클로드(Claude) 스킬 만들기 완전 가이드

전체 번역본 (한글)

원문: The Complete Guide to Building Skills for Claude

목차

소개(Introduction)

스킬(skill)은 Claude가 특정 작업이나 워크플로를 처리하는 방법을 '간단한 폴더' 형태로 묶어 가르치는 지침 집합입니다. 매번 대화마다 선호도/프로세스/도메인 지식을 다시 설명하지 않아도, 한 번 스킬로 만들어두면 반복적으로 혜택을 볼 수 있습니다.

스킬은 다음과 같은 반복 가능한 워크플로에서 특히 강력합니다: 사양에서 프론트엔드 디자인 생성, 일관된 방법론으로 조사 수행, 팀 스타일가이드를 따르는 문서 생성, 여러 단계의 프로세스를 순서대로 오케스트레이션하는 작업 등.

또한 스킬은 코드 실행, 문서 생성 같은 Claude의 내장 기능과 잘 결합됩니다. MCP 통합을 만드는 경우에는 '툴 접근' 위에 '신뢰 가능한 워크플로 지식'을 더하는 층으로서, 도구를 실제 가치로 연결해 주는 역할을 합니다.

이 가이드는 스킬을 효과적으로 만드는 데 필요한 계획, 구조, 테스트, 배포 전 과정을 다룹니다.

이 가이드로 배우는 것

- 스킬 구조에 대한 기술 요구사항과 모범 사례
- 단독 스킬 및 MCP 강화 워크플로 패턴
- 다양한 사용 사례에서 검증된 패턴
- 테스트, 반복 개선, 배포 방법

대상 독자

- Claude가 특정 워크플로를 '항상 같은 방식'으로 따르도록 만들고 싶은 개발자
- 반복 업무를 표준화하려는 파워 유저
- 조직 내에서 Claude의 동작을 표준화하려는 팀

가이드 읽는 2가지 경로

- 단독 스킬을 만들려면: 1장 기본, 2장 계획과 설계, 그리고 카테고리 1-2를 중심으로 읽습니다.
- MCP 통합을 강화하려면: 'Skills + MCP' 관련 설명과 카테고리 3을 참고합니다.
- 두 경로 모두 기술 요구사항은 동일하며, 본인 목적에 맞는 부분을 선택해 적용하면 됩니다.

이 가이드를 마치면 한 번 앉아서도 첫 스킬을 만들고 테스트할 수 있도록(원문은 15-30분 정도를 언급) 구성되어 있습니다.

1장 기본(Fundamentals)

스킬이란 무엇인가?

스킬은 다음 요소를 담은 폴더입니다.

- SKILL.md (필수): YAML 프론트매터가 포함된 Markdown 지침 파일
- scripts/ (선택): 실행 가능한 코드(Python, Bash 등)
- references/ (선택): 필요 시 로드할 문서/가이드
- assets/ (선택): 템플릿, 폰트, 아이콘 등 산출물에 쓰는 리소스

핵심 설계 원칙

점진적 공개(Progressive Disclosure)

- 1단계(YAML 프론트매터): Claude의 시스템 프롬프트에 항상 로드됨. 어떤 상황에서 이 스킬을 써야 하는지 판단할 최소 정보만 제공.
- 2단계(SKILL.md 본문): Claude가 현재 작업과 관련 있다고 판단하면 로드됨. 전체 지침과 가이드를 포함.
- 3단계(연결 파일): 스킬 디렉터리 내 추가 파일들을 Claude가 필요할 때만 찾아보고 활용.

이 구조는 토큰 사용을 줄이면서도 전문 지식을 유지하기 위한 것입니다.

조합성(Composability)

Claude는 여러 스킬을 동시에 로드할 수 있으므로, 스킬은 '나만 존재한다'고 가정하지 않고 다른 스킬과 함께 동작하도록 설계해야 합니다.

이식성(Portability)

스킬은 Claude.ai, Claude Code, API 전반에서 동일하게 동작하도록 설계되었습니다(단, 필요한 의존성이 환경에 존재해야 함).

