

## Claude Skills 개발자 매뉴얼

### 소개: Claude Skills란 무엇이며 무엇을 할 수 있나요?

Claude Skills는 Anthropic의 AI 비서 Claude에게 새로운 능력을 부여하는 **맞춤형 도구 세트**입니다 <sup>1</sup>. 간단히 말해, Skill은 Claude가 특정 작업이나 도메인에 특화된 지식을 **동적으로 불러올 수 있도록 해주는 패키지**입니다. 각 Skill은 하나의 폴더로 구성되며, 그 안에 해당 기능을 수행하기 위한 **지침서(SKILL.md)**, 필요한 **스크립트나 코드**, **템플릿 파일**, **참고 자료** 등이 포함됩니다 <sup>2</sup>. Claude는 사용자의 질문이나 요청을 분석하여, **관련 Skill이 있을 경우 자동으로 해당 Skill을 불러와 사용합니다** <sup>3</sup>. 이러한 메커니즘 덕분에 개발자와 조직은 일회성 프롬프트에 의존하지 않고, **재사용 가능한 워크플로우와 전문 지식을 Claude에 주입할 수 있게 됩니다** <sup>4</sup>.

Claude Skills의 주요 특징은 다음과 같습니다 <sup>5</sup> <sup>3</sup>:

- **모듈식(Modular)**: 각 Skill은 독립적인 폴더로 존재하는 **자급자족형 모듈**입니다. 필요에 따라 개별 Skill을 추가하거나 제거해도 다른 Skill에 영향을 미치지 않습니다 <sup>6</sup>. 예를 들어, PDF 처리용 Skill과 코드 리뷰용 Skill은 서로 분리되어 관리될 수 있습니다.
- **발견 가능(Discoverable)**: Skill은 Claude가 **자동으로 찾아서 사용합니다**. 사용자가 특정 Skill을 직접 호출할 필요 없이, Claude가 대화 내용을 보고 해당 Skill의 **설명(description)**과 일치한다고 판단하면 Skill을 로드합니다 <sup>7</sup>.
- **휴대성(Portable)**: 같은 Skill 패키지가 **Claude 웹 애플리케이션(claude.ai)**, **Claude Code(데스크톱/CLI)**, **API/에이전트 SDK** 등 모든 환경에서 동일하게 동작합니다 <sup>3</sup>. 한 번 만든 Skill을 다양한 Claude 플랫폼에서 활용할 수 있습니다.
- **합성 가능(Composable)**: 여러 Skill이 **조합되어 동시에 사용될 수 있습니다**. Claude는 복잡한 작업일수록 여러 Skill을 한 번에 불러와 협업시키며 작업을 진행할 수 있습니다 <sup>5</sup>.
- **효율적(Efficient)**: Skill은 **필요할 때만 로드됩니다**. Claude는 사용자의 요청과 Skill 설명을 대조하여 정말 관련성이 있을 때에만 해당 Skill의 내용을 읽고 실행합니다 <sup>3</sup>. 또한 Skill 내에서도 **점진적 로딩(progressive disclosure)**을 활용하여, 처음에는 메타데이터만 읽고 필요한 경우에만 추가 파일을 불러오기 때문에 **컨텍스트 창을 효율적으로 사용합니다** <sup>8</sup>.
- **강력함(Powerful)**: Skill은 단순 지침뿐만 아니라 **실행 가능한 코드를 포함할 수 있습니다** <sup>9</sup>. Claude의 **코드 실행 도구(Code Execution)**를 통해 Skill 내 스크립트를 실제로 실행함으로써, 예컨대 데이터베이스 조회나 파일 변환과 같은 전통적인 프로그래밍 작업도 Skill을 통해 수행할 수 있습니다 <sup>9</sup>. 단, 이러한 기능을 사용할 때는 보안과 권한 관리에 유의해야 합니다 (뒤에서 자세히 설명합니다).

**요약하면**: Claude Skills는 조직이나 개발자가 **클로드에게 자기만의 업무 방식이나 전문 지식을 가르쳐주는 매뉴얼**이라고 볼 수 있습니다. 이를 통해 Claude를 일반적인 조수에서 나만의 **특화된 전문가**로 변모시킬 수 있습니다 <sup>10</sup>. 예를 들어, 재무팀은 재무 보고서 작성 Skill을 만들어 Claude가 매번 동일한 양식과 규칙을 따르도록 할 수 있고, 마케팅팀은 브랜드 가이드라인 Skill을 통해 모든 산출물이 브랜드 규격을 준수하게 만들 수 있습니다. 이러한 Skill은 **프롬프트 엔지니어링을 문서화된 자산으로 전환하여 일관성과 재현성을 높여주며, 필요할 때만 불러오기 때문에 Claude의 기본 성능에도 영향을 최소화합니다** <sup>11</sup>.

이전에는 비슷한 작업을 할 때마다 일일이 긴 지시문(프롬프트)을 다시 쓰거나, 특정 출력을 얻기 위해 별도의 소프트웨어를 만들어야 했습니다. Claude Skills를 활용하면 이런 **프롬프트 노하우나 워크플로우를 하나의 패키지로 만들어 체계화할 수 있습니다** <sup>11</sup>. 이는 에이전트의 동작을 더 **재현 가능하고 감사 가능하게** 만들어주며, 조직 차원에서 AI 활용 지침을 공유하고 관리하기 쉽게 해줍니다. 특히 기업 환경에서는 일관된 출력과 절차 준수가 중요한데, Skills를 통해 **출력 품질의 표준화와 절차 준수의 자동화**를 실현할 수 있습니다. 이제 본 매뉴얼을 통해 Claude Skills의 세계를 자세히 탐험해봅시다.

## Claude Skills 시작하기: 준비 및 인터페이스 사용법

Claude의 웹 인터페이스( [claude.ai](https://claude.ai) )에서 Skills 기능을 사용하려면 몇 가지 **사전 준비 조건**을 충족해야 합니다. 먼저 **무료 요금제**(Pro, Max, Team, Enterprise 중 하나)를 사용 중이어야 하며, **코드 실행 및 파일 생성(Code execution and file creation)** 기능이 활성화되어야 합니다 <sup>12</sup> <sup>13</sup> . **Team/Enterprise 플랜**에서는 관리자가 조직 설정에서 Skills를 켤 수 있으며, Pro/Max 사용자는 개인 설정에서 바로 활성화할 수 있습니다 <sup>14</sup> .

### Skills 기능 활성화 방법

1. Claude 웹 앱에서 우측 상단의 **Settings(설정)** 메뉴로 이동합니다.
2. **Capabilities(기능)** 탭을 선택합니다 <sup>13</sup> .
3. 먼저 **"Code execution and file creation"** 옵션이 켜져 있는지 확인하세요 <sup>15</sup> . (Skills는 Claude의 안전한 샌드박스 환경인 코드 실행 기능을 필요로 합니다.)
4. 아래로 스크롤하여 **Skills** 섹션을 찾습니다 <sup>15</sup> . 이 섹션이 보이지 않는다면 코드 실행 기능이 켜져 있는지 다시 확인해야 합니다 <sup>16</sup> .
5. **"Skills (Preview)"** 섹션에서 Anthropic이 제공하는 여러 **예제 Skill** 목록을 볼 수 있습니다. 각 Skill 옆의 토글 스위치를 사용하여 해당 Skill을 **활성화(ON)** 또는 **비활성화(OFF)** 할 수 있습니다 <sup>17</sup> <sup>18</sup> . 필요한 Skill을 켜두면 Claude가 적절한 상황에서 그것을 활용하게 됩니다.
6. **사용자 정의 Skill을 추가하려면**, Skills 섹션에서 **Upload skill(스킬 업로드)** 버튼을 클릭합니다 <sup>19</sup> . 파일 선택 창이 나타나면, 미리 준비한 Skill 폴더를 ZIP 파일로 압축한 것을 선택하여 업로드합니다 <sup>20</sup> . 업로드가 성공하면 해당 Skill이 목록에 추가되고 토글을 통해 활성화할 수 있습니다 <sup>21</sup> .

Claude 웹 애플리케이션의 설정 화면에서 Skills 기능을 활성화하고 사용자 정의 Skill을 업로드하는 인터페이스. 목록에 기본 제공 Skill들이 나열되어 있으며, 우측에 각 Skill의 ON/OFF 토글 스위치가 표시됩니다. 예시 화면에서는 "algorithmic-art", "brand-guidelines" 등 여러 기본 제공 Skill이 리스트업되어 있고, 사용자가 슬라이더를 통해 개별 Skill을 활성화/비활성화할 수 있습니다. 또한 화면 하단에는 'Upload skill' 버튼이 있어 새로운 Skill의 ZIP 파일을 추가할 수 있습니다.

위 스크린샷에서 볼 수 있듯이, Anthropic이 기본 제공하는 Skill들이 미리 포함되어 있습니다. 예를 들어 **Excel 통합 문서 생성/편집**, **Word 문서 작성**, **PowerPoint 프레젠테이션 생성**, **PDF 처리** 등의 Skill들은 모든 사용자가 사용할 수 있는 내장 Skill입니다 <sup>22</sup> . 이러한 Skill들은 코드 실행 기능만 켜져 있으면 Claude가 **자동으로 적절한 작업에 사용**하므로, 별도로 호출할 필요가 없습니다 <sup>23</sup> . (예: "Q3 실적에 대한 프레젠테이션을 만들어줘"라고 물으면 Claude는 자동으로 PowerPoint Skill을 활용하여 슬라이드를 생성합니다 <sup>24</sup> .)

사용자가 직접 추가한 **커스텀 Skill**은 해당 계정에만 **개인적으로 저장**됩니다 <sup>25</sup> <sup>26</sup> . 조직 내에서 Skill을 공유하려면 각 사용자들이 개별적으로 해당 ZIP 파일을 업로드해야 하며, 아직 웹 인터페이스 상에서 조직 전체에 Skill을 배포하는 중앙 관리 기능은 제공되지 않습니다 (API를 통한 배포 방법은 뒤에서 설명합니다) <sup>25</sup> . 따라서 팀원들과 Skill을 공유하려면 GitHub 등의 저장소에 Skill 폴더를 올려두거나, ZIP 파일을 배포하여 각자가 업로드하도록 안내해야 합니다.

## Claude Skill 작성: 새 스킬 만들기 단계별 안내

이제 자신만의 **커스텀 Skill**을 만들어보겠습니다. 전체 과정은 **Skill 기획 → Skill 폴더 및 파일 구성 → 매니페스트(YAML) 작성 → 본문 작성 → 패키징 → 업로드**의 단계로 진행됩니다. 아래에 각 단계를 상세히 설명합니다.

### 1. 스킬 기획 및 요구사항 정의

새로운 Skill을 작성하기 전에, **어떤 문제를 해결할 것인지** 명확히 정의해야 합니다. **Skill의 목표와 범위**를 구체적으로 정하세요 <sup>27</sup> . 강력한 Skill은 **구체적인 니즈**를 다룹니다. 예를 들어 막연히 "재무 관련 도와줘"보다는, "PDF 형식의 재

무제표에서 특정 데이터를 추출해 CSV로 정리"처럼 **입력 형식, 수행 작업, 기대 출력**이 분명한 목표가 좋습니다 <sup>27</sup>. 이를 통해 Skill이 다루는 **사용 시나리오**를 명확히 하고, Claude가 언제 이 Skill을 사용해야 하는지 판단하기 쉬워집니다.

- **발동 트리거 정의:** 어떤 사용자 요청이 들어올 때 이 Skill이 활성화되어야 할까요? Skill의 동작을 이끌어내는 **키워드나 문맥**을 생각해 보세요. (예: "캘린더 일정 확인" Skill이라면 "일정 보여줘", "meeting", "캘린더" 등의 단어가 요청에 있을 때 사용되어야 할 것입니다.)
- **성공 기준 정의:** Skill이 제대로 작동했을 때의 **출력 결과물**은 무엇이며 어떻게 측정할 수 있나요? 예를 들어 "요약 보고서 생성" Skill이라면, 보고서에 반드시 포함되어야 할 항목이나 형식이 있을 것입니다. 이러한 **성공 기준**을 Skill 설계 시에 정해두면, 나중에 결과물을 검증하거나 개선할 때 기준점이 됩니다 <sup>28</sup>.
- **한계와 예외 정의:** Skill이 **처리하지 않을 범위**는 어디까지인지 미리 정해두세요. 또한 입력이 불완전하거나 엉뚱할 때 어떻게 대응할지 (오류 메시지 출력 등) 계획합니다. 이렇게 하면 Skill 설명에 **명확한 경계 조건**을 포함시켜 오작동을 줄일 수 있습니다. (예: "이 Skill은 금융 데이터에만 사용되며 일반 문서 요약에는 사용하지 않습니다"라고 명시해 두는 식입니다.)

