> ## Documentation Index
>
> Fetch the complete documentation index at: https://code.claude.com/docs/llms.txt
> Use this file to discover all available pages before exploring further.

使用 skills 扩展 Claude

> 创建、管理和共享 skills 以在 Claude Code 中扩展 Claude 的功能。包括自定义命令和捆绑 skills。

Skills 扩展了 Claude 能做的事情。创建一个 `SKILL.md` 文件，其中包含说明，Claude 会将其添加到其工具包中。Claude 在相关时使用 skills，或者你可以使用 `/skill-name` 直接调用一个。

对于内置命令（如 `/help` 和 `/compact`），请参阅交互模式。

自定义命令已合并到 skills 中。 `.claude/commands/deploy.md` 中的文件和 `.claude/skills/deploy/SKILL.md` 中的 skill 都会创建 `/deploy` 并以相同的方式工作。你现有的 `.claude/commands/` 文件继续工作。Skills 添加了可选功能：支持文件的目录、控制你或 Claude 是否调用它们的 frontmatter，以及 Claude 在相关时自动加载它们的能力。

Claude Code skills 遵循 Agent Skills 开放标准，该标准适用于多个 AI 工具。Claude Code 使用额外功能扩展了该标准，如调用控制、subagent 执行和动态上下文注入。

捆绑 skills

捆绑 skills 随 Claude Code 一起提供，在每个会话中都可用。与内置命令不同，内置命令直接执行固定逻辑，捆绑 skills 是基于提示的：它们为 Claude 提供详细的操作手册，让它使用其工具来编排工作。这意味着捆绑 skills 可以生成并行代理、读取文件并适应你的代码库。

你调用捆绑 skills 的方式与调用任何其他 skill 相同：输入 / 后跟 skill 名称。

- **/simplify**：审查你最近更改的文件以查找代码重用、质量和效率问题，然后修复它们。在实现功能或错误修复后运行它来清理你的工作。它并行生成三个审查代理（代码重用、代码质量、效率），汇总他们的发现，并应用修复。传递可选文本以专注于特定问题：/simplify focus on memory efficiency。
- **/batch <instruction>**：在整个代码库中并行编排大规模更改。提供更改的描述，/batch 研究代码库，将工作分解为 5 到 30 个独立单元，并提出计划供你批准。批准后，它为每个单元生成一个后台代理，每个都在隔离的 git worktree 中。每个代理实现其单元、运行测试并打开拉取请求。需要 git 存储库。示例：/batch migrate src/ from Solid to React。
- **/debug [description]**：通过读取会话调试日志来排查你当前的 Claude Code 会话。可选地描述问题以专注分析。
- **/loop [interval] <prompt>**：在会话保持打开状态时按间隔重复运行提示。Claude 解析间隔、安排循环 cron 任务并确认节奏。对于轮询部署、监视 PR 或定期重新运行另一个 skill 很有用。示例：/loop 5m check if the deploy finished。请参阅按计划运行提示。
- **/claude-api**：为你的项目语言（Python、TypeScript、Java、Go、Ruby、C#、PHP 或 cURL）加载 Claude API 参考资料，以及 Python 和 TypeScript 的 Agent SDK 参考。涵盖工具使用、流式传输、批处理、结构化输出和常见陷阱。当你的代码导入 anthropic、@anthropic-ai/sdk 或 claude_agent_sdk 时也会自动激活。

入门

创建你的第一个 skill

此示例创建一个 skill，教 Claude 使用视觉图表和类比来解释代码。由于它使用默认 frontmatter，Claude 可以在你询问某事如何工作时自动加载它，或者你可以使用 /explain-code 直接调用它。

在你的个人 skills 文件夹中为 skill 创建一个目录。个人 skills 在你的所有项目中都可用。

```bash  theme={null}
mkdir -p ~/.claude/skills/explain-code
```