MCP 빌더를 위한: Skills + Connectors

이미 MCP 서버(커넥터)가 준비되어 있다면, 스킬은 그 위에 얹는 '지식 레이어'입니다.

주방 비유

- MCP는 '전문 주방'(도구, 재료, 장비 접근)을 제공.
- 스킬은 '레시피'(가치 있는 결과를 만드는 단계별 지침)를 제공.

둘을 결합하면 사용자는 매번 절차를 스스로 설계하지 않아도 복잡한 일을 수행할 수 있습니다.

함께 작동하는 방식(개념 비교)

- MCP(연결성): Notion/Asana/Linear 등 서비스에 대한 실시간 데이터 접근과 툴 호출을 제공(무엇을 할 수 있는지).
- 스킬(지식): 서비스를 '어떻게 잘 활용하는지'에 대한 워크플로와 모범 사례를 제공(어떻게 해야 하는지).

왜 중요한가(원문 요지)

- 스킬이 없으면: 사용자는 연결만 해두고 다음 행동을 몰라 지원 문의가 늘고, 대화마다 처음부터 시작하며, 결과가 들쭉날쭉해짐.
- 스킬이 있으면: 필요할 때 워크플로가 자동으로 활성화되고, 툴 사용이 일관되며, 학습 곡선이 낮아지고, 결과 신뢰도가 올라감.

2장 계획과 설계(Planning and design)

사용 사례(use case)에서 시작하기

코드를 쓰기 전에, 스킬이 가능하게 해야 하는 2-3개의 구체적 사용 사례를 정의합니다.

좋은 사용 사례 정의 예: 스프린트 계획

- 트리거: 사용자가 “이번 스프린트 계획 도와줘”, “스프린트 작업 만들어줘”라고 말함
- 단계: 1) Linear에서 현재 상태 가져오기(MCP) 2) 팀 속도/수용량 분석 3) 우선순위 제안 4) 라벨/견적이 포함된 작업 생성
- 결과: 스프린트 계획 완료 및 작업 생성

스스로에게 던질 질문

- 사용자가 최종적으로 이루고 싶은 것은 무엇인가?
- 이를 위해 필요한 다단계 워크플로는 무엇인가?
- 필요한 툴은 무엇인가(내장 도구 vs MCP)?
- 어떤 도메인 지식/베스트 프랙티스를 스킬에 내장해야 하는가?

흔한 스킬 사용 사례 카테고리(원문: Anthropic 관찰 3가지)

카테고리 1: 문서/자산 생성

일관된 고품질 결과물(문서, 프레젠테이션, 앱, 디자인, 코드 등)을 생성하는 데 사용.

- 기법: 스타일 가이드/브랜드 표준 내장, 템플릿 구조, 최종 품질 체크리스트
- 외부 툴이 없어도 Claude 내장 기능만으로 구현 가능

카테고리 2: 워크플로 자동화

여러 단계의 프로세스를 일관된 방법으로 수행하도록 안내(여러 MCP 서버를 조합하기도 함).

- 기법: 단계별 진행 + 검증 게이트, 공통 템플릿, 리뷰/개선 제안, 반복 개선 루프

카테고리 3: MCP 강화

MCP 서버가 제공하는 ‘툴 접근’을 실제 업무 워크플로로 연결하는 안내 레이어.

- 기법: 여러 MCP 호출을 순차/조합 수행, 도메인 전문성 내장, 사용자가 매번 적어야 할 맥락을 기본 제공, 오류 처리

성공 기준 정의하기

스킬이 잘 동작하는지 어떻게 판단할지 미리 정합니다(정밀 임계값이라기보다 목표 지표).

정량 지표 예

- 관련 요청의 90%에서 자동 트리거: 10-20개의 테스트 쿼리로 자동 로드 비율을 기록.
- 워크플로 완료에 필요한 툴 호출 수: 스킬 유무 비교로 툴 호출/토큰 소비를 측정.
- 워크플로 당 실패 API 호출 0회: 테스트 중 MCP 서버 로그로 재시도/에러 코드를 추적.

