AIO 1기 · 최종 프로젝트 특강

Agent Skills로 최종 프로젝트 제대로 하기

addyosmani/agent-skills — 시니어 엔지니어의 작업 순서를 AI 코딩 에이전트에 입히는 방법

2026년 11월 · 출처: github.com/addyosmani/agent-skills

왜 필요한가

AI 코딩 에이전트는 가장 짧은 길로 '완료'합니다

빠뜨리는 것 ①

스펙 없이 바로 코딩

요구사항 명세(FR/NFR)와 실제 구현이 따로 놀고, 발표 Q&A에서 "왜 이렇게 만들었나"에 답하지 못합니다.

빠뜨리는 것 ②

테스트는 "나중에"

Auto-Healing 시연과 통합 테스트 결과서를 쓸 때가 되어서야 깨진 곳이 한꺼번에 드러납니다.

빠뜨리는 것 ③

리뷰·보안 점검 생략

API 키가 커밋되고, 승인 없이 위험한 도구가 실행되는 경로가 그대로 남습니다.

agent-skills는 이 빠진 단계를 에이전트가 반드시 밟도록 강제하는 워크플로우 모음입니다.

한눈에 보기

addyosmani/agent-skills 구성

25

스킬

생명주기 스킬 24개 + 라우팅 메타 스킬 1개

9

슬래시 커맨드

/spec /plan /build /test /review /ship 등

4

전문가 페르소나

코드 리뷰어 · 테스트 엔지니어 · 보안 감사 · 웹 성능 감사

70+

지원 에이전트

Claude Code · Codex · Gemini CLI · Cursor · Copilot 등

MIT

라이선스

자유롭게 복사·수정해 팀 전용으로 바꿔 쓸 수 있음

11.7만

GitHub 스타

2026년 11월 기준, 포크 1.2만

출처: github.com/addyosmani/agent-skills README (2026-11-18 확인)

스킬이란

스킬 = 읽는 문서가 아니라 실행하는 절차

```
skills/
  spec-driven-development/
    SKILL.md    # 진입점
references/ # 필요할 때만
agents/     # 페르소나
```

SKILL.md는 frontmatter(이름·언제 쓰는지) + 단계별 절차 + 체크포인트 + 종료 조건으로 된 마크다운 파일입니다.

Process, not prose

절차형

설명문이 아니라 체크포인트와 종료 조건이 있는 단계별 워크플로우

Anti-rationalization

핑계 차단표

에이전트가 단계를 건너뛸 때 대는 핑계와 반박을 미리 적어 둠

Verification

증거로 끝남

통과한 테스트, 빌드 출력, 런타임 데이터 없이는 '완료'가 아님

Progressive disclosure

필요한 것만 로드

메타 스킬이 지금 단계에 맞는 스킬만 불러와 컨텍스트를 아낌

워크플로우

6단계 생명주기와 슬래시 커맨드

DEFINE
정의
/spec
코드보다 스펙 먼저

›

PLAN
계획
/plan
작고 원자적인 작업

›

BUILD
구현
/build
한 번에 한 조각

›

VERIFY
검증
/test
테스트가 증거

›

REVIEW
리뷰
/review
코드 건강 개선

›

SHIP
배포
/ship
빠를수록 안전

/constraints 품질 기준을 한 번 정하고 끝까지 강제

/webperf 최적화 전에 먼저 측정

/code-simplify 영리함보다 명료함

/build auto 계획 1회 승인 → 작업별 테스트·커밋 자동, 실패·위험 단계에선 멈춤

Claude Code에서는 /review가 내장 명령과 겹치므로 **/agent-skills:review**로 호출합니다. 커맨드 없이도 작업 내용에 따라 스킬이 자동 활성화됩니다.

스킬 카탈로그

단계별 24개 스킬 — 굵은 글씨가 최종 프로젝트 핵심

단계	스킬
정의	interview-me · idea-refine · **spec-driven-development** · **constraint-driven-development**
계획	**planning-and-task-breakdown**
구현	**incremental-implementation** · **test-driven-development** · **context-engineering** · **source-driven-development** · doubt-driven-development · frontend-ui-engineering · **api-and-interface-design**
검증	browser-testing-with-devtools · **debugging-and-error-recovery**
리뷰	**code-review-and-quality** · code-simplification · **security-and-hardening** · performance-optimization
배포	**git-workflow-and-versioning** · **ci-cd-and-automation** · deprecation-and-migration · **documentation-and-adrs** · **observability-and-instrumentation** · shipping-and-launch