每个 skill 都需要一个 `SKILL.md` 文件，包含两部分：YAML frontmatter（在 `---` 标记之间）告诉 Claude 何时使用该 skill，以及包含 Claude 在调用该 skill 时遵循的说明的 markdown 内容。`name` 字段变成 `/slash-command`，`description` 帮助 Claude 决定何时自动加载它。

创建 `~/.claude/skills/explain-code/SKILL.md`：

```yaml  theme={null}
---
name: explain-code
description: Explains code with visual diagrams and analogies. Use when explaining how code works, teaching about a codebase, or when the user asks "how does this work?"
---

When explaining code, always include:

1. **Start with an analogy**: Compare the code to something from everyday life
2. **Draw a diagram**: Use ASCII art to show the flow, structure, or relationships
3. **Walk through the code**: Explain step-by-step what happens
4. **Highlight a gotcha**: What's a common mistake or misconception?
```

````
Keep explanations conversational. For complex concepts, use multiple analogies.
```
````

你可以通过两种方式测试它：

````
**让 Claude 自动调用它**，通过询问与描述匹配的内容：

```text  theme={null}
How does this code work?
```

**或直接使用 skill 名称调用它**：

```text  theme={null}
/explain-code src/auth/login.ts
```

无论哪种方式，Claude 都应该在其解释中包含类比和 ASCII 图表。
````

Skills 的位置

你存储 skill 的位置决定了谁可以使用它：

位置	路径	适用于
企业	请参阅托管设置	你的组织中的所有用户
个人	`~/.claude/skills/<skill-name>/SKILL.md`	你的所有项目
项目	`.claude/skills/<skill-name>/SKILL.md`	仅此项目
插件	`<plugin>/skills/<skill-name>/SKILL.md`	启用插件的位置
````

当 skills 在各个级别共享相同名称时，更高优先级的位置获胜：企业 > 个人 > 项目。插件 skills 使用 `plugin-name:skill-name` 命名空间，因此它们不能与其他级别冲突。如果你在 `.claude/commands/` 中有文件，它们的工作方式相同，但如果 skill 和命令共享相同名称，skill 优先。

从嵌套目录自动发现

当你在子目录中处理文件时，Claude Code 会自动从嵌套的 `.claude/skills/` 目录中发现 skills。例如，如果你正在编辑 `packages/frontend/` 中的文件，Claude Code 也会在 `packages/frontend/.claude/skills/` 中查找 skills。这支持 monorepo 设置，其中包有自己的 skills。

每个 skill 都是一个以 `SKILL.md` 作为入口点的目录：

```
my-skill/
├── SKILL.md              # 主要说明（必需）
├── template.md           # Claude 要填写的模板
├── examples/
│   └── sample.md         # 显示预期格式的示例输出
└── scripts/
    └── validate.sh       # Claude 可以执行的脚本
```

`SKILL.md` 包含主要说明并且是必需的。其他文件是可选的，让你构建更强大的 skills：Claude 要填写的模板、显示预期格式的示例输出、Claude 可以执行的脚本或详细的参考文档。从你的 `SKILL.md` 引用这些文件，以便 Claude 知道它们包含什么以及何时加载它们。有关更多详细信息，请参阅添加支持文件。

`.claude/commands/` 中的文件仍然有效并支持相同的 frontmatter。建议使用 Skills，因为它们支持额外功能，如支持文件。

来自其他目录的 skills

在通过 `--add-dir` 添加的目录中的 `.claude/skills/` 中定义的 skills 会自动加载并由实时更改检测拾取，因此你可以在会话期间编辑它们而无需重新启动。

来自 `--add-dir` 目录的 CLAUDE.md 文件默认不加载。要加载它们，请设置 `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1`。请参阅从其他目录加载。

配置 skills

Skills 通过 `SKILL.md` 顶部的 YAML frontmatter 和随后的 markdown 内容进行配置。

Skill 内容的类型

Skill 文件可以包含任何说明，但思考你想如何调用它们有助于指导要包含的内容：

参考内容添加 Claude 应用于你当前工作的知识。约定、模式、风格指南、领域知识。此内容内联运行，以便 Claude 可以将其与你的对话上下文一起使用。

```yaml
---
name: api-conventions
description: API design patterns for this codebase
---

When writing API endpoints:
- Use RESTful naming conventions
- Return consistent error formats
- Include request validation
```

