

## Agent Skills로 실제 세계를 위한 에이전트 장비하기

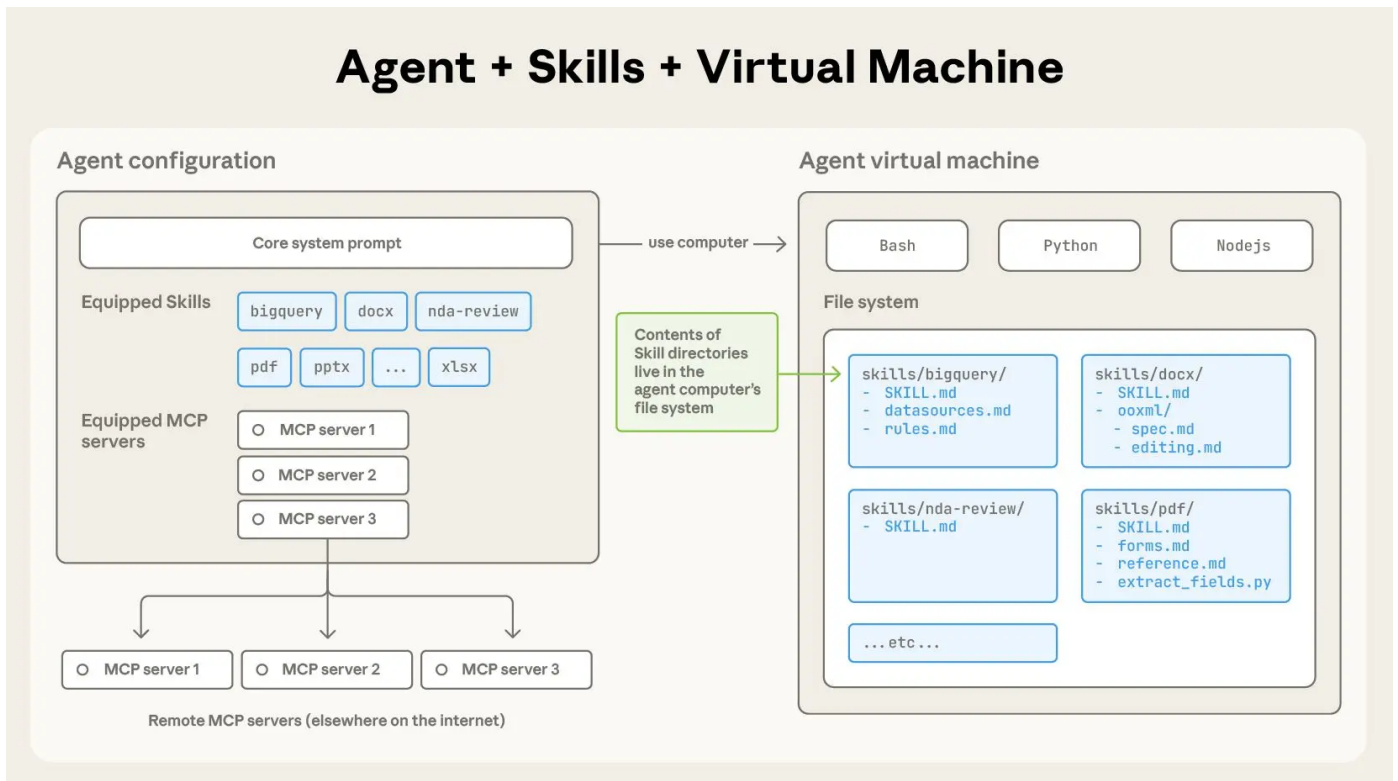
게시됨 2025년 10월 16일

Claude는 강력하지만, 실제 업무에는 절차적 지식과 조직적 맥락이 필요합니다. 파일과 폴더를 사용하여 전문화된 에이전트를 구축하는 새로운 방법인 Agent Skills를 소개합니다.

모델 기능이 향상됨에 따라, 이제 완전한 컴퓨팅 환경과 상호작용하는 범용 에이전트를 구축할 수 있습니다. [Claude Code](#) 예를 들어, 로컬 코드 실행과 파일시스템을 사용하여 도메인 전반에 걸쳐 복잡한 작업을 수행할 수 있습니다. 하지만 이러한 에이전트가 더욱 강력해짐에 따라, 도메인별 전문성을 갖추기 위한 더욱 조합 가능하고, 확장 가능하며, 이식 가능한 방법이 필요합니다.

이것이 우리가 다음을 만들게 된 계기입니다 **AgentSkills**: 에이전트가 특정 작업을 더 잘 수행하기 위해 동적으로 발견하고 로드할 수 있는 지침, 스크립트, 리소스로 구성된 체계적인 폴더입니다. 스킬은 당신의 전문 지식을 Claude가 활용할 수 있는 구성 가능한 리소스로 패키징하여 Claude의 기능을 확장하고, 범용 에이전트를 당신의 필요에 맞는 전문화된 에이전트로 변환합니다.

