23

Agent Skills でエージェントを実世界に備えさせる

— Equipping agents for the real world with Agent Skills —

公開日	2025-10-16
原題	Equipping agents for the real world with Agent Skills
著者	Anthropic Engineering Team
原文	https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills
翻訳	Claude（機械翻訳／Anthropic）
編集	2026-04-17

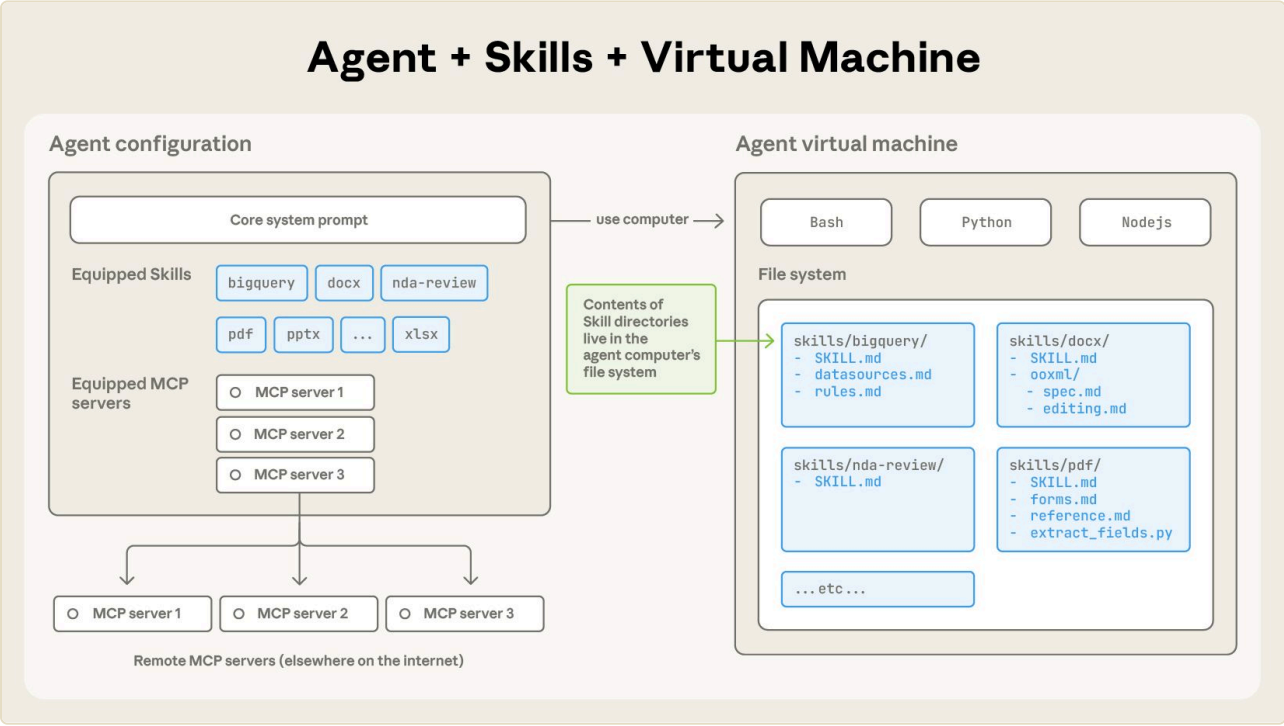
Agent Skills でエージェントを実世界に備えさせる

追記: Agent Skills を、プラットフォームをまたいで持ち運べるオープン標準として公開しました。(2025 年 12 月 18 日)

モデルの能力が高まるにつれて、本格的な計算環境とやり取りする汎用エージェントを作れるようになってきました。たとえば Claude Code は、ローカルでのコード実行とファイルシステムを使って、領域をまたいだ複雑なタスクをこなせます。しかし、こうしたエージェントが強力になるほど、領域固有の専門知識を与えるための、より組み合わせやすく、スケールしやすく、移植性のある方法が必要になります。

そこで私たちが作ったのが **Agent Skills** です。skill とは、特定のタスクでエージェントがよりうまく振る舞えるように、動的に発見・読み込みができる、指示・スクリプト・リソースをまとめた整理されたフォルダです。Skills は、あなたの専門知識を Claude 向けの組み合わせ可能なリソースとしてパッケージ化することで Claude の能力を拡張し、汎用エージェントをあなたのニーズに合った特化型エージェントへと変身させます。