任务内容为 Claude 提供特定操作的分步说明，如部署、提交或代码生成。这些通常是你想使用 `/skill-name` 直接调用的操作，而不是让 Claude 决定何时运行它们。添加 `disable-model-invocation: true` 以防止 Claude 自动触发它。

```yaml
---
name: deploy
description: Deploy the application to production
context: fork
disable-model-invocation: true
---

Deploy the application:
1. Run the test suite
2. Build the application
3. Push to the deployment target
```

你的 `SKILL.md` 可以包含任何内容，但思考你想如何调用该 skill（由你、由 Claude 或两者）以及你想在哪里运行它（内联或在 subagent 中）有助于指导要包含的内容。对于复杂的 skills，你也可以添加支持文件来保持主 skill 的专注。

Frontmatter 参考

除了 markdown 内容外，你可以使用 `SKILL.md` 文件顶部 `---` 标记之间的 YAML frontmatter 字段来配置 skill 行为：

```yaml
---
name: my-skill
description: What this skill does
disable-model-invocation: true
allowed-tools: Read, Grep
---

Your skill instructions here...
```

所有字段都是可选的。建议使用 `description`，以便 Claude 知道何时使用该 skill。

字段	必需	描述
`name`	否	Skill 的显示名称。如果省略，使用目录名称。仅小写字母、数字和连字符（最多 64 个字符）。
`description`	推荐	Skill 的功能以及何时使用它。Claude 使用它来决定何时应用该 skill。如果省略，使用 markdown 内容的第一段。
`argument-hint`	否	自动完成期间显示的提示，指示预期的参数。示例：`[issue-number]` 或 `[filename] [format]`。
`disable-model-invocation`	否	设置为 `true` 以防止 Claude 自动加载此 skill。用于你想使用 `/name` 手动触发的工作流。默认值：`false`。
`user-invocable`	否	设置为 `false` 以从 `/` 菜单中隐藏。用于用户不应直接调用的背景知识。默认值：`true`。
`allowed-tools`	否	当此 skill 处于活动状态时，Claude 可以使用而无需请求权限的工具。
`model`	否	当此 skill 处于活动状态时要使用的模型。
`context`	否	设置为 `fork` 以在分叉的 subagent 上下文中运行。
`agent`	否	当设置 `context: fork` 时要使用的 subagent 类型。
`hooks`	否	限定于此 skill 生命周期的 hooks。有关配置格式，请参阅 Skills 和代理中的 Hooks。

可用的字符串替换

Skills 支持 skill 内容中动态值的字符串替换：

变量	描述
`$ARGUMENTS`	调用 skill 时传递的所有参数。如果 `$ARGUMENTS` 不在内容中，参数作为 `ARGUMENTS: <value>` 追加。
`$ARGUMENTS[N]`	按 0 基索引访问特定参数，如 `$ARGUMENTS[0]` 表示第一个参数。
`$N`	`$ARGUMENTS[N]` 的简写，如 `$0` 表示第一个参数或 `$1` 表示第二个参数。
`${CLAUDE_SESSION_ID}`	当前会话 ID。用于日志记录、创建会话特定文件或将 skill 输出与会话关联。
`${CLAUDE_SKILL_DIR}`	包含 skill 的 `SKILL.md` 文件的目录。对于插件 skills，这是插件内 skill 的子目录，而不是插件根目录。在 bash 注入命令中使用它来引用与 skill 捆绑的脚本或文件，无论当前工作目录如何。

使用替换的示例：

```
---
name: session-logger
description: Log activity for this session
---

Log the following to logs/${CLAUDE_SESSION_ID}.log:

$ARGUMENTS
```