에이전트를 위한 스킬을 구축하는 것은 신입 사원을 위한 온보딩 가이드를 만드는 것과 같습니다. 각 사용 사례마다 단편적이고 맞춤 설계된 에이전트를 구축하는 대신, 이제 누구나 절차적 지식을 포착하고 공유함으로써 구성 가능한 기능으로 에이전트를 전문화할 수 있습니다. 이 글에서는 스킬이 무엇인지 설명하고, 작동 방식을 보여주며, 자신만의 스킬을 구축하기 위한 모범 사례를 공유합니다.



스킬은 에이전트에게 추가적인 기능을 제공하는 지시사항, 스크립트, 리소스들이 정리된 폴더들을 포함하는 SKILL.md 파일이 있는 디렉토리입니다.

## 스킬의 구조

스킬이 실제로 어떻게 작동하는지 보기 위해, 실제 예시를 살펴보겠습니다: **Claude의 최근 출시된 문서 편집 기능**을 지원하는 스킬 중 하나입니다. Claude는 이미 PDF를 이해하는 것에 대해 많이 알고 있지만, PDF를 직접 조작하는 능력(예: 양식 작성)에는 제한이 있습니다. 이 **PDF 스킬**을 통해 Claude에게 이러한 새로운 능력을 부여할 수 있습니다.

가장 간단하게 말하면, 스킬은 다음을 포함하는 디렉토리입니다 **SKILL.md 파일**. 이 파일은 몇 가지 필수 메타데이터를 포함하는 YAML 프론트매터로 시작해야 합니다: **이름** 과 **설명**. 시작 시, 에이전트는 설치된 모든 스킬의 **이름** 과 **설명** 을 시스템 프롬프트에 미리 로드합니다.

이 메타데이터는 **첫 번째 레벨의 점진적 공개** : 모든 정보를 컨텍스트에 로드하지 않으면서도 Claude가 각 기술을 언제 사용해야 하는지 알 수 있을 만큼의 정보만 제공합니다. 이 파일의 실제 본문은 **두 번째 수준**의 세부사항입니다. Claude가 해당 기술이 현재 작업과 관련이 있다고 판단하면, 전체 내용을 읽어 기술을 로드할 것입니다 **SKILL.md** 맥락에 넣어보세요.

## A simple SKILL.md file

pdf/SKILL.md

### YAML Frontmatter

```
name: pdf
description: Comprehensive PDF toolkit for extracting text and tables,
merging/splitting documents, and filling-out forms.
```

### Markdown

#### ## Overview

This guide covers essential PDF processing operations using Python libraries and command-line tools. For advanced features, JavaScript libraries, and detailed examples, see ./reference.md. If you need to fill out a PDF form, read ./forms.md and follow its instructions.

#### ## Quick Start

```
python
from pypdf import PdfReader, PdfWriter

# Read a PDF
reader = PdfReader("document.pdf")
print(f"Pages: {len(reader.pages)}")
```

SKILL.md 파일은 파일명과 설명을 포함하는 YAML Frontmatter로 시작해야 하며, 이는 시작 시 시스템 프롬프트에 로드됩니다.

스킬이 복잡해짐에 따라 단일 **SKILL.md**에 맞지 않을 정도로 많은 컨텍스트를 포함하거나, 특정 시나리오에서만 관련된 컨텍스트를 포함할 수 있습니다. 이러한 경우, 스킬은 스킬 디렉토리 내에 추가 파일들을 번들로 포함하고 **SKILL.md**에서 이름으로 참조할 수 있습니다. 이러한 추가 연결 파일들이 **세 번째 레벨**(그리고 그 이상의) 세부 사항으로, Claude는 필요에 따라서만 탐색하고 발견하도록 선택할 수 있습니다.

아래 표시된 PDF 스킬에서 **SKILL.md**는 스킬 작성자가 핵심 **reference.md**과 **forms.md**과 함께 번들로 묶기로 선택한 두 개의 추가 파일(**SKILL.md**)을 참조합니다. 양식 작성 지침을 별도 파일로 이동함으로써(**forms.md**), 스킬 작성자는 Claude가 읽을 것이라고 신뢰하면서 스킬의 핵심을 간결하게 유지할 수 있습니다 **forms.md** 양식을 작성할 때만.

# Bundling additional content

pdf/SKILL.md

## YAML Frontmatter

```
---
name: pdf
description: Comprehensive PDF toolkit for extracting text and
tables, merging/splitting documents, and filling-out forms.
---
```

## Markdown

### ## Overview

This guide covers essential PDF processing operations using Python libraries and command-line tools. For advanced features, JavaScript libraries, and detailed examples, see `./reference.md`. If you need to fill out a PDF form, read `./forms.md` and follow its instructions.

