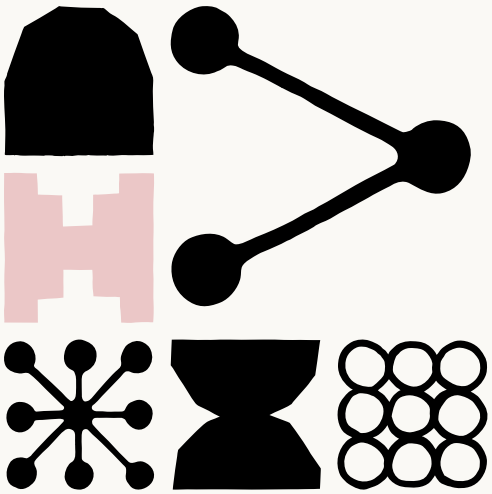


Equipping agents for the real world with Agent Skills

Published Oct 16, 2025

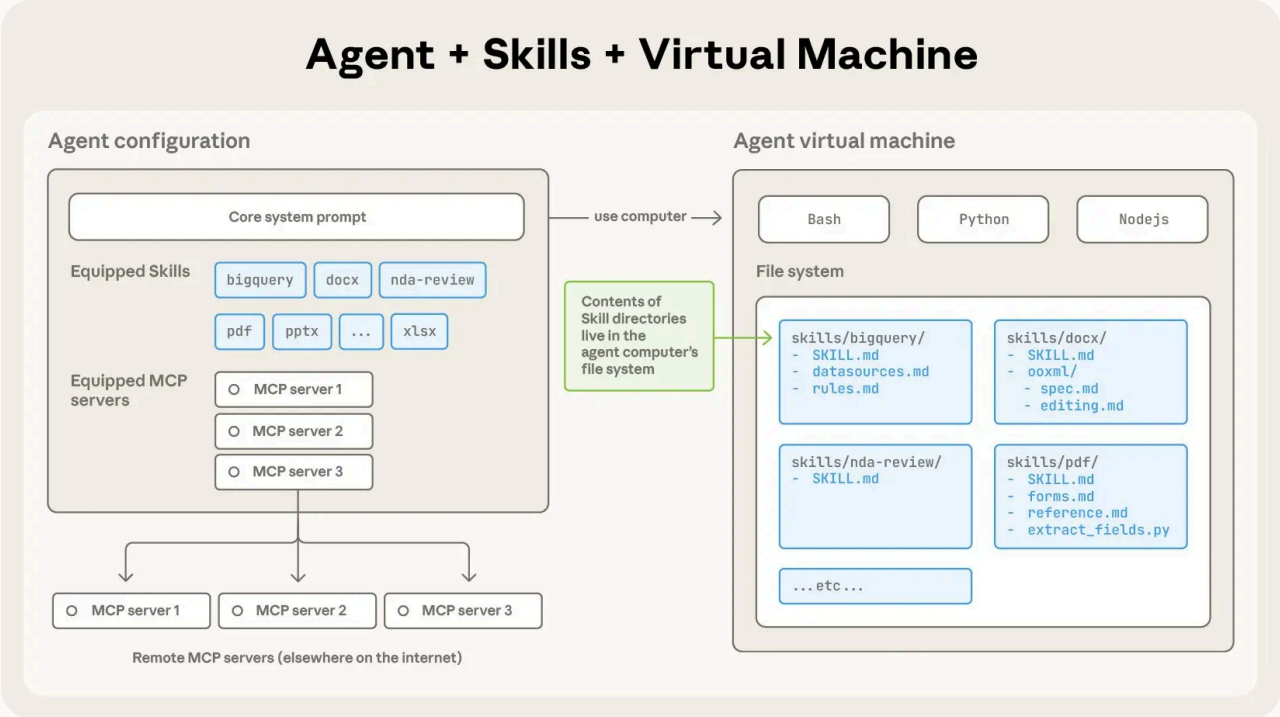
Claude is powerful, but real work requires procedural knowledge and organizational context. Introducing Agent Skills, a new way to build specialized agents using files and folders.

Update: We've published Agent Skills as an open standard for cross-platform portability. (December 18, 2025)

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.