## 2. 스킬 폴더 구조 설계

Claude Skill은 **특정 디렉토리(폴더)** 안에 파일들을 모아 구성합니다 <sup>2</sup>. 기본적으로 각 Skill 폴더에는 `SKILL.md` 파일 하나만 있으면 되지만, 필요에 따라 여러 파일을 포함할 수 있습니다. Skill 폴더의 **이름이 곧 Skill의 이름**이 되므로, 폴더명을 간단하고 알기 쉽게 정하세요. 가능하면 영문 **소문자와 하이픈(-)**으로 구성하는 것이 좋습니다 <sup>29</sup>. (예: `pdf-editor`, `brand-guidelines` 등) 폴더 이름은 Claude가 Skill을 식별하는데 사용되며, Settings 화면의 목록에도 이 이름으로 표시됩니다.

폴더 구조 예시:

```
my-skill/
├── SKILL.md           # 필수: Skill의 매니페스트 및 지침서
├── reference.md       # 선택: 추가 참고 자료
├── templates/        # 선택: 템플릿 파일 모음 폴더
│   └── template.docx
├── scripts/          # 선택: 실행 스크립트 모음 폴더
│   └── helper.py
└── assets/           # 선택: 이미지 등 기타 자산 폴더
    └── logo.png
```

위 예시는 CometAPI 블로그에서 소개된 Skill 폴더 구조 예시를 각색한 것입니다 <sup>30</sup>. 이처럼 **SKILL.md를 중심으로**, 필요시 관련 리소스를 하위 폴더로 구성하여 **모듈식**으로 관리할 수 있습니다 <sup>31</sup> <sup>32</sup>. 예를 들어, `reference.md` 나 여러 `.md` 파일들을 만들어 SKILL.md 본문에서 참조하면, Claude는 상황에 따라 그 파일들만 추가로 읽어들이는 <sup>8</sup> . 이는 Skill 내용을 **여러 조각으로 나누어** 필요한 내용만 로드함으로써 효율을 높이는 기법입니다. (이 점진적 로딩 방식에 대해서는 잠시 후에 더 설명합니다.)

**참고:** Anthropic의 공식 Skills 예제 저장소나 일부 문서에서는 별도의 `metadata.yaml` 파일을 두어 버전, 작성자, 권한 정보 등을 관리하기도 합니다 <sup>33</sup> <sup>30</sup>. 그러나 일반적으로 **SKILL.md 하나로** 충분하며, 추가 메타데이터가 필요하면 YAML 프론트매터 내에 작성해도 됩니다.

## 3. SKILL.md 매니페스트(YAML 헤더) 작성

Skill 폴더의 핵심인 `SKILL.md` 파일을 만들어봅시다. 이 파일은 Skill의 **메타데이터(앞부분 YAML)**와 **본체(Markdown 포맷 지침)**로 구성됩니다. 우선 파일 맨 위에 YAML 형식의 **프론트매터**를 작성해야 합니다. YAML 프론트매터는 `---`로 시작하고 `---`로 끝나는 영역으로, 그 사이에 키-값 쌍으로 메타데이터를 정의합니다.

필수 필드는 **name**과 **description** 두 가지입니다 34 :

- **name**: Skill의 짧은 이름입니다. 64자 이내의 간결하면서도 구체적인 이름을 짓습니다 34 . 공백 대신 하이픈(-)을 사용하는 것이 좋습니다 (예: pdf-editor , brand-guidelines ) 29 . 가능하면 영문 소문자로 작성하고 특수문자는 피하세요 35 . 이 이름은 Claude가 Skill을 식별하는 데 사용되며, 업로드 후 Settings 화면의 목록에도 표시됩니다.
- **description**: Skill의 역할과 언제 사용되어야 하는지를 설명하는 한 문장입니다. 200자 이내로 작성해야 합니다 36 . 이 설명이 Skill 트리거의 핵심이 되므로 가장 중요합니다 37 . Claude는 사용자의 요청과 이 description을 대조하여 Skill 적용 여부를 판단합니다 38 . 따라서 description에는 해당 Skill이 다루는 구체 작업, 활용 사례, 적용 대상, 한계 등을 명확히 담아야 합니다 39 40 .

예를 들어, "이 Skill은 PDF와 문서 작업을 도와줍니다"라는 약한 설명보다는 41 ,

**강한 설명 예시:** "Comprehensive PDF manipulation toolkit for extracting text and tables, creating new PDFs, merging/splitting documents, and handling forms. When Claude needs to fill in a PDF form or programmatically process, generate, or analyze PDF documents at scale. Use for document workflows and batch operations. Not for simple PDF viewing or basic conversions." 40

처럼 구체적인 기능(추출, 생성, 병합 등)과 사용 시점(대량 처리, 양식 기입), 그리고 비사용 조건(단순 조회 등)까지 명시하는 것이 좋습니다. 강한 버전은 Claude에게 여러 데이터 포인트를 제공합니다: 구체적 동사 (추출, 생성, 등), 명확한 사용 사례 (폼 채우기, 배치 작업 등), 그리고 명확한 경계 ("단순 조회에는 사용하지 않음")를 담고 있습니다 42 . 이렇게 작성된 description은 Claude에게 Skill의 쓰임새를 정확히 알려주어, 관련 없는 질문에는 함부로 개입하지 않게 하고 꼭 필요한 상황에서만 Skill이 발동되도록 합니다 43 44 .

이 외에 YAML 메타데이터에는 필요한 경우 몇 가지 선택 필드를 추가할 수 있습니다:

- **dependencies**: Skill 실행에 필요한 외부 패키지나 라이브러리 목록입니다. 예를 들어 Python의 pandas 패키지가 필요하다면 dependencies: python>=3.8, pandas>=1.5.0 과 같이 명시합니다 45 . Claude나 Claude Code는 Skill 로드 시 이러한 패키지를 PyPI나 npm에서 자동 설치할 수 있습니다 46 (단, API 환경에서는 런타임 패키지 설치가 불가능하므로 미리 설치되어 있어야 합니다 47 ).
- **allowed-tools**: Claude가 이 Skill을 실행하는 동안 사용할 도구 권한을 제한할 수 있습니다. 예를 들어 allowed-tools: ["python"] 으로 지정하면 Claude가 코드 실행 시 Python만 사용하고 Shell 명령 (Bash)은 쓰지 않도록 할 수 있습니다. 특별히 지정하지 않으면 Claude는 필요한 경우 Bash , Python 등 사용 가능한 모든 도구를 활용합니다. 이 필드는 고급 설정에 속하므로 대부분의 경우 필수는 아닙니다.
- **license**: Skill의 라이선스 정보를 명시합니다. 공개 공유를 염두에 둔 Skill이라면 MIT, Apache-2.0 등의 라이선스명을 넣을 수 있고, 사내 전용이라면 Proprietary 등으로 표시합니다. (실제로 Anthropic 제공 Skill들의 YAML에 license: Proprietary 등이 포함되어 있습니다 48 .)
- **version**: Skill의 버전 번호를 정할 수 있습니다. 예: version: "1.0.0" . 버전을 명시해 두면 나중에 Skill을 업데이트하거나 여러 버전을 관리할 때 유용합니다. (Claude 콘솔이나 API에서는 버전 관리를 지원하지 않습니다. 웹 UI에서는 수동 관리 필요.)

YAML 프론트매터 작성이 끝나면 닫는 --- 줄을 넣어 마무리합니다.

## 4. Skill 본문 작성 (Markdown 지침 설계)

메타데이터 아래에는 Skill의 **핵심 로직과 가이드**를 Markdown 형식으로 작성합니다. 이 부분이 실제 Claude에게 주어지는 "추가 프롬프트" 역할을 합니다. 작성 시 다음 사항을 고려하세요:

- **개요 섹션 추가:** `## Overview` 등으로 Skill의 전반적인 개요를 간략히 설명합니다. 여기에는 Skill의 목적과 작동 방식 요약도 적습니다. Overview는 다른 섹션들의 맥락을 제공하며, Claude가 Skill을 불러들였을 때 먼저 읽는 부분이 됩니다.
- **단계별 가이드:** Skill이 수행해야 할 작업을 **여러 단계**로 나누어 서술합니다. 가능한 경우 `## Step 1`, `## Step 2` 또는 목록 `1.`, `2.` 등을 사용해 **절차를 명확히** 나타냅니다 <sup>49</sup>. 각 단계에는 구체적인 지시를 내리고, Claude가 어떤 도구를 사용하거나 어떤 출력을 생성해야 하는지 써줍니다.
- **예시와 코드 블록:** Claude에게 기대하는 입력/출력 형태나 특정 명령 사용법 등을 보여주려면 **예시**를 들어주세요. Markdown의 **코드 블록**(`` ``)을 활용하여 예시 입력, 출력, 또는 실행할 코드 조각을 제시하면 Claude가 더 정확히 이해합니다 <sup>49</sup>. 예를 들어, Bash 명령을 알려주려면 `bash ...` 블록으로, 예상 출력 형식을 보여주려면 `json ...` 또는 `text ...` 블록으로 제시할 수 있습니다.
- **조건부 지침:** 상황에 따라 다른 동작을 해야 한다면 헤더로 섹션을 구분해 안내하세요. 예를 들어 `### 상황 A:`, `### 상황 B:`처럼 나누어, Claude가 해당 경우에만 그 부분을 읽도록 합니다. Claude는 사용자의 요청과 가장 관련 있는 섹션을 골라 읽으므로, **여러 시나리오를 한 Skill 내에서 다룰 때** 특히 유용합니다 <sup>50 51</sup>.
- **한계와 오류 처리:** Skill이 다루지 않을 사례나 실패 시 대응 방안을 명시합니다. 예를 들어 "만약 입력이 X 형태가 아니라면 오류 메시지를 제시하고 중단할 것" 등의 지침을 포함하세요. 이를 통해 Claude가 억지로 무리한 동작을 시도하지 않게 하고, 사용자에게 필요한 정보를 요청하거나 적절히 대응할 수 있습니다 <sup>52</sup>.
- **추가 자료 참조:** Skill 내용이 너무 방대하거나 분기별로 상세 매뉴얼이 필요한 경우, 별도의 파일로 분리한 뒤 본문에서 **상대 경로 링크**로 참조할 수 있습니다. 예를 들어 Skill 폴더에 `REFERENCE.md`를 두고 `SKILL.md` 본문에 `[관련 참고사항](REFERENCE.md)` 식으로 링크를 걸어두면, Claude가 Skill을 실행하며 해당 부분에 이르렀을 때 자동으로 그 파일을 읽습니다 <sup>53</sup>. 이렇게 하면 한 파일에 모든 내용을 넣지 않고도 필요한 내용만 로드시켜 효율을 높일 수 있습니다 <sup>53</sup>.

마지막으로, **Skill 본문 작성 시의 전반적인 원칙**은 "**명확성(clarity)**"과 "**구체성(specificity)**"입니다. Claude는 사람과 달리 모호한 지침을 알아서 추론하기보다는, Skill에 적합한 대로 충실히 따르려 합니다. 따라서 사람이 읽는 매뉴얼을 작성한다는 느낌으로, **누구나 따라도 동일한 결과를 낼 정도로 상세히** 적는 것이 좋습니다. 하지만 동시에 **불필요하게 장황한 설명은 피하고 핵심만 명시**해야 컨텍스트 낭비를 줄일 수 있습니다. 항상 "이 부분이 Claude의 문제 해결에 직접 필요한 내용인가?"를 자문하며 작성하세요 <sup>54 51</sup>.

### 예시: 브랜드 가이드라인 Skill 내용

다음은 Anthropic에서 제공하는 **Brand Guidelines** (브랜드 가이드라인) Skill의 예시 내용입니다 <sup>55 56</sup>. 이 Skill은 회사의 브랜드 사용 수칙을 정리한 것으로, Claude가 문서나 자료를 만들 때 브랜드 규정을 따르도록 도와줍니다.