### ## Quick Start

```
```python
from pypdf import PdfReader, PdfWriter
```

```
# Read a PDF
reader = PdfReader("document.pdf")
print(f"Pages: {len(reader.pages)}")
```

```
# Extract text
text = ""
for page in reader.pages:
    text += page.extract_text()
```

pdf/reference.md

## # PDF Processing Advanced Reference

This document contains advanced PDF processing features, detailed examples, and additional libraries not covered in the main skill instructions.

### ## pypdfium2 Library (Apache/BSD License)

#### ### Overview

pypdfium2 is a Python binding for PDFium (Chromium's PDF library). It's excellent for fast PDF rendering, image generation, and serves as ...

pdf/forms.md

If you need to fill out a PDF form, first check to see if the PDF has fillable form fields. Run this script from this file's directory:

```
`python scripts/check_fillable_fields <file.pdf>`,
and depending on the result go to either the "Fillable
fields" or "Non-fillable fields" and follow those
instructions.
```

#### # Fillable fields

If the PDF has fillable form fields:

- Run this script from this file's directory:

```
`python scripts/extract_form_field_info.py <input.pdf>
<fields.json>`.
...
```

시스템 프롬프트를 기반으로 Claude가 트리거할 수 있는 추가적인 컨텍스트(추가 파일을 통해)를 스킬에 통합할 수 있습니다.

점진적 공개는 Agent Skills를 유연하고 확장 가능하게 만드는 핵심 설계 원칙입니다. 목차로 시작해서 특정 챕터, 그리고 마지막에 상세한 부록으로 구성된 잘 정리된 매뉴얼처럼, 스킬은 Claude가 필요에 따라서만 정보를 로드할 수 있게 해줍니다:

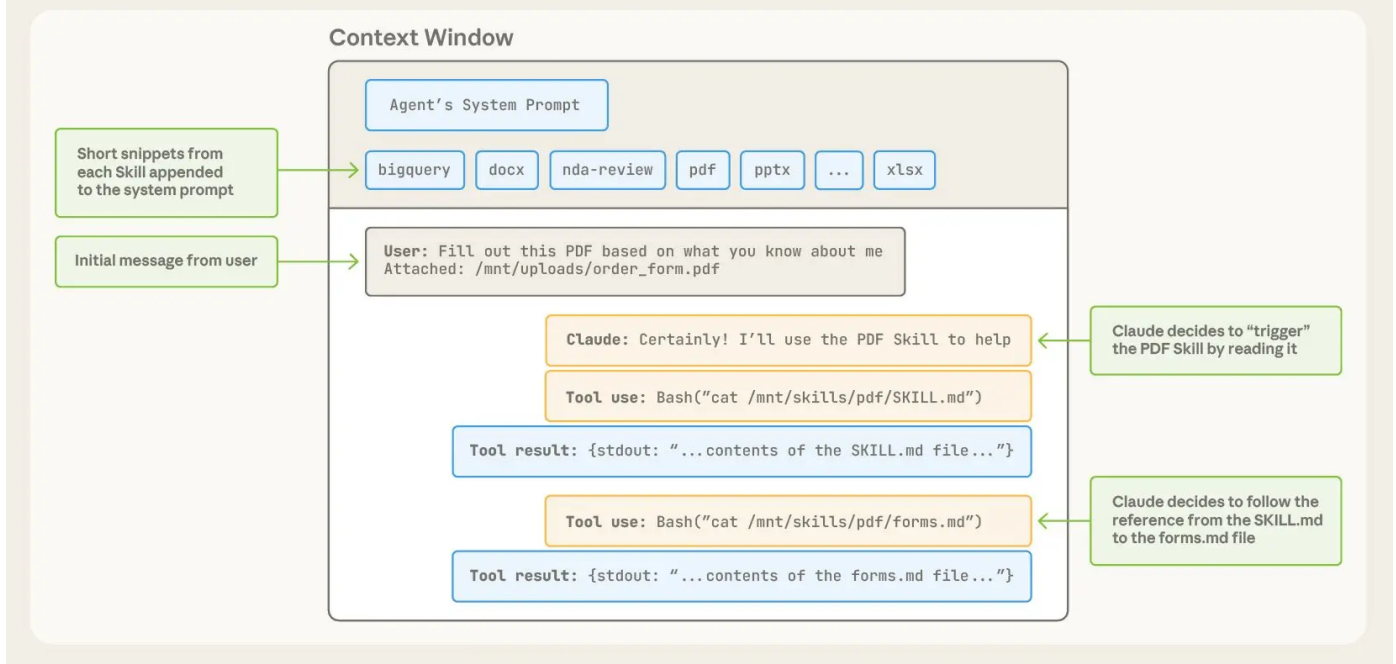
Level	File	Context Window	# Tokens
1	SKILL.md Metadata (YAML)	Always loaded	~100
2	SKILL.md Body (Markdown)	Loaded when Skill triggers	<5k
3+	Bundled files (text files, scripts, data)	Loaded as-needed by Claude	unlimited*

파일시스템과 코드 실행 도구를 가진 에이전트는 특정 작업을 수행할 때 스킬의 전체 내용을 컨텍스트 윈도우에 읽어들이 필요가 없습니다. 이는 스킬에 포함될 수 있는 컨텍스트의 양이 사실상 무제한이라는 것을 의미합니다.

## 스킬과 컨텍스트 윈도우

다음 다이어그램은 사용자의 메시지에 의해 스킬이 트리거될 때 컨텍스트 윈도우가 어떻게 변화하는지 보여줍니다.

# Skills and the Context Window



스킬은 시스템 프롬프트를 통해 컨텍스트 윈도우에서 트리거됩니다.

표시된 작업 순서:

1. 시작할 때, 컨텍스트 윈도우에는 핵심 시스템 프롬프트와 설치된 각 스킬의 메타데이터, 그리고 사용자의 초기 메시지가 포함됩니다;
2. Claude는 다음 내용을 읽기 위해 Bash 도구를 호출하여 PDF 스킬을 트리거합니다 `pdf/SKILL.md` ;
3. Claude는 스킬과 함께 번들된 `forms.md` 파일을 읽기로 선택합니다;
4. 마지막으로, Claude는 PDF 스킬에서 관련 지침을 로드한 후 사용자의 작업을 진행합니다.

## 스킬과 코드 실행

스킬에는 Claude가 재량에 따라 도구로 실행할 수 있는 코드도 포함될 수 있습니다.

대형 언어 모델은 많은 작업에서 뛰어난 성능을 보이지만, 특정 작업들은 전통적인 코드 실행에 더 적합합니다. 예를 들어, 토큰 생성을 통해 리스트를 정렬하는 것은 단순히 정렬 알고리즘을 실행하는 것보다 훨씬 비용이 많이 듭니다. 효율성 문제를 넘어서, 많은 애플리케이션에서는 코드만이 제공할 수 있는 결정론적 신뢰성이 필요합니다.

우리 예시에서 PDF 스킬은 PDF를 읽고 모든 양식 필드를 추출하는 미리 작성된 Python 스크립트를 포함합니다. Claude는 스크립트나 PDF를 컨텍스트에 로드하지 않고도 이 스크립트를 실행할 수 있습니다. 그리고 코드는 결정론적이기 때문에 이 워크플로우는 일관되고 반복 가능합니다.

# Bundling executable scripts

pdf/forms.md