添加支持文件

Skills 可以在其目录中包含多个文件。这使 `SKILL.md` 专注于要点，同时让 Claude 仅在需要时访问详细的参考资料。大型参考文档、API 规范或示例集合不需要在每次 skill 运行时加载到上下文中。

```
my-skill/
├─── SKILL.md (required - overview and navigation)
├─── reference.md (detailed API docs - loaded when needed)
├─── examples.md (usage examples - loaded when needed)
└─── scripts/
    └─── helper.py (utility script - executed, not loaded)
```

从 `SKILL.md` 引用支持文件，以便 Claude 知道每个文件包含什么以及何时加载它：

```
## Additional resources

- For complete API details, see [reference.md](reference.md)
- For usage examples, see [examples.md](examples.md)
```

将 `SKILL.md` 保持在 500 行以下。将详细的参考资料移到单独的文件中。

控制谁调用 skill

默认情况下，你和 Claude 都可以调用任何 skill。你可以输入 `/skill-name` 直接调用它，Claude 可以在与你的对话相关时自动加载它。两个 frontmatter 字段让你限制这一点：

- **`disable-model-invocation: true`**：只有你可以调用该 skill。用于有副作用或你想控制时间的工作流，如 `/commit`、`/deploy` 或 `/send-slack-message`。你不希望 Claude 因为你的代码看起来准备好了就决定部署。
- **`user-invocable: false`**：只有 Claude 可以调用该 skill。用于不可作为命令操作的背景知识。`legacy-system-context` skill 解释了旧系统的工作原理。Claude 在相关时应该知道这一点，但 `/legacy-system-context` 对用户来说不是一个有意义的操作。

此示例创建一个只有你可以触发的部署 skill。`disable-model-invocation: true` 字段防止 Claude 自动运行它：

```yaml
---
name: deploy
description: Deploy the application to production
disable-model-invocation: true
---

Deploy $ARGUMENTS to production:

1. Run the test suite
2. Build the application
3. Push to the deployment target
4. Verify the deployment succeeded
```

以下是两个字段如何影响调用和上下文加载：

Frontmatter	你可以调用	Claude 可以调用	何时加载到上下文
（默认）	是	是	描述始终在上下文中，调用时加载完整 skill
`disable-model-invocation: true`	是	否	描述不在上下文中，你调用时加载完整 skill
`user-invocable: false`	否	是	描述始终在上下文中，调用时加载完整 skill

在常规会话中，skill 描述被加载到上下文中，以便 Claude 知道什么可用，但完整 skill 内容仅在调用时加载。具有预加载 skills 的 Subagents 的工作方式不同：完整 skill 内容在启动时注入。

限制工具访问

使用 `allowed-tools` 字段来限制当 skill 处于活动状态时 Claude 可以使用哪些工具。此 skill 创建一个只读模式，其中 Claude 可以浏览文件但不能修改它们：

```yaml
---
name: safe-reader
description: Read files without making changes
allowed-tools: Read, Grep, Glob
---
```

将参数传递给 skills

你和 Claude 都可以在调用 skill 时传递参数。参数可通过 `$ARGUMENTS` 占位符获得。

此 skill 按编号修复 GitHub 问题。`$ARGUMENTS` 占位符被替换为 skill 名称后面的任何内容：

```yaml
---
name: fix-issue
description: Fix a GitHub issue
disable-model-invocation: true
---

Fix GitHub issue $ARGUMENTS following our coding standards.

1. Read the issue description
2. Understand the requirements
3. Implement the fix
4. Write tests
5. Create a commit
```

当你运行 `/fix-issue 123` 时，Claude 收到"Fix GitHub issue 123 following our coding standards..."