#### ## Metadata

name: Brand Guidelines

description: Apply Acme Corp brand guidelines to all presentations and documents

#### ## Overview

This Skill provides Acme Corp's official brand guidelines for creating consistent, professional materials. When creating presentations, documents, or marketing materials, apply these standards to ensure all outputs match Acme's visual identity. Claude should reference these guidelines whenever

creating external-facing materials or documents that represent Acme Corp.

## ## Brand Colors

Our official brand colors are:

- Primary: #FF6B35 (Coral)
- Secondary: #004E89 (Navy Blue)
- Accent: #F7B801 (Gold)
- Neutral: #2E2E2E (Charcoal)

## ## Typography

Headers: Montserrat Bold

Body text: Open Sans Regular

Size guidelines:

- H1: 32pt
- H2: 24pt
- Body: 11pt

## ## Logo Usage

Always use the full-color logo on light backgrounds. Use the white logo on dark backgrounds. Maintain minimum spacing of 0.5 inches around the logo.

## ## When to Apply

Apply these guidelines whenever creating:

- PowerPoint presentations
- Word documents for external sharing
- Marketing materials
- Reports for clients

## ## Resources

See the resources folder for logo files and font downloads.

위 내용을 보면, 브랜드 컬러, 타이포그래피, 로고 사용법 등 **스타일 가이드**를 항목별로 명확히 제시하고 있습니다. description에는 이 Skill이 언제 쓰이는지 잘 설명되어 있고, 본문에는 구체적인 규칙과 예시가 나열되어 있습니다. Claude는 이 Skill을 활성화한 상태에서 "회사 소개서 만들어줘" 같은 요청을 받으면, 위 지침을 참고하여 색상이나 폰트를 준수하려고 노력할 것입니다.

이처럼 Skill 내용은 **해당 작업을 잘 아는 사람이 작성한 매뉴얼**과 크게 다르지 않습니다. 차이점은, 이 매뉴얼을 사람이 아닌 Claude가 읽고 따른다는 것이죠. 따라서 글의 조직, 명확성, 완전성이 중요합니다. 위 예시를 참고하여, 여러분도 자신의 분야에서 이런 식으로 Skill 콘텐츠를 구성해 보세요.

## 5. 부가 리소스 및 스크립트 첨부 (Optional)

Skill에 필요한 부가 자료나 코드가 있다면, 앞서 설계한 대로 추가 파일을 포함할 수 있습니다. **참고 자료**(예: 회사 정책 문서, 추가 설명서 등)는 별도 `.md` 파일로 작성하여 Skill 폴더에 넣고, 본문에서 링크하면 됩니다 <sup>53</sup>. **이미지나 아이콘, 템플릿 파일**(예: 워드 템플릿, PPT 테마 등)은 `assets/` 나 `templates/` 폴더를 만들어 넣어두고, 본문에 경로를 언급하면 Claude가 필요 시 해당 파일을 활용합니다.

만약 **코드 실행이 필요한 작업**이라면, 해당 코드를 `scripts/` 폴더에 파이썬(`.py`)이나 자바스크립트(`.js`) 파일로 포함시킬 수 있습니다. 그리고 SKILL.md 본문에 그 스크립트를 어떻게 사용할지 지시합니다. 예를 들어 "데이터 분

석을 위해 `scripts/analyze.py`를 실행"과 같은 내용을 본문에 쓰면 Claude는 그 파일을 찾아 실행합니다. 이때 Claude와 Claude Code는 필요한 패키지가 dependencies에 명시되어 있으면 자동으로 **pip/npm 설치**를 시도합니다 <sup>46 47</sup>.

코드 파일을 첨부할 때는 **민감한 정보**를 하드코딩해서는 안 됩니다. API 키나 비밀번호 등은 절대로 코드에 직접 넣지 말고, 필요 시 Claude에게 사용자에게 입력을 요청하게 하거나 안전한 저장소를 연동하세요 <sup>57</sup>. 또한 외부 서비스와 연동하는 코드의 경우 Anthropic이 제공하는 **MCP (Managed Connection Points)** 등을 활용해 안전하게 접근하세요 <sup>58</sup>.

**Note:** 현재 Claude 웹과 Claude Code 환경에서 Skill 실행 시 **인터넷에 직접 액세스**하는 기능은 제한되어 있습니다. 따라서 외부 API 호출이 필요한 경우, 해당 기능을 위해 별도로 허용된 MCP 서버를 세팅하거나, 사용자로부터 필요한 데이터를 업로드 받아 처리하는 방식으로 우회해야 합니다. Claude Skills의 코드 실행 환경은 **격리된 샌드박스**이며, 기본적으로 파일시스템이나 네트워크에 임의 접근이 불가하도록 보호됩니다 <sup>59 60</sup>.

## 6. Skill 패키징 및 검증

이제 Skill 폴더와 내용을 다 만들었다면, 이를 Claude에 등록하기 위해 **패키징**해야 합니다. 패키징은 간단히 말해 **Skill 폴더를 ZIP 압축**하는 것입니다. 이때 주의할 점:

- **폴더 이름과 Skill 이름 일치:** Skill 폴더의 이름과 SKILL.md의 `name` 필드가 가급적 동일해야 합니다. 예를 들어 name이 "calendar-assistant"이면 폴더명도 `calendar-assistant/`로 하는 것이 안전합니다. 만약 불일치하거나 특수문자가 포함되면 업로드가 거부될 수 있습니다 <sup>35</sup>.
- **ZIP 구조:** ZIP 파일 안에 **폴더 자체가 포함되어** 있어야 합니다. 예를 들어 `my-skill.zip` 내 최상위에 `my-skill/` 폴더가 있고 그 안에 SKILL.md가 있도록 압축해야 합니다 <sup>61</sup>. 잘못 압축하여 `my-skill.zip` 최상위에 바로 SKILL.md 파일이 있거나, 폴더가 두 번 중첩되면 Claude가 Skill을 인식하지 못합니다 <sup>62</sup>.

올바른 ZIP 구조:

```
my-skill.zip
├─ my-skill/
│  └─ SKILL.md
│  └─ ...
```

잘못된 ZIP 구조 예:

```
my-skill.zip
├─ SKILL.md
└─ ...
```

위와 같이 잘못 압축한 경우 업로드 시 에러가 발생합니다.

- **크기 제한:** Skill ZIP 파일은 너무 크면 업로드가 제한됩니다 <sup>63</sup>. 정확한 한도는 플랜에 따라 다를 수 있지만, 일반적인 Markdown과 스크립트, 소량의 리소스만 담는다면 수 MB 이내로 유지하는 것이 좋습니다. 불필요하게 용량을 차지하는 데이터가 있다면 제거하거나 외부 저장소를 활용하세요.
- **필수 파일 확인:** ZIP 내에 **SKILL.md 파일이 꼭 있어야** 합니다. 간혹 이름을 잘못 입력하거나 확장자를 빼먹는 실수로 업로드가 실패할 수 있으니 확인하세요 <sup>35</sup>.

- **이름/설명 유효성:** Skill의 name이나 description에 지원되지 않는 문자가 없는지 확인합니다. 예를 들어 특수 문자나 너무 긴 내용은 문제가 될 수 있습니다 <sup>64</sup>.

패키징을 완료했으면, **업로드 전에 간단한 자체 검증**을 해봅시다:

- SKILL.md 내용이 명확하고 오타자가 없는지 점검합니다.
- Skill description이 정확히 이 Skill의 사용 조건을 반영하는지 확인합니다 (너무 포괄적이거나 모호하지 않은 지).
- SKILL.md 본문에서 참조하는 파일 경로들이 실제로 존재하는지 확인합니다 (없는 파일을 링크하면 실행 시 오류가 발생합니다).
- 가상의 사용 시나리오를 상정하고, 해당 요청 문구가 Skill description과 맞아떨어지는지 검토합니다 (Claude 입장에서 이 요청에 이 Skill을 쓸지 판단해볼 것).

## 7. Skill 업로드 및 활성화

이제 준비된 Skill을 Claude에 **등록**할 차례입니다. 앞서 "Skills 기능 활성화 방법" 섹션에서 소개한 대로, Settings > Capabilities의 Skills 영역에서 **Upload skill** 버튼을 클릭하고, 방금 만든 ZIP 파일을 업로드하세요 <sup>65</sup>. 업로드가 성공하면 Skills 목록에 새 Skill이 표시될 것입니다 <sup>66</sup>. 해당 Skill을 **토글 ON**으로 켜줘야 Claude가 이를 활용할 수 있으니, 업로드 후 반드시 활성화하는 것을 잊지 마세요 <sup>66</sup>.

마지막으로, Skill이 제대로 동작하는지 **테스트**를 진행합니다. 다음 섹션에서는 Skill을 테스트하고 디버그하는 방법에 대해 알아보겠습니다.

## Claude AI를 활용한 Skill 생성 (Skill-Creator 사용)

지금까지는 파일을 직접 작성하고 업로드하는 **수동 방식**으로 Skill을 만드는 절차를 살펴보았습니다. Anthropic는 Skill 작성 과정을 돕기 위해 **Skill-Creator**라는 특별한 스킬을 제공하고 있습니다 <sup>67</sup> <sup>68</sup>. Skill-Creator 스킬을 활성화하면 Claude가 대화형으로 질문을 던지며 새로운 Skill의 뼈대를 만들어줍니다. 초보자라면 이 방법을 활용하여 보다 쉽게 Skill을 구축할 수 있습니다.

### Skill-Creator 사용 방법:

1. Settings > Skills 목록에서 **"skill-creator"** 스킬을 찾아 활성화합니다 <sup>67</sup>. (만약 목록에 없다면 Anthropic의 GitHub에서 `skill-creator` 폴더를 다운로드하여 업로드할 수도 있습니다 <sup>68</sup>.)
2. Claude와의 채팅에서, 새 Skill에 대해 알려줍니다. 예: "안녕 Claude, '프로젝트 상태 보고'라는 새로운 스킬을 만들고 싶어. 도와줄래?"
3. 그러면 Claude (Skill-Creator의 도움을 받아)가 다음과 같이 물어볼 것입니다:
4. "새로운 스킬의 목표는 무엇인가요? 어떤 상황에서 사용되나요?"
5. "이 스킬을 통해 Claude가 수행해야 할 작업은 구체적으로 무엇인가요?"
6. "스킬의 이름을 정하고 설명을 한 문장으로 적어볼까요?"
7. 사용자는 각각의 질문에 답변합니다. Claude는 답변을 기반으로 **name, description, 주요 지침** 등을 초안합니다. 예컨대:
8. 사용자: "프로젝트 상태 보고서를 자동으로 작성해주는 스킬이야. 매주 팀원들이 올리는 진행 상황을 요약하고 위험 요소를 하이라이트 해줘."
9. Claude: "좋습니다. 이 스킬의 이름은 `project-status-report` 가 적절해 보입니다. description은 '프로젝트 주간 진행 상황을 요약하고 위험 요소를 강조하여 보고하는 스킬'로 해볼게요." <sup>38</sup> <sup>39</sup>
10. Claude: "이제, 보고서 작성에 필요한 단계들을 알려주세요. 예를 들어 데이터를 수집하고, 요약하고, 형식화하는 과정이 있나요?"
11. 사용자: "1) Jira에서 지난 주 closed된 이슈 목록 정리, 2) 각 이슈의 상태 요약, 3) 지연된 작업이나 리스크 항목 나열."



12. Claude: "좋습니다. 그 정보를 반영해서 스킬 지침을 작성해볼게요..."
13. Claude는 이러한 Q&A를 거쳐 **SKILL.md**의 초안을 만들어냅니다. 여기에는 YAML 메타데이터(이름과 설명)와 Markdown 지침 구조가 포함됩니다. 예:

**project-status-report**

**description:** 프로젝트 주간 진행 상황을 요약하고 지연/위험 요소를 강조하여 보고하는 스킬...

## Overview

이 스킬은 프로젝트 관리 도구(Jira)에서 수집한 지난 주 작업 현황을 기반으로, 진행 상황 보고서를 자동 생성합니다...

## Steps

1. Jira에서 지난 주에 완료된 이슈 목록을 추출합니다...
2. 각 이슈의 상태와 완료 여부를 요약합니다...
3. 마감 기한이 지나거나 위험이 있는 항목을 별도로 표시합니다...

...(이하 생략)...