+ 메타 스킬 using-agent-skills: 들어온 작업을 알맞은 스킬로 연결하고 공통 규칙(요청받은 것만 건드린다 등)을 정의

설치

설치 — 한눈에 보기 (도구별 자세한 사용법은 다음 장)

```
# Claude Code (권장)
/plugin marketplace add addyosmani/agent-skills
/plugin install agent-skills@addy-agent-skills
# Codex CLI (v0.122+)
codex plugin marketplace add addyosmani/agent-skills
# Antigravity CLI
agy plugin install https://github.com/addyosmani/agent-skills.git
```

SSH 오류가 나면

marketplace add에 HTTPS 주소(…/agent-skills.git)를 넣기

Cursor · Gemini

Cursor는 .cursor/skills/에 복사, Gemini CLI는 gemini skills install

저장소 전체 설치

스킬 하나씩 npx로 깔면 공용 references/가 빠짐

설치 후 확인

/help로 /spec 등이 보이는지 먼저 확인

도구별 사용법 ① · Claude Code

플러그인으로 설치하고, 슬래시 커맨드로 단계를 진행

```
# 1. 설치 — Claude Code 세션 안에서 입력
/plugin marketplace add addyosmani/agent-skills
/plugin install agent-skills@addy-agent-skills
# 2. 확인 → 3. 단계별 실행
/help
/spec → /plan → /build → /test → /agent-skills:review → /ship
```

/review 충돌

내장 /review와 겹치므로 /agent-skills:review로 호출

자동 활성화

커맨드 없이 “API 설계해줘”라고 해도 맞는 스킬이 붙음

페르소나

code-reviewer · test-engineer · security-auditor를 서브에이전트로 지정

CLAUDE.md

“커밋 전 pytest”, “스펙 승인 전 코딩 금지” 같은 팀 규칙

도구별 사용법 ② · Codex CLI

플러그인으로 설치하고, @스킬이름으로 호출

```
# 1. 설치 — 터미널에서 (Codex CLI v0.122 이상)
codex plugin marketplace add addyosmani/agent-skills
codex plugin add agent-skills@agent-skills
# 2. 새 Codex 세션을 열고 채팅에서 @로 호출
@spec-driven-development InfraCare AI 스펙을 써줘
@planning-and-task-breakdown 승인된 스펙으로 작업 분해
@test-driven-development T04부터 실패하는 테스트 먼저
```

슬래시 커맨드 없음

/spec 대신 @spec-driven-development가 공식 권장 방법

자동 선택

작업만 설명해도 Codex가 맞는 스킬을 고름

AGENTS.md

저장소 루트의 팀 규칙을 작업 전에 먼저 읽음

새 세션 필수

설치 후 세션을 새로 열어야 스킬이 인식됨

도구별 사용법 ③ · Antigravity CLI (agy)

agy 플러그인으로 설치하고, /agent-skills:스킬이름으로 호출

```
# 1. 설치 (Gemini CLI에 이미 깔았다면: agy plugin import gemini)
agy plugin install https://github.com/addyosmani/agent-skills.git
# 2. 확인
agy plugin list
# 3. 새 세션에서 /agent-skills: 를 입력하면 스킬 목록이 뜸
/agent-skills:spec-driven-development
/agent-skills:code-review-and-quality
```

/build가 안 될 때

짧은 커맨드가 “No matches”면 알려진 제한 → 스킬 이름으로 호출

페르소나

code-reviewer 등 서브에이전트가 등록됨, /agents에서 선택

AGENTS.md

작업 공간 루트에 두면 팀 계획 규칙을 따름

안전하게

agy --sandbox로 실행, 커밋 전 Ctrl+r로 변경 검토

도구별 사용법 · 한 장 정리

같은 단계, 도구별 호출법

단계	Claude Code	Codex CLI	Antigravity CLI
정의	/spec	@spec-driven-development	/agent-skills:spec-driven-development
계획	/plan	@planning-and-task-breakdown	/agent-skills:planning-and-task-breakdown
구현	/build	@incremental-implementation	/agent-skills:incremental-implementation
검증	/test	@test-driven-development	/agent-skills:test-driven-development
리뷰	/agent-skills:review	@code-review-and-quality	/agent-skills:code-review-and-quality
배포	/ship	@shipping-and-launch	/agent-skills:shipping-and-launch

팀원마다 도구가 달라도 **같은 스킬**을 부르면 같은 절차가 돌아갑니다. 어느 도구든 이름이 기억나지 않으면 “follow spec-driven-development”처럼 말로 지시해도 됩니다.