如果你使用参数调用 skill 但 skill 不包含 `$ARGUMENTS`，Claude Code 会将 `ARGUMENTS: <your input>` 追加到 skill 内容的末尾，以便 Claude 仍然看到你输入的内容。

要按位置访问单个参数，使用 `$ARGUMENTS[N]` 或较短的 `$N`：

```yaml
---
name: migrate-component
description: Migrate a component from one framework to another
---

Migrate the $ARGUMENTS[0] component from $ARGUMENTS[1] to $ARGUMENTS[2].
Preserve all existing behavior and tests.
```

运行 `/migrate-component SearchBar React Vue` 将 `$ARGUMENTS[0]` 替换为 `SearchBar`，`$ARGUMENTS[1]` 替换为 `React`，`$ARGUMENTS[2]` 替换为 `Vue`。使用 `$N` 简写的相同 skill：

```yaml
---
name: migrate-component
description: Migrate a component from one framework to another
---

Migrate the $0 component from $1 to $2.
Preserve all existing behavior and tests.
```

高级模式

注入动态上下文

`!command`` 语法在将 skill 内容发送给 Claude 之前运行 shell 命令。命令输出替换占位符，所以 Claude 接收实际数据，而不是命令本身。

此 skill 通过使用 GitHub CLI 获取实时 PR 数据来总结拉取请求。`!gh pr diff`` 和其他命令首先运行，它们的输出被插入到提示中：

```yaml
---
name: pr-summary
description: Summarize changes in a pull request
context: fork
agent: Explore
allowed-tools: Bash(gh *)
```

```
---

## Pull request context
- PR diff: !`gh pr diff`
- PR comments: !`gh pr view --comments`
- Changed files: !`gh pr diff --name-only`

## Your task
Summarize this pull request...
```

当此 skill 运行时：

1. 每个 `!command`` 立即执行（在 Claude 看到任何东西之前）
2. 输出替换 skill 内容中的占位符
3. Claude 接收带有实际 PR 数据的完全呈现的提示

这是预处理，不是 Claude 执行的东西。Claude 只看到最终结果。

要在 skill 中启用扩展思考，在你的 skill 内容中的任何地方包含单词"ultrathink"。

在 subagent 中运行 skills

当你想让 skill 在隔离中运行时，在你的 frontmatter 中添加 `context: fork`。skill 内容变成驱动 subagent 的提示。它将无法访问你的对话历史。

`context: fork` 仅对具有明确说明的 skills 有意义。如果你的 skill 包含"使用这些 API 约定"之类的指南而没有任务，subagent 接收指南但没有可操作的提示，并返回而没有有意义的输出。

Skills 和 subagents 以两个方向协同工作：

方法	系统提示	任务	也加载
带有 `context: fork` 的 Skill	来自代理类型（`Explore`、`Plan` 等）	SKILL.md 内容	CLAUDE.md
带有 `skills` 字段的 Subagent	Subagent 的 markdown 正文	Claude 的委派消息	预加载的 skills + CLAUDE.md

使用 `context: fork`，你在你的 skill 中编写任务并选择一个代理类型来执行它。对于反向（定义使用 skills 作为参考资料的自定义 subagent），请参阅 Subagents。

示例：使用 Explore 代理的研究 skill

此 skill 在分叉的 Explore 代理中运行研究。skill 内容变成任务，代理提供针对代码库探索优化的只读工具：

```
---
name: deep-research
description: Research a topic thoroughly
context: fork
agent: Explore
---

Research $ARGUMENTS thoroughly:

1. Find relevant files using Glob and Grep
2. Read and analyze the code
3. Summarize findings with specific file references
```

当此 skill 运行时：

1. 创建新的隔离上下文
2. Subagent 接收 skill 内容作为其提示（"Research $ARGUMENTS thoroughly..."）
3. `agent` 字段确定执行环境（模型、工具和权限）

4. 结果被总结并返回到你的主对话

`agent` 字段指定使用哪个 subagent 配置。选项包括内置代理（`Explore`、`Plan`、`general-purpose`）或来自 `.claude/agents/` 的任何自定义 subagent。如果省略，使用 `general-purpose`。