1. 사용자는 Claude가 제안한 내용을 검토하고, 잘못된 부분이나 추가하고 싶은 내용을 피드백합니다. "2단계에서 위험 항목은 Bold로 표시해줘" 같은 요청을 하면 Claude가 SKILL.md 초안을 수정합니다.
2. 초안이 만족스러우면 Claude에게 "**완성된 SKILL.md 파일을 보여줘**"라고 요청합니다. Claude는 최종 결과물을 Markdown 형태로 제공할 것입니다. 이를 복사하여 `.md` 파일로 저장하고, 폴더 구조에 맞게 준비한 후 ZIP으로 압축하여 업로드하면 Skill이 완성됩니다 <sup>68</sup> <sup>69</sup>.

이 대화형 접근 방식은 Skill 초안을 빠르게 얻는 데 매우 유용합니다. Claude가 제안한 내용은 일반적인 모범 사례를 반영하고 있으므로 <sup>68</sup>, 이를 토대로 약간의 수정만 거치면 되는 경우가 많습니다. 특히 **description** 문구 개선이나 **단계 구조화**에서 Claude의 도움을 받을 수 있어 편리합니다. 다만, Skill-Creator가 만들어 준 초안이라고 해도 직접 테스트하고 조율하는 과정은 여전히 필요합니다. Claude의 출력이 완벽하지 않을 수 있으므로, 결과물을 검토하여 실제 자신의 업무 맥락에 맞게 다듬은 후 업로드하세요.

또한 Skill-Creator는 Claude의 기본 능력을 활용하는 것이므로, 보안이나 품질 면에서 최종 책임은 사용자에게 있습니다. 민감한 내부 정보가 Skill 초안에 노출되지 않도록 하고, 필요한 경우 일부 내용은 수동으로 추가해야 할 수도 있습니다.

이 방법을 통해 **비개발자도** 비교적 쉽게 Skill 작성을 시작할 수 있습니다. Claude와 협업하여 Skill을 만들어나가는 과정 자체가 하나의 **프롬프트 엔지니어링 학습**이 되기도 합니다. 익숙해지면, 수동 편집 방식과 결합하여 더욱 정교한 Skill을 개발할 수 있을 것입니다.

## Skill의 호출과 실행 흐름

완성된 Skill을 Claude가 실제 대화에서 **어떻게 사용하는지** 그 흐름을 이해해봅시다. Skill 호출 과정은 다음과 같은 단계로 이뤄집니다 <sup>70</sup> <sup>71</sup>:

Claude에서 Skill을 탐색하고 실행하는 전체 흐름을 나타낸 도식화. 사용자의 요청을 받아 Claude가 가용한 Skill 목록에서 적절한 Skill을 식별하고, 해당 Skill의 내용을 로드하여 실행한 뒤, 결과를 응답에 반영하는 일련의 과정을 보여줌

니다. Claude의 내부 "Skill" 도구가 어떻게 스킬 목록에서 해당 스킬을 찾아 로드하고, 필요한 경우 코드를 실행하여 결과를 반환하는지를 시각화한 것입니다.

1. **사용자 요청** - 사람이 Claude에게 작업을 요청합니다 (예: "이번 분기의 마케팅 성과를 우리 회사 **브랜드 가이드**에 맞춰 PPT로 만들어줘.").
2. **Skill 검사 및 선택** - Claude는 현재 활성화되어 있는 모든 Skill의 `name`과 `description` (메타데이터)을 훑어보며, 사용자 요청과 **의미적으로 관련 있는 Skill**이 있는지 판단합니다 <sup>72</sup>. 이 단계는 단순 키워드 매칭이 아니라 Claude의 언어 모델을 통해 이루어지며, 요청의 의도와 Skill 설명 간의 **포괄적인 유사성**을 고려합니다 <sup>73</sup> <sup>74</sup>. 관련 Skill이 **없다**고 판단되면 Claude는 내장 지식과 일반 능력만으로 질문에 답합니다. **관련 Skill이 발견되면**, Claude는 그 Skill을 사용하기로 **내부 결정**을 내립니다.
3. **Skill 로드** - Claude가 특정 Skill을 사용하기로 결정하면, 해당 Skill 폴더의 내용을 불러옵니다. 먼저 `SKILL.md`의 전체 내용을 읽어 현재 대화 컨텍스트에 추가합니다 <sup>75</sup>. 만약 `SKILL.md` 안에 추가 참조 파일 링크가 있거나 실행해야 할 코드가 언급되어 있다면 Claude는 필요에 따라 그 파일들도 열람하거나 실행합니다 <sup>8</sup>. (이 때 Claude의 도구 중 하나인 **"Skill" 메타-도구**가 이러한 파일 로드 작업을 수행합니다 <sup>76</sup>.)
4. **실행 및 문제 해결** - Skill에 코드 실행이 포함된 경우 Claude는 안전한 샌드박스 환경에서 그 코드를 실행합니다 <sup>71</sup>. 예를 들어 Skill에 Python 스크립트가 있고 본문 지침에 따라 그 스크립트를 실행해야 한다면, Claude는 `Python` 도구를 사용해 해당 코드를 수행하고 결과를 얻습니다. 코드 실행 중에 파일을 읽거나 쓰거나, 또는 MCP(외부 연동)가 필요한 경우, Claude는 Skill 지침에 따라 이러한 작업들을 순차적으로 수행하며 **문제 해결 과정**을 진행합니다.
5. **응답 생성** - Skill 지침대로 모든 작업(문서 생성, 데이터 처리 등)이 완료되면, Claude는 그 결과물을 토대로 사용자에게 줄 최종 **답변**을 생성합니다. 이 답변 내용에는 Skill을 통해 얻은 정보나 실행 결과가 반영됩니다. 또한 Claude는 투명성을 위해 응답 중에 **해당 Skill을 사용했음을 언급**할 수 있습니다 <sup>77</sup>. 예를 들어 Claude가 "Brand Guidelines" Skill을 활용했다면 답변 앞부분에 "... (Brand Guidelines 스킬을 사용하여 작업했습니다)"와 같은 문구가 추가될 수 있습니다. 이를 통해 사용자는 Claude가 특별한 워크플로우를 적용했음을 알 수 있습니다.
6. **검토 및 로깅** - Claude의 응답이 완료되면, 해당 Skill 사용에 대한 정보가 **로그**로 남습니다. 팀이나 조직 관리자는 Claude Console 등을 통해 어떤 Skill이 언제 사용되었고, 어떤 코드가 실행되었는지 **감사(audit)**할 수 있습니다 <sup>71</sup>. 이는 기업 환경에서 AI 사용에 대한 투명성을 확보하는 데 도움이 됩니다.

이 과정은 대부분 **자동으로** 이루어지며, 사용자는 평소와 다름없이 자연어로 요청을 입력하면 됩니다. 단, Skill이 잘 작동하는지 확인하기 위해 사용자가 Skill 이름을 직접 언급하여 강제로 Skill을 사용하도록 유도할 수도 있습니다 (예: "내 **브랜드 가이드라인** 스킬을 사용해서 자료를 만들어줘"). Claude는 기본적으로 자율적으로 Skill 사용 여부를 판단하지만, 이런 직접적인 지시도 따라주므로 디버깅 시 유용합니다 <sup>78</sup> <sup>79</sup>.

또한 한 번에 **여러 Skill이 연쇄적으로 사용**될 수도 있습니다. 예를 들어 요청이 복잡해서 두 개 이상의 Skill이 각기 일부를 도울 수 있다면, Claude는 여러 Skill의 설명이 모두 관련 있다고 판단하고 **복수의 Skill을 순차적으로 로드**하여 작업할 수 있습니다 <sup>5</sup> <sup>43</sup>. Claude는 각 Skill의 역할을 조정하여 마치 전문팀이 협업하듯 문제를 해결합니다. 다만, 지나치게 범용적인 Skill 설명을 작성하면 엉뚱한 상황에서 Skill이 발동되거나 충돌할 수 있으므로, **Skill 간 역할이 겹치지 않도록 명확히 구분**하는 것이 중요합니다.

## 테스트와 디버깅: Skill 동작 확인 및 문제 해결

Skill을 업로드하고 활성화한 다음에는, 의도대로 작동하는지 **테스트**해야 합니다. 테스트는 다양한 각도에서 이루어져야 하며, 일반적으로 다음과 같은 시나리오를 검토합니다:

1. **정상 시나리오 테스트** - Skill이 염두에 둔 **대표적인 요청**으로 Claude를 실행해 봅니다. 예를 들어 금융 보고서 생성 Skill이라면, "마이크로소프트의 최신 분기 실적을 분석해 보고서를 만들어줘" 같은 **전형적인 요청**을 입력해 보는 것입니다 <sup>80</sup>. 기대한 대로 Skill이 발동되고, 지침서에 따라 Claude가 결과물을 산출하는지 확인합니다 <sup>81</sup>. 이때 Claude의 응답에 Skill 이름 언급이 나오거나, 출력 형식이 예시와 일치하는지 등을 살펴봅니다.

2. **엣지 케이스 테스트** - 비정상적이거나 **예외적인 상황**에서 Skill이 어떻게 반응하는지도 실험해야 합니다. 예를 들어 입력 데이터가 일부 누락되었거나 포맷이 예상과 다를 때 Claude가 어떤 답을 내놓는지 확인합니다 <sup>82</sup>. Skill 지침에서 이런 경우에 대비한 안내를 넣었다면, Claude가 그에 따라 "추가 정보가 필요합니다" 또는 "지원하지 않는 형식입니다" 등의 응답을 하는지 살펴봅니다.
3. **범위 벗어난 요청 테스트** - Skill이 **의도하지 않은 요청**에 대해 불필요하게 활성화되지 않는지도 중요합니다 <sup>83</sup>. Skill과 유사하지만 실제로는 다른 맥락의 질문을 던져보세요. 예를 들어 NDA(비밀 유지 계약) 검토 Skill을 만들었다면, "이 임대차 계약서 좀 검토해줘" 같은 요청에는 **Skill이 개입하지 않고 일반적인 Claude 능력으로 답변**하는 게 바람직합니다 <sup>83</sup>. 이러한 테스트를 통해 Skill의 **description이 충분히 특정적인지**, 또는 너무 광범위해서 오작동하지는 않는지 점검할 수 있습니다.

테스트를 진행하면서 **Claude의 반응을 면밀히 관찰**하세요. Claude가 Skill을 사용한 경우 응답 내용이나 내부 로그(Claude's thinking)에 Skill 로드 관련 언급이 나타납니다 <sup>84</sup>. Claude가 Skill을 사용하지 않아야 할 상황인데도 사용했다면 description을 더 제한적으로 바꾸어야 합니다. 반대로 사용해야 하는데도 사용하지 않는다면 description을 보완하여 해당 요청 키워드를 포함시키거나, Skill의 우선순위를 높일 수 있는 어휘를 추가해야 합니다.

추가로 고려할 테스트들:

- **트리거 테스트** - Skill이 **예상한 상황에서만 발동되고 다른 상황에서는 조용히 대기하는지** 확인합니다 <sup>85</sup>. 예를 들어 금융 데이터 분석 Skill을 만들었다면, "이 회사의 재무제표 분석해줘" (Skill 활용 기대)와 "이 회사 경영진 소개해줘" (Skill 비활용 기대)처럼 유사하지만 범위가 다른 요청들을 모두 시도해 봅니다. Skill이 불필요할 때는 개입하지 않고 넘어가는지 살펴보세요 <sup>85</sup> <sup>86</sup>.
- **일관성 테스트** - 동일하거나 유사한 입력에 대해 Skill이 **일관된 출력**을 내는지 확인합니다. 랜덤성 없이 결정적인 결과가 필요한 Skill(예: 포맷 변환)은 여러 번 실행해보아도 결과가 같아야 합니다. 만약 매번 출력이 다르다면 지침을 더 구체화하거나 예시를 추가해 일관성을 높일 필요가 있습니다.
- **사용성 테스트** - Skill을 전혀 모르는 사용자가 해당 Skill을 염두에 두지 않고 평범하게 질문해도 의도한 대로 동작하는지 확인합니다. Skill의 description이 충분히 직관적이라면, 사용자는 Skill의 존재를 몰라도 자연스럽게 그 혜택을 누리게 됩니다. 반대로 Skill이 작동하지 않는다면 description이나 triggers가 너무 협소한지 점검하세요.
- **문서/예시 정확성 검증** - SKILL.md에 포함한 사용 예시나 출력 예시가 실제 Claude의 동작과 맞는지 검증합니다. 예시에 어긋나는 출력이 나올 경우, Skill 지침을 업데이트하여 설명과 실제 동작을 일치시킵니다.

