Claude Code 完整中文文档

> 本文档由 Claude Code 官方英文文档翻译而来

Claude 代码 概述

> ## 文档 索引
>
> 获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
> 使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.

Claude 代码 概述

> Claude 代码 is an 代理ic coding 工具 that 读取s your 代码基础, 编辑s 文件, 运行s 命令, and integ速率s with your 开发 工具. 可用 in your 终端, IDE, 桌面 应用, and 浏览器.

Claude 代码 is an AI-权力ed coding assistant that 帮助s you 构建 features, 修复 缺陷s, and automate 开发 任务s. It understands your 整个 代码基础 and can 工作 across mul 提示le 文件 and 工具 to get 薄gs 完成.

开始使用

选择您的环境 to 开始使用. 大多数界面需要 a Claude 订阅 or Anthropic 控制台 说明. The 终端 CLI and VS 代码 also 支持 third-部分y 提供者s.

```
The 满-featured CLI for工作 with Claude 代码 直接ly in your 终端. 编辑 文件,

To 安装 Claude 代码, 使用 one of the跟随 方法:



    **macOS、Linux、WSL:**

    ```bash  主题={空}
    curl -fsSL https://claude.ai/安装.sh | bash
    ```

    风ows 权力命令行:

    ```权力命令行  主题={空}
    irm https://claude.ai/安装.ps1 | iex
    ```

    风ows CMD:

    ```batch  主题={空}
    curl -fsSL https://claude.ai/安装.cmd -o 安装.cmd && 安装.cmd && del 安
    ```

    If you看见 `The 令牌 '&&' is not a 有效 州ment separator`, you're in 权

    风ows requ怒s [Git for 风ows](https://Git-scm.com/下载s/win). 安装

     原生安装ations 自动所有y 更新 in the 背景 to keep you on the 最新 版本.


    ```bash  主题={空}
    brew 安装 --cask claude-代码
```
```

```
    ```


      Homebrew 安装s do not auto-更新. 运行 `brew 升级 claude-代码` 期间ic所有


    ```权力命令行  主题={空}
    WinGet 安装 Anthropic.Clau解码
    ```


      WinGet 安装s do not auto-更新. 运行 `WinGet 升级 Anthropic.Clau解码` 期

Then 启动 Claude 代码 in 任何 项目:

```bash  主题={空}
cd your-项目
claude
```

You'll be 及时ed to 日志 in on 第一个 使用. That's it! [继续 with the 快速入门

 看见 [先进 设置](/en/设置) for 安装 选项, 手册 更新s, or un安装 说明. Visit [故

The VS 代码 扩展 provides in行 diffs, @-mentions, 计划 re视图, and 对话 histo

* [安装 for VS 代码](vs代码:扩展/anthropic.claude-代码)
* [安装 for Cursor](cursor:扩展/anthropic.claude-代码)
```
```

```
Or 搜索 for "Claude 代码" in the 扩展s视图 (`Cmd+Shift+X` on Mac, `Ctrl+Shi

[开始使用 with VS 代码 →](/en/vs-代码#get-开始)



A 独立 应用 for 运行中 Claude 代码 外部 your IDE or 终端. Re视图 diffs visu所有

下载 and 安装:

* [macOS](https://claude.ai/api/桌面/darwin/普遍/dmg/最新/重定向?utm_来源=cl
* [风ows](https://claude.ai/api/桌面/win32/x64/设置/最新/重定向?utm_来源=clau
* [风ows ARM64](https://claude.ai/api/桌面/win32/arm64/设置/最新/重定向?utm_

之后安装, 启动 Claude, 标志 in, and click the **代码** tab to 启动 coding. A

[Learn 更多 about the 桌面 应用 →](/en/桌面-快速入门)



运行 Claude 代码 in your 浏览器 with no 本地 设置. Kick off 长-运行中 任务s and

启动 coding at [claude.ai/代码](https://claude.ai/代码).

[开始使用 on the 网页 →](/en/claude-代码-on-the-网页#getting-开始)



A 插件 for IntelliJ 想法, PyCharm, 网页风暴, and other JetB雨s IDEs with 交互

安装 the [Claude 代码 插件](https://插件s.jetb雨s.com/插件/27310-claude-代码-

[开始使用 with JetB雨s →](/en/jetb雨s)
```

您可以做什么

Here are 一些 of the 方式s you can 使用 Claude 代码:

Claude 代码 handles the 乏味 任务s that eat 上 your day:写作 测试s for un测试

```bash  主题={空}
claude "写入 测试s for the auth 模块, 运行 them, and 修复 任何 失败s"
```

Describe what you 想要 in 简单 language. Claude 代码 计划s the 方法, 写入s th

For 缺陷s, 粘贴 an 错误 消息 or describe the 症状. Claude 代码 跟踪s the 问题 t

Claude 代码 工作s 直接ly with Git. It 阶段s 更改s, 写入s 提交 消息s, 创建s 分支e

```bash  主题={空}
claude "提交 my 更改s with a de脚本ive 消息"
```

In CI, you can automate 代码 re视图 and 问题 triage with [GitHub 行动s](/en/

The [模型上下文协议 (MCP)](/en/mcp) is an 打开 标准 for连接 AI 工具 to 外部 数据

[`CLAUDE.md`](/en/记忆) is a mark下 文件 you 添加 to your 项目 根 that Claude

创建 [习俗 命令](/en/技能) to 包 repea表 工作流s your 团队 can share, 像 `/re视

[钩子](/en/钩子) let you 运行 命令行 命令 之前 or 之后 Claude 代码 行动s, 像 auto

```
Spawn [mul提示le Claude 代码 代理s](/en/sub-代理s) that 工作 on 不同 部分s of

For 满y 习俗 工作流s, the [代理 SDK](https://平台.claude.com/docs/en/代理-sdk



Claude 代码 is compos能够 and fol低s the Unix pH值ilosopH值y. 管道 日志s into

```bash  主题={空}
#分析 最近 日志 输出
tail -200 应用.日志 | claude -p "松弛 me if you看见 任何 anomalies"

# Automate translations in CI
claude -p "tran板岩 新 字符串s into French and raise a PR for re视图"

# Bulk 运营 across 文件
Git diff 主 --名称-only | claude -p "re视图 these 更改d 文件 for 安全 问题s"
```
```

看见 the CLI 参考 for the 满 设置 of 命令 and 标志.

```
 运行 Claude on a 时间表 to automate 工作 that repeats: morning PR re视图s, 

* [云 计划 任务s](/en/网页-计划-任务s) 运行 on Anthropic-管理 基础设施, so they
* [桌面 计划 任务s](/en/桌面#时间表-recurring-任务s) 运行 on your machine, wit
* [`/loop`](/en/计划-任务s) repeats a 及时 在...内 a CLI 会话 for 快 polling



会话s aren't tied to a single 表面. 移动 工作 between 环境s as your 上下文 更

*步骤 a方式 from your desk and keep工作 from your pH值one or 任何 浏览器 with
* 消息 [Dis补丁](/en/桌面#会话s-from-dis补丁) a 任务 from your pH值one and 打
* Kick off a 长-运行中 任务 on the [网页](/en/claude-代码-on-the-网页) or [iO
* Hand off a 终端 会话 to the [桌面 应用](/en/桌面) with `/桌面` for visual d
* 路由 任务s from 团队 chat: mention `@Claude` in [松弛](/en/松弛) with a 缺
```

使用 Claude 代码 每个where

每个 表面 connects to the 相同 under躺 Claude 代码 engine, so your CLAUDE.md 文件, 设置, and MCP 服务器 工作 across 所有 of them.

超出 the 终端, VS 代码, JetB雨s, 桌面, and 网页 环境s above, Claude 代码 integ速率s with CI/CD, chat, and 浏览器 工作流s:

I 想要 to...	最佳 选项
继续 a 本地 会话 from my pH值one or another device	远程 控制
推送 事件s from Telegram, 不和, i消息, or my own 网页 钩子 into a 会话	通道s
启动 a 任务 本地ly, 继续 on 移动	网页 or Claude iOS 应用
运行 Claude on a recurring 时间表	云 计划 任务s or 桌面 计划 任务s
Automate PR re视图s and 问题 triage	GitHub 行动s or GitLab CI/CD
Get 自动 代码 re视图 on 每个 PR	GitHub 代码 Re视图
路由 缺陷 报告s from 松弛 to 拉取请求s	松弛
调试 live 网页 应用程序s	Chrome
构建 习俗 代理s for your own 工作流s	代理 SDK

下一步

Once you've 安装ed Claude 代码, these 指南s 帮助 you 前往 深er.