우리 프로젝트에 적용

최종 프로젝트 5단계에 그대로 얹기

단계	필수 산출물	쓰는 커맨드 · 스킬
STEP 1 기획·설계	기획서(요구사항 명세), WBS	/spec · interview-me · /plan
STEP 2 API·FC	API 설계 문서, DB 스키마 설계서	api-and-interface-design · /build · /test
STEP 3 단일 에이전트	상태 전이 그래프 설계서, 자기 성찰 테스트 결과서	context-engineering · source-driven · TDD
STEP 4 협업·Healing	협업 아키텍처, AI 보안 가드레일, Auto-Healing 결과서	debugging · security-and-hardening · doubt-driven
STEP 5 최적화·배포	대화형 AI 웹 앱, 통합 테스트 결과서, 배포 아키텍처	ci-cd · observability · /ship

평가는 문서의 존재가 아니라 **설계 · 구현 · 검증의 연결**을 봅니다. 스킬을 쓰면 그 연결이 저장소 안에 **spec.md → tasks/todo.md → 테스트 → 리뷰 기록**으로 남습니다.

STEP 1 · 기획·설계

/spec — 코드 전에 PRD부터

프롬프트 예시 (주요산출물 문서의 교육용 예시 서비스)

```
/spec
InfraCare AI: IT 운영자가 장애 접수→진단→
승인→복구→검증을 한 화면에서 처리하는
멀티 에이전트 서비스.
스택: FastAPI · LangGraph · Supabase · AWS
MVP: 장애 목록, AI 진단, 복구 승인, 이력
요구사항은 FR-xx / NFR-xx ID로 정리해줘.
```

스펙의 6개 핵심 영역

목표 · 명령어 · 프로젝트 구조 · 코드 스타일 · 테스트 전략 · 경계(Boundaries)

경계는 3단계로

Always 커밋 전 pytest 실행
Ask First DB 스키마 변경, 의존성 추가
Never .env·API 키 커밋, 승인 없는 복구 실행

PRD (Product Requirements Document, 제품 요구사항 문서) — 무엇을, 누구를 위해, 왜 만드는지와 '완료'의 기준을 코드보다 먼저 정리한 문서입니다. 우리 과정의 **기획서(요구사항 명세 포함)**에 해당합니다.

STEP 1 · 승인 게이트

스펙은 ‘쓰고 바로 코딩’이 아니라 ‘승인 후 다음 단계’

PHASE 0
범위 점검
기능이 여러 개 묶였으면 모듈 지도부터 승인

›

PHASE 1
명세
가정 목록과 스펙 작성 후 **멈추고** 승인 요청

›

PHASE 2
계획
tasks/plan.md

›

PHASE 3
작업
tasks/todo.md

›

PHASE 4
구현
한 번에 한 작업씩

팀 운영 팁

승인 = 팀장 + 담당자가 PR로 docs/spec.md를 머지. 이것이 기획서 v1.0의 근거입니다.

살아있는 문서

범위가 바뀌면 스펙부터 고쳐 커밋하고, PR에서 해당 스펙을 참조합니다.

위험 신호

스펙을 쓴 같은 턴에 계획이나 코드까지 시작했다면 멈추세요.

STEP 1 · 작업 분해

/plan — WBS가 저절로 나오는 작업 목록

```
## T03 진단 Agent 노드 구현
- 담당: AI / 선행: T02
- 수용 기준: 장애 ID → 원인·근거 로그·신뢰도 반환
- 검증: pytest tests/test_diagnosis.py 통과
- 파일: app/agents/diagnosis.py

## T04 복구 승인 가드 (FR-03)
- 수용 기준: approval_id 없으면 403
  APPROVAL_REQUIRED
- 검증: test_recover_requires_approval
```

WBS 표를 그대로 채움

ID · 담당 · 선행 · 산출물 열이 이미 들어 있습니다.

일일 기록이 쉬워짐

대시보드에는 “오늘 끝낸 T번호 + 검증 결과”만 적으면 됩니다.

한 작업 = 하루 이내

넘으면 다시 쪼갭니다.

STEP 2~3 · 구현과 검증

/build + /test — 얇은 수직 조각, 테스트 먼저

1 · RED

실패하는 테스트를 쓰고 실제로 실패하는 것을 확인

2 · GREEN

통과할 만큼만 최소한의 코드를 작성

3 · REFACTOR

정리한 뒤 다시 통과 확인 → 작업 단위로 커밋

```
def test_recover_requires_approval(client, token):
    res = client.post(
        "/api/incidents/INC-104/recover",
        json={"strategy": "restart"},
        headers={"Authorization": f"Bearer {token}"},
    )
    assert res.status_code == 403
    assert res.json()["error"]["code"] \
        == "APPROVAL_REQUIRED"
```

한 조각 = 라우터 + 서비스 + 테스트 + 커밋. 테스트 비율은 단위 80 / 통합 15 / E2E 5를 기준으로 삼습니다.