테스트 과정에서 발견된 개선점은 곧바로 Skill에 반영해 **반복(iteration)**하세요 <sup>87</sup>. description을 다듬거나, 지침 단계를 수정하거나, 추가 예외 처리를 넣는 등 지속적인 개선을 통해 Skill의 신뢰성을 높일 수 있습니다 <sup>88</sup>. 여러 번의 실전 테스트를 거치면서 Skill은 점점 안정화되고, 다양한 상황에서도 예측 가능한 도구로 자리잡게 될 것입니다.

## 문제 상황별 디버깅 팁

- **Skill이 발동하지 않을 때:** 우선 **Settings에서 Skill이 활성화(ON) 상태인지** 확인합니다 <sup>89</sup>. 활성화되어 있다면, 사용자의 요청이 Skill description과 충분히 일치하는지 검토합니다. description이 너무 모호하면 Claude가 관련성을 인지하지 못할 수 있습니다 <sup>43</sup>. 또한 SKILL.md의 내용이 복잡하거나 비정형적이면 Claude가 제대로 이해하지 못할 수 있으므로, **설명을 더 명확하고 간결하게** 다듬어 보세요 <sup>90</sup>. 그래도 안 된다면, 사용자 프롬프트에 Skill 이름을 직접 언급하여 ("... 스킬을 사용해 ...") Skill 호출을 유도해보면 단서를 얻을 수 있습니다 <sup>79</sup>. Claude가 이에 반응한다면, description에 그 표현을 포함시키거나 더 명확한 키워드를 추가하는 등 트리거 조건을 조정하세요.
- **Skill이 불필요한 상황에서 작동할 때:** 이는 description이 **지나치게 포괄적**이어서 관계 없는 요청에도 매치되는 경우입니다 <sup>44</sup>. 이런 경우 description에 **"Not for ..."** (어떤 경우에는 사용하지 않음)과 같은 부정 조건이나 예외를 명시하는 것을 고려하세요 <sup>40</sup>. 혹은 트리거 키워드를 좀 더 구체적으로 포함시켜 범위를 좁히세요. 여러 Skill을 운용 중이라면, 서로 description이 겹쳐 충돌하는지 검토해보고 조정해야 합니다.
- **업로드 오류 발생 시:** Claude 웹에 Skill ZIP을 업로드할 때 에러가 난다면, 오류 메시지를 확인합니다. 일반적으로 ZIP 파일 구조 불일치, 파일 용량 초과, name/description 필드 형식 오류 등이 원인입니다 <sup>35</sup>. 앞서

"Skill 패키징" 부분을 다시 한번 점검하고, 폴더 구조와 파일명을 바로잡은 뒤 재압축하여 시도합니다. (특히 ZIP 안에 SKILL.md가 빠져있거나 폴더 없이 파일만 있으면 인식이 안 됩니다.)

- **코드 실행 오류 시:** Skill 내 스크립트가 실행되다가 에러가 발생한 경우, Claude의 응답에 오류 메시지가 포함되기도 합니다. (예: "NameError: 'X' is not defined" 등) Claude가 오류를 자체적으로 요약해 말해줄 수도 있으므로 응답을 잘 확인하세요. 필요하다면 Claude에게 "어떤 오류가 발생했는지 자세히 알려줘"라고 물어볼 수도 있습니다. 오류 원인을 파악했으면, 코드를 수정하거나 예외 처리를 추가한 뒤 Skill을 재업로드합니다. (개발 단계에서 작은 수정이 잦다면, 버전 번호를 올려가며 업로드하면 추적에 도움이 됩니다.)
- **결과가 기대와 다를 때:** Claude의 출력이 SKILL.md의 의도와 다르다면, 지침의 표현을 더욱 구체화해야 합니다. 예를 들어 일정 요약 Skill이 너무 장황하게 답한다면 "한 문단으로 요약"이라는 지시를 추가합니다. 또는 일부 항목이 빠진다면 해당 항목을 언급했는지 SKILL.md를 점검합니다. 필요한 경우 **예시**를 추가하여 Claude의 출력 스타일을 교정합니다.
- **Skill이 회색으로 비활성화됨:** 만약 Settings > Skills 목록에서 Skill 이름이 회색으로 표시되고 토글할 수 없는 경우, 이는 현재 계정이나 조직에서 Skills 기능이 완전히 꺼져 있음을 의미합니다 <sup>91</sup>. Team/Enterprise라면 관리자가 조직에 대해 Skills 사용을 허용했는지 확인하고, 개인 계정 설정에서 Code execution이 켜져 있는지 살펴보세요.
- **보안 경고 발생 시:** 드물지만 Skill의 내용이나 코드가 Anthropic의 사용 정책에 저촉될 경우 Claude가 경고를 표시하거나 실행을 거부할 수 있습니다. 이럴 땐 해당 내용을 수정하여 정책을 준수해야 합니다. 예를 들어 과도한 개인 정보가 하드코딩되어 있거나, 외부 통신 시도가 문제가 될 수 있습니다. Skill을 재검토하여 안전하게 고치세요.

위와 같은 과정을 거쳐 Skill을 지속적으로 개선하면, 결국 실사용 환경에서 신뢰성 있게 돌아가는 **안정된 Skill**을 얻을 수 있습니다. Skill 개발은 한번에 끝내기보다는, 실제 사용 피드백을 반영하여 **점진적으로 다듬어가는 과정**임을 염두에 두세요 <sup>87</sup>.

## 모범 사례 및 팁

마지막으로, Claude Skill을 설계하고 운영할 때 염두에 둘 몇 가지 **모범 사례**를 정리합니다:

- **단일 책임 원칙:** 하나의 Skill에는 **하나의 구체적인 기능이나 워크플로우**만 담습니다. 여러 업무를 한 Skill에 욕여넣기보다는, 용도별로 Skill을 나누는 것이 좋습니다 <sup>92</sup> <sup>93</sup>. 이렇게 하면 필요할 때 해당 Skill만 불러와 사용할 수 있고, 유지보수도 쉬워집니다.
- **명확한 트리거 설명:** Skill의 description에는 **어떤 상황에서 사용해야 하는지**를 분명히 적습니다 <sup>90</sup>. "언제, 무엇을 위해 이 Skill을 쓰라"는 식으로 서술하면 Claude가 정확한 맥락에서 Skill을 불러옵니다. 모호한 설명은 피하고, 특정 키워드나 개념을 포함하세요 (필요시 "이럴 때는 사용, 저럴 때는 사용 금지" 형태로).
- **컨텍스트 최소화:** Skill에 지나치게 방대한 내용을 한꺼번에 넣지 마세요. **Progressive Disclosure(점진적 노출)** 원칙을 활용하여, 필수 핵심 지침은 SKILL.md에 넣되 상세 참조는 별도 파일로 분리합니다 <sup>8</sup>. Claude는 필요한 순간에만 추가 파일을 읽으므로, 이를 통해 컨텍스트 윈도우를 효율적으로 사용하고 성능 저하를 막을 수 있습니다.
- **코드 사용은 신중하게:** Skill에 코드를 포함하면 강력한 기능을 구현할 수 있지만, 항상 **보안**을 고려해야 합니다 <sup>57</sup>. 외부 API 키나 민감 정보를 넣지 말고, 서드파티 라이브러리를 사용할 때는 신뢰할 수 있는 것인지 검토하세요. 또한 `allowed-tools` 등을 활용해 Claude가 불필요한 도구를 실행하지 않게 제어할 수 있습니다.
- **작게 시작하여 개선 반복:** 처음부터 완벽한 Skill을 만들려고 하기보다, **기본 버전을 빨리 만들어 테스트**해보는 것이 좋습니다 <sup>94</sup>. 작은 성공 케이스부터 확인하고, 점진적으로 Skill을 확장하거나 세부 조정합니다. 테스트 → 피드백 → 수정 과정을 여러 번 돌면서 Skill의 품질을 높여가세요 <sup>87</sup>.
- **예시로 학습 효과 극대화:** 가능하면 SKILL.md에 **예시 입력과 출력**을 넣어 Claude에게 기대 행동을 보여주세요 <sup>95</sup>. 예시는 일종의 few-shot 학습 역할을 하여, 모델이 당신이 원하는 포맷이나 스타일을 더 잘 따르게 해줍니다.
- **Skill 간 충돌 주의:** 여러 개의 Skill을 운영할 경우, 서로 description이 겹쳐 Claude가 둘 이상을 혼동하거나 과도하게 로드하지 않도록 합니다. 각 Skill의 역할을 명확히 구분하고, 이름도 고유하게 짓습니다. 혹시 유사한 영역이라면 하나의 Skill로 통합하거나, description에 의도적으로 차이를 두어 구별하세요.

- **팀 협업과 공유:** 조직에서 Skill을 활용한다면, Git 저장소나 내부 포털을 통해 Skill 파일을 **공유 및 버전 관리**하세요 <sup>96</sup> <sup>97</sup>. Claude Code 환경에서는 `.claude/skills/` 폴더를 프로젝트에 포함해 팀원이 자동으로 Skill을 가져가게 할 수 있습니다 <sup>96</sup> <sup>97</sup>. 웹 환경에서는 각자가 ZIP을 업로드해야 하므로, 최신 버전 배포 시 모두에게 공지하여 동기화하는 방식으로 운영합니다.
- **보안 및 정책 준수:** Anthropic Claude는 사용 정책에 어긋나는 내용이 Skill에 포함될 경우 해당 Skill 실행을 차단할 수 있습니다. 예를 들어 차별적이거나 부적절한 지침은 금지됩니다. Skill을 만들 때 **Anthropic 사용 정책**을 숙지하고 준수하세요 <sup>57</sup>. 또한 Skill의 코드 실행 기능을 남용하면 예기치 않은 결과가 발생할 수 있으므로, 항상 필요한 최소한으로 사용하고, 검증된 코드만 포함합니다.

이상의 모범 사례를 따르면 Claude Skills를 더욱 효과적이고 안전하게 활용할 수 있습니다. 마지막으로, Skill 개발은 단발성이 아니라 **지속적인 관리**가 필요한 작업입니다. AI 모델과 환경의 업데이트에 따라 Skill을 주기적으로 점검하고 개선함으로써, 항상 최상의 성능을 유지하도록 합니다.

## 예제 1: 일정 관리 비서 Skill

**사용 사례:** 임직원의 Google/Outlook 캘린더 등의 일정을 확인하고 요약하는 **일정 관리 비서** Skill을 만들어보겠습니다. 이 Skill은 사용자가 "내일 일정 알려줘" 또는 "다음 주 회의 일정 알려줘"처럼 **개인 캘린더 일정을** 묻는 요청을 하면, 캘린더에서 해당 기간의 이벤트들을 읽어와 정리된 목록을 제공하는 역할을 합니다. (이 예제에서는 macOS의 기본 캘린더 데이터를 불러오는 시나리오를 가정합니다.)

**예상 폴더 구조:**

```
calendar-assistant/
├── SKILL.md
└── scripts/
    └── get_events.sh
```

이 Skill에서는 `scripts/get_events.sh` 스크립트를 통해 캘린더 이벤트를 추출한다고 가정합니다. (macOS에서는 **icalBuddy**라는 커맨드라인 도구로 캘린더 이벤트를 얻을 수 있습니다.)

**YAML 매니페스트 예시 (SKILL.md 상단):**

```
name: calendar-assistant
description: "사용자의 캘린더에서 특정 날짜 범위의 일정을 읽어 요약해주는 스킬. 사용자가 자신의 오늘/내일/이번 주 일정이나 다음 회의 등을 물어볼 때 사용. 이벤트의 시간, 장소, 참석자를 포함한 일정을 목록으로 반환."
dependencies: ical-buddy
allowed-tools: ["Bash"]
```

위에서 `dependencies`에 `ical-buddy`를 명시해 두면, Claude Code 환경 등에서 해당 패키지를 사전에 설치해 둘 수 있습니다. `allowed-tools: ["Bash"]`는 이 Skill 실행 시 Bash 셸 명령 사용만 허용하고 Python 등은 쓰지 않도록 제한한 것입니다 (일정 추출은 Bash를 통해 icalBuddy로 할 것이므로 Python 불필요).

**SKILL.md 본문 구성 예시:**

```
## When to Use
- "내 일정 보여줘"
- "오늘/내일/이번주 일정 어때?"
```

- "다음 미팅 언제야?"

## ## Overview

이 스킬은 사용자의 macOS 캘린더에서 일정 정보를 추출해 요약합니다. 주어진 날짜 범위를 지정된 포맷으로 조회하여, 각 이벤트의 시간, 제목, 위치를 정리해 제공합니다.