```
If you need to fill out a PDF form, first check
to see if the PDF has fillable form fields.
Run this script from this file's directory:
`python scripts/check_fillable_fields <file.pdf>`,
and depending on the result go to either the
"Fillable fields"
or "Non-fillable fields" and follow those
instructions.

# Fillable fields If the PDF has fillable form
fields:
- Run this script from this file's directory:
`python ./extract_fields.py
<input.pdf> <fields.json>`.
...
```

pdf/extract\_fields.py

```
from pypdf import PdfReader

def write_field_info(pdf_path: str, output_path: str):
    """Extract form fields from PDF and store as JSON."""
    reader = PdfReader(pdf_path)
    fields = get_fields(reader)
    with open(output_path, "w") as f:
        json.dump(fields, f)

# ... omitted ...

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print(f"Usage: python {sys.argv[0]} <pdf_path> <output_json_path>")
        sys.exit(1)
    write_field_info(sys.argv[1], sys.argv[2])
```

스킬은 또한 작업의 성격에 따라 Claude가 재량껏 도구로 실행할 수 있는 코드를 포함할 수도 있습니다.

## 스킬 개발 및 평가

스킬 작성 및 테스트를 시작하는 데 도움이 되는 몇 가지 유용한 가이드라인은 다음과 같습니다:

- **평가부터 시작하세요:** 대표적인 작업에서 에이전트를 실행하고 어려움을 겪거나 추가적인 맥락이 필요한 부분을 관찰하여 에이전트 역량의 구체적인 격차를 파악하세요. 그런 다음 이러한 부족한 부분을 해결하기 위해 점진적으로 기술을 구축하세요.
- **규모를 위한 구조:** 번역할 텍스트가 불완전합니다. "When the"로 시작하는 문장이 끝나지 않았습니다. 완전한 텍스트를 제공해 주시면 번역해드리겠습니다. **SKILL.md** 파일이 다루기 어려워지면 내용을 별도 파일로 분할하고 참조하세요. 특정 컨텍스트가 상호 배타적이거나 함께 사용되는 경우가 드물다면, 경로를 분리해 두는 것이 토큰 사용량을 줄일 수 있습니다. 마지막으로, 코드는 실행 가능한 도구와 문서 역할을 모두 할 수 있습니다. Claude가 스크립트를 직접 실행해야 하는지 아니면 참조용으로 컨텍스트에 읽어들여야 하는지 명확해야 합니다.
- **클로드의 관점에서 생각해 보세요:** 실제 시나리오에서 Claude가 당신의 스킬을 어떻게 사용하는지 모니터링하고 관찰을 바탕으로 반복 개선하세요: 예상치 못한 궤적이나 특정 맥락에 대한 과도한 의존을 주의 깊게 살펴보세요. 특히 **이름**과 **설명**에 특별한 주의를 기울이세요. Claude는 현재 작업에 대응하여 스킬을 실행할지 결정할 때 이를 사용합니다.
- **Claude와 함께 반복하세요:** Claude와 함께 작업을 진행할 때, Claude가 성공적인 접근 방식과 일반적인 실수들을 스킬 내의 재사용 가능한 컨텍스트와 코드로 정리하도록 요청하세요. 스킬을 사용하여 작업을 완료할 때 방향을 잃는다면, 무엇이 잘못되었는지 스스로 성찰하도록 요청하세요. 이 과정은 미리 예상하려고 하는 대신, Claude가 실제로 필요로 하는 컨텍스트가 무엇인지 발견하는 데 도움이 될 것입니다.

## 스킬 사용 시 보안 고려사항

스킬은 지시사항과 코드를 통해 Claude에게 새로운 기능을 제공합니다. 이는 스킬을 강력하게 만들지만, 동시에 악의적인 스킬이 사용되는 환경에 취약점을 도입하거나 Claude가 데이터를 유출하고 의도하지 않은 행동을 취하도록 지시할 수 있음을 의미합니다.

신뢰할 수 있는 출처에서만 스킬을 설치하는 것을 권장합니다. 신뢰도가 낮은 출처에서 스킬을 설치할 때는 사용하기 전에 철저히 검토하세요. 먼저 스킬에 포함된 파일들의 내용을 읽어 스킬이 무엇을 하는지 이해하고, 특히 코드 종속성과 이미지나 스크립트 같은 번들 리소스에 주의를 기울이세요. 마찬가지로 Claude가 신뢰할 수 없는 외부 네트워크 소스에 연결하도록 지시하는 스킬 내의 지침이나 코드에도 주의를 기울이세요.

## 스킬의 미래

에이전트 스킬은 현재 지원됩니다 전반에 걸쳐 [Claude.ai](#), Claude Code, Claude Agent SDK, 그리고 Claude Developer Platform입니다.

앞으로 몇 주 동안 Skills의 생성, 편집, 발견, 공유, 사용의 전체 생명주기를 지원하는 기능들을 계속 추가할 예정입니다. 특히 Skills가 조직과 개인이 자신들의 맥락과 워크플로우를 Claude와 공유하는 데 도움이 되는 기회에 대해 매우 기대하고 있습니다. 또한 Skills가 어떻게 보완할 수 있는지도 탐구할 것입니다. Model Context Protocol (MCP) 서버를 통해 에이전트에게 외부 도구와 소프트웨어를 포함하는 더 복잡한 워크플로우를 가르칩니다.

더 나아가, 우리는 에이전트들이 스스로 스킬을 생성, 편집, 평가할 수 있도록 하여, 자신만의 행동 패턴을 재사용 가능한 능력으로 체계화할 수 있기를 희망합니다.

스킬은 간단한 개념이며 그에 상응하는 간단한 형식을 가지고 있습니다. 이러한 단순함은 조직, 개발자, 최종 사용자가 맞춤형 에이전트를 구축하고 새로운 기능을 부여하는 것을 더 쉽게 만듭니다.