STEP 3~4 · 에이전트와 Auto-Healing

LangGraph 에이전트 단계에서 꺼내 쓰는 스킬

debugging-and-error-recovery

재현 → 위치 특정 → 축소 → 수정 → 재발 방지

→ Auto-Healing 결과서의 장애 주입·감지·복구·검증 로그 구조로 그대로 사용

security-and-hardening

OWASP Top 10 · 시크릿 관리 · 3단계 경계

→ AI 보안 가드레일: 프롬프트 인젝션, 도구 권한, 승인 게이트 점검

source-driven-development

공식 문서에 근거하고 출처를 남김

→ API가 자주 바뀌는 LangGraph·LangChain에서 옛 문법 환각 방지

doubt-driven-development

중요한 결정을 새 컨텍스트에서 반박 검토

→ 상태 전이 그래프와 자기 성찰 루프의 종료 조건 점검

context-engineering 규칙 파일(CLAUDE.md·AGENTS.md)·컨텍스트 묶기·MCP 연동으로 “필요한 정보를 필요한 때” 주는 법 — 에이전트 프롬프트 설계에도 그대로 적용됩니다.

모든 STEP 공통 · 머지 전 품질 게이트

/review — 다섯 축으로 보고, 심각도를 붙인다

리뷰 축	확인할 것
정확성	요구사항 충족, 예외·경계 케이스, 테스트 충분성
가독성·단순성	명확한 이름, 불필요한 복잡도 없음
아키텍처	기존 패턴 준수, 결합도, 파일 크기
보안	시크릿·입력 검증·인증, 외부 데이터 불신
성능	N+1 없음, 목록 API 페이지네이션

심각도 라벨

Critical 머지 차단 · (라벨 없음) 필수 수정 · **Optional** 제안 · **Nit** 사소함 · **FYI** 참고

변경 크기

약 100줄이 이상적, 300줄은 한 가지 변경일 때만, 1000줄은 쪼개기

페르소나 활용

code-reviewer · test-engineer · security-auditor를 각각 따로 돌려 관점을 나눔

STEP 5 · 최적화·배포

/ship — 배포는 체크리스트와 되돌리기 계획까지

git-workflow-and-versioning

트렁크 기반 개발, 원자적 커밋, 커밋을 세이브 포인트로

ci-cd-and-automation

Shift Left와 품질 게이트 파이프라인 → GitHub Actions에서 ruff·pytest 통과해야 머지

observability-and-instrumentation

구조화 로그, RED 지표, OpenTelemetry, 증상 기반 알림 → MLOps 모니터링

shipping-and-launch

출시 전 체크리스트, 단계적 롤아웃, 롤백 절차 → 배포 아키텍처 설계도

documentation-and-adrs

ADR로 '왜'를 기록 → 발표 Q&A의 설계 근거

/ship이 하는 일

전문가 페르소나에게 나눠 점검을 맡기고 **go / no-go** 판단을 돌려줌

핑계 차단표 (Anti-rationalization)

에이전트도, 우리도 쓰는 핑계와 그 반박

핑계	스킬에 적힌 반박
“너무 간단해서 스펙은 필요 없어요”	수용 기준은 여전히 필요합니다. 다섯 줄이면 충분하지만 0줄은 안 됩니다.
“테스트는 나중에 쓸게요”	‘나중’은 없습니다. 실패하는 테스트부터 씁니다.
“테스트 통과했으니 배포하죠”	통과는 증거이지 증명이 아닙니다. 실제 동작, 사용자 화면, 사람이 diff를 읽었는지 확인합니다.
“요구사항은 어차피 바뀌어요”	살아있는 스펙이 변경을 감당합니다. 낡은 스펙도 없는 것보다 낫습니다.

실습 · 10분

우리 팀이 실제로 써 온 핑계 3개와 반박을 CONSTRAINTS.md에 적고 커밋하세요. Osmani도 팀 고유의 핑계를 직접 적어 두라고 권합니다.