## ## Steps

1. **\*\*Retrieve Events\*\***: 지정된 날짜 범위의 일정을 가져옵니다 (macOS의 경우 `icalBuddy` 도구 활용). 예를 들어 오늘 날짜의 일정을 가져오려면:

```
```bash
icalBuddy -n -iep "title,datetime,location" -df "%Y-%m-%d" -tf "%H:%M" eventsFrom:
2025-12-10 to:2025-12-10
```
```

위 명령은 2025-12-10의 이벤트를 '제목, 날짜시간, 위치' 속성만 추려서 출력합니다 <sup>98</sup>. icalBuddy를 사용할 수 없는 환경이라면 `get\_events.sh` 스크립트를 통해 동일한 기능을 구현합니다.

2. **\*\*Parse & Format\*\***: 가져온 이벤트를 날짜별로 그룹화하고, 보기 좋은 형식으로 정리합니다. 예를 들어:

- 날짜별로 헤더를 달아 구분 (예: "### 2025-12-10 (화)").
- 각 이벤트 항목에 시간, 행사명, 위치를 표시.
- 오전/오후 또는 시간 순서대로 정렬.

3. **\*\*Present Results\*\***: 추출한 일정을 사용자가 이해하기 쉽게 출력합니다. 예를 들면:

```
```
2025-12-10 (화)
- 09:00 팀 미팅 @ 회의실 A (참석자: 홍길동, 김철수)
- 14:00 프로젝트 Kick-off @ Zoom (참석자: 전체 팀)
```
```

만약 해당 날짜에 이벤트가 없다면 "일정 없음"이라고 표시합니다. 여러 날의 범위를 요청한 경우 각 날짜별로 위와 같은 목록을 반복하여 보여줍니다.

## ## Error Handling

- 캘린더 데이터에 접근할 수 없으면 "캘린더 정보를 불러올 수 없습니다. 권한을 확인해주세요."라고 답변합니다.
- 요청에 날짜 범위가 명시되지 않은 경우 기본으로 오늘 날짜를 기준으로 동작합니다.

위와 같이 작성해두면, Claude는 사용자가 "이번 주 내 일정 알려줘"라고 물었을 때 이 Skill을 자동으로 불러와서, macOS 시스템에서 해당 기간의 캘린더 이벤트를 추출한 후 요약 결과를 알려줄 것입니다. 실제 실행에서는 Bash 도구로 icalBuddy를 호출하여 데이터를 얻고, Markdown 형식의 목록으로 답변을 구성하게 됩니다 <sup>99</sup> <sup>100</sup>.

### 실행 예시:

- **사용자**: "이번 주 내 일정 알려줘."
- **Claude**: (calendar-assistant 스킬을 사용하여)

```
2025-12-08 (월)
- 10:00 팀 주간 회의 @ 본사 3층 회의실
- 15:00 프로젝트 Kick-off @ Zoom

2025-12-09 (화)
- 일정 없음
```

2025-12-10 (수)

- 14:00 고객 미팅 @ 온라인 (참석자: 김지원, 피터)

"이번 주 중요한 미팅은 수요일 14시에 예정되어 있습니다. 그 외에는 비교적 한가한 편입니다."

Claude는 이처럼 주중 일정을 날짜별로 정리하여 표시해주었습니다. 월요일과 수요일에 이벤트가 있으며, 화요일 등 다른 날은 "일정 없음"으로 나타냅니다. 마지막에는 요약 코멘트를 덧붙여 사용자에게 중요한 포인트를 알려주었습니다. 이는 calendar-assistant Skill의 지침에 따른 형식입니다.

## 예제 2: 파일 요약 Skill

**사용 사례:** 사용자가 긴 텍스트 파일이나 문서를 업로드하고 "이 파일 요약해줘"라고 요청하는 상황을 가정해봅시다. 이를 위해 **파일 요약 Skill**을 만들어, Claude가 해당 파일의 내용을 읽고 핵심을 정리해 반환하도록 할 수 있습니다. 이 Skill은 리포트, 기사, 로그 파일 등 다양한 텍스트 기반 파일의 요점을 신속히 파악하는 데 활용될 수 있습니다.

예상 폴더 구조:

```
file-summarizer/  
└─ SKILL.md
```

(이 Skill은 별도의 코드나 리소스 없이 순수한 지침만으로 구성한다고 가정합니다.)

YAML 매니페스트 예시:

```
name: file-summarizer  
description: "긴 문서나 텍스트 파일의 핵심 내용을 추려주는 요약 Skill. 사용자가 파일을 첨부하거나 텍스트를 주  
면서 요약을 요청할 때 사용. 문서의 주요 주장, 중요한 데이터 포인트 등을 간결히 정리하여 제공."
```

이 Skill의 description은 "**파일 첨부 + 요약 요청**" 시나리오에 초점을 맞추고 있습니다. 사용자가 "다음 텍스트를 요약해줘:"라며 본문을 주거나, PDF/텍스트 파일을 업로드하고 요약을 부탁하면 Claude가 이 Skill을 고려하게 됩니다.

SKILL.md 본문 구성 예시:

### ## When to Use

- "요약"이라는 단어가 포함된 요청이며, 큰 텍스트 블록이나 파일이 주어졌을 때
- 사용자가 파일을 업로드하고 그 내용을 요약해 달라고 할 때 (예: "첨부된 보고서 요약좀")

### ## Overview

이 Skill은 문서의 길이나 형식에 관계없이 주요 요점을 파악하여 간결한 요약을 제공합니다. 보고서, 기사, 기술 문서 등 어떤 텍스트든 핵심만 추립니다.

### ## Instructions

1. **\*\*Read the entire content\*\***: 파일이나 텍스트 전체를 처음부터 끝까지 **\*\*꼼꼼하게 읽습니다\*\***. 중요한 부분을 놓치지 않도록 모든 문장을 분석하세요. (Claude의 **`Read`** 도구를 사용하여 첨부 파일 내용을 열람할 수 있습니다.)
2. **\*\*Identify key points\*\***: 문서의 주제, 저자가 전달하려는 주요 메시지, 데이터/사실 중 중요한 것들을 추출합니다. 특히 처음과 끝 부분(서론/결론)에 요약이 담겨있을 가능성이 높으니 유심히 봅니다.

3. **\*\*Produce a concise summary\*\***: 문체는 간결하고 명확하게 유지합니다. 필요한 경우 **\*\*bullet 포인트 목록\*\***으로 정리하거나, 한 두 문단의 요약본을 작성합니다 (요청에 따라 포맷 선택). 예를 들어:
  - 보고서인 경우: **"\*\*보고서 요약\*\*"**: ...로 시작하는 한 단락의 요약.
  - 기술 문서인 경우: 주요 섹션별로 한 줄 요약.
  - 대화 로그인 경우: 대화의 결론이나 결정 사항 중심으로 요약.
4. **\*\*Preserve important terms\*\***: 원문에 등장하는 중요한 용어, 고유명사, 숫자 데이터는 가능한 요약에 포함합니다. 단, 지나치게 상세한 수치는 맥락을 해치지 않는 선에서 간략화하거나 생략합니다.
5. **\*\*Review for completeness\*\***: 요약에 빠진 핵심 내용이 없는지 스스로 검토합니다. 질문 형태로 주어진 요청이라면, 그 질문에 대한 답이 요약에 포함되었는지도 확인하세요.

### ## Examples

- (예시 입력) **"다음 기사 내용을 요약해줘."** [이후 수백 자의 기사 본문]  
 (예시 출력) 기사 주요 내용: 1) 최근 경제 성장률이 예상치를 상회함, 2) 전문가들은 내년 전망을 긍정적으로 봄, 3) 위험 요소로 인플레이션을 지목.
- (예시 입력) **"첨부된 PDF 보고서 요약 부탁해"** (첨부: `Q3\_report.pdf`)  
 (예시 출력) 보고서 요약: 분기 매출 15% 상승, 신제품 라인 호조, 향후 투자 확대 계획. 세부적으로 A상품군 매출 증가가 두드러짐.

위와 같이 구성된 Skill은 사용자가 큰 파일을 제시하며 요약을 요구할 때 자동으로 활성화될 것입니다. Claude는 우선 **Read** 도구 등을 사용해 파일 내용을 입수하고, 모든 텍스트를 분석한 뒤, **간결하고 체계적인 요약**을 생성해줍니다. 이 Skill을 통해 사용자들은 장문의 문서를 일일이 읽지 않고도 핵심만 빠르게 파악할 수 있습니다.

### 실행 예시:

- **사용자**: (긴 텍스트 파일을 업로드하며) "이 문서를 3문장으로 요약해줘."
- **Claude**: (file-summarizer 스킬을 사용하여) "이 문서는 최근 시장 동향을 분석한 보고서입니다. 주요 내용은 다음과 같습니다:
  1. 시장 성장률이 지난 분기 대비 5% 상승했습니다.
  2. 기술 부문이 성장을 주도하였으며, 소비자 수요가 꾸준히 증가하고 있습니다.
  3. 향후 인플레이션이 잠재적 위험 요인으로 지목되었습니다."

Claude는 위와 같이 file-summarizer 스킬의 지침에 따라 문서를 읽고 핵심만 간추려 응답합니다. 사용자가 요청한 **3 문장 요약** 형식을 인지하여, 내용을 번호 목록으로 정리해주었습니다. 이제 사용자는 이 짧은 답변만 읽고도 문서의 핵심을 파악할 수 있게 됩니다.

## 예제 3: 엑셀 보고서 자동 생성 Skill

**사용 사례**: 경영진이 매주 특정 형식의 Excel 보고서를 요구한다면, 이를 자동으로 생성해주는 **엑셀 보고서 생성 Skill**을 만들어볼 수 있습니다. 예를 들어 매출 데이터가 주어지면, 그에 대한 요약 표와 차트를 포함한 Excel 파일을 만들어주는 것입니다. 사용자가 "이 데이터로 대시보드 엑셀 만들어줘"라고 요청하면, Claude가 이 Skill을 활용해 **프로그래밍 방식**으로 Excel 파일을 생성하고, 결과물을 첨부하는 흐름입니다.

### 예상 폴더 구조:

```
excel-report-skill/
├── SKILL.md
├── scripts/
│   └── generate_report.py
```



이 Skill은 파이썬을 이용해 Excel 파일을 생성하므로, 핵심 로직을 `scripts/generate_report.py`에 구현했다고 가정합니다.

#### YAML 매니페스트 예시:

```
name: excel-report
description: "제공된 데이터로 요약 표와 차트를 포함한 Excel 보고서를 자동 생성하는 스킬. 사용자가 테이블 데이터나 통계치를 주고 엑셀 보고서를 요청할 때 사용."
dependencies: python>=3.8, pandas>=1.5.0, openpyxl>=3.1.0
allowed-tools: ["Python"]
```

- 여기서 `dependencies`에는 Python과 데이터 처리를 위한 라이브러리(pandas) 및 Excel 파일 작성을 위한 라이브러리(openpyxl)를 지정했습니다. Claude의 코드 실행 환경은 이 목록을 보고 필요 시 해당 패키지를 설치합니다  
46. - `allowed-tools: ["Python"]`으로 설정하여, 이 Skill 실행 중에는 Python 스크립트만 사용하도록 제한했습니다. (Excel 생성은 Python으로 충분하므로 Bash 등 다른 도구는 불필요합니다.)

#### SKILL.md 본문 구성 예시:

##### ## When to Use

- 사용자가 "엑셀 보고서", "스프레드시트 생성" 등을 언급하며 데이터 제공
- 테이블 형식 데이터 요약 및 차트 시각화를 요구할 때

##### ## Overview

이 스킬은 입력된 데이터에 기반하여 자동으로 Excel 형식 보고서를 생성합니다. 보고서에는 데이터 요약 표와 간단한 시각화(예: 막대 차트)가 포함됩니다.

