좋은 스킬 만들기 매뉴얼

Claude가 당신처럼 일하게 만드는 실전 가이드

개발곰곰 | youtube.com/@devgomgom

기업용 업무 시스템 기획·개발 | 그룹웨어·ERP·업무 자동화·사내 AI 환경 구축

> 업무 전문성을 스킬로 풀어내면 AI가 당신처럼 일한다.

섹션 01

스킬이 필요한 진짜 이유

왜 같은 Claude인데 어떤 사람은 10배 빠르게 일하는가?

- 매번 같은 지시를 반복하는 비효율
- 범용 프롬프트 템플릿의 한계 -- 복붙 프롬프트는 왜 안 되는가
- 컨텍스트 유실 -- 대화가 길어지면 품질이 떨어지는 이유

스킬의 본질 정의

- 스킬 = 당신의 업무 전문성을 코드화한 것
- 프롬프트와 스킬의 차이: 일회성 지시 vs. 재사용 가능한 업무 패턴
- 스킬은 Claude의 "근무 매뉴얼"이다

> 범용 프롬프트 템플릿은 쓸모없다 -- 작동하는 건 당신의 업무 전문성을 코드화하는 것이다.

지표	수치	의미
스킬 생태계	1,400+	커뮤니티에서 공유된 스킬 수
Frontend Design	277K+	가장 많이 설치된 스킬
Caveman 스킬	75%	토큰 절감률

섹션 02

좋은 스킬의 5가지 조건

이 5가지를 모두 충족하면, 그 스킬은 반복해서 가치를 만든다.

1. 반복성 -- 5회 이상 반복하는 작업인가?

한 번만 할 작업에 스킬을 만들면 시간 낭비다.

✓ 주 2회 이상 반복하는 작업

✓ 팀 내 여러 사람이 같은 작업을 수행

☒ 일회성 분석, 탐색적 작업

2. 명확성 -- 설명(description)이 공격적으로 구체적인가?

Claude는 기본적으로 스킬을 덜 호출한다. 트리거 조건을 넓게 써야 한다.

✓ when_to_use에 구체적 상황 나열

✓ 부정 트리거(when NOT to use)로 과잉 호출 방지

☒ "코드 관련 작업에 사용" 같은 모호한 설명

3. 간결성 -- SKILL.md가 500줄 / 5,000토큰 이내인가?

5,000토큰 초과 시 자동 압축 -- 의도가 왜곡될 수 있다.

✓ 핵심 지시만 SKILL.md에 배치

✓ 상세 참조는 references/ 폴더로 분리

☒ 모든 케이스를 SKILL.md 하나에 담기

4. 시범성 -- 규칙 대신 예제가 있는가?

한 개의 좋은 예제가 한 페이지의 규칙을 이긴다.

✓ 입력 -> 출력 쌍으로 된 예제 1~2개

✓ 예제에 "왜 이렇게 하는지" 주석 포함

☒ 추상적 규칙만 나열

5. 적응성 -- 지시 구체성이 작업 취약성에 비례하는가?

유연한 작업엔 WHY를, 깨지기 쉬운 작업엔 정확한 순서를 적어라.

✓ 창의적 작업: 원칙과 목표 중심 기술

✓ 정밀 작업: 단계별 정확한 시퀀스 기술

☒ 모든 작업에 같은 수준의 지시

종합 판정

✓ 5/5 --> 프로덕션 투입 준비 완료

● 3~4/5 --> 개선 후 배포 가능

☒ 2/5 이하 --> 재설계 필요

섹션 03

SKILL.md 해부학

SKILL.md의 모든 구성 요소를 한눈에 파악한다.

3-1. 파일 구조

```
.claude/skills/my-skill/
+-- SKILL.md <- 메인 지시 파일 (필수)
+-- scripts/ <- 보조 스크립트
| +-- validate.sh
+-- references/ <- 참조 문서 (도메인 지식)
| +-- style-guide.md
| +-- api-spec.md
+-- assets/ <- 정적 리소스
+-- template.json
```

3-2. Frontmatter 필드

필드	필수	제한	설명
name	필수	64자, 소문자+하이픈	스킬 고유 이름
description	필수	1536자 (합산)	스킬이 하는 일 설명
when_to_use	권장	description과 합산	트리거 조건

3-3. 로딩 우선순위

- Enterprise 스킬 (관리자 설정) --> 최우선
- Personal 스킬 (~/.claude/skills/) --> 개인 설정
- Project 스킬 (.claude/skills/) --> 프로젝트별
- Plugin 스킬 --> 최하위

3-4. 컨텍스트 관리

- 시작 시: description만 로드 (가벼움)
- 호출 시: 전체 SKILL.md 로드
- 5,000토큰 초과 시: 자동 압축 (auto-compaction) --> 의도 손실 위험

