SKILLS学习

概述

一种简单开放的格式，旨在为agent提供新的能力和专业知识。 agent技能文件夹包含各种指令、脚本和资源，agent可以查找并使用这些资源来更准确、更高效地完成工作。

为什么需要SKILLS?

智能体的功能日益强大，但往往缺乏可靠完成实际工作所需的上下文信息。技能通过赋予智能体访问程序知识以及公司、团队和用户特定上下文信息的能力来解决这一问题，智能体可以按需加载这些信息。拥有技能集的智能体可以根据正在执行的任务扩展自身能力。

- 对于技能开发者：只需构建一次功能，即可将其部署到多个智能体产品中。
- 对于兼容的智能体：对技能的支持使用户能够开箱即用地为智能体赋予新功能。
- 对于团队和企业：将组织知识捕获到可移植的、版本控制的软件包中。

SKILLS可以实现什么?

- 领域专业知识：将专业知识打包成可重用的指令，涵盖从法律审查流程到数据分析管道等各个方面。
- 新功能：赋予智能体新功能（例如，创建演示文稿、构建 MCP 服务器、分析数据集）。
- 可重复的工作流程：将多步骤任务转化为一致且可审计的工作流程。
- 互操作性：在不同的、技能兼容的智能体产品中重用相同的技能。

什么是SKILLS?

智能体技能是一种轻量级的开放格式，用于通过专业知识和工作流程扩展人工智能智能体的功能。

技能的核心是一个包含 SKILL.md 文件的文件夹。该文件包含元数据（至少包括名称和描述）以及指示智能体如何执行特定任务的指令。技能还可以包含脚本、模板和参考资料。

代码块

```
my-skill/
├── SKILL.md          # Required: instructions + metadata
├── scripts/          # Optional: executable code
├── references/       # Optional: documentation
```

```
└── assets/           # Optional: templates, resources
```

技能的工作原理

技能采用渐进式披露来高效管理上下文：

- 发现：启动时，代理仅加载每个可用技能的名称和描述，仅足以了解其何时可能相关。
- 激活：当任务与技能描述匹配时，代理会读取完整的 SKILL.md 指令并将其整合到上下文中。
- 执行：代理遵循指令，并可根据需要加载引用的文件或执行捆绑代码。

这种方法既能保持代理的快速运行，又能使其按需访问更多上下文。

SKILL.md 文件

每个技能都以一个包含 YAML 前置元数据和 Markdown 指令的 SKILL.md 文件开始：

代码块

```
---
name: pdf-processing
description: Extract text and tables from PDF files, fill forms, merge documents.
---

# PDF Processing

## When to use this skill
Use this skill when the user needs to work with PDF files...

## How to extract text
1. Use pdfplumber for text extraction...

## How to fill forms
...
```

SKILL.md 文件顶部必须包含以下 frontmatter：

- name：简短的标识符
- description：何时使用此技能

Markdown 正文包含实际的指令，对结构或内容没有任何特定限制。

这种简单的格式具有以下几个关键优势：

- 自文档化：技能作者或用户可以阅读 SKILL.md 文件并理解其功能，从而便于技能的审核和改进。
- 可扩展性：技能的复杂程度可以从纯文本指令到可执行代码、资源和模板不等。
- 可移植性：技能只是文件，因此易于编辑、版本控制和共享。

后续步骤

- 查看规范以了解完整格式。
- 将技能支持添加到您的代理，以构建兼容的客户端。
- 在 GitHub 上查看示例技能。
- 阅读编写有效技能的最佳实践。
- 使用参考库验证技能并生成提示 XML。

格式

目录结构

一个技能是一个目录，其中至少包含一个 SKILL.md 文件：

```
skill-name/
└── SKILL.md          # Required
```

