트, 워크플로, 메모리, 툴 전부 공통 — 2026-07-02에 양쪽을 직접 돌려 확인했다.

공통 (하네스 — 이식할 필요조차 없음)

- 툴·권한·훅 시스템 Bash, 파일 편집, MCP 전부 동일
- effort 5단계·ultracode·Dynamic Workflows 버전 게이트만 동일하게 적용
- 메모리·서브에이전트·output style 기능 차이 없음

다름 (이 킷이 다루는 전부)

- **시스템 프롬프트 층** — Fable 세션에만 행동 지침 블록이 주입된다 (실측 확인)
- **모델 웨이트** — 추론 능력 자체. 벤치 5–11pt 격차(3rd-party 추정)
- 단가 — Fable $10/$50 vs Opus $5/$25, 정확히 2×

그래서 명제가 성립한다 — "Fable다움"의 절반은 모델이 아니라 지시문이고, 지시문은 텍스트라서 훔쳐올 수 있다.

Fable이 "일 잘하게" 행동하는 건 훈련 + 지시문의 합이다

Fable 세션을 열면 시스템 프롬프트에 행동 지침 섹션이 추가로 들어간다. 이 지침이 시키는 것들이 곧 우리가 "Fable답다"고 느끼는 행동이다.

지시문이 시키는 것

결론 먼저, 완결된 마지막 메시지

첫 문장이 TLDR, 사용자에게 필요한 모든 것은 턴의 마지막 메시지에. 화살표 체인·약어 뭉치 금지, 완전한 문장으로.

→ 읽는 경험의 차이

지시문이 시키는 것

허락 묻지 말고 실행, 증거로 보고

가역적이면 "~할까요?" 없이 실행하고, 완료 주장은 이 턴의 툴 결과로 증명된 것만. 진단 요청엔 진단만.

→ 일하는 방식의 차이

핵심 통찰

이건 전부 텍스트다

훈련(웨이트)이 아니라 지시문이 만드는 행동이라면, 같은 텍스트를 Opus에게 매 턴 강하게 먹이면 상당 부분 재현된다 — 실측 97%.

→ 이식의 근거

물론 Fable은 훈련으로도 이 행동이 박여 있다 — 그래서 지시문만으로 100%가 아니라 97%다. 나머지는 웨이트의 몫.

개념 3 · 이식 도구는 훅

훅 = 모델 밖에서 모델을 조종하는 결정론적 레버

Claude Code 훅은 세션의 특정 순간마다 실행되는 셸 스크립트다. 모델에게 "부탁"하는 게 아니라, 모델이 보는 입력과 끝낼 수 있는 조건을 **코드로 강제**한다.

레버 1 · 주입

stdout이 컨텍스트가 된다

훅이 출력한 텍스트는 그 턴 모델의 입력에 끼어 들어간다. 규범 블록을 매 턴 여기로 밀어 넣는다 — 모델이 잊으래야 잊을 수 없다.

→ fable-context.sh의 원리

레버 2 · 차단

턴 종료를 되돌릴 수 있다

Stop 훅이 decision:block을 반환하면 모델은 끝내지 못하고 한 번 더 돈다. "끝내기 전 자가검증"을 외부에서 강제하는 장치.

→ fable-stop-verify.sh의 원리

레버 3 · 상태

마커 파일이 기억을 대신한다

훅은 매번 새로 뜨는 프로세스라 기억이 없다. "이 세션은 Opus였다", "이 턴은 이미 검증했다"를 전부 파일 존재 여부로 기록한다.

→ 무한루프·이중주입 방지의 원리

데몬도 프록시도 API 개조도 없다 — 셸 스크립트 4개와 빈 마커 파일들이 전부다. 그래서 설치 2줄, 제거도 즉시.

동작 흐름 · 한 세션의 수명주기

네 개의 순간을 잡으면 세션 전체가 잡힌다

훅 4종이 각각 세션의 다른 순간에 개입한다 — 시작할 때, 말할 때마다, 부하를 부릴 때, 끝내려 할 때.