エージェントのために skill を作るのは、新入社員のためにオンボーディングガイドをまとめるのに似ています。ユースケースごとに断片的で特注のエージェントを作るのではなく、誰もが手続き的な知識を切り出して共有することで、組み合わせ可能な能力でエージェントを特化できるようになりました。本稿では、Skills とは何か、どう動くのかを説明し、自分で作るためのベストプラクティスを共有します。



skill とは、SKILL.md ファイルを含むディレクトリであり、エージェントに追加の能力を与える指示・スクリプト・リソースの整理されたフォルダです。

skill の構造

Skills がどう動くかを見るために、実例を 1 つ追ってみましょう。先ごろ公開された Claude の文書編集能力を支えている skill の 1 つです。Claude はすでに PDF を理解することに長けていますが、PDF を直接操作する（たとえばフォームに記入する）能力は限定的です。この PDF skill は、Claude にそうした新しい能力を与えるためのものです。

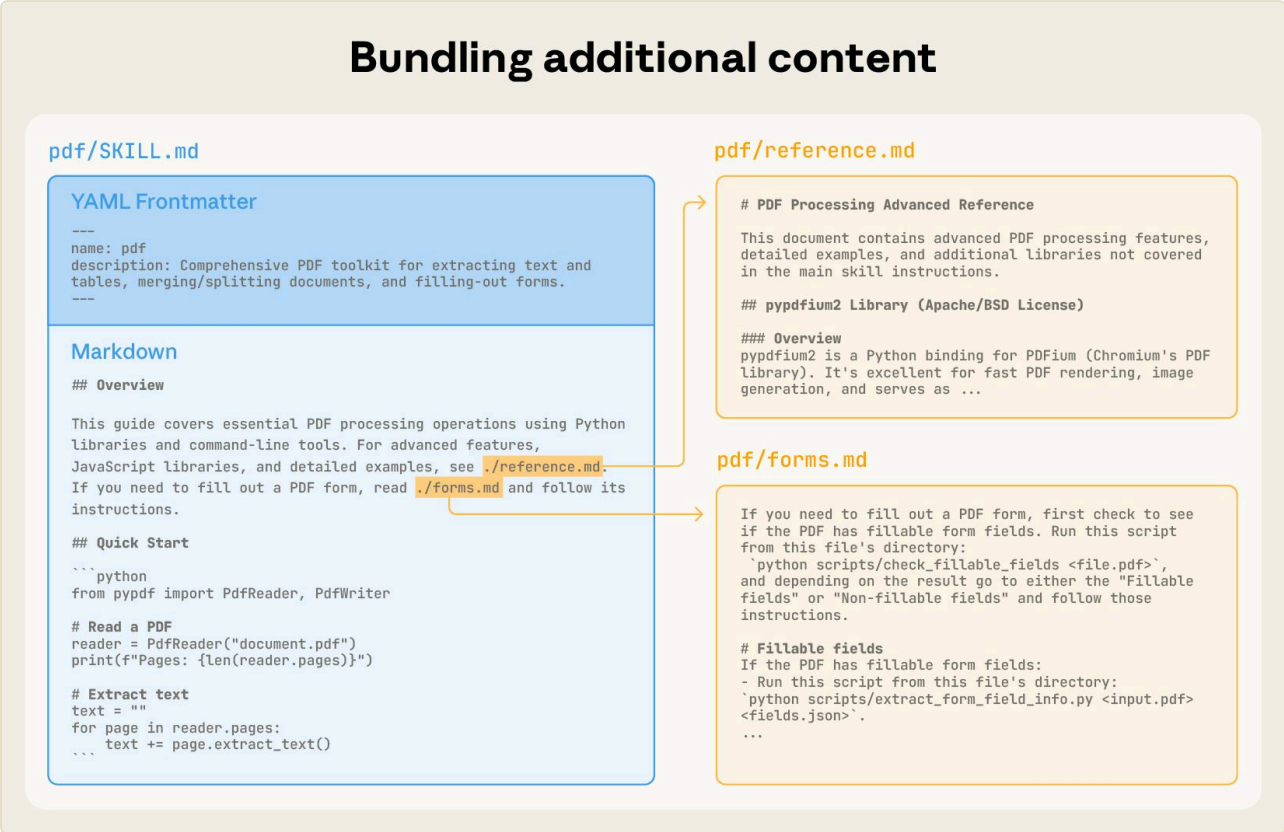
最もシンプルな形の skill は、`SKILL.md file` を含むディレクトリです。このファイルは、`name` と `description` という必須のメタデータを含む YAML フロントマターで始まらなければなりません。起動時に、エージェントはインストール済みのすべての skill の `name` と `description` をシステムプロンプトに事前に読み込みます。