限制 Claude 的 skill 访问

默认情况下，Claude 可以调用任何没有设置 `disable-model-invocation: true` 的 skill。定义 `allowed-tools` 的 Skills 在 skill 处于活动状态时授予 Claude 对这些工具的访问权限，无需每次使用批准。你的权限设置仍然管理所有其他工具的基线批准行为。内置命令如 `/compact` 和 `/init` 不能通过 Skill 工具获得。

三种方式来控制 Claude 可以调用哪些 skills：

通过在 `/permissions` 中拒绝 Skill 工具来禁用所有 skills：

```
# Add to deny rules:
Skill
```

使用权限规则允许或拒绝特定 skills：

```
# Allow only specific skills
Skill(commit)
Skill(review-pr *)

# Deny specific skills
Skill(deploy *)
```

权限语法：`Skill(name)` 用于精确匹配，`Skill(name *)` 用于带有任何参数的前缀匹配。

通过在其 frontmatter 中添加 `disable-model-invocation: true` 来隐藏单个 skills。这将 skill 从 Claude 的上下文中完全删除。

`user-invocable` 字段仅控制菜单可见性，不控制 Skill 工具访问。使用 `disable-model-invocation: true` 来阻止程序调用。

共享 skills

Skills 可以根据你的受众在不同范围内分发：

- **项目 skills**：将 `.claude/skills/` 提交到版本控制
- **插件**：在你的插件中创建 `skills/` 目录
- **托管**：通过托管设置部署组织范围内

生成视觉输出

Skills 可以捆绑并运行任何语言的脚本，为 Claude 提供超越单个提示可能的功能。一个强大的模式是生成视觉输出：在浏览器中打开的交互式 HTML 文件，用于探索数据、调试或创建报告。

此示例创建一个代码库浏览器：一个交互式树视图，你可以在其中展开和折叠目录、一目了然地查看文件大小，并按颜色识别文件类型。

创建 Skill 目录：

```
mkdir -p ~/.claude/skills/codebase-visualizer/scripts
```

创建 `~/.claude/skills/codebase-visualizer/SKILL.md`。描述告诉 Claude 何时激活此 Skill，说明告诉 Claude 运行捆绑脚本：

```
---
name: codebase-visualizer
description: Generate an interactive collapsible tree visualization of your codebase. Use when exploring a new repo, understanding project structure, or identifying large files.
allowed-tools: Bash(python *)
---

# Codebase Visualizer

Generate an interactive HTML tree view that shows your project's file structure with collapsible directories.

## Usage
```

````
Run the visualization script from your project root:

```bash
python ~/.claude/skills/codebase-visualizer/scripts/visualize.py .
```text

This creates `codebase-map.html` in the current directory and opens it in your default browser.

## What the visualization shows

- **Collapsible directories**: Click folders to expand/collapse
- **File sizes**: Displayed next to each file
- **Colors**: Different colors for different file types
- **Directory totals**: Shows aggregate size of each folder
````

创建 `~/.claude/skills/codebase-visualizer/scripts/visualize.py`。此脚本扫描目录树并生成一个自包含的 HTML 文件，包含：

- 一个**摘要侧边栏**，显示文件计数、目录计数、总大小和文件类型数量
- 一个**条形图**，按文件类型（按大小排名前 8）分解代码库
- 一个**可折叠树**，你可以在其中展开和折叠目录，带有颜色编码的文件类型指示器

该脚本需要 Python，但仅使用内置库，因此无需安装包：

```
#!/usr/bin/env python3
"""Generate an interactive collapsible tree visualization of a codebase."""

import json
import sys
import webbrowser
from pathlib import Path
from collections import Counter