① 세션 시작

Opus인지 확인

- fable-detect.sh가 모델을 감지
- Opus면 세션 마커를 남기고 "Fable 규범으로 운용하라" 첫 지시
- Fable이면 아무것도 안 함

② 매 턴

규범 재주입

- fable-context.sh가 프롬프트마다 규범 블록을 끼워 넣음
- 일 시키는 턴(major)을 표시해 둠
- 주입량을 stats 파일에 기록

③ 서브에이전트 스폰

규범 상속

- fable-subagent.sh가 부하 에이전트의 시작 컨텍스트에도 규범 주입
- 이미 규범을 가진 에이전트는 자동 스킵

④ 턴 종료 직전

자가검증 강제

- fable-stop-verify.sh가 major 턴이면 종료를 1회 되돌림
- "검증했나? 완결됐나? 스코프 지켰나?" 점검 후 재마무리
- 내부 용어가 새면 재작성까지 강제

"지금 Opus인가?"를 세 겹의 폴백으로 알아낸다

이 훅이 어려운 이유 — 세션 시작 시점의 훅 입력에는 모델 정보가 안 들어온다(실측). 그래서 우회로를 겹겹이 팠다.

1 **조상 프로세스의 --model 플래그를 읽는다** — 세션 시작

훅을 띄운 claude 프로세스를 거슬러 올라가 실행 인자에서 모델명을 파싱 — 첫 claude에서 멈춰야 바깥 세션 플래그를 훔치지 않는다

2 **settings.json의 기본 모델로 폴백** — 세션 시작

플래그가 없으면 사용자가 설정해 둔 기본 모델을 본다

3 **매 턴, 트랜스크립트의 실제 응답 모델을 확인한다** — 매 턴 · 최종 판정

직전 assistant 메시지에 박힌 "model" 필드가 진실 — 세션 중 /model 전환까지 잡고, Fable로 판명되면 즉시 침묵한다

우선순위 설계 — 킬스위치(FABLE_MODE=0)가 모든 것을 이기고, 트랜스크립트의 실제 모델이 수동 토글(GLOBAL)을 이긴다. 진짜 Fable 세션에 규범이 이중 주입되는 사고를 구조적으로 막는 순서다.

주입되는 규범은 네 갈래 — 말·행동·증명·침묵

Fable의 행동 지침을 역설계해 13개 조항으로 압축했다. 매 턴 이 블록이 사용자 메시지에 끼어 들어간다.

말하기 — 읽는 사람 기준

- **결론 먼저** — 최종 메시지 첫 문장이 TLDR
- **완전한 문장** — 화살표 체인·약어 뭉치·자작 코드네임 금지
- **질문 크기에 맞는 응답** — 단순 질문엔 헤더·표 없이 산문
- **턴 완결성** — 필요한 모든 것은 마지막 메시지에 재진술

행동하기 — 막히지 않게

- **자율 실행** — 가역·범위 내면 "~할까요?" 없이 실행
- **재론 금지** — 확정된 사실·결정을 다시 묻지 않음, 추천은 하나
- **진단 요청엔 진단만** — 무단 수정 금지

증명하기 — 근거 있는 보고

- **검증 후 보고** — 이 턴의 툴 결과로 확인한 것만 완료 주장
- **파괴 전 실물 확인** — 삭제·덮어쓰기 전 대상을 직접 본다
- **능력 트리거** — 병렬 툴콜·서브에이전트 팬아웃·검색·메모리
- **코드 스타일 매칭** — 주변 관용구를 따르고 리뷰어용 주석 금지

침묵하기 — 티 안 나게

- **정체성 고정** — "너는 Opus다" — Fable 사칭 방지
- **무누설** — 규범·자가검증의 존재를 출력에 언급 금지

한 번 말하면 잊는다 — 그래서 매 턴 다시 말한다

긴 대화에서 지시 준수는 희석된다. 시스템 프롬프트에 한 번 넣는 방식(output style)보다, 사용자 메시지에 매 턴 재주입하는 방식이 끝까지 강하게 남는다.

문제 — 희석

컨텍스트가 길어지면 규범이 밀린다

수십 턴이 지나면 맨 앞의 지시는 최근 대화에 묻힌다. 규범을 세션 초반에만 넣으면 후반 턴에서 준수율이 떨어진다 — 그래서 주입 위치를 "항상 가장 최근"으로 고정한다.