실제 사용 사례 ① · ScrapingBee 기술 블로그 (2026-08)

같은 패턴으로 '우리 팀 전용 스킬'을 만들었다

만든 것

가장 저렴한 스크래핑 방법부터 시작하고, 근거가 있을 때만 단계를 올리는 스킬 — SKILL.md + scripts/fetch.py, extract.py

결과

정적 페이지: 표준 라이브러리 스크래퍼로 해결, 크레딧 0
JS 페이지: 정적 확인이 실패한 근거로 렌더링 단계로 승급
같은 스킬이 상황에 따라 **다른 결정**을 내림

그들이 얻은 교훈

- SDK 버전부터 고정한다
- 마켓플레이스 SSH 오류 → HTTPS
- 200 응답이어도 차단 페이지일 수 있다 → 필드 검증
- 스킬은 가이드일 뿐 보장이 아니다 → 테스트·린트와 함께
- 컬렉션 전체보다 1~5개만 설치
- 남이 만든 스킬은 설치 전 스크립트·훅까지 검토

우리 적용: 팀 도메인의 외부 API 연동 규칙(공공 API 페일오버, 재시도 제한 등)을 팀 전용 SKILL.md로 만들어 두면 누가 에이전트를 돌려도 같은 방식으로 구현됩니다.

실제 사용 사례 ② · 개발자 커뮤니티 반응 (Hacker News · GeekNews, 2026)

현업 개발자들의 평가 — 좋은 점과 한계

좋았다는 경험

- 수년 걸릴 레거시 PHP 리팩터링을 한 달 안에 해냄
- 스펙 기반 워크플로로 몇 달째 운영 중인 서비스에 기능을 꾸준히 추가
- 코드베이스를 이미 이해하고 있으면 생성 코드 리뷰가 빠름

아쉬웠다는 경험

- 지시 파일의 필수 규칙을 LLM이 빠뜨림 → 사람 리뷰가 유일한 확실한 검사
- 스킬·MCP를 많이 올리면 토큰과 속도 손해
- 효과를 A/B로 검증한 사례가 드묾

“AI가 만들어낸 코드를 검토하고 이해하는 능력이 핵심”

— GeekNews 댓글, 사내에서 바이브 코딩만으로 개발해 본 개발자. 공통 결론: **작고 집중된, 팀에 맞춘 스킬**이 가치 있다.

주의사항

우리 과정에 맞게 쓰는 5가지 원칙

1 **다 깔지 말고 핵심부터**

/spec · /plan · /build · /test · /review로 시작. 공식 가이드도 24개를 한꺼번에 로드하지 말라고 권합니다.

2 **웹 프론트엔드 편향 걸러 내기**

Core Web Vitals·브라우저 테스트 스킬은 UI 담당만. 백엔드·에이전트는 Python(pytest·ruff) 기준으로 읽습니다.

3 **스킬은 권고, 강제는 CI**

모델은 규칙을 빠뜨릴 수 있습니다. 테스트·린트를 GitHub Actions 품질 게이트로 막아 두세요.

4 **커맨드가 안 보이면 이름으로 부르기**

설치 후 /help로 확인하고, 안 되면 “follow spec-driven-development”처럼 스킬 이름으로 지시합니다.

5 **내가 설명할 수 있는 코드만 머지**

최종 발표는 예외 대응, 한계, **개인 기여 설명**까지 평가합니다.

도입 로드맵 (권장 예시 · 실제 마감은 대시보드 기준)

8주 동안 이렇게 씁니다 — 지금 2주차부터

1~2주
/spec /plan
기획서 · WBS

›

2~3주
/build /test
API 설계서 · DB 스키마 설계서

›

3~5주
debugging security
그래프 설계서 · 보안 가드레일 · Auto-Healing

›

5~6주
/review
통합 테스트 결과서 · 결함 재검증

›

7~8주
/ship
배포 아키텍처 · 모니터링 · 시연

/review는 2주차부터 모든 PR에서 매번. 단계 사이에 넘어갈 때마다 spec.md와 tasks/todo.md를 갱신하고, 그 변경 이력이 곧 중간·최종 발표의 '계획 대비 실적' 근거가 됩니다.

이번 주 할 일

팀 체크리스트 6개

1 **전원 같은 방식으로 설치**
/help로 커맨드가 보이는지 확인

2 **docs/spec.md 승인**
/spec 결과를 팀장이 PR로 리뷰·머지

3 **CONSTRAINTS.md 작성**
/constraints로 테스트·린트·금지 사항 확정

4 **tasks/todo.md ↔ WBS**
T번호를 대시보드 WBS·일일 기록과 맞춤

5 **PR 템플릿**
T번호 · 테스트 결과 · /review 심각도 라벨

6 **첫 ADR 1건**
“LangGraph 상태를 왜 이렇게 나눴나”

마무리

AI가 짠 코드를 설명할 수 있는 팀이 이깁니다

저장소 · github.com/addyosmani/agent-skills

문서 · skills.addy.ie

저자 해설 · addyosmani.com/blog/agent-skills