##### ## Instructions

1. **\*\*Prepare Data\*\***: 사용자가 업로드한 데이터 파일(CSV 등)을 읽거나, 대화에 포함된 표를 파싱합니다. Python pandas 라이브러리를 사용해 데이터를 DataFrame으로 불러옵니다.
2. **\*\*Generate Summary\*\***: 주요 지표를 계산합니다. (예: 합계, 평균, 증감율 등) 필요한 경우 새로운 요약 테이블을 만듭니다.
3. **\*\*Create Excel Workbook\*\***: `openpyxl` 등을 사용해 새로운 Excel 워크북을 생성합니다. 첫 번째 시트에 원본 데이터 또는 요약 데이터를 표 형태로 작성합니다. 두 번째 시트에는 `openpyxl`의 차트 기능을 이용해 간단한 차트를 추가합니다.

```
```python
from openpyxl import Workbook
from openpyxl.chart import BarChart, Reference

wb = Workbook()
ws1 = wb.active
ws1.title = "Summary"
# (여기에 pandas DataFrame 내용을 ws1에 써넣는 코드)
chart = BarChart()
data = Reference(ws1, min_col=2, min_row=1, max_col=2, max_row=5)
chart.add_data(data, titles_from_data=True)
ws1.add_chart(chart, "E2")
wb.save("report.xlsx")
```
```

위 코드는 단순 예시로, 워크북 생성 후 요약 데이터와 차트를 추가하는 모습을 보여줍니다.

4. **\*\*Attach Excel File\*\***: 생성된 `report.xlsx` 파일을 Claude의 **\*\*파일 출력(artifact)\*\*** 기능을 통해 사용자에게 제공합니다. Claude는 "엑셀 보고서를 생성했어요:"와 함께 파일을 첨부할 것입니다.
5. **\*\*Explain Results\*\***: 최종 응답에는 보고서에 포함된 주요 결과를 간략히 설명하여 첨부된 파일의 내용을 요약합니다. 예를 들어 "보고서가 생성되었습니다. 총 매출 합계는 1.2억 원이며, 전년 동기 대비 10% 상승한 수치입니다. 상세 내역과 그래프는 첨부된 Excel 파일을 확인하세요."라고 답변합니다.

### ## Error Handling

- 데이터 형식이 올바르지 않거나 누락된 경우, "제공된 데이터에서 표를 생성할 수 없습니다. 형식을 확인해주세요."라는 메시지를 반환하고 Skill 실행을 중단합니다.
- Python 스크립트 실행 중 오류가 발생하면 해당 오류를 캐치하여 Claude가 사용자에게 이해하기 쉽게 설명하도록 합니다. (예: "보고서 생성 중 오류가 발생했습니다: 데이터 범위가 유효하지 않습니다.")

이 Skill을 사용하면 반복적인 Excel 보고서 작성 업무를 자동화할 수 있습니다. 예를 들어, 매주 영업팀 실적을 취합해 동일한 양식의 보고서를 만들어야 하는 경우, 이제 Claude에게 CSV만 주고 "보고서 만들어줘"라고 말하면 곧바로 완성된 Excel을 받을 수 있습니다. 실제 Anthropic에서 제공하는 Excel 생성 Skill도 이와 비슷한 원리로 동작하며, Claude는 해당 Skill을 통해 복잡한 수식이나 서식도 수작업 없이 만들어냅니다 <sup>101</sup>.

### 실행 예시:

- **사용자**: (CSV 데이터 파일을 업로드하며) "이 자료로 분기별 매출 보고 엑셀 만들어줘."
- **Claude**: (excel-report 스킬을 사용하여) "분석한 결과, 보고서를 생성했습니다. 주요 내용은 Q1부터 Q4까지 매출이 꾸준히 증가하여 **연간 총 매출이 15% 상승**했다는 점입니다. 상세 내역과 그래프는 첨부된 Excel 파일에서 확인하실 수 있습니다." (첨부: `report.xlsx`)

Claude는 excel-report Skill을 활용하여 Python으로 엑셀 파일을 만들고, 그 파일을 답변에 첨부했습니다. 또한 응답 텍스트에서 주요 결과(연간 매출 15% 상승)를 언급하여 사용자가 파일을 열지 않고도 핵심을 알 수 있게 배려했습니다. 이러한 처리 방식은 SKILL.md의 지침 "최종 답변에 주요 지표를 설명하라"는 내용을 따른 것입니다.

## Skill 업데이트와 버전 관리

Skill을 만들고 난 이후에도, **업데이트와 버전 관리**가 필요합니다. 업무 절차가 바뀌거나 새로운 사례를 반영해야 한다면 Skill 내용을 수정해야 할 것입니다. Claude 웹 인터페이스에서는 Skill을 수정했을 경우 **새로운 ZIP 파일로 다시 업로드**하는 방식으로 업데이트를 적용합니다. 이때 몇 가지 고려사항이 있습니다:

- **메타데이터의 버전 표기**: YAML 프론트매터에 `version` 필드를 추가해 버전 번호를 명시해두면, 나중에 관리할 때 편리합니다 (예: `version: "1.1.0"`). 새로운 기능을 추가하거나 큰 변경을 했을 때 이 버전 값을 올려서 기록해두세요.
- **변경점 기록**: Skill 폴더 내에 `CHANGELOG.md` 같은 파일을 두어, 어떤 변경이 있었는지 기술해 둘 수 있습니다. 아직 Claude가 이를 자동 인식하지는 않지만, 팀 내 공유를 위해 사람이 읽을 변경 이력을 남겨두는 것은 좋습니다.
- **재배포**: 업데이트된 Skill을 팀원들과 공유해야 한다면, 이전과 마찬가지로 새로운 ZIP을 전달하여 각자가 업로드하도록 해야 합니다. 다만 **동일한 이름**의 Skill을 다시 업로드하면 Claude 웹에서는 기존 Skill을 대체하지 않고 **중복 추가**되므로 주의가 필요합니다. 현재 웹 UI 상에서는 동일 이름 Skill을 식별하여 "업그레이드"하는 개념이 없기 때문에, 버전업을 할 경우 `name` 이나 폴더명을 바꾸거나, 먼저 옛 버전을 **삭제**한 뒤 새 버전을 업로드하는 것이 좋습니다.

Anthropic은 **개발자 콘솔** 및 API를 통해 보다 체계적인 Skill 버전 관리를 지원하고 있습니다 <sup>102</sup> <sup>103</sup>. 예를 들어 API의 `/v1/skills` 엔드포인트를 사용하면 프로그래밍 방식으로 Skill을 등록, 조회, 업데이트할 수 있으며, 버전을 파라미터로 관리할 수도 있습니다 <sup>104</sup>. Team/Enterprise 고객의 경우 Anthropic이 제공하는 **Claude Console**에서 중앙 집중적으로 Skill들을 관리하고 버전 업그레이드를 배포할 수 있는 기능이 제공됩니다 <sup>105</sup>. 이를 활용하면 조직 전체 Skill을 한 번에 새 버전으로 교체하거나, 이전 버전으로 롤백하는 등의 작업이 가능합니다.

**팁:** 개발 중인 Skill은 작은 수정 후에도 Claude에 바로 반영하기 위해 여러 번 업로드/테스트를 반복하게 됩니다. 이때 버전을 0.1, 0.2 처럼 세분하여 올려두면 나중에 안정 버전(예: 1.0)을 결정하기 쉽습니다. 또한 오래된 임시 Skill들은 Settings 화면에서 **삭제(Delete)**하여 목록을 정리하세요 106. 복수의 비슷한 Skill이 활성화되어 있으면 Claude가 혼동할 수 있으므로, 쓰지 않는 실험용 Skill은 지워두는 것이 좋습니다.

지속적인 버전 관리를 통해 Skill의 품질을 유지하고, 변화하는 요구사항에 대응할 수 있습니다. 특히 업무 프로세스가 변경될 때 Skill을 업데이트하는 것을 잊지 말고, **AI 지식 베이스를 최신 상태로 유지**하는 것이 중요합니다.

## 응용: 다양한 Skill 아이디어

마지막으로, Claude Skills로 구현할 수 있는 응용 아이디어 몇 가지를 소개합니다. 조직이나 업무 분야에 맞춰 다음과 같은 Skill들을 만들어볼 수 있습니다:

- **브랜드 콘텐츠 생성 Skill** - 마케팅 부서용. 주어진 주제에 대해 **회사 브랜드 가이드라인**에 맞는 블로그 글이나 홍보 문구를 작성하는 Skill입니다. 회사의 톤앤매너, 금지어 목록, 슬로건 등을 SKILL.md에 담아 두고, Claude가 이를 참고해 일관된 브랜드 메시지로 콘텐츠를 생성하도록 할 수 있습니다.
- **코드 리뷰 Skill** - 개발팀용. Pull Request의 코드 변경 사항을 분석하고, **회사 코딩 표준**에 맞는 코드 리뷰 피드백을 자동 생성해주는 Skill입니다. 예를 들어 들여쓰기 스타일, 변수명 규칙, 보안 취약점 패턴 등을 SKILL.md에 포함시켜 Claude가 변경 코드를 검토하고 지적 사항을 정리하게 할 수 있습니다.
- **로그 분석 Skill** - SRE/운영팀용. 서버 로그 파일을 첨부하면 주요 에러나 이상 패턴을 찾아주는 Skill입니다. 예: "이 서버 로그에서 지난 1시간 동안 발생한 500 오류 원인을 요약해줘." Skill 로직은 로그의 에러 스택트레이스를 분류하고 빈도 높은 이슈를 추려내도록 지침을 작성합니다.
- **FAQ 답변 Skill** - 고객지원팀용. 자주 묻는 질문들과 그에 대한 **공식 답변**을 Skill에 담아두면, Claude가 사용자의 질문에 FAQ 기반 표준 답변을 제공합니다. 예: "환불 정책 알려줘." 질문에 회사 정책 Skill을 참고하여 정확한 조건과 절차를 안내하는 식입니다.
- **계약서 검토 Skill** - 법무팀용. NDA나 라이선스 계약서 등의 문서를 업로드하면, **중요 조항 누락 여부나 위험 조항**을 점검해주는 Skill입니다. 법무 부서의 체크리스트를 SKILL.md에 넣어, Claude가 문서를 읽고 해당 체크 항목을 모두 찾아 표시하거나 부족한 부분을 알려주도록 할 수 있습니다.
- **데이터베이스 질의 Skill** - 데이터팀용. 자연어 질문을 **데이터베이스 SQL 질의**로 변환하고, 결과를 요약해주는 Skill입니다. (이 Skill은 Claude에게 DB 직접 접속 권한을 주는 것은 위험하므로, 안전한 샘플 데이터로 동작을 시험해볼 수 있습니다.) 예: "우리 2023년 상품별 판매량 알려줘." Claude가 미리 정의된 쿼리 템플릿을 활용해 SQL을 생성하고, 결과를 표 형태로 보여주는 방식입니다.
- **번역 및 어조 교정 Skill** - 콘텐츠팀용. 문장을 번역하거나 글쓰기 **스타일을 교정**하는 Skill입니다. 예: 문장을 영어로 번역할 때 기업의 특정 용어 스타일 가이드를 반영하거나, 격식을 조절하는 등의 지침을 포함시켜, Claude가 번역/교정 시 일관된 스타일을 유지하도록 합니다.
- **UI/UX 카피라이팅 Skill** - 디자인팀용. UI 문구(버튼 텍스트, 오류 메시지 등)를 작성할 때 일관성을 유지하도록 돕는 Skill입니다. "우리 앱의 UI 문구 작성 원칙"을 SKILL.md에 기록해두고, Claude가 새 문구 작성 시 그 원칙(어조, 길이, 금치어 등)을 자동으로 적용하도록 합니다.
- **재무 보고 자동화 Skill** - 재무팀용. 분기별 재무 보고서를 작성하는 Skill입니다. 입력으로 원시 재무 지표(매출, 비용 등)를 주면, Skill이 **정해진 양식의 보고서 문장과 표**를 자동으로 생성합니다. (Excel 보고서 Skill과 유사하지만, Word 또는 PDF 형식의 텍스트 위주 보고서를 생성합니다.)
- **의료 기록 요약 Skill** - 의료 분야. 환자의 장문 진료 기록을 입력하면 중요한 진단 정보, 투약 내역 등을 발췌해 **의료진 인계용 요약본**을 만들어주는 Skill입니다. 의료 전문 용어와 각종 약어의 뜻도 포함하도록 하여, 다른 의료진이 빠르게 파악할 수 있게 해줍니다.

이처럼 Claude Skills를 활용하면 **각 분야의 전문 지식과 업무 절차를 AI에게 가르쳐서**, 반복 작업을 자동화하거나 복잡한 정보를 빠르게 처리하는 데 응용할 수 있습니다. 아이디어를 구상할 때, 현재 Claude가 잘하지 못하는 일이나 매번 사람이 수작업으로 설명해 주어야 했던 부분이 무엇인지 떠올려보세요. 그런 부분이 있다면 Skill로 만들어 **한 번만 가르치고 두고두고 쓰는** 것을 목표로 할 수 있습니다 11. 창의적인 활용 사례는 무궁무진하며, 앞으로도 커뮤니티에서 다양한 Skill들이 공유되고 발전될 것입니다.