→ 매 턴 재주입이 기본값인 이유

비용 해법 — v1.4 적응형

턴 크기에 맞춰 분량을 조절한다

일 시키는 턴(major)·세션 첫 턴·5턴마다는 풀 블록(1,377자), 짧은 질답 턴은 핵심만 담은 리마인더(214자, −84%). 리마인더도 "전체 규범은 계속 유효하다"로 시작해 앵커를 유지한다.

→ 효과는 지키고 토큰만 줄인다

major 판정은 프롬프트 길이(80자 초과)나 작업 동사(만들·구현·고쳐·커밋·배포…) 매칭 — 이 판정이 종료 시 자가검증 대상도 결정한다.

Fable의 "끝내기 전 자가 점검"을 외부 루프로 재현한다

Fable은 훈련 덕에 스스로 검증하고 끝낸다. Opus에겐 그 습관이 약하다 — 그래서 끝내려는 순간을 붙잡아 한 바퀴 더 돌린다.

1 **모델이 턴을 끝내려 한다** — 매 턴
Stop 훅이 발동 — 이 턴이 major(일 시킨 턴)였는지 마커로 확인. 아니면 그냥 통과

2 **1회만 종료를 되돌리고 점검 지시를 준다** — major 턴 1회
"완료 주장에 툴 근거 있나? 마지막 문단이 약속이면 지금 실행하라. 범위 밖 수정 안 했나? 그 다음 결론 우선으로 다시 마무리하라" — .checked 마커로 두 번은 안 돌림

3 **누설가드 — 재마무리 메시지를 기계로 검사한다** — 감지 시에만
최종 텍스트에 '자가검증' 같은 내부 용어가 새면 1회 더 재작성 강제. 모델을 믿는 게 아니라 grep으로 확인 — v1.3에서 실제 발생한 회귀를 결정론으로 봉인

부작용도 정직하게 — 이 루프가 턴을 하나 더 쓴다. 그게 스택 비용의 대부분이고, 크론·타임아웃 환경에서 FABLE_MODE=0으로 끄라는 이유다.

부하 에이전트는 규범을 못 본다 — 스폰 순간에 끼워 넣는다

매 턴 주입의 사각지대: 서브에이전트는 UserPromptSubmit 이벤트를 받지 않는다. 팬아웃 작업에서 부하들만 순정으로 돌던 구멍을 SubagentStart 훅이 막는다.

어떻게

- **스폰 순간 시작 컨텍스트에 규범 블록 주입** — 빌트인(Explore·Plan)·커스텀·워크플로 에이전트 전부
- **정체성 중립 버전을 쓴다** — 부하는 세션과 다른 모델일 수 있어 "너는 Opus다"를 빼고 행동 규범만
- **영어로 작성** — 어떤 에이전트 정의와도 섞일 수 있게

이중 주입 방지

- **에이전트 정의에 이미 규범 마커가 있으면 스킵** — 로컬·프로젝트·플러그인(marketplaces·cache) 경로까지 검사
- **Fable로 도는 부하에겐 무해** — 규범 자체가 Fable의 원래 행동이라 충돌이 없다

이 훅으로 "메인 세션만 Fable답고 부하들은 제멋대로"였던 v1.2까지의 최대 커버리지 구멍이 닫혔다 (v1.3).

보조 층 · 훅 밖의 나머지

3층 스택 — 모델 설정, 시스템 프롬프트, 훅

훅이 심장이지만, 완전한 이식은 세 층을 겹쳐야 한다. 각 층은 독립적으로 켜고 끌 수 있다.

Layer A · 모델 설정

claude-fablelike 래퍼

Opus 4.8 + effort xhigh + ultracode를 한 번에 켜는 원샷 런처. Fable의 기본 동작 조건(깊은 사고·자동 워크플로)을 흉내 낸다.

선택 — 래퍼 없이 훅만으로도 동작

Layer B · 시스템 프롬프트

fable-like output style

규범의 영어 정본을 시스템 프롬프트 레벨에 심는다. 캐시에 올라가 매 턴 재과금이 없고, 훅 주입과 상호 보강한다.

