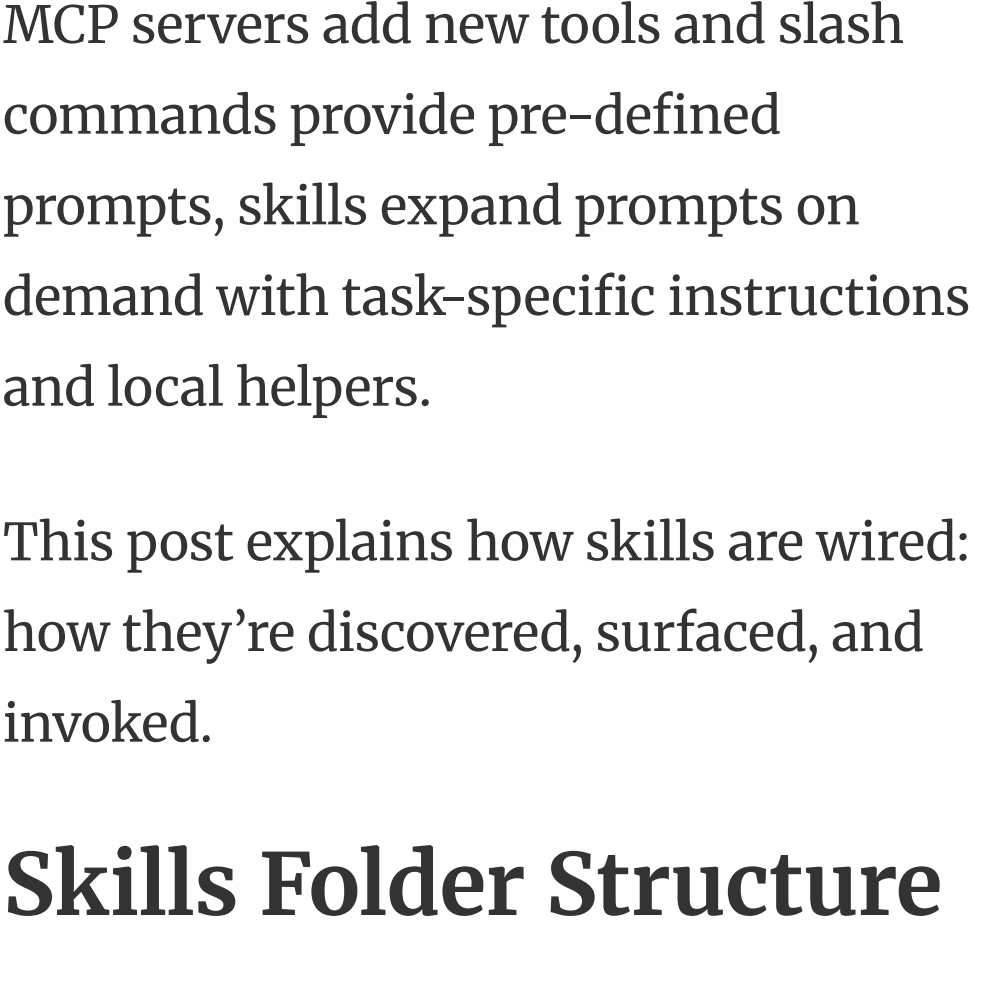


Inside Claude Code Skills: Structure, prompts, invocation

Published on Oct 28, 2025 · 5 min read



Claude Code recently added **skills** as an extensibility mechanism alongside MCP servers and slash commands. While MCP servers add new tools and slash commands provide pre-defined prompts, skills expand prompts on demand with task-specific instructions and local helpers.

This post explains how skills are wired: how they're discovered, surfaced, and invoked.

Skills Folder Structure

Skills are folders containing a `SKILL.md` file and optional scripts or other resources. Sub-folders are allowed (and encouraged) for organizing helper scripts, templates, and data files. Here's an example with two skills:

```
.claude/skills/
├── pdf/
│   ├── SKILL.md
│   ├── extract_text.py
│   └── templates/
│       └── summary.html
└── csv/
    ├── SKILL.md
    ├── analyze.py
    └── utils/
        ├── parser.py
        └── visualizer.py
```

The `pdf/SKILL.md` might contain:

```
---
name: pdf
description: Extract and analyze text from PDF documents
---

# PDF Processing Skill

Use the extract_text.py script in this folder to process PDF files.

python3 extract_text.py <input_file>

After extraction, summarize the key points.
```

Each skill packages instructions alongside executable scripts and reference materials, creating self-contained capability extensions.