📌 您还可以选择添加其他目录，例如 scripts/、references/ 和 assets/，以增强您的技能。

SKILL.md 格式

`SKILL.md` 文件必须包含 YAML 前置元数据，后跟 Markdown 内容。

前置元数据（必填）

代码块

```
---
name: skill-name
description: 描述此技能的功能及其适用场景。

---
```

可选字段：

代码块

```
---
name: pdf-processing
description: 从 PDF 文件中提取文本和表格，填写表单，合并文档。

license: Apache-2.0

metadata:

author: example-org

version: "1.0"

---
```

字段	必填	限制
name	是	最多 64 个字符。仅限小写字母、数字和连字符。不能以连字符开头或结尾。
description	是	最多 1024 个字符。非空。描述技能的功能和使用时机。
license	否	许可证名称或捆绑许可证文件的引用。
compatibility	否	最多 500 个字符。指示环境要求（目标产品、系统软件包、网络访问等）。
metadata	否	用于附加元数据的任意键值映射。

allowed-tools	否	技能可使用的预先批准工具列表，以空格分隔。（实验性功能）

`name` 字段

必填的 `name` 字段：

- 必须为 1-64 个字符
- 只能包含 Unicode 小写字母数字字符和连字符（`a-z` 和 `-`）
- 不能以 `-` 开头或结尾
- 不能包含连续的连字符 (`--`)
- 必须与父目录名称匹配

有效示例：

```
代码块
1   name: pdf-processing
```

```
代码块
1   name: data-analysis
```

```
代码块
1   name: code-review
```

无效示例：

```
代码块
1   name: PDF-Processing # 不允许大写
```

```
代码块
1   name: -pdf # 不能以连字符开头
```

```
name: pdf--processing # 不允许连续的连字符
```

`description` 字段

必填的 `description` 字段：

- 必须为1-1024 个字符
- 应描述技能的功能及其适用场景
- 应包含帮助客服人员识别相关任务的特定关键词

优秀示例：

```
description: 从 PDF 文件中提取文本和表格，填写 PDF 表单，并合并多个 PDF 文件。适用于处理 PDF 文档或用户提及 PDF、表单或文档提取的情况。
```

不合格示例：

```
description: 帮助处理 PDF 文件。
```

`license` 字段

可选的 `license` 字段：

- 指定应用于该技能的许可证
- 我们建议保持简短（许可证名称或捆绑的许可证文件名称）

示例：

```
license: 专有。 LICENSE.txt 文件包含完整条款
```

`compatibility` 字段

可选的 `兼容性` 字段：

- 如果提供，则必须为 1-500 个字符
- 仅当您的技能有特定的环境要求时才应包含此字段
- 可以指明目标产品、所需的系统软件包、网络访问需求等。

示例：

```
compatibility: 专为 Claude Code（或类似产品）设计
```

```
compatibility: 需要 git、docker、jq 和互联网访问权限
```

大多数技能不需要 compatibility字段。

`metadata` 字段

可选的 `metadata` 字段：

- 一个从字符串键到字符串值的映射
- 客户端可以使用此字段存储 Agent Skills 规范中未定义的其他属性
- 我们建议您使用尽可能唯一的键名，以避免意外冲突

示例：

```
metadata:
  author: example-org
  version: "1.0"
```

`allowed-tools` 字段

可选的 `allowed-tools` 字段：

- 一个以空格分隔的已预先批准运行的工具列表
- 实验性功能。不同 Agent 实现对此字段的支持可能有所不同

示例：

```
```

```
1
2   allowed-tools: Bash(git:*) Bash(jq:*) Read
```

正文内容

前置元数据之后的 Markdown 正文包含技能指令。格式没有限制。编写任何有助于 Agent 有效执行任务的内容即可。

推荐章节：

- 分步说明
- 输入和输出示例
- 常见边界情况

请注意，代理一旦决定激活某项技能，就会加载整个文件。建议将较长的 `SKILL.md` 内容拆分为多个引用文件。

可选目录

scripts/

包含代理可以运行的可执行代码。脚本应：

- 独立运行或清晰地记录依赖关系
- 包含有用的错误信息
- 妥善处理边界情况

支持的语言取决于代理的实现。常用语言包括 Python、Bash 和 JavaScript

references/

包含代理在需要时可以阅读的其他文档：

- `REFERENCE.md` - 详细的技术参考
- `FORMS.md` - 表单模板或结构化数据格式
- 特定领域的文档（例如 `finance.md`、`legal.md` 等）