사람들이 스킬로 무엇을 만들어낼지 기대됩니다. 오늘 우리의 스킬을 확인하여 시작해보세요 문서와 요리책.

## 감사의 말

Barry Zhang, Keith Lazuka, Mahesh Murag가 작성했으며, 이들은 모두 폴더를 정말 좋아합니다. Skills를 지지하고, 지원하고, 구축한 Anthropic의 많은 다른 분들께 특별한 감사를 드립니다.

# Equipping agents for the real world with Agent Skills

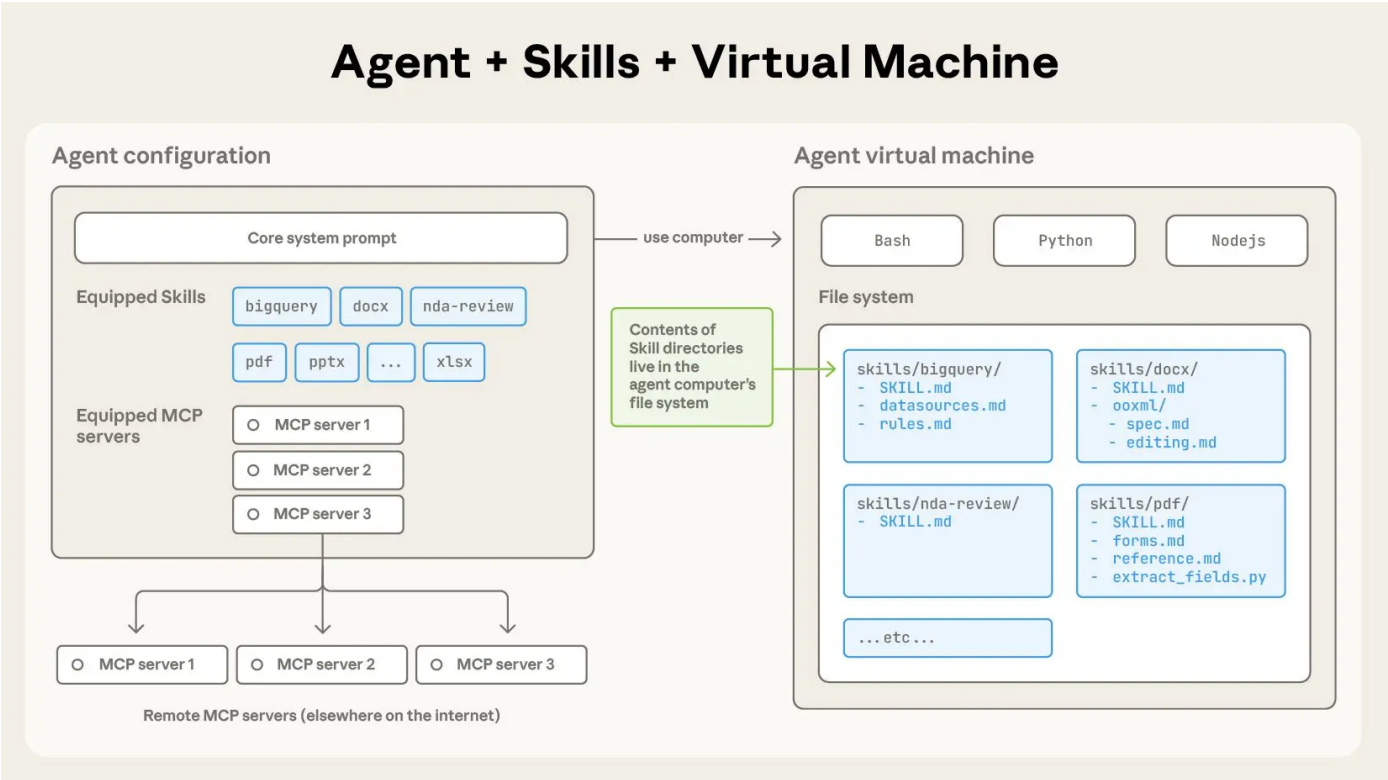
Published Oct 16, 2025

Claude is powerful, but real work requires procedural knowledge and organizational context. Introducing Agent Skills, a new way to build specialized agents using files and folders.

As model capabilities improve, we can now build general-purpose agents that interact with full-fledged computing environments. [Claude Code](#), for example, can accomplish complex tasks across domains using local code execution and filesystems. But as these agents become more powerful, we need more composable, scalable, and portable ways to equip them with domain-specific expertise.

This led us to create [Agent Skills](#): organized folders of instructions, scripts, and resources that agents can discover and load dynamically to perform better at specific tasks. Skills extend Claude’s capabilities by packaging your expertise into composable resources for Claude, transforming general-purpose agents into specialized agents that fit your needs.

Building a skill for an agent is like putting together an onboarding guide for a new hire. Instead of building fragmented, custom-designed agents for each use case, anyone can now specialize their agents with composable capabilities by capturing and sharing their procedural knowledge. In this article, we explain what Skills are, show how they work, and share best practices for building your own.



A skill is a directory containing a SKILL.md file that contains organized folders of instructions, scripts, and resources that give agents additional capabilities.

## The anatomy of a skill

To see Skills in action, let's walk through a real example: one of the skills that powers [Claude's recently launched document editing abilities](#). Claude already knows a lot about understanding PDFs, but is limited in its ability to manipulate them directly (e.g. to fill out a form). This [PDF skill](#) lets us give Claude these new abilities.

At its simplest, a skill is a directory that contains a [SKILL.md file](#). This file must start with YAML frontmatter that contains some required metadata: [name](#) and [description](#). At startup, the agent pre-loads the [name](#) and [description](#) of every installed skill into its system prompt.

This metadata is the **first level** of *progressive disclosure*: it provides just enough information for Claude to know when each skill should be used without loading all of it into context. The actual body of this file is the **second level** of detail. If Claude thinks the skill is relevant to the current task, it will load the skill by reading its full [SKILL.md](#) into context.

## A simple SKILL.md file

pdf/SKILL.md

### YAML Frontmatter