3-5. 특수 문법

문법	용도	예시
$ARGUMENTS	사용자 입력 전체	/my-skill 이 텍스트
$1, $2	위치 인자	$1 = 첫번째 인자
${CLAUDE_SKILL_DIR}	스킬 디렉토리 경로	스킬 내 파일 참조
!`command`	동적 컨텍스트 주입	!`cat package.json`

3-6. 확장 필드 (Claude 전용)

필드	용도	예시
user-invocable	수동 호출 전용 (/명령 형태)	true
disable-model-invocation	AI 자동 호출 차단	true
allowed-tools	사용 가능 도구 제한	Read Grep Bash
argument-hint	슬래시 명령 인자 힌트	[environment] [branch]

섹션 04

내 업무에서 스킬 뽑아내기

FCB 역도출 워크시트 -- 스킬 아이디어 발굴

> 스킬의 출발점은 기술이 아니라 당신이 매일 반복하는 '그 작업'이다.

STEP 1: 이득(Benefit) 정의

이 스킬이 완성되면, 나(또는 팀)에게 어떤 이득이 생기는가?

예) "PR 리뷰 시간이 30분 -> 5분으로 줄어든다"

STEP 2: 문제(Problem) 역도출

그 이득이 없는 지금, 구체적으로 어떤 문제가 있는가?

예) "매 PR마다 같은 체크리스트를 수동으로 확인"

STEP 3: 현재 작업 흐름 분해

지금 이 작업을 어떻게 하고 있는가? (단계별로 적고, '매번 같은 방식'인 단계에 표시)

1단계: ___ 2단계: ___ 3단계: ___

STEP 4: 스킬 범위 결정

표시한 단계들을 묶어서 하나의 스킬로 만들 수 있는가?

스킬 이름: ___ 입력: ___ 출력: ___

STEP 5: 5가지 조건 사전 검증

반복성, 명확성, 간결성, 시범성, 적응성을 체크한다.

3개 이상 체크 --> 스킬로 만들 가치가 있다!

핵심 원칙

- STEP 1~2: 기술 기능이 아니라 업무 이득에서 출발하라
- STEP 3: 자동화할 가치가 있는 건 '매번 같은' 부분이다
- STEP 4: 하나의 스킬 = 하나의 명확한 책임

섹션 05

스타터 템플릿으로 첫 스킬 만들기

즉시 복사해서 쓸 수 있는 템플릿 + 작성 가이드

```
---
name: my-skill-name
description: |
[이 스킬이 하는 일을 한 문장으로 적는다.]
구체적으로 -- "코드를 도와준다" (X)
"Python FastAPI 엔드포인트의 입력 검증 코드를 생성한다" (O)
when_to_use: |
다음 상황에서 이 스킬을 사용한다:
- [트리거 상황 1]
- [트리거 상황 2]
다음 상황에서는 사용하지 않는다:
- [제외 상황 1]
---

# [스킬 이름]

## 목적
[이 스킬이 존재하는 이유를 1~2문장으로]

## 핵심 원칙
1. [가장 중요한 판단 기준]
2. [품질을 결정하는 핵심 규칙]
3. [흔한 실수를 방지하는 가드레일]

## 작업 절차
1. [입력 확인]
2. [처리]
3. [출력]
4. [검증]

## 예제
### 입력
```

```
33 [사용자가 실제로 입력할 내용]
34 ### 기대 출력
35 [스킬이 만들어야 하는 결과물]
36 ### 왜 이렇게 하는가
37 [이 출력이 올바른 이유]
```

수동 전용 스킬 템플릿

AI가 자동으로 호출하지 않고, /명령으로만 실행되는 스킬 템플릿.

```
1 ---
2 name: deploy-preview
3 description: 프리뷰 배포가 필요할 때 사용한다.
4 user-invocable: true
5 disable-model-invocation: true
6 argument-hint: "[environment] [branch]"
7 allowed-tools: Read Grep Bash
8 ---
```

작성 팁

섹션	작성 팁	흔한 실수
name	동사-목적어 형식	너무 일반적 (helper, utils)
description	"~를 ~한다" 구체적으로	"다양한 작업을 도와줍니다"
when_to_use	부정 트리거 반드시 포함	누락 시 과잉 호출
핵심 원칙	3개 이하로 제한	10개 이상 --> Claude 혼란
예제	입력->출력 + "왜" 설명	예제 없이 규칙만 나열