정성 지표 예

- 사용자가 다음 단계 지시를 거의 하지 않아도 된다.
- 워크플로가 사용자 수정 없이 완료된다(여러 번 반복 실행해 일관성 확인).
- 새 사용자도 최소 안내로 첫 시도에 목표를 달성한다.

기술 요구사항

파일 구조 예

your-skill-name/ ├── SKILL.md (필수) ├── scripts/ (선택) ├── references/ (선택) └── assets/ (선택)

중요 규칙

- SKILL.md는 정확히 'SKILL.md'여야 함(대소문자 포함).
- 폴더명은 kebab-case, 공백/언더스코어/대문자 금지.
- 스킬 폴더 내부에 README.md를 넣지 말 것(문서는 SKILL.md 또는 references/).

YAML 프론트매터: 가장 중요

Claude가 스킬을 로드할지 결정하는 핵심이므로, 특히 description을 정교하게 작성합니다.

최소 형식

--- name: your-skill-name description: 무엇을 하는지. 사용자가 어떤 말/요청을 할 때 쓰는지. ---

필드 요구사항

- name(필수): kebab-case, 공백/대문자 금지, 폴더명과 일치 권장.
- description(필수): '무엇을'+'언제'(트리거 조건) 모두 포함. 1024자 이하, < > 금지, 사용자가 말할 구체적 문구/파일 타입 언급.
- license(선택): 오픈소스라면 MIT, Apache-2.0 등.
- compatibility(선택): 환경 요구사항(제품, 시스템 패키지, 네트워크 필요 등).
- metadata(선택): author/version/mcp-server 등 커스텀 키-값.

보안 제한(프론트매터)

- XML 꺾쇠 괄호(< >) 금지.
- 이름에 'claude' 또는 'anthropic' 사용 금지(예약).
- 이유: 프론트매터는 시스템 프롬프트에 들어가므로, 악의적 주입을 막기 위함.

좋은 description 작성법

구조: [무엇을 한다] + [언제 쓴다(트리거)] + [핵심 기능]

좋은 예(요지)

- Figma 파일을 분석해 개발자 핸드오프 문서를 생성. 사용자가 .fig 업로드, "design specs", "component documentation" 등을 요청할 때 사용.
- Linear에서 스프린트 계획/작업 생성/상태 추적. 사용자가 "sprint", "create tickets" 등을 말할 때 사용.

- PayFlow 고객 온보딩 워크플로. “onboard new customer”, “create account” 등의 요청에 사용.

  나쁜 예(요지)
- 너무 모호함: “프로젝트를 도와줌”
- 트리거가 없음: “고급 문서 시스템 생성”
- 너무 기술적이며 사용자 언어가 없음: “엔터티 모델 구현”

SKILL.md 본문 지침 작성

프론트매터 이후에는 실제 지침을 Markdown으로 작성합니다. 권장 구조는 다음과 같습니다.

- 스킬 이름/목적
- 단계별 절차(각 단계의 기대 결과 포함)
- 예시(사용자 발화 → 수행 행동 → 결과)
- 트러블슈팅(오류 메시지/원인/해결)

지침 작성 모범 사례

- 구체적이고 실행 가능하게: 예) 특정 명령어와 입력 형식, 실패 시 흔한 원인과 수정법을 명시.
- 에러 처리 포함: 예) MCP 연결 실패 시 Settings에서 확인/재연결 등의 절차.
- 번들 리소스(reference)를 명확히 링크: 예) references/api-patterns.md에서 레이트리밋/페이지네이션/에러코드 확인.
- 점진적 공개 유지: SKILL.md는 핵심만, 상세 문서는 references/로 옮겨 링크.

3장 테스트와 반복 개선(Testing and iteration)

테스트 수준

- Claude.ai에서 수동 테스트: 빠르고 설정이 거의 없음.
- Claude Code에서 스크립트 기반 테스트: 변경에 대해 반복 가능한 검증.
- Skills API로 프로그램적 테스트: 정의된 테스트 세트로 체계적인 평가.