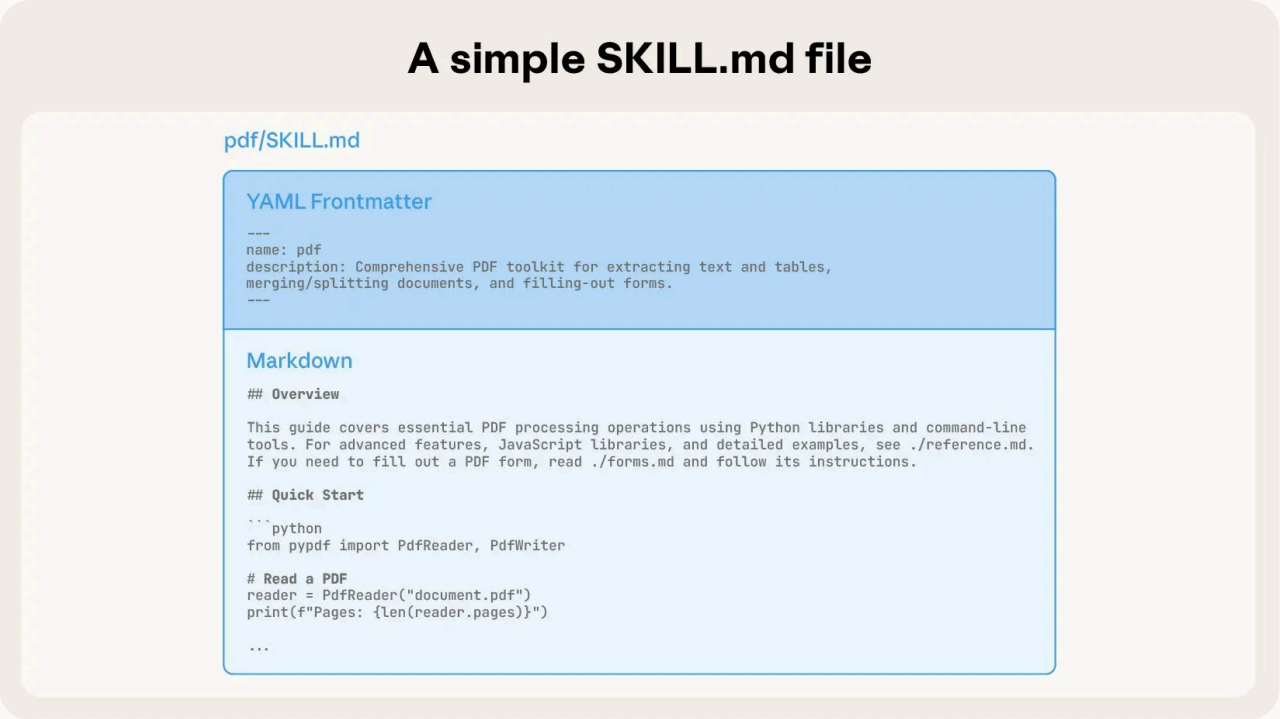
A skill is a directory containing a SKILL.md file that contains organized folders of instructions, scripts, and resources that give agents additional capabilities.

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.