섹션 06

실전 예제 -- 주석 달린 스킬 분석

실제 작동하는 스킬을 해부하며 "왜 이렇게 썼는지" 학습

예제: 커밋 메시지 작성 스킬

```
---
name: write-commit-message # (A) 동사-목적어
description: | # (B) 구체적 3문장
Git 스테이징된 변경사항을 분석하여
Conventional Commits 형식의 커밋 메시지를 작성한다.
변경 범위, 영향도, breaking change 여부를 자동 판단.
when_to_use: | # (C) 부정 트리거 포함
사용할 때:
- 사용자가 "커밋", "commit"을 언급할 때
- git add 후 다음 단계를 물을 때
사용하지 않을 때:
- 이미 커밋 메시지를 작성해서 보여줄 때
- git log나 히스토리 조회를 요청할 때
---

# 커밋 메시지 작성기 # (D) 목적 명시

## 목적
일관되고 의미 있는 커밋 메시지를 빠르게 작성.
"무엇이 바뀌었는가"보다 "왜 바꿨는가"에 초점.

## 핵심 원칙 # (E) 3개만
1. 제목 50자, 본문 72자 줄바꿈
2. type은 변경의 의도로 판단 (파일 종류 X)
3. breaking change 시 BREAKING CHANGE 푸터 필수

## 작업 절차 # (F) 정밀 작업 = 순서
1. git diff --staged로 변경사항 확인
2. 변경 파일 목록과 diff 분석
3. type과 scope 판단
4. Conventional Commits 형식으로 생성
```

```
32 5. breaking change 확인 후 최종 출력
33
34 ## 예제 # (G) 입력->출력->왜
35 ### 입력: 로그인에 rate limiting 추가
36 ### 출력: feat(auth): add rate limiting
37 ### 왜: type=feat(신기능), scope=auth(인증 모듈)
```

주석 해설

주석	포인트	원칙 연결
(A)	동사-목적어 형식 이름	조건 2: 명확성
(B)	3문장으로 구체적 설명	조건 2: 명확성
(C)	부정 트리거 2개 포함	조건 2: 명확성
(D)	바로 아래에 "왜 존재하는가"	조건 5: 적응성
(E)	원칙 3개만 -- 더 필요하면 references/	조건 3: 간결성
(F)	정밀 작업이므로 구체적 순서	조건 5: 적응성
(G)	입력->출력->"왜" 구조	조건 4: 시범성

실패 패턴 7가지

커뮤니티에서 반복적으로 관찰되는 스킬 실패 유형과 대응 가이드.

실패 패턴	원인	가이드 문장
under/over-trigger	description이 모호하거나 내부 구현만 설명	설명문은 사용자 의도를 적는다
commands->skills 혼선	레거시 경로와 신규 경로 혼재	skills 디렉터리만 사용
flat file/소문자 무시	디렉터리+대문자 SKILL.md 필수	name/SKILL.md 구조 필수
모델이 스킬 인식 못함	registry injection 실패	/skills와 실제 호출 둘 다 검증
validator 오탐	확장 필드를 에러로 표시	런타임 테스트가 진짜 기준
prose가 권한 체크에 걸림	markdown 본문을 코드로 스캔	shell 메타문자 예시 최소화

API와 로컬 차이	컨테이너는 네트워크 차단	portable != identical runtime

섹션 07

스킬을 더 똑똑하게 -- 고급 패턴 4가지

첫 스킬이 작동하면, 더 어려운 작업을 스킬화할 차례다.

패턴 1: Plan-Validate-Execute

언제: 배치 작업, 되돌리기 어려운 작업 (DB 마이그레이션, 파일 일괄 수정)

- 실행 계획을 먼저 보여주고
- 사용자 확인을 받은 후
- 실행 --> 결과 보고

> 파괴적 작업은 반드시 이 패턴을 써라

패턴 2: 점진적 컨텍스트 로딩

언제: 도메인 지식이 많은 스킬 (금융, 의료, 법률)

- SKILL.md --> 핵심 원칙만
- references/domain.md --> 상세 규칙
- !`cat`으로 필요할 때만 로드

> SKILL.md는 목차, references/는 본문

패턴 3: 반복 개선 + 종료 조건

언제: 코드 리뷰, 문서 교정 등 품질 수렴이 필요한 작업

- 초안 생성
- 체크리스트 기반 자체 검증
- 통과 못하면 수정 반복 (최대 N회)

> 종료 조건 없으면 무한 루프에 빠진다

패턴 4: Two-Claude 개발법

언제: 복잡한 스킬을 처음 만들 때

- Claude A: 스킬 설계 및 작성
- Claude B: 설계된 스킬을 실행하며 테스트
- 피드백 루프를 3회 이상 반복

> 자기가 만든 스킬을 자기가 테스트하면 맹점이 생긴다

섹션 08

스킬 테스트 & 반복 개선 프로세스

8-1. 3회 반복 원칙

> 최소 3번의 반복 개선 전에는 안정적이라고 판단하지 마라.