```
name: pdf
description: Comprehensive PDF toolkit for extracting text and tables,
merging/splitting documents, and filling-out forms.
```

### Markdown

#### ## Overview

This guide covers essential PDF processing operations using Python libraries and command-line tools. For advanced features, JavaScript libraries, and detailed examples, see [./reference.md](#). If you need to fill out a PDF form, read [./forms.md](#) and follow its instructions.

#### ## Quick Start

```
```python
from pypdf import PdfReader, PdfWriter
```

```
# Read a PDF
reader = PdfReader("document.pdf")
print(f"Pages: {len(reader.pages)}")
```

```
...
```

A SKILL.md file must begin with YAML Frontmatter that contains a file name and description, which is loaded into its system prompt at startup.

As skills grow in complexity, they may contain too much context to fit into a single [SKILL.md](#), or context that's relevant only in specific scenarios. In these cases, skills can bundle additional files within the skill directory and reference them by name from [SKILL.md](#). These additional linked files are the **third level** (and beyond) of detail, which Claude can choose to navigate and discover only as needed.

In the PDF skill shown below, the [SKILL.md](#) refers to two additional files ( [reference.md](#) and [forms.md](#) ) that the skill author chooses to bundle alongside the core [SKILL.md](#). By moving the form-filling instructions to a separate file ( [forms.md](#) ), the skill author is able to keep the core of the skill lean, trusting that Claude will read [forms.md](#) only when filling out a form.