## (고급) Claude Skills의 내부 동작 원리

※ 이 섹션은 Claude Skills의 동작 원리에 대한 심화 기술 설명으로, 일반 사용자는 건너뛰어도 무방합니다.

Claude Skills의 작동 방식은 겉보기에는 마치 **함수를 호출**하거나 **코드를 실행**하는 것처럼 느껴지지만, 실제 내부에서는 **프롬프트 기반 컨텍스트 주입** 기법으로 이루어집니다 <sup>107 108</sup>. 이를 이해하면 왜 Skill을 특정 방식으로 작성해야 하는지, 그리고 Claude가 Skill을 어떻게 다루는지 더 깊이 파악할 수 있습니다.

### 프롬프트 컨텍스트로서의 Skill

Claude에게는 기본적으로 다양한 **내장 도구(tools)**들이 있습니다. 예를 들어 `Read` (파일 읽기), `Write` (파일 쓰기), `Bash` / `Python` (코드 실행) 등의 도구입니다. Skill은 이러한 도구들과는 달리, **“Skill”이라는 특수한 메타-도구**로 관리됩니다 <sup>76</sup>. Claude는 대화를 시작할 때, 사용 가능한 Skill들의 이름과 description을 하나의 큰 목록으로 합쳐서 이 **Skill 메타-도구의 설명**에 포함시킵니다 <sup>109 110</sup>. 쉽게 말해, Claude는 매 요청마다 "다음은 사용할 수 있는 스킬들입니다: - 스킬A: 설명..., - 스킬B: 설명..., ..."이라는 정보를 시스템 메시지에 가지고 있는 셈입니다.

이때 Claude는 사용자의 질문을 이해하는 과정에서, 이 Skill 목록(특히 각 Skill의 description)을 참고하여 **가장 관련성 높은 Skill의 이름**을 결정합니다 <sup>73 74</sup>. 그리고는 내부적으로 그 Skill 이름을 “사용된 도구”로 표기하면서, 해당 Skill의 실제 콘텐츠를 컨텍스트에 주입합니다 <sup>75</sup>. 예를 들어 Claude가 `internal-comms` 라는 Skill을 사용하기로 했다면, Skill 메타-도구를 호출하면서 `command: "internal-comms"` 와 같이 호출하는 식입니다 <sup>111</sup>. 그러면 시스템은 `internal-comms` Skill의 SKILL.md 파일 내용을 Claude에게 **추가 컨텍스트**로 제공하게 됩니다.

이 과정은 전적으로 **LLM 모델의 자율적인 판단**으로 이루어집니다. Skill 선택을 위한 별도의 머신러닝 분류기나 하드코딩된 규칙은 없습니다 <sup>112</sup>. Claude의 Transformer 모델 내부에서, 사용자 메시지와 Skill 설명들이 함께 고려되어 attention 메커니즘을 통해 적절한 Skill이 선택되는 것입니다. 이것이 Anthropic이 Skills를 설계한 핵심 아이디어로, **규칙 기반이 아닌 순수한 언어 이해를 통한 도구 선택**을 구현한 부분입니다 <sup>112</sup>.

### Skill의 컨텍스트 주입과 실행 컨텍스트

Skill이 선택되면, 해당 Skill의 모든 내용(SKILL.md 본문)이 Claude의 대화 컨텍스트에 추가됩니다 <sup>75</sup>. 이는 마치 사용자가 아주 긴 매뉴얼을 Claude에게 일시에 던져준 것과 비슷한 효과입니다. Claude는 그 내용을 읽고, 거기에 쓰인 지시에 따라 이후 답변을 구성합니다.

동시에, Skill에 스크립트 파일이 있거나 특정 도구 사용이 필요한 경우, Claude의 **실행 컨텍스트**도 조정됩니다 <sup>75 113</sup>. 예를 들어 Skill에서 `allowed-tools: ["Python"]` 으로 제한했다면, Claude는 해당 Skill을 수행하는 동안 Bash 명령은 시도조차 하지 않을 것입니다. 또, Skill이 특정 모델 사용을 지정하는 메타데이터를 갖고 있다면, Claude가 더 높은 성능의 모델(예: Sonnet 4.5)로 자동 전환하는 등의 동작도 가능해집니다 (실제로 Anthropic API에서는 Skill 호출 시 모델 전환을 지원하기도 합니다).

이러한 맥락에서, Skill은 **프롬프트 템플릿**의 일종으로 볼 수 있습니다 <sup>108</sup>. 우리가 수동으로 "이런 상황에서는 이렇게 해"라고 대화창에 입력하던 것을, Skill 파일에 정리해두고 Claude가 스스로 불러쓰도록 한 것입니다. 그래서 Skill 작성시 언어는 **명령형(imperative)**으로 쓰며, Claude에게 직접 말한다고 생각하고 지침을 적는 것이 좋습니다. (예: "너는 반드시 ~~해야 한다" 보다는 "항상 ~~해."처럼 간결한 명령조로 작성합니다.)

### Tools vs Skills 차이

Claude Skills의 개념을 이해할 때, 기존의 도구(function call) 기반 확장과 비교하면 다음과 같은 차이가 있습니다 <sup>114 115</sup>:

| 구분    | 전통적 도구 (예: 함수 호출)                  | Claude Skill  |
|-------|------------------------------------|---|
| 실행 방식 | 코드/함수를 실행하여 즉각 결과 반환 (동기적)         | 프롬프트를 추가하여 컨텍스트 내 문제 해결 (비동기적)                                |
| 용도    | 특정 작업을 바로 수행 (예: 계산, DB조회)         | 복잡한 절차나 작업 방식을 안내 (예: 문서 작성 워크플로우)                            |
| 반환값   | 함수 반환값이나 API 응답 등 즉각적 결과           | 대화 컨텍스트에 추가된 지침 및 그에 따른 Claude의 응답                            |
| 예시    | <code>Read</code> 도구: 파일 내용을 읽어 반환 | <code>pdf-summarizer</code> Skill: PDF 처리 방법을 컨텍스트에 주입해 요약 생성 |
| 병렬 처리 | 여러 함수 병렬 호출 가능 (일반적으로 안전)          | 한 번에 한 Skill 주 컨텍스트 사용 (여러 Skill 동시 발동 시 Claude가 순차 처리)       |
| 정의 위치 | 애플리케이션 코드 (개발자가 작성)                | 외부 파일(Skill 폴더 내) - LLM이 불러서 사용                               |
| 안전성   | 샌드박스 내 실제 코드 실행 -> 외부 시스템 영향 가능    | 모델의 텍스트 생성 -> 외부 영향은 Claude가 별도 도구 쓸 때만 발생                    |

위 표에서 보듯, Skills는 **직접 실행**된다기보다 Claude의 **사고 과정(Chain-of-Thought)을 확장**하는 개념입니다. 따라서 결과도 즉각적인 반환값이 아니라, Claude가 Skill 지식을 기반으로 생성한 **대화 응답** 형태로 나타납니다. 장점은 여러 Skill의 지침이 **조합**되어 더 복잡한 작업도 수행 가능하다는 점이고 <sup>5</sup>, 단점은 모든 게 모델의 응답 생성에 달려 있으므로 가끔 예측이 빗나가거나 지침을 일부만 따르는 경우도 있을 수 있다는 점입니다. (예: Skill 지침을 부분적으로만 따르는 현상이 발생할 수 있는데, 이를 막기 위해 Skill에 중요한 부분을 **강조**하거나, 충분한 테스트를 통해 보완해야 합니다.)

## 안전성과 권한 제어

Anthropic Claude의 Skill 기능은 강력하지만, 보안 면에서도 신중하게 설계되었습니다. Skill은 기본적으로 Claude의 **격리된 샌드박스** 환경에서만 코드를 실행할 수 있으며, 외부 네트워크 접속은 명시적으로 허용된 MCP 경로를 통해서만 이뤄집니다 <sup>59 60</sup>. 또한 각 Skill은 사용자의 계정 영역에 국한되어 작동하며, 한 사용자의 Skill이 다른 사용자에게 영향을 주지 않습니다 (공유하지 않는 한). Anthropic은 권고사항으로 **신뢰할 수 없는 출처의 Skill을 설치하지 말** 것을 안내하고 있으며 <sup>116 117</sup>, Skill 내용을 업로드하기 전에 사람이 반드시 **검토**하도록 합니다 <sup>118</sup>.

특히 **프롬프트 인젝션** 공격에 주의해야 합니다 <sup>60 119</sup>. 악의적인 사용자가 대화 중에 "다른 Skill을 비활성화해"와 같은 지시를 하거나, Skill 내부에 의도치 않은 명령이 숨겨져 있을 가능성을 항상 염두에 두고 Claude는 동작합니다. Anthropic은 이러한 위험을 낮추기 위해 Claude의 내부 정책 필터를 적용하고 있으며, Skill 실행 시에도 안전성 점검을 수행합니다. 그러나 궁극적으로 Skill 제작자가 **악의적이거나 오류가 있는 내용**을 포함하지 않도록 주의하는 것이 1차 방어입니다.

## 성능 최적화와 한계

마지막으로, Skill 사용 시 고려해야 할 **성능 요소**로는 **컨텍스트 길이 한계**와 **응답 시간**이 있습니다. Skill 내용이 아주 많다면 Claude의 한 번 대화 컨텍스트에 모두 들어가기 어려울 수 있습니다. (Claude 2 등 최신 모델은 수만 토큰까지 입력을 처리하지만, 여전히 한계는 존재합니다.) 그러므로 **가벼운 Skill 여러 개**로 쪼개는 편이 유리하고 <sup>120 121</sup>, 한 Skill 내에서도 꼭 필요한 내용만 남기고 과도한 예시는 제거하는 등 튜닝이 필요할 수 있습니다. 응답 시간 측면에서는, Skill을 로드하면 Claude가 해야 할 읽기 작업이 늘어나므로 약간 딜레이가 생길 수 있습니다. 하지만 Anthropic이 Skill 설계를 효율적으로 해 두었기에, **필요 최소한만 읽도록** (progressive disclosure) 최적화되어 있어 이 영향은 크지 않습니다 <sup>122</sup>.

정리하면, Claude Skills는 언뜻 복잡해 보이지만 결국 "상황에 따라 불러오는 작은 지침서"입니다. Claude는 이를 통해 자신의 능력을 확장하고, 더욱 전문화된 답변을 제공할 수 있게 됩니다. 개발자는 Skill의 내부 메커니즘을 이해함으로써, 더욱 효과적으로 Skill을 설계하고 문제 발생 시 디버깅을 할 수 있을 것입니다.

---

1 2 3 4 8 9 10 11 30 33 70 71 72 77 122 What are Claude Skills and how do they work? -

CometAPI - All AI Models in One API

<https://www.cometapi.com/what-are-claude-skills-and-how-do-they-work/>

5 67 69 101 102 103 104 105 Introducing Agent Skills | Claude

<https://claude.com/blog/skills>

6 7 31 32 93 96 97 About Claude Skills - A comprehensive guide · GitHub

<https://gist.github.com/stevenringo/d7107d6096e7d0cf5716196d2880d5bb>

12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 35 59 60 63 64 78 79 89 90 91 94 95 106 116

117 118 119 Using Skills in Claude | Claude Help Center

<https://support.claude.com/en/articles/12512180-using-skills-in-claude>

27 28 29 37 38 39 40 41 42 43 44 48 49 50 51 52 54 65 66 68 80 81 82 83 85 86 87 88 How

to create Skills for Claude: steps and examples | Claude

<https://claude.com/blog/how-to-create-skills-key-steps-limitations-and-examples>

34 36 45 46 47 53 55 56 57 58 61 62 84 92 120 121 How to create custom Skills | Claude Help

Center

<https://support.claude.com/en/articles/12512198-how-to-create-custom-skills>

73 74 75 76 107 108 109 110 111 112 113 114 115 Claude Agent Skills: A First Principles Deep Dive

<https://leehanchung.github.io/blogs/2025/10/26/claude-skills-deep-dive/>

98 99 100 read-calendar - Claude Skills

<https://claude-plugins.dev/skills/@ronnycoding/my-personal-assistant/read-calendar>