请保持各个[参考文件](#file-references)的重点突出。代理程序按需加载这些文件，因此文件越小，对上下文的使用就越少。

assets/

包含静态资源：

- 模板（文档模板、配置模板）
- 图片（图表、示例）

- 数据文件（查找表、模式）

渐进式披露

技能应结构化，以便高效利用上下文：

1. **Metadata**（约 100 个标记）：所有技能在启动时都会加载 `name` 和 `description` 字段。
2. **Instructions**（建议少于 5000 个标记）：技能激活时会加载完整的 `SKILL.md` 文件。
3. **Resources**（按需加载）：仅在需要时加载文件（例如，位于 `scripts/`、`references/` 或 `assets/` 中的文件）。

请将主 `SKILL.md` 文件控制在 500 行以内。将详细的参考资料移至单独的文件中。

文件引用

在技能中引用其他文件时，请使用相对于技能根目录的相对路径：

```
代码块
1  See [the reference guide](references/REFERENCE.md) for details.
2
3  Run the extraction script:
4  scripts/extract.py
5
6
7
8
9  详情请参阅[参考指南](references/REFERENCE.md)。
10
11 运行提取脚本：
12
13 scripts/extract.py
```

文件引用应保持在 `SKILL.md` 的一级。避免使用嵌套过深的引用链。

验证

使用 skills-ref 引用库来验证您的技能：

```
代码块
1
2  skills-ref validate ./my-skill
```

此命令会检查您的 `SKILL.md` 前置元数据是否有效，并符合所有命名规范。

使用script

技能可以指示代理运行 shell 命令，并将可重用的脚本打包在 scripts/ 目录中。本指南涵盖一次性命令、具有自身依赖项的独立脚本，以及如何为代理设计脚本接口。

一次性命令

如果现有软件包已经满足您的需求，则可以直接在 SKILL.md 指令中引用它，而无需使用 scripts/ 目录。许多生态系统都提供了在运行时自动解析依赖项的工具。

uvx

uvx 在隔离环境中运行 Python 包，并采用积极的缓存策略。它随 uv 一起提供。

```
代码块
1   uvx ruff@0.8.0 check .
2   uvx black@24.10.0 .
```

- 未与 Python 捆绑，需要单独安装。 速度快。
- 缓存机制完善，重复运行几乎瞬间完成。

pipx

pipx 在隔离的环境中运行 Python 包。可通过操作系统包管理器安装（apt install pipx，brew install pipx）。

```
代码块
1   pipx run 'black==24.10.0' .
2   pipx run 'ruff==0.8.0' check .
```

- 未与 Python 捆绑，需要单独安装。 一个成熟的 uvx 替代方案。
- 虽然 uvx 已成为标准推荐方案，但 pipx 仍然是一个可靠的选择，并且具有更广泛的操作系统包管理器支持。

npx

npx 运行 npm 包，按需下载。它随 npm 一起发布（npm 随 Node.js 一起发布）。

```
代码块
1   npx eslint@9 --fix .
2   npx create-vite@6 my-app
```

- 已与 Node.js 捆绑，无需额外安装。
- 下载软件包，运行它，并将其缓存以供将来使用。
- 使用 `npx package@version` 命令锁定版本，以确保可复现性。

bunx

bunx 是 Bun 的 npx 版本，它随 Bun 一起提供。

```
代码块
1   bunx eslint@9 --fix .
2   bunx create-vite@6 my-app
```