このメタデータが、*progressive disclosure（漸進的開示）* の **第 1 層** です。すべてをコンテキストに読み込まなくても、いつ各 skill を使うべきかを Claude が判断できるだけの情報を提供します。このファイルの本体は、**第 2 層** の詳細です。Claude が現在のタスクにその skill が関連しそうだと判断した場合、`SKILL.md` の全文をコンテキストに読み込んで skill を展開します。

SKILL.md ファイルは、ファイル名と説明を含む YAML フロントマターで始まり、起動時にシステムプロンプトに読み込まれます。

skill が複雑になるにつれて、単一の `SKILL.md` に収まりきらないほど多くの文脈を含んだり、特定の場面でしか関係しない文脈が出てきたりします。そうした場合、skill は skill ディレクトリ内に追加のファイルをバンドルし、`SKILL.md` から名前で参照できます。これら追加のリンク先ファイルが、詳細の **第 3 層**（およびそれ以降）であり、Claude は必要に応じて辿って発見するかを選べます。

以下に示す PDF skill では、`SKILL.md` が skill 作者によってコアの `SKILL.md` と一緒にバンドルされた 2 つの追加ファイル（`reference.md` と `forms.md`）を参照しています。フォーム記入の指示を別ファイル（`forms.md`）に移すことで、skill 作者は、Claude がフォームを埋めるときだけ `forms.md` を読むと信頼しつつ、skill のコアを軽く保つことができます。



（追加ファイルによって）より多くの文脈を skill に組み込むと、Claude がシステムプロンプトに基づいて必要なタイミングでそれらを引き出せます。

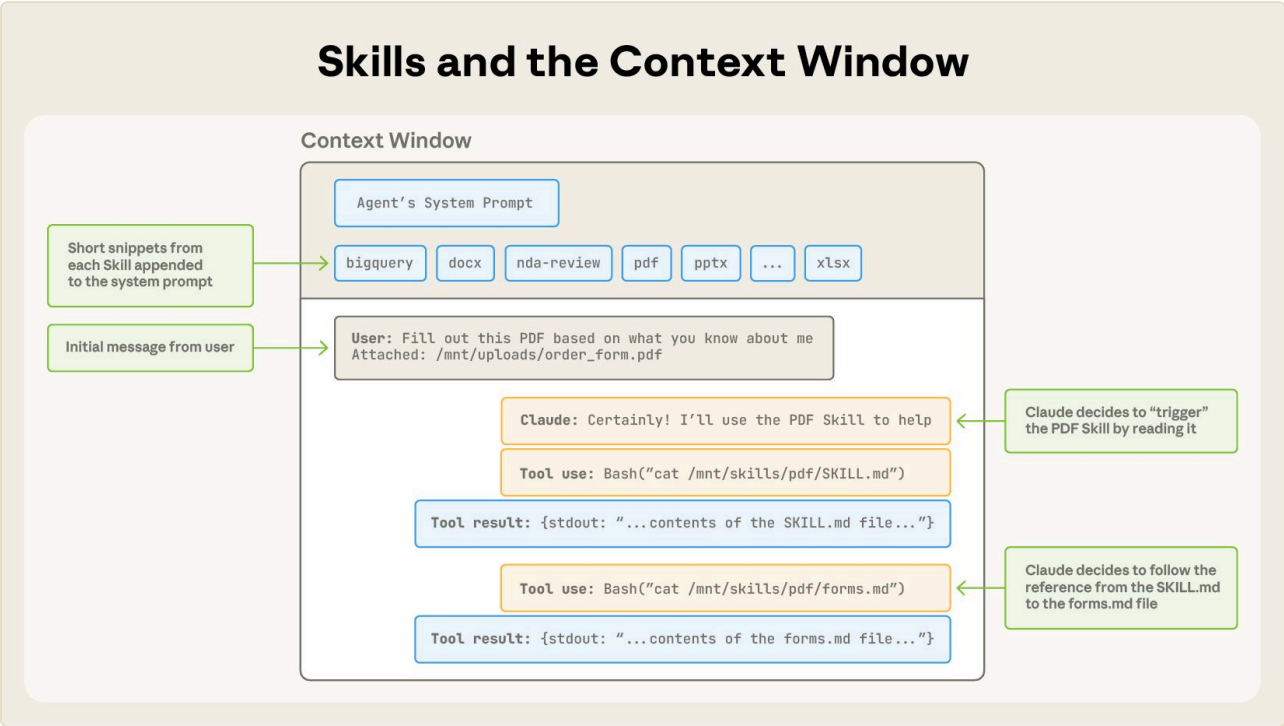
progressive disclosure（漸進的開示）は、Agent Skills を柔軟でスケーラブルにしているコアの設計原則です。よく整理されたマニュアルが、目次から始まり、個別の章に進み、最後に詳細な付録へと至るように、skill は Claude が必要に応じてのみ情報を読み込めるようにしてくれます。