```

```
IGNORE = {'.git', 'node_modules', '__pycache__', '.venv', 'venv', 'dist', 'build'}

def scan(path: Path, stats: dict) -> dict:
    result = {"name": path.name, "children": [], "size": 0}
    try:
        for item in sorted(path.iterdir()):
            if item.name in IGNORE or item.name.startswith('.'):
                continue
            if item.is_file():
                size = item.stat().st_size
                ext = item.suffix.lower() or '(no ext)'
                result["children"].append({"name": item.name, "size": size, "ext": ext})
                result["size"] += size
                stats["files"] += 1
                stats["extensions"][ext] += 1
                stats["ext_sizes"][ext] += size
            elif item.is_dir():
                stats["dirs"] += 1
                child = scan(item, stats)
                if child["children"]:
                    result["children"].append(child)
                    result["size"] += child["size"]
    except PermissionError:
        pass
    return result

def generate_html(data: dict, stats: dict, output: Path) -> None:
    ext_sizes = stats["ext_sizes"]
    total_size = sum(ext_sizes.values()) or 1
    sorted_exts = sorted(ext_sizes.items(), key=lambda x: -x[1])[:8]
    colors = {
        '.js': '#f7df1e', '.ts': '#3178c6', '.py': '#3776ab', '.go': '#00add8',
```

```
        '.rs': '#dea584', '.rb': '#cc342d', '.css': '#264de4', '.html': '#e34c26',
        '.json': '#6b7280', '.md': '#083fa1', '.yaml': '#cb171e', '.yml': '#cb171e',
        '.mdx': '#083fa1', '.tsx': '#3178c6', '.jsx': '#61dafb', '.sh': '#4eaa25',
    }
    lang_bars = "".join(
        f'<div class="bar-row"><span class="bar-label">{ext}</span>'
        f'<div class="bar" style="width:{(size/total_size)*100}%;background:{colors.get(ext,"#6b7280")}"></div>'
        f'<span class="bar-pct">{(size/total_size)*100:.1f}%</span></div>'
        for ext, size in sorted_exts
    )
    def fmt(b):
        if b < 1024: return f"{b} B"
        if b < 1048576: return f"{b/1024:.1f} KB"
        return f"{b/1048576:.1f} MB"

    html = f'''<!DOCTYPE html>
<html><head>
  <meta charset="utf-8"><title>Codebase Explorer</title>
  <style>
    body {{ font: 14px/1.5 system-ui, sans-serif; margin: 0; background: #1a1a2e; color: #eee; }}
    .container {{ display: flex; height: 100vh; }}
    .sidebar {{ width: 280px; background: #252542; padding: 20px; border-right: 1px solid #3d3d5c; overflow-y: auto; flex-shrink: 0; }}
    .main {{ flex: 1; padding: 20px; overflow-y: auto; }}
    h1 {{ margin: 0 0 10px 0; font-size: 18px; }}
    h2 {{ margin: 20px 0 10px 0; font-size: 14px; color: #888; text-transform: uppercase; }}
    .stat {{ display: flex; justify-content: space-between; padding: 8px 0; border-bottom: 1px solid #3d3d5c; }}
```

```
    .stat-value {{ font-weight: bold; }}
    .bar-row {{ display: flex; align-items: center; margin: 6px 0; }}
    .bar-label {{ width: 55px; font-size: 12px; color: #aaa; }}
    .bar {{ height: 18px; border-radius: 3px; }}
    .bar-pct {{ margin-left: 8px; font-size: 12px; color: #666; }}
    .tree {{ list-style: none; padding-left: 20px; }}
    details {{ cursor: pointer; }}
    summary {{ padding: 4px 8px; border-radius: 4px; }}
    summary:hover {{ background: #2d2d44; }}
    .folder {{ color: #ffd700; }}
    .file {{ display: flex; align-items: center; padding: 4px 8px; border-radius: 4px; }}
    .file:hover {{ background: #2d2d44; }}
    .size {{ color: #888; margin-left: auto; font-size: 12px; }}
    .dot {{ width: 8px; height: 8px; border-radius: 50%; margin-right: 8px; }}
  </style>
</head><body>
  <div class="container">
    <div class="sidebar">
      <h1>📊 Summary</h1>
      <div class="stat"><span>Files</span><span class="stat-value">{stats["files"]:,}</span></div>
      <div class="stat"><span>Directories</span><span class="stat-value">{stats["dirs"]:,}</span></div>
      <div class="stat"><span>Total size</span><span class="stat-value">{fmt(data["size"])}</span></div>
      <div class="stat"><span>File types</span><span class="stat-value">{len(stats["extensions"])}</span></div>
      <h2>By file type</h2>
      {lang_bars}
    </div>
    <div class="main">
      <h1>📁 {data["name"]}</h1>
```

```
      <ul class="tree" id="root"></ul>
    </div>
  </div>
  <script>
    const data = {json.dumps(data)};
    const colors = {json.dumps(colors)};
    function fmt(b) {{ if (b < 1024) return b + ' B'; if (b < 1048576) return (b/1024).toFixed(1) + ' KB'; return (b/1048576).toFixed(1) + ' MB'; }}
    function render(node, parent) {{
      if (node.children) {{
        const det = document.createElement('details');
        det.open = parent === document.getElementById('root');
        det.innerHTML = `<summary><span class="folder">📁 ${{node.name}}</span><span class="size">${{fmt(node.size)}}</span></summary>`;
        const ul = document.createElement('ul'); ul.className = 'tree';
        node.children.sort((a,b) => (b.children?1:0)-(a.children?1:0) || a.name.localeCompare(b.name));
        node.children.forEach(c => render(c, ul));
        det.appendChild(ul);
        const li = document.createElement('li'); li.appendChild(det); parent.appendChild(li);
      }} else {{
        const li = document.createElement('li'); li.className = 'file';
        li.innerHTML = `<span class="dot" style="background:${{colors[node.ext]||'#6b7280'}}"></span>${{node.name}}<span class="size">${{fmt(node.size)}}</span>`;
        parent.appendChild(li);
      }}
    }}
    data.children.forEach(c => render(c, document.getElementById('root')));
  </script>
</body></html>'''
```

```
    output.write_text(html)

if __name__ == '__main__':
    target = Path(sys.argv[1] if len(sys.argv) > 1 else '.').resolve()
    stats = {"files": 0, "dirs": 0, "extensions": Counter(), "ext_sizes": Counter()}
    data = scan(target, stats)
    out = Path('codebase-map.html')
    generate_html(data, stats, out)
    print(f'Generated {out.absolute()}')
    webbrowser.open(f'file://{out.absolute()}')
```