팁: 하나의 어려운 작업에서 먼저 반복 개선

원문은 가장 효과적인 스킬 제작자가 '특히 어려운 단일 작업'을 Claude가 성공할 때까지 반복 개선하고, 그 성공 패턴을 스킬로 추출한 뒤 범위를 확장한다고 말합니다.

권장 테스트 접근(3가지)

1) 트리거 테스트

- 명확한 요청에서 트리거되는지
- 말을 바꿔 표현한 요청에서도 트리거되는지
- 무관한 주제에서는 트리거되지 않는지

예시 테스트 세트(요지)

- 트리거되어야 함: "ProjectHub 워크스페이스 새로 세팅해줘", "프로젝트 만들어줘" 등
- 트리거되면 안 됨: "샌프란시스코 날씨?", "파이썬 코드 작성", "스프레드시트 만들어줘"(스킬 범위가 아니라면)

2) 기능 테스트

- 유효한 결과물이 생성되는지
- API/MCP 호출이 성공하는지
- 에러 핸들링이 동작하는지
- 엣지 케이스를 커버하는지

예: '작업 5개가 포함된 프로젝트 생성' 테스트

- Given: 프로젝트명과 작업 5개 설명
- When: 워크플로 실행
- Then: 프로젝트 생성, 작업 5개 생성 및 연결, API 에러 없음

3) 성능 비교

스킬이 실제로 개선을 만드는지 '기준선(baseline)'과 비교합니다.

비교 예(원문 요지)

- 스킬 없음: 왕복 대화 많음, API 실패/재시도 발생, 토큰 소비 증가
- 스킬 있음: 자동 워크플로, 최소한의 확인 질문, 실패 0회, 토큰 감소

skill-creator 스킬 사용

원문은 Claude.ai의 디렉터리 또는 Claude Code에서 사용할 수 있는 'skill-creator'가 스킬 제작/리뷰/개선에 도움이 된다고 설명합니다.

- 자연어 설명으로 스킬 초안 생성
- 올바른 프론트매터/구조 제안
- 트리거 문구/구조 문제/과다·과소 트리거 위험 플래그
- 엣지 케이스에서 실패한 대화를 가져와 개선 제안

피드백 기반 반복 개선

- 언더 트리거: 스킬이 켜져야 하는데 안 켜짐 → description에 키워드/트리거 문구/기술 용어 추가.
- 오버 트리거: 무관한 요청에도 켜짐 → 부정 트리거 추가, 범위를 더 구체화.
- 실행 문제: 결과 편차/실패/사용자 수정 필요 → 지침 명확화, 에러 처리 보강.

4장 배포와 공유(Distribution and sharing)

스킬은 MCP 통합을 더 '완성도 높게' 만들어 주며, MCP만 있는 대안 대비 빠른 가치 전달 경로를 제공합니다.

현재 배포 모델(원문: 2026년 1월 기준)

- 개인: 스킬 폴더 다운로드 → (필요 시) zip → Claude.ai에서 Settings > Capabilities > Skills로 업로드 또는 Claude Code 디렉터리에 배치.
- 조직: 관리자가 워크스페이스 전역 배포 가능(원문은 2025-12-18에 기능 출시 언급), 자동 업데이트 및 중앙 관리.

오픈 표준

스킬은 MCP처럼 오픈 표준으로 공개되어 있으며, 동일한 스킬이 여러 플랫폼에서 이식 가능하도록 하는 방향을 지향합니다(단, 특정 플랫폼 기능을 활용하는 스킬은 compatibility에 요구사항을 적어야 함).

API로 스킬 사용

- /v1/skills 엔드포인트로 스킬 목록/관리
- Messages API 요청에 container.skills 파라미터로 스킬 추가
- Claude Console에서 버전 관리
- Claude Agent SDK와 함께 사용 가능

API vs Claude.ai 사용 기준(요지)

- 엔드유저 대화: Claude.ai / Claude Code
- 개발 중 수동 테스트/개선: Claude.ai / Claude Code
- 프로덕션 규모 앱/에이전트/자동 파이프라인: API

참고(원문 요지): Skills API는 안전한 실행 환경을 제공하는 Code Execution Tool 베타가 필요하다고 언급합니다.