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*

ファイルシステムとコード実行ツールを持つエージェントは、特定のタスクに取り組むときに skill 全体をコンテキストウィンドウに読み込む必要はありません。これは、skill にバンドルできる文脈の量が事実上無制限であることを意味します。

skill とコンテキストウィンドウ

次の図は、ユーザーのメッセージによって skill がトリガーされたときに、コンテキストウィンドウがどう変化するかを示しています。



skill は、システムプロンプトを経由してコンテキストウィンドウの中でトリガーされます。

図に示される一連の動作は次のとおりです。

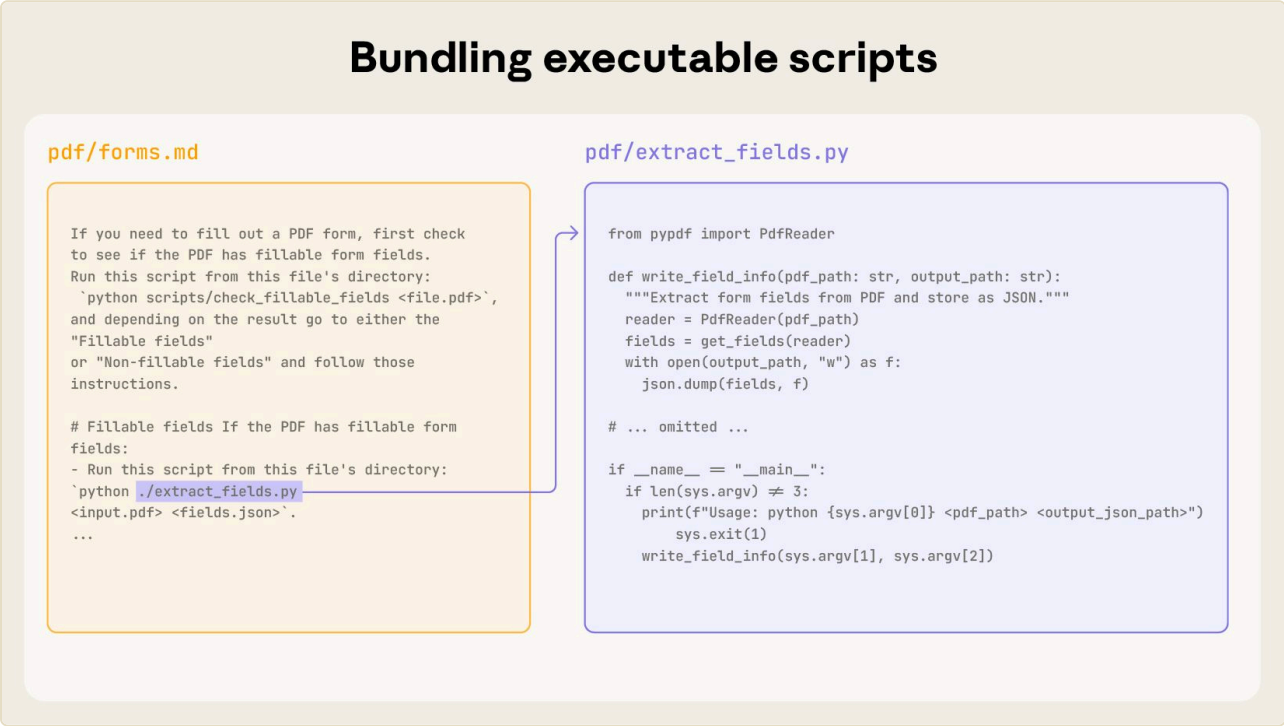
1. 最初、コンテキストウィンドウには、コアのシステムプロンプトと、インストール済みの各 skill のメタデータ、そしてユーザーの最初のメッセージがあります。
2. Claude は Bash ツールを呼び出して `pdf/SKILL.md` の中身を読み、PDF skill を発動させます。
3. Claude は、その skill にバンドルされた `forms.md` ファイルを読むことを選びます。
4. 最後に Claude は、PDF skill から関連する指示を読み込んだ状態でユーザーのタスクを進めます。

skill とコード実行

skill には、Claude が自らの裁量でツールとして実行できるコードを含めることもできます。

大規模言語モデルは多くのタスクに優れていますが、操作によっては従来のコード実行のほうが向いているものもあります。たとえば、リストをトークン生成でソートするのは、単にソートアルゴリズムを走らせるよりはるかにコストがかかります。効率だけでなく、多くのアプリケーションは、コードだけが提供できる決定論的な信頼性を必要とします。