要测试，在任何项目中打开 Claude Code 并询问"Visualize this codebase."Claude 运行脚本，生成 `codebase-map.html`，并在浏览器中打开它。

这个模式适用于任何视觉输出：依赖关系图、测试覆盖率报告、API 文档或数据库架构可视化。捆绑脚本做繁重的工作，而 Claude 处理编排。

故障排除

Skill 未触发

如果 Claude 在预期时不使用你的 skill：

1. 检查描述是否包含用户会自然说的关键字
2. 验证 skill 是否出现在 `What skills are available?` 中
3. 尝试重新表述你的请求以更接近描述
4. 如果 skill 是用户可调用的，使用 `/skill-name` 直接调用它

Skill 触发过于频繁

如果 Claude 在你不想要时使用你的 skill：

1. 使描述更具体
2. 如果你只想手动调用，添加 `disable-model-invocation: true`

Claude 看不到我的所有 skills

Skill 描述被加载到上下文中，以便 Claude 知道什么可用。如果你有许多 skills，它们可能超过字符预算。预算在上下文窗口的 2% 处动态扩展，回退为 16,000 个字符。运行 `/context` 来检查有关排除的 skills 的警告。

要覆盖限制，设置 `SLASH_COMMAND_TOOL_CHAR_BUDGET` 环境变量。

相关资源

- **Subagents**：将任务委派给专门的代理
- **Plugins**：打包和分发 skills 与其他扩展
- **Hooks**：围绕工具事件自动化工作流
- **Memory**：管理 CLAUDE.md 文件以获得持久上下文
- **Interactive mode**：内置命令和快捷键
- **Permissions**：控制工具和 skill 访问