# Bundling additional content

pdf/SKILL.md

## YAML Frontmatter

```
---
name: pdf
description: Comprehensive PDF toolkit for extracting text and
tables, merging/splitting documents, and filling-out forms.
---
```

## Markdown

### ## Overview

This guide covers essential PDF processing operations using Python libraries and command-line tools. For advanced features, JavaScript libraries, and detailed examples, see `./reference.md`. If you need to fill out a PDF form, read `./forms.md` and follow its instructions.

### ## Quick Start

```
```python
from pypdf import PdfReader, PdfWriter
```

#### # Read a PDF

```
reader = PdfReader("document.pdf")
print(f"Pages: {len(reader.pages)}")
```

#### # Extract text

```
text = ""
for page in reader.pages:
    text += page.extract_text()
```

pdf/reference.md

## # PDF Processing Advanced Reference

This document contains advanced PDF processing features, detailed examples, and additional libraries not covered in the main skill instructions.

### ## pypdfium2 Library (Apache/BSD License)

#### ### Overview

pypdfium2 is a Python binding for PDFium (Chromium's PDF library). It's excellent for fast PDF rendering, image generation, and serves as ...

pdf/forms.md

If you need to fill out a PDF form, first check to see if the PDF has fillable form fields. Run this script from this file's directory:

```
`python scripts/check_fillable_fields <file.pdf>`,
and depending on the result go to either the "Fillable
fields" or "Non-fillable fields" and follow those
instructions.
```

#### # Fillable fields

If the PDF has fillable form fields:

- Run this script from this file's directory:

```
`python scripts/extract_form_field_info.py <input.pdf>
<fields.json>`
...
```

You can incorporate more context (via additional files) into your skill that can then be triggered by Claude based on the system prompt.

Progressive disclosure is the core design principle that makes Agent Skills flexible and scalable. Like a well-organized manual that starts with a table of contents, then specific chapters, and finally a detailed appendix, skills let Claude load information only as needed:

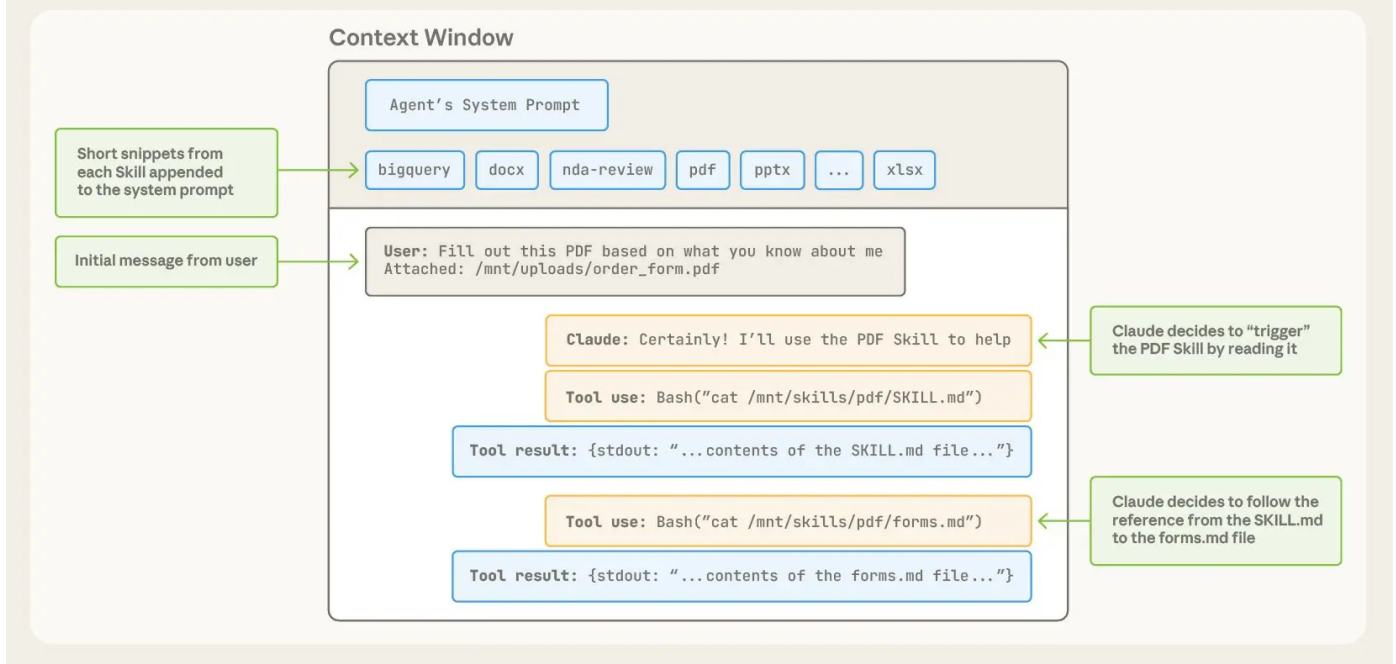
Level	File	Context Window	# Tokens
1	SKILL.md Metadata (YAML)	Always loaded	~100
2	SKILL.md Body (Markdown)	Loaded when Skill triggers	<5k
3+	Bundled files (text files, scripts, data)	Loaded as-needed by Claude	unlimited*

Agents with a filesystem and code execution tools don't need to read the entirety of a skill into their context window when working on a particular task. This means that the amount of context that can be bundled into a skill is effectively unbounded.

## Skills and the context window

The following diagram shows how the context window changes when a skill is triggered by a user's message.

# Skills and the Context Window



Skills are triggered in the context window via your system prompt.

The sequence of operations shown:

1. To start, the context window has the core system prompt and the metadata for each of the installed skills, along with the user's initial message;
2. Claude triggers the PDF skill by invoking a Bash tool to read the contents of **pdf/SKILL.md** ;
3. Claude chooses to read the **forms.md** file bundled with the skill;
4. Finally, Claude proceeds with the user's task now that it has loaded relevant instructions from the PDF skill.

## Skills and code execution

Skills can also include code for Claude to execute as tools at its discretion.

Large language models excel at many tasks, but certain operations are better suited for traditional code execution. For example, sorting a list via token generation is far more expensive than simply running a sorting algorithm. Beyond efficiency concerns, many applications require the deterministic reliability that only code can provide.

In our example, the PDF skill includes a pre-written Python script that reads a PDF and extracts all form fields. Claude can run this script without loading either the script or the PDF into context. And because code is deterministic, this workflow is consistent and repeatable.

# Bundling executable scripts

pdf/forms.md