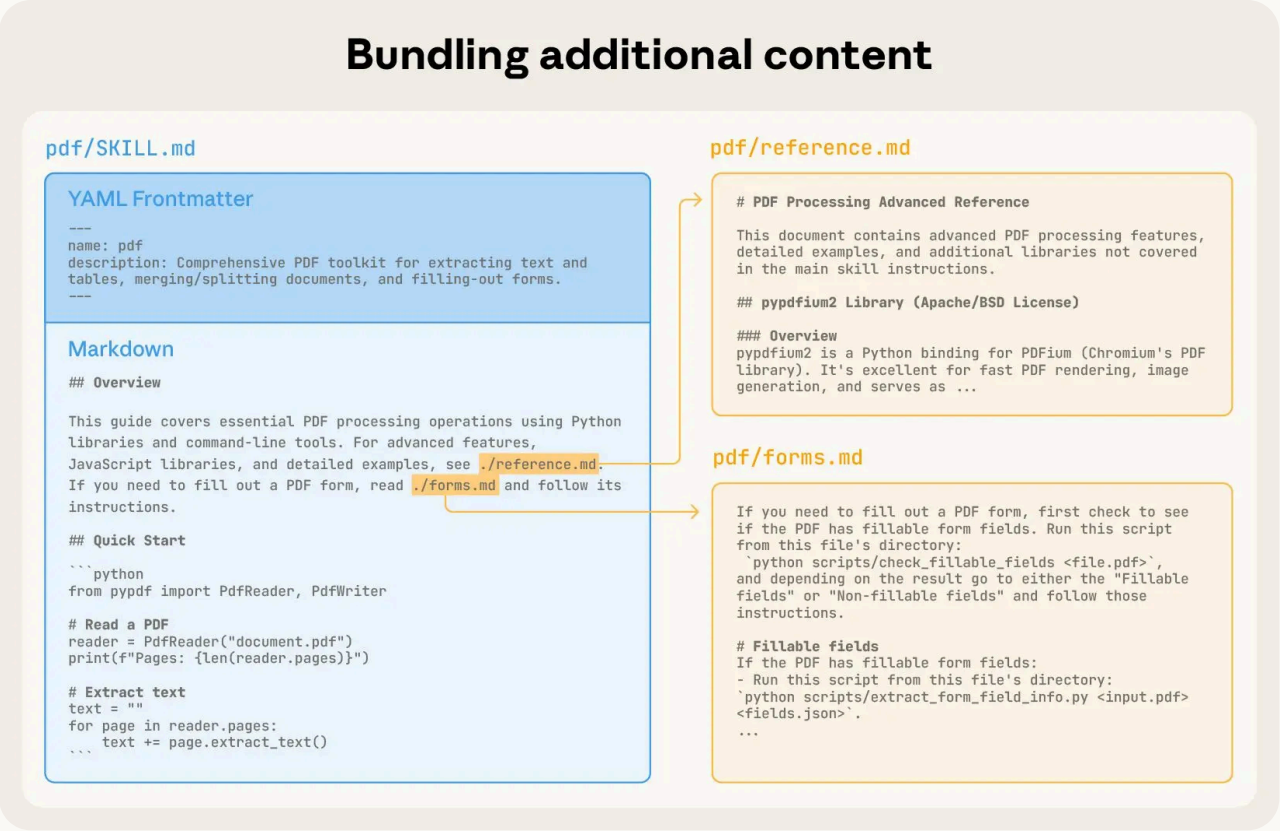
This metadata is the **first level** of *progressive disclosure*: it provides just enough information for Claude to know when each skill should be used without loading all of it into context. The actual body of this file is the **second level** of detail. If Claude thinks the skill is relevant to the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



A 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.



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

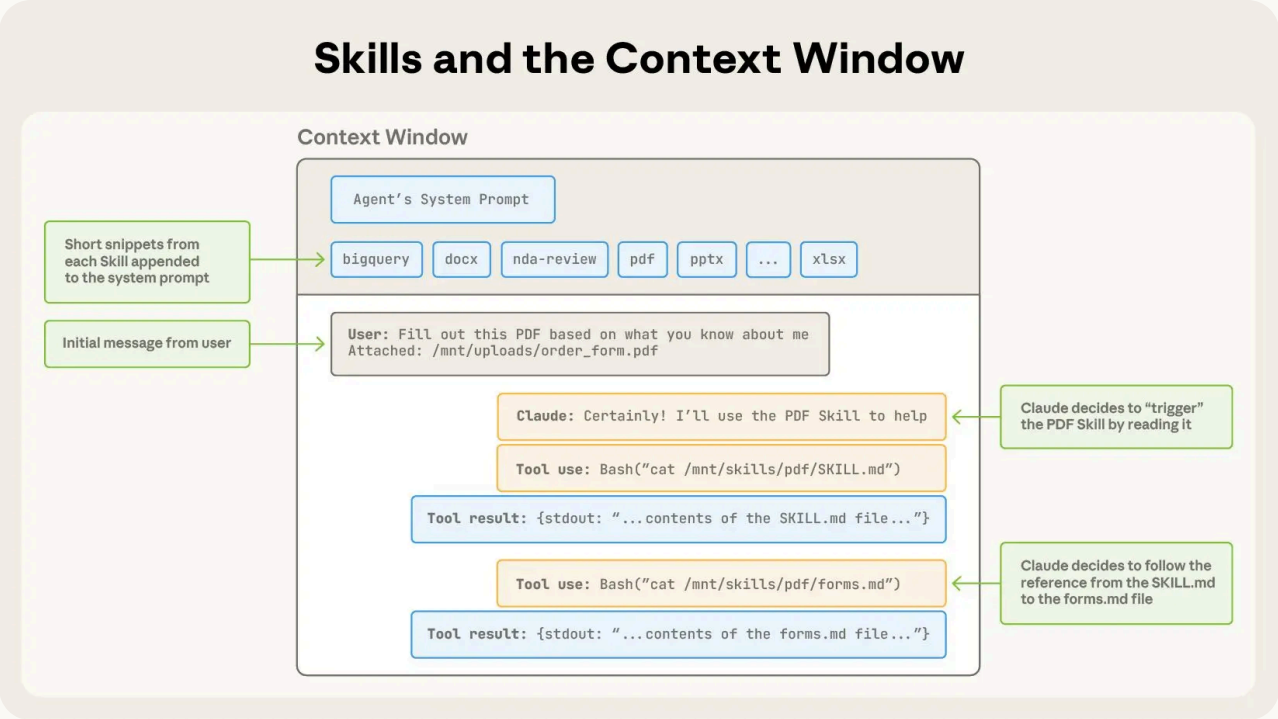
Progressive disclosure is the core design principle that makes Agent Skills flexible and scalable. Like a well-organized manual that starts with a table of contents, then specific chapters, and finally a detailed appendix, skills let Claude load information only as needed:

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 are triggered in the context window via your system prompt.