- 在基于 Bun 的环境中，可直接替代 npx。
- 仅适用于用户环境中使用的是 Bun 而非 Node.js 的情况。

deno run

deno run 可以直接从 URL 或指定符运行脚本。它随 Deno 一起提供。

```
代码块
1   deno run npm:create-vite@6 my-app
2   deno run --allow-read npm:eslint@9 -- --fix .
```

- 文件系统/网络访问需要权限标志（例如 `--allow-read`）。
- 使用 `--` 将 Deno 标志与工具自身的标志分隔开来。

go run

`go run` 命令直接编译并运行 Go 包。它已内置于 `go` 命令中。

```
代码块
1   go run golang.org/x/tools/cmd/goimports@v0.28.0 .
2   go run github.com/golangci/golangci-lint/cmd/golangci-lint@v1.62.0 run
```

- Go 内置版本控制，无需额外工具。
- 锁定版本或使用 `@latest` 显式指定命令。

技能中一次性命令的技巧：

固定版本（例如，npx eslint@9.0.0），以确保命令行为始终一致。

在 SKILL.md 文件中声明先决条件（例如，“需要 Node.js 18+”），而不是假设代理环境已具备这些条件。对于运行时级别的要求，请使用兼容性 frontmatter 字段。

将复杂命令移至脚本中。一次性命令适用于仅使用少量标志调用工具的情况。但当命令变得过于复杂，难以一次性正确执行时，在 scripts/ 目录下编写经过测试的脚本会更加可靠。

从 SKILL.md 文件中引用脚本

使用技能目录根目录的相对路径来引用捆绑的文件。代理会自动解析这些路径，无需绝对路径。

在 SKILL.md 文件中列出可用脚本，以便代理知道它们的存在：

代码块

```
## Available scripts

- **`scripts/validate.sh`** – Validates configuration files
- **`scripts/process.py`** – Processes input data
```

然后指示代理人运行它们：

代码块

```
## Workflow

1. Run the validation script:
   ```bash
   bash scripts/validate.sh "$INPUT_FILE"
   ```

2. Process the results:
   ```bash
   python3 scripts/process.py --input results.json
   ```
```

🎉 同样的相对路径约定也适用于参考文件/*.md 等支持文件——脚本执行路径（在代码块中）相对于技能目录根目录，因为代理从那里运行命令。

独立脚本

当您需要可重用的逻辑时，请将一个脚本打包到 scripts/ 目录下，该脚本可以内联声明自身的依赖项。代理只需一条命令即可运行该脚本，无需单独的清单文件或安装步骤。

多种语言支持内联依赖项声明：

Python

PEP 723 定义了内联脚本元数据的标准格式。在 # /// 标记内的 TOML 代码块中声明依赖关系：

代码块

```
# /// script
# dependencies = [
#   "beautifulsoup4",
# ]
# ///

from bs4 import BeautifulSoup

html = '<html><body><h1>Welcome</h1><p class="info">This is a test.</p></body></html>'
print(BeautifulSoup(html, "html.parser").select_one("p.info").get_text())
```

Run with uv (recommended):

代码块

```
uv run scripts/extract.py
```

`uv run` 会创建一个隔离环境，安装声明的依赖项，并运行脚本。`pipx`（`pipx run scripts/extract.py`）也支持 PEP 723。

- 使用 PEP 508 规范指定版本：`"beautifulsoup4>=4.12,<5"`。
- 使用 `requires-python` 来限制 Python 版本。
- 使用 `uv lock --script` 创建一个锁定文件，以确保完全可复现问题。

Deno

Deno 的 npm 和 jsr 导入说明符默认情况下使每个脚本都是自包含的：

代码块

```
#!/usr/bin/env -S deno run

import * as cheerio from "npm:cheerio@1.0.0";

const html = `<html><body><h1>Welcome</h1><p class="info">This is a test.</p></body></html>`;
```

```
const $ = cheerio.load(html);
console.log($("p.info").text());
```