오늘의 권장 접근(요지)

- GitHub에 공개 저장소로 호스팅(사람을 위한 repo-level README 포함, 단 스킬 폴더 내부에는 README.md 금지).
- MCP 문서에 스킬 링크와 '둘을 함께 쓰면 좋은 이유'를 포함.
- 설치 가이드 제공: 다운로드/업로드/활성화/테스트 절차.

포지셔닝(설명 방식)

- 기능 나열보다 '성과/결과'를 중심으로 설명.
- MCP(툴 접근) + 스킬(워크플로 지식)의 결합 스토리를 강조.

5장 패턴과 트러블슈팅(Patterns and troubleshooting)

원문은 초기 사용자와 내부 팀이 만든 스킬에서 나타난 공통 패턴을 정리합니다(처방적 템플릿이 아니라 참고용).

문제 중심 vs 툴 중심 접근

- 문제 중심(Problem-first): “프로젝트 워크스페이스를 세팅하고 싶다” → 스킬이 적절한 MCP 호출을 순서대로 처리.
- 툴 중심(Tool-first): “Notion MCP를 연결했다” → 스킬이 최적 워크플로/모범 사례를 가르쳐 활용도를 높임.

패턴 1: 순차 워크플로 오케스트레이션

사용자 온보딩 같은 ‘정해진 순서’가 중요한 프로세스에 적합.

- 단계 순서 명시, 단계 간 의존성(이전 단계 결과를 다음 입력으로), 각 단계 검증, 실패 시 롤백 지침

패턴 2: 멀티 MCP 조합

한 워크플로가 여러 서비스에 걸칠 때(예: 디자인→자산 저장→태스크 생성→알림).

- 단계/페이즈 분리, 서비스 간 데이터 전달, 다음 단계 전에 검증, 중앙 에러 처리

패턴 3: 반복 개선(Iterative refinement)

- 초안 생성 → 검증 스크립트 실행 → 이슈 목록화 → 수정/재생성 → 재검증 반복 → 최종 마감

패턴 4: 컨텍스트 기반 도구 선택

- 같은 목표라도 파일 타입/크기/협업 필요 등에 따라 다른 저장소/툴 선택
- 선택 기준을 의사결정 트리로 명시하고, 선택 이유를 사용자에게 투명하게 설명

패턴 5: 도메인 특화 지능

- 툴 접근만으로 부족한 도메인 규칙(예: 컴플라이언스)을 워크플로에 포함
- 처리 전 점검(제재 리스트, 관할 허용, 위험 수준) → 처리 → 감사 추적(audit trail)

트러블슈팅

업로드가 안 될 때

- 오류: “Could not find SKILL.md …” → 원인: 파일명이 정확히 SKILL.md가 아님 → 해결: 대소문자 포함 정확히 변경.
- 오류: “Invalid frontmatter” → 원인: YAML 구분자(---) 누락, 따옴표 미닫힘 등 → 해결: 형식 수정.
- 오류: “Invalid skill name” → 원인: name에 공백/대문자 포함 → 해결: kebab-case로 변경.

스킬이 트리거되지 않을 때

- description이 너무 일반적이지 않은지 확인(예: “프로젝트 도움”은 약함).
- 사용자가 실제로 쓸 표현/키워드를 추가.
- Claude에게 “이 스킬은 언제 쓰나요?”라고 물어 description을 되돌려받아 점검.

스킬이 너무 자주 트리거될 때

- 부정 트리거 추가("~에는 사용하지 말 것").
- 범위를 더 구체화('문서 처리'→'계약서 PDF 검토' 등).
- 적용 범위를 명확히 제한(결제 워크플로 전용 등).

MCP 연결 문제(스킬은 켜지는데 호출이 실패)

- Settings에서 MCP 서버 연결 상태 확인(Connected).
- API 키/권한 범위/토큰 만료 확인.
- 스킬 없이 MCP 툴을 직접 호출해보며 원인을 분리.
- 툴 이름이 정확한지 확인(대소문자/문서와 일치).