The sequence of operations shown:

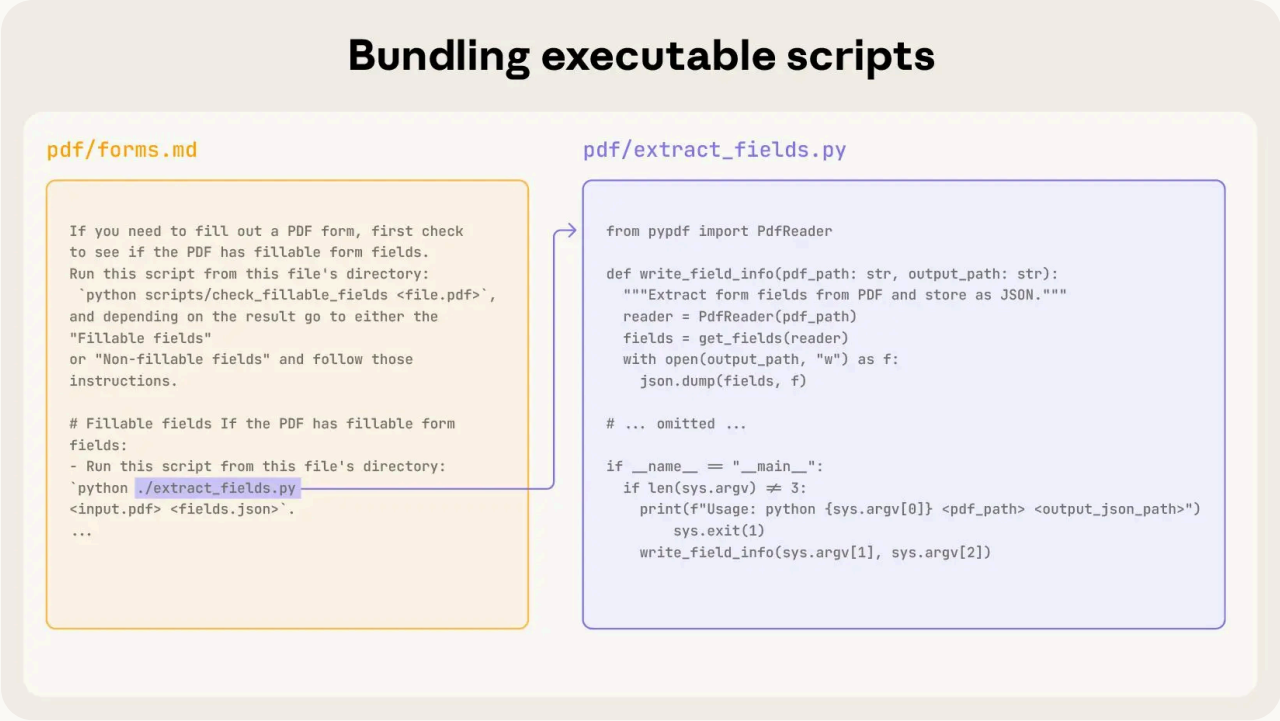
1. To start, the context window has the core system prompt and the metadata for each of the installed skills, along with the user's initial message;
2. Claude triggers the PDF skill by invoking a Bash tool to read the contents of `pdf/SKILL.md`;
3. Claude chooses to read the `forms.md` file bundled with 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.



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.

Enter your email

Please provide your email address if you'd like to receive our monthly developer newsletter. You can unsubscribe at any time.

Nonprofits

Learn

Blog

Claude partner network

Connectors

Courses

Customer stories

Engineering at Anthropic

Events

Powered by Claude

Service partners

Startups program

Use cases

Company

Anthropic

Careers

Economic Futures

Research

News

Responsible Scaling Policy

Security and compliance

Transparency

Help and security

Availability

Status

Support center

Terms and policies

Privacy choices

Privacy policy

Responsible disclosure policy

Terms of service: Commercial

Terms of service: Consumer

Usage policy