代码块

```
deno run scripts/extract.ts
```

- 使用 npm：适用于 npm 包，使用 jsr：适用于 Deno 原生包。
- 版本说明符遵循语义化版本控制：@1.0.0（精确版本），@^1.0.0（兼容版本）。
- 依赖项已全局缓存。使用 `--reload` 强制重新获取依赖项。
- 带有原生插件（node-gyp）的包可能无法正常工作——提供预编译二进制文件的包效果最佳。

Ruby

Ruby 2.6 版本起，Bundler 就随 Ruby 一起发布了。使用 bundler/inline 可以直接在脚本中声明 gems：

代码块

```
require 'bundler/inline'

gemfile do
  source 'https://rubygems.org'
  gem 'nokogiri'
end

html = '<html><body><h1>Welcome</h1><p class="info">This is a test.</p></body></html>'
doc = Nokogiri::HTML(html)
puts doc.at_css('p.info').text
```

代码块

```
ruby scripts/extract.rb
```

- 显式锁定版本（gem 'nokogiri', '~> 1.16'）——没有锁定文件。
- 工作目录中已存在的 Gemfile 或 BUNDLE_GEMFILE 环境变量可能会造成干扰。

为代理设计脚本

当代理运行您的脚本时，它会读取标准输出 (stdout) 和标准错误输出 (stderr) 来决定下一步操作。一些设计选择可以显著简化代理使用脚本的过程。

避免交互式提示

这是代理执行环境的硬性要求。代理在非交互式 shell 中运行——它们无法响应 TTY 提示、密码对话框或确认菜单。阻塞于交互式输入的脚本将无限期地挂起。

接受所有通过命令行标志、环境变量或标准输入 (stdin) 的输入：

代码块

```
# Bad: hangs waiting for input
$ python scripts/deploy.py
Target environment: _

# Good: clear error with guidance
$ python scripts/deploy.py
Error: --env is required. Options: development, staging, production.
Usage: python scripts/deploy.py --env staging --tag v1.2.3
```

使用 `--help`

--help 输出是代理学习脚本界面的主要方式。请包含简要说明、可用标志和使用示例：

代码块

```
Usage: scripts/process.py [OPTIONS] INPUT_FILE

Process input data and produce a summary report.

Options:
  --format FORMAT    Output format: json, csv, table (default: json)
  --output FILE      Write output to FILE instead of stdout
  --verbose          Print progress to stderr

Examples:
  scripts/process.py data.csv
  scripts/process.py --format csv --output report.csv data.csv
```

保持简洁——输出结果会与代理正在处理的所有其他内容一起进入代理的上下文窗口。

编写清晰易懂的错误信息

当智能体遇到错误时，错误信息会直接影响其下一次尝试。一条晦涩难懂的“错误：输入无效”会浪费一次机会。相反，应该说明哪里出了问题，预期结果是什么，以及应该尝试什么：

代码块

```
Error: --format must be one of: json, csv, table.
       Received: "xml"
```

使用结构化输出

优先使用结构化格式（例如 JSON、CSV、TSV）而非自由文本。结构化格式既可以被代理程序读取，也可以被标准工具（例如 jq、cut、awk）读取，从而使您的脚本能够在流水线中组合使用。

代码块

```
# Whitespace-aligned — hard to parse programmatically
NAME          STATUS    CREATED
my-service    running   2025-01-15

# Delimited — unambiguous field boundaries
{"name": "my-service", "status": "running", "created": "2025-01-15"}
```

将数据与诊断信息分离：将结构化数据发送到标准输出 (stdout)，将进度消息、警告和其他诊断信息发送到标准错误输出 (stderr)。这样，代理程序既可以捕获清晰、可解析的输出，又可以在需要时访问诊断信息