지침을 잘 따르지 않을 때

- 지침이 너무 장황함 → 핵심을 상단에 두고, 불릿/번호로 간결화, 상세는 references로 분리.
- 중요 지침이 묻힘 → 'CRITICAL' 헤더로 강조, 반복해서 상기.
- 모호한 표현 → 검증 기준을 체크리스트로 구체화.
- 고급 기법: 중요한 검증은 언어 지시보다 스크립트로 자동화(결정론적).
- 성능 노트: '천천히 꼼꼼히, 검증 생략 금지' 같은 문구는 SKILL.md보다 사용자 프롬프트에 넣는 것이 더 효과적이라는 언급.

컨텍스트가 너무 커질 때(느려지거나 품질 저하)

- 원인: 스킬 내용 과대, 동시에 너무 많은 스킬 활성화, 모든 내용을 한 번에 로드
- 해결: SKILL.md를 5,000단어 이하로 유지, 상세 문서를 references로 분리, 활성 스킬 수를 줄이고 팩 형태로 구성

6장 자료와 참고(Resources and references)

원문은 첫 스킬을 만드는 경우 ‘Best Practices Guide’를 먼저 읽고, 필요 시 API 문서를 참고하라고 안내합니다.

공식 문서(원문 목록 요지)

- Best Practices Guide
- Skills Documentation
- API Reference
- MCP Documentation

블로그/글(원문 목록 요지)

- Introducing Agent Skills
- Engineering Blog: Equipping Agents for the Real World
- Skills Explained
- How to Create Skills for Claude
- Building Skills for Claude Code
- Improving Frontend Design through Skills

예시 스킬/저장소

- 공개 저장소: anthropics/skills (커스터마이즈용 예시 포함)

도구 및 유틸리티

- skill-creator: 스킬 생성/리뷰/추천 제공
- 검증: skill-creator로 스킬을 평가하고 개선점 확인 가능

지원 받기

- 기술 질문: Claude Developers Discord의 커뮤니티 포럼(원문 언급)
- 버그 리포트: GitHub Issues(스킬명, 오류 메시지, 재현 절차 포함)

부록 A: 빠른 체크리스트(Reference A)

시작 전

- 구체적 use case 2-3개 정의
- 필요 툴(내장/MCP) 확정
- 가이드 및 예시 스킬 검토
- 폴더 구조 계획

개발 중

- 폴더명 kebab-case
- SKILL.md 존재(정확한 철자)
- 프론트매터 --- 구분자 확인
- name: kebab-case, 공백/대문자 없음
- description: WHAT + WHEN 포함
- 어디에도 < > 없음
- 지침이 명확하고 실행 가능
- 에러 처리 포함
- 예시 제공
- references 링크 명확

업로드 전

- 트리거 테스트(명확/패러프레이즈/무관 요청)
- 기능 테스트 통과
- 툴 통합(해당 시) 동작
- zip 압축

업로드 후

- 실제 대화에서 테스트
- 과소/과다 트리거 모니터링
- 사용자 피드백 수집
- description/지침 반복 개선
- metadata 버전 업데이트

부록 B: YAML 프론트매터(Reference B)

필수 필드

--- name: kebab-case-skill-name description: 무엇을 하고, 언제 사용하는지. 구체적인 트리거 문구 포함. ---

선택 필드 예(요지)

- license: MIT
- allowed-tools: "Bash(python:*) Bash(npm:*) WebFetch"
- metadata: author/version/mcp-server/category/tags/documentation/support 등

보안 메모(요지)

- 허용: 표준 YAML 타입 및 커스텀 metadata, 1024자까지의 description
- 금지: < >, YAML 내 코드 실행(안전 파서), 이름에 claude/anthropic(예약)

부록 C: 전체 스킬 예시(Reference C)

원문은 다음과 같은 ‘완성형 예시 스킬’들을 참고하라고 안내합니다.

- 문서 스킬: PDF, DOCX, PPTX, XLSX 생성
- 다양한 워크플로 패턴 예시 스킬
- 파트너 스킬 디렉터리(Asana, Atlassian, Canva, Figma, Sentry, Zapier 등 예시)

작성일: 2026-01-30