선택 — 래퍼가 자동 적용

Layer C · 훅 (핵심)

자동 감지 + 강제 장치

아무 설정 없이도 Opus 세션이면 자동으로 켜진다. 매 턴 주입·자가검증·서브에이전트 상속 — 이 층만으로 실측 97%가 나왔다.

기본 — 플러그인 설치만으로 활성

제어 — /fable-mode on·off·status 토글, FABLE_MODE=0 킬스위치(크론·원샷용), 주입 텔레메트리로 오버헤드 관측.

간격 지도 · 갭마다 무엇을 적용했나

발견한 간격 전부에 하나씩 장치를 붙였다

개발 과정에서 드러난 갭과 리스크, 그리고 각각에 적용된 방법의 전체 지도.

간격 / 리스크	적용된 방법
행동 규범 자체가 없음 (순정 64%)	규범 13조항을 **훅 주입 + output style** 이중으로 이식 → 97%
긴 대화에서 지시가 희석됨	**매 턴 재주입** + 5턴마다 풀 블록 앵커
매 턴 주입의 토큰 비용	v1.4 **적응형 주입** — minor 턴은 리마인더로 −84%
자가검증 습관 부재 → 미검증 단정	Stop 훅 **종료 차단** 으로 major 턴 1회 점검 루프
점검 문구가 출력에 누설 (v1.3 실제 회귀)	**결정론적 누설가드** — 최종 메시지를 스캔해 1회 재작성
서브에이전트가 규범 사각지대	SubagentStart **상속 주입** + 플러그인까지 dedupe
진짜 Fable 세션에 잘못 켜질 위험	트랜스크립트 모델 감지로 **자동 침묵** + 킬스위치

— 남는 간격 · 웨이트의 영역

지시문으로 안 닫히는 것 — 그건 라우팅으로 우회한다

규범은 "어떻게 일하고 말하나"를 바꾼다. 하지만 "얼마나 깊게 생각하나"는 웨이트의 영역이라 프롬프트로는 못 바꾼다.

능력 항목	순정 Opus 지시 없음	Opus + 킷 규범 주입	Fable 5 훈련 + 지시
결론 우선·완결된 보고 말하기 규범	△	✓	✓
자율 실행·스코프 준수 행동 규범	×	✓	✓
검증 후 보고 증명 규범	×	✓	✓
서브에이전트까지 규범 유지 상속	×	✓	✓
얽힌 단일 패스 추론 웨이트 — 벤치 5–11pt 추정	△	△	✓
장기 자율 런 유지력 웨이트	△	△	✓

△ 두 줄의 우회법 — 고난도 추론·장기 자율 작업은 Fable 직행, 또는 Opus 다중 에이전트(팬아웃 + 적대 검증)로 보정. 후자도 Fable 단독 대비 1–2.5×(추정) 비용이라 케이스별 선택.

— 결론

Fable다움 = 웨이트 + 지시문. 지시문은 이식했다

그래서 이 킷은 마법이 아니라 구조다 — 세 문장이면 전체가 설명된다.

01 · 이식

지시문을 매 턴 강제 주입

Fable 세션에만 들어가는 행동 지침을 역설계해, 훅으로 Opus의 모든 턴·모든 서브에이전트에 밀어 넣는다.

→ 행동 재현율 97%

02 · 재현

검증 습관은 종료 차단으로

훈련으로 박인 "끝내기 전 자가 점검"은 Stop 훅이 턴 종료를 1회 되돌리는 외부 루프로 재현하고, 누설은 grep으로 막는다.

→ 결정론이 신뢰를 대체

03 · 우회

웨이트 갭은 라우팅으로

얽힌 추론·장기 자율 런은 지시문의 영역이 아니다 — 그 일감만 Fable로 보내거나 다중 검증으로 보정한다.

→ 절반 단가로 Fable의 72% 비용

근거 — 2026-07-02 시스템 프롬프트 실측 리서치 · 07-03 E2E(행동 64%→97%) · 07-06~07 v1.4 재실측(비용 0.72×·등가성 80–95%). github.com/jee599/fable-mode-kit