其他注意事项

- 幂等性。代理程序可以重试命令。“如果不存在则创建”比“创建但重复则失败”更安全。
- 输入约束。拒绝含义模糊的输入时应给出明确的错误信息，而不是猜测。尽可能使用枚举和闭集。
- 试运行支持。对于破坏性操作或有状态操作，`--dry-run` 标志允许代理程序预览将要发生的情况。
- 有意义的退出代码。为不同的失败类型（未找到、参数无效、身份验证失败）使用不同的退出代码，并在 `--help` 输出中记录这些代码，以便代理程序了解每个代码的含义。
- 安全的默认值。考虑破坏性操作是否应该需要显式的确认标志（`--confirm`、`--force`）或其他与风险级别相适应的安全措施。
- 可预测的输出大小。许多代理程序会自动截断超过阈值（例如 10-30K 个字符）的工具输出，这可能会丢失关键信息。如果您的脚本可能会产生大量输出，请默认显示摘要或设置合理的输出限制，并支持类似 `--offset` 的标志，以便代理在需要时请求更多信息。或者，如果输出量很大且不适合分页，则要求代理传递 `--output` 标志，该标志可以指定输出文件，也可以使用 `-` 显式选择输出到标准输出。

将SKILLS融入你的代理

集成方法

集成技能主要有两种方法：

- 基于文件系统的代理在计算机环境（bash/unix）中运行，是功能最强大的选择。当模型发出类似 `cat /path/to/my-skill/SKILL.md` 的 shell 命令时，技能会被激活。捆绑的资源通过 shell 命令访问。
- 基于工具的代理无需专用计算机环境即可运行。它们通过实现工具，使模型能够触发技能并访问捆绑的资源。具体的工具实现由开发者决定。

概述

一个兼容技能的代理需要：

- 在配置的目录中发现技能
- 在启动时加载元数据（名称和描述|name and description）
- 将用户任务与相关技能匹配
- 通过加载完整指令来激活技能
- 根据需要执行脚本并访问资源

技能发现

技能是包含 `SKILL.md` 文件的文件夹。您的代理应该扫描配置的目录以查找有效的技能。

加载元数据metadata

在启动时，仅解析每个 `SKILL.md` 文件的 frontmatter。这样可以降低初始上下文的使用量。

解析前言

代码块

```
function parseMetadata(skillPath):
    content = readFile(skillPath + "/SKILL.md")
    frontmatter = extractYAMLFrontmatter(content)

    return {
        name: frontmatter.name,
        description: frontmatter.description,
        path: skillPath
    }
```

注入上下文

在系统提示中包含技能元数据，以便模型了解有哪些技能可用。

请遵循平台关于系统提示更新的指南。例如，对于 Claude 模型，推荐使用 XML 格式：

```
<available_skills>
  <skill>
    <name>pdf-processing</name>
    <description>Extracts text and tables from PDF files, fills forms, merges documents.</description>
    <location>/path/to/skills/pdf-processing/SKILL.md</location>
  </skill>
  <skill>
    <name>data-analysis</name>
    <description>Analyzes datasets, generates charts, and creates summary reports.</description>
    <location>/path/to/skills/data-analysis/SKILL.md</location>
  </skill>
</available_skills>
```

对于基于文件系统的代理，请在位置字段中包含 SKILL.md 文件的绝对路径。对于基于工具的代理，可以省略位置字段。

保持元数据简洁。每个技能应向上下文添加大约 50-100 个标记。

安全注意事项

脚本执行会带来安全风险。请考虑以下措施：

- 沙箱(Sandboxing)：在隔离环境中运行脚本
- 允许列表(Allowlisting)：仅执行来自受信任技能的脚本
- 确认(Confirmation)：在执行潜在危险操作之前询问用户
- 日志记录(Logging)：记录所有脚本执行以进行审计

参考实现

skills-ref 库提供用于处理技能的 Python 实用程序和命令行界面 (CLI)。

- 例如： 验证技能目录：

```
skills-ref validate <path>
```

- 为agent提示生成 <available_skills> XML：

代码块

```
skills-ref to-prompt <path>...
```

使用库源代码作为参考实现。