- 반복 1: 기본 동작 확인
- 반복 2: 엣지 케이스 발견 및 수정
- 반복 3: 다른 모델(Haiku/Sonnet/Opus)에서 교차 테스트

8-2. 테스트 체크리스트

✓ 정상 입력으로 기대 출력이 나오는가?

✓ 빈 입력 / 잘못된 입력에서 적절히 대응하는가?

✓ 트리거 조건 외 상황에서 불필요하게 호출되지 않는가?

✓ 5,000토큰 이내인가? (wc -w로 대략 확인)

✓ Haiku에서도 핵심 동작이 작동하는가?

✓ 다른 사람이 처음 사용해도 의도대로 작동하는가?

8-3. 평가를 먼저 만들어라

- 문서를 쓰기 전에 평가 기준을 먼저 만들어라
- 정확도(%), 일관성(동일 입력 -> 동일 출력), 완전성(필수 항목 포함)

8-4. Eval Harness (평가 자동화)

- 8~10개 should-trigger + 8~10개 should-not-trigger 쿼리로 평가 세트를 만들어라

```
{
"skill_name": "release-notes",
"evals": [
{"id": 1, "prompt": "릴리스 노트 써줘",
"expected": "should_trigger=true"},
{"id": 2, "prompt": "README 오탈자 수정",
"expected": "should_trigger=false"}
]
}
```

8-5. 개선 로그

버전	변경 내용	트리거	결과
v1	초기 버전	-	기본 동작 확인
v2	엣지 케이스 추가	빈 입력 발견	방어 코드 추가
v3	모델 교차 테스트	Haiku에서 실패	지시 간결화

섹션 09

Quick Reference Card

책상 옆에 붙여두고 쓰는 1페이지 요약

구조 & 문법

```
-- SKILL.md 구조 --
frontmatter: name / description / when_to_use
본문: 목적 -> 원칙 -> 절차 -> 예제

-- 파일 위치 --
개인: ~/.claude/skills/
프로젝트: .claude/skills/

-- 특수 문법 --
$ARGUMENTS 사용자 입력 전체
$1, $2 위치 인자
${CLAUDE_SKILL_DIR} 스킬 디렉토리
!`command` 동적 컨텍스트 주입

-- 제한 --
name: 64자, 소문자+하이픈
description + when_to_use: 1536자
SKILL.md 권장: 500줄 / 5,000토큰 이내
```

체크리스트 & 패턴

✓ 반복성 (5회+)
✓ 명확성 (공격적 description)
✓ 간결성 (500줄 이내)
✓ 시범성 (예제 > 규칙)
✓ 적응성 (유연/정밀 매칭)

고급 패턴

- Plan-Validate-Execute -- 파괴적 작업
- Progressive Disclosure -- 도메인 지식

- Iterative Refinement -- 품질 수렴
- Two-Claude Method -- 복잡한 신규 스킬

개발 프로세스

> 평가 기준 --> 초안 --> 3회 반복 --> 배포

섹션 10

마무리 + 다음 단계

핵심 요약 3줄

- 스킬은 당신의 업무 전문성을 코드화한 것이다.
- 5가지 조건을 충족하면 반복적으로 가치를 만든다.
- FCB 워크시트로 시작하고, 스타터 템플릿으로 바로 만들어보라.

다음 단계

- 첫 스킬을 만들었으면 --> 커뮤니티에서 공유하세요
- 더 깊은 내용은 --> YouTube 채널에서 확인하세요
- 스킬 생태계 탐색 --> agentskills.io

개발곰곰 | youtube.com/@devgomgom

AI 업무 자동화 & 기업 도입 컨설팅

> 이 매뉴얼의 구조 자체가 '좋은 스킬의 5가지 조건'을 따른다.

섹션

참고 자료

공식 문서

- Claude Skills 공식 문서 -- 구조, frontmatter, allowed-tools, distribution
- Complete Guide to Building Skills -- Anthropic 공식 설계/테스트/배포 가이드

Agent Skills 표준

- Agent Skills Specification -- SKILL.md, name/description, 디렉터리 구조 규격
- Best Practices -- 스코프, 실행 기반 개선, 템플릿 사용 원칙
- Optimizing Descriptions -- description 기반 트리거 품질 개선
- Using Scripts -- 비대화형 실행, structured output 설계

공식 예제 저장소

- anthropics/skills -- 공식 Skill 예제, spec, template 모음
- skill-creator -- 스킬 생성/설계 핵심 예제
- doc-coauthoring -- 문서 협업형 스킬 패턴

실무 트러블슈팅 (공식 이슈)

- #9716 -- .claude/skills/ 디렉터리 인식 실패
- #10768 -- 자연어 intent matching 문제
- #11266 -- ~/.claude/skills/ 자동 발견 안 됨