本稿の例では、PDF skill は、PDF を読み込んですべてのフォームフィールドを抽出する、あらかじめ書かれた Python スクリプトを含んでいます。Claude は、このスクリプトもその入力 PDF もコンテキストに読み込まずにスクリプトを実行できます。そしてコードは決定論的であるため、このワークフローは一貫性があり、再現可能です。



skill は、タスクの性質に応じて、Claude が裁量でツールとして実行できるコードを含めることもできます。

skill の開発と評価

skill の執筆とテストを始めるにあたっての指針をいくつか挙げておきます。

- **評価から始める:** エージェントを代表的なタスクで走らせ、どこで詰まるか、どこで追加の文脈を必要とするかを観察することで、エージェントの能力における具体的な穴を特定します。そのうえで、そうした不足点を埋めるために skill を段階的に作っていきます。
- **スケールを見据えた構造にする:** `SKILL.md` ファイルが手に負えないほど大きくなってきたら、中身を別ファイルに分割して参照させます。互いに排他的な文脈や、ほとんど同時に使われない文脈は、経路を分けておくとトークン使用量を減らせます。また、コードは実行可能なツールとしても、ドキュメントとしても使

えます。Claude にスクリプトを直接実行させるのか、それともリファレンスとしてコンテキストに読み込ませるのかが明確である必要があります。

- **Claude の視点で考える:** 実際のシナリオで Claude が skill をどう使うかを観察し、その観察に基づいて反復します。予想外の経路や、特定の文脈への依存のしすぎがないか見張ってください。skill の `name` と `description` には特に注意を払いましょう。Claude は現在のタスクに応じて skill を発動させるかどうかを判断する際に、これらを使います。
- **Claude とともに反復する:** Claude と一緒にタスクに取り組みながら、うまくいったアプローチやよくある失敗を、skill 内で再利用可能な文脈とコードへとまとめさせていきます。skill を使ってタスクを進める途中で脱線したら、何が悪かったのかを自己省察させましょう。このプロセスは、先回りして推測するのではなく、Claude が実際にどんな文脈を必要としているかを発見するのに役立ちます。

Skills を使うときのセキュリティ上の考慮

Skills は、指示とコードによって Claude に新しい能力を与えます。これは強力である反面、悪意のある skill が使用環境に脆弱性を持ち込んだり、Claude を誘導してデータを流出させたり意図しない動作を取らせたりする可能性がある、ということでもあります。

skill は信頼できるソースからのみインストールすることを推奨します。信頼しきれないソースから skill をインストールする場合は、使用前に徹底的に監査してください。まずは skill にバンドルされたファイルの中身を読み、何をするものかを理解するところから始めます。特にコードの依存関係や、画像・スクリプトといったバンドルされたリソースに注意を払ってください。同様に、信頼できない可能性のある外部ネットワークに Claude を接続させるような指示やコードが skill 内に含まれていないかにも注意しましょう。

Skills のこれから

Agent Skills は、Claude.ai、Claude Code、Claude Agent SDK、Claude Developer Platform にわたって本日より利用可能 です。

今後数週間で、Skills を作成・編集・発見・共有・利用するライフサイクル全体を支える機能を追加していきます。特に、Skills が組織や個人にとって自分たちの文脈とワークフローを Claude と共有する手段になり得ることには、大きな期待を寄せています。また、Model Context Protocol (MCP) サーバーを補完する形で、外部のツールやソフトウェアを伴うより複雑なワークフローをエージェントに教える手段としての Skills も探っていきます。

さらに先を見据えると、エージェント自身が Skills を作成・編集・評価できるようにし、自分自身の振る舞いのパターンを再利用可能な能力として明文化できるようにしたいと考えています。

Skills はシンプルな概念であり、それに対応してシンプルな形式を持っています。この単純さのおかげで、組織・開発者・エンドユーザーがカスタマイズされたエージェントを作り、新しい能力を持たせることが容易になります。

皆さんが Skills で何を作るのか、楽しみにしています。Skills の ドキュメント と クックブック をぜひ覗いて、今日から始めてみてください。

謝辞

執筆は Barry Zhang、Keith Lazuka、Mahesh Murag。3 人ともフォルダが本当に好きです。Skills を支え、育て、実装してくれた Anthropic の多くの仲間たちに特別な感謝を捧げます。