```
If you need to fill out a PDF form, first check
to see if the PDF has fillable form fields.
Run this script from this file's directory:
`python scripts/check_fillable_fields <file.pdf>`,
and depending on the result go to either the
"Fillable fields"
or "Non-fillable fields" and follow those
instructions.

# Fillable fields If the PDF has fillable form
fields:
- Run this script from this file's directory:
`python ./extract_fields.py
<input.pdf> <fields.json>`.
...
```

pdf/extract\_fields.py

```
from pypdf import PdfReader

def write_field_info(pdf_path: str, output_path: str):
    """Extract form fields from PDF and store as JSON."""
    reader = PdfReader(pdf_path)
    fields = get_fields(reader)
    with open(output_path, "w") as f:
        json.dump(fields, f)

# ... omitted ...

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print(f"Usage: python {sys.argv[0]} <pdf_path> <output_json_path>")
        sys.exit(1)
    write_field_info(sys.argv[1], sys.argv[2])
```

Skills can also include code for Claude to execute as tools at its discretion based on the nature of the task.

## Developing and evaluating skills

Here are some helpful guidelines for getting started with authoring and testing skills:

- **Start with evaluation:** Identify specific gaps in your agents' capabilities by running them on representative tasks and observing where they struggle or require additional context. Then build skills incrementally to address these shortcomings.
- **Structure for scale:** When the `SKILL.md` file becomes unwieldy, split its content into separate files and reference them. If certain contexts are mutually exclusive or rarely used together, keeping the paths separate will reduce the token usage. Finally, code can serve as both executable tools and as documentation. It should be clear whether Claude should run scripts directly or read them into context as reference.
- **Think from Claude's perspective:** Monitor how Claude uses your skill in real scenarios and iterate based on observations: watch for unexpected trajectories or overreliance on certain contexts. Pay special attention to the `name` and `description` of your skill. Claude will use these when deciding whether to trigger the skill in response to its current task.
- **Iterate with Claude:** As you work on a task with Claude, ask Claude to capture its successful approaches and common mistakes into reusable context and code within a skill. If it goes off track when using a skill to complete a task, ask it to self-reflect on what went wrong. This process will help you discover what context Claude actually needs, instead of trying to anticipate it upfront.

## Security considerations when using Skills

Skills provide Claude with new capabilities through instructions and code. While this makes them powerful, it also means that malicious skills may introduce vulnerabilities in the environment where they're used or direct Claude to exfiltrate data and take unintended actions.

We recommend installing skills only from trusted sources. When installing a skill from a less-trusted source, thoroughly audit it before use. Start by reading the contents of the files bundled in the skill to understand what it does, paying particular attention to code dependencies and bundled resources like images or scripts. Similarly, pay attention to instructions or code within the skill that instruct Claude to connect to potentially untrusted external network sources.

## The future of Skills

---

Agent Skills are [supported today](#) across [Claude.ai](#), Claude Code, the Claude Agent SDK, and the Claude Developer Platform.

In the coming weeks, we'll continue to add features that support the full lifecycle of creating, editing, discovering, sharing, and using Skills. We're especially excited about the opportunity for Skills to help organizations and individuals share their context and workflows with Claude. We'll also explore how Skills can complement [Model Context Protocol](#) (MCP) servers by teaching agents more complex workflows that involve external tools and software.

Looking further ahead, we hope to enable agents to create, edit, and evaluate Skills on their own, letting them codify their own patterns of behavior into reusable capabilities.

Skills are a simple concept with a correspondingly simple format. This simplicity makes it easier for organizations, developers, and end users to build customized agents and give them new capabilities.

We're excited to see what people build with Skills. Get started today by checking out our Skills [docs](#) and [cookbook](#).

## Acknowledgements

---

Written by Barry Zhang, Keith Lazuka, and Mahesh Murag, who all really like folders. Special thanks to the many others across Anthropic who championed, supported, and built Skills.