Skill Tool Definition

Claude Code provides a **Skill** tool to Claude. Here's the complete tool definition (captured from an actual Claude Code session):

```
## Skill

**Input Schema:**

object:
  - command (required):
    string
    # The skill name (no arguments). E.g. 'pdf'

**Description:**

Execute a skill within the main conversation.

<skills_instructions>
When users ask you to perform tasks, check below can help complete the task more effectively by leveraging available capabilities and domain knowledge.

How to use skills:
- Invoke skills using this tool with the skill name.
- When you invoke a skill, you will see <command> in the response.
- The skill's prompt will expand and provide detailed instructions.
- Examples:
  - `command: "pdf"` - invoke the pdf skill
  - `command: "xlsx"` - invoke the xlsx skill
  - `command: "ms-office-suite:pdf"` - invoke the ms-office-suite skill

Important:
- Only use skills listed in <available_skills>.
- Do not invoke a skill that is already running.
- Do not use this tool for built-in CLI commands.

</skills_instructions>

<available_skills>
[Skills are listed here - see next section]
</available_skills>
```

The tool is simple: just a `command` parameter with the skill name. But the description contains both instructions for using skills and an embedded `<available_skills>` section.

Available Skills List

Within the tool definition's description, Claude Code builds an `<available_skills>` section based on the skill folders. The `name` and `description` fields come directly from the YAML frontmatter in each skill's `SKILL.md` file:

```
<available_skills>
<skill>
  <name>pdf</name>
  <description>
    Extract and analyze text from PDF documents. Use this skill to process or read PDFs.
  </description>
  <location>user</location>
</skill>
<skill>
  <name>csv</name>
  <description>
    Analyze and visualize CSV data. Use this skill to process or analyze CSV files.
  </description>
  <location>user</location>
</skill>
</available_skills>
```

Notice how the `pdf` skill's name and description match exactly what was defined in the `pdf/SKILL.md` frontmatter shown earlier. The metadata includes:

- **name:** From frontmatter, used as identifier for invoking the skill
- **description:** From frontmatter, tells Claude when to use this skill
- **location:** Either user (machine-scoped) or project (defined in the current folder)

The list of skills is embedded in the tool definition.

Runtime Invocation

Skills operate via a tool call/tool response pair. For example, when the user asks "Extract text from report.pdf", Claude recognizes this matches the `pdf` skill and sends an assistant message with a tool use:

```
{
  "role": "assistant",
  "content": [
    {
      "type": "text",
      "text": "Now let me use the pdf skill to extract text from report.pdf."
    },
    {
      "type": "tool_use",
      "id": "toolu_01JRBZGD3vy9gDsifuT89L8f...",
      "name": "Skill",
      "input": {
        "command": "pdf"
      }
    }
  ]
}
```

The system responds with a user message containing a `tool_result` and two `text` blocks:

```
{
  "role": "user",
  "content": [
    {
      "type": "tool_result",
      "tool_use_id": "toolu_01JRBZGD3vy9gDsifuT89L8f...",
      "content": "Launching skill: pdf"
    },
    {
      "type": "text",
      "text": "<command-message>The \"pdf\" skill was invoked."
    },
    {
      "type": "text",
      "text": "Base Path: /Users/username/.claude"
    }
  ]
}
```

The third block contains both the skill's base path and the `SKILL.md` body (without frontmatter). The base path enables Claude Code to locate and execute scripts bundled with the skill relative to that folder.

From this point, Claude Code follows the expanded instructions: it runs the extraction script, processes the output, and creates a summary. Skills aren't separate processes, sub-agents, or external tools: they're injected instructions that guide Claude's behavior within the main conversation.

Conclusion

Skills in Claude Code are a simple mechanism for extensibility. The mechanics:

1. **Folder structure:** Each skill is a folder with `SKILL.md` (containing YAML frontmatter and instructions) plus optional scripts/resources
2. **Tool definition:** The Skill tool embeds an `<available_skills>` list built from the frontmatter of all skills
3. **Runtime invocation:** When invoked, the tool response includes the base path and `SKILL.md` body, expanding the context and referencing additional resources

What makes this design clever is that it achieves on-demand prompt expansion without modifying the core system prompt. Skills are executable knowledge packages that Claude loads only when needed, extending capabilities while keeping the main prompt lean.

The complete tool definition shown here (captured from actual Claude Code sessions) hasn't been published by Anthropic, making this reverse-engineered view a novel contribution to understanding how Claude Code's extensibility works under the hood.

Author: Mikhail Shilkov
Cloud and AI developer and researcher. Principal engineer at Pulum.

Follow @MikhailShilkov

Responses Write a response

Visit the [Github Issue](#) to comment on this page