- 快速入门:走 th粗糙 your 第一个 真实 任务, from探索 a 代码基础 to 提交ting a 修复
- Store 说明 and memories: give Claude 持续 说明 with CLAUDE.md 文件 and auto 记忆
- 通用工作流 and 最佳实践: 模式s for getting the 最多 out of Claude 代码
- 设置: 习俗ize Claude 代码 for your 工作流
- 故障排除: 解决方案 for 常见问题

- 代码.claude.com: demos,定价, and 产品 详情s

快速入门

> ### 文档 索引
>
> 获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
> 使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.

快速入门

> Welcome to Claude 代码!

出口 常量 安装Configurator = () => {
常量 TERM = {
mac: {
标签: 'macOS / Linux',
cmd: 'curl -fsSL https://claude.ai/安装.sh | bash'
},
win: {
标签: '风ows'
},
brew: {
标签: 'Homebrew',
cmd: 'brew 安装 --cask claude-代码'
},
WinGet: {
标签: 'WinGet',
cmd: 'WinGet 安装 Anthropic.Clau解码'
}

```
};
常量 WIN_VARIANTS = {
ps: 'irm https://claude.ai/安装.ps1 | iex',
cmd: 'curl -fsSL https://claude.ai/安装.cmd -o 安装.cmd && 安装.cmd && del 安
装.cmd'
};
常量 TABS = [{
键: '终端',
标签: '终端'
}, {
键: '桌面',
标签: '桌面'
}, {
键: 'vs代码',
标签: 'VS 代码'
}, {
键: 'jetb雨s',
标签: 'JetB雨s'
}];
常量 ALT_目标S = {
桌面: {
名称: '桌面',
安装标签: '下载 the 应用',
安装Href: 'https://claude.com/下载?utm_来源=claude_代码&utm_中=docs&utm_满
意=configurator_桌面_下载',
指南Href: '/en/桌面-快速入门'
},
vs代码: {
名称: 'VS 代码',
安装标签: '安装 from 市场place',
安装Href: 'https://市场place.visualstudio.com/项s?项名称=anthropic.claude-代码',
altCmd: '代码 --安装-扩展 anthropic.claude-代码',
指南Href: '/en/vs-代码'
},
jetb雨s: {
名称: 'JetB雨s',
安装标签: '安装 from 市场place',
```

```
安装Href: 'https://插件s.jetb雨s.com/插件/27310-claude-代码-beta-',
指南Href: '/en/jetb雨s'
}
};
常量 提供者S = [{
键: 'anthropic',
标签: 'Anthropic'
}, {
键: '基岩',
标签: 'Amazon 基岩'
}, {
键: 'foun干',
标签: 'Micro软 Foun干'
}, {
键: 'vertex',
标签: '前往ogle Vertex AI'
}];
常量 提供者_通知 = {
基岩: <>
Con图 your AWS 说明 第一个. 运行中 on 基岩
requ怒s 模型 access 启用 in the AWS 控制台 and IAM credentials.{' '}
基岩 设置 指南 →
,
vertex: <>
Con图 your GCP 项目 第一个. 运行中 on Vertex AI
requ怒s the Vertex API 启用 and a 服务 说明 with the 正确
权限.{' '}
Vertex 设置 指南 →
,
foun干: <>
Con图 your Azure 资源 第一个. 运行中 on
Micro软 Foun干 requ怒s an Azure sub脚本ion with a Foun干 资源
and 模型 部署ments provisioned.{' '}
Foun干 设置 指南 →

};
常量 图标检查 = (尺寸 = 14) =>
```

```
;
```

常量 图标复制 = (尺寸 = 14) =>

```
;
```

常量 图标Ar行正确 = (尺寸 = 13) =>

```
;
```

常量 图标Ar行上正确 = (尺寸 = 14) =>

```
;
```

常量 图标信息 = (尺寸 = 16) =>

```
;
```

常量 [目标, 设置目标] = 使用州('终端');
常量 [团队, 设置团队] = 使用州(虚假);
常量 [提供者, 设置提供者] = 使用州('anthropic');
常量 [pkg, 设置Pkg] = 使用州(() => (/Win/).测试(navigator.用户代理) ? 'win' : 'mac');
常量 [winCmd, 设置WinCmd] = 使用州(虚假);
常量 [复制, 设置复制] = 使用州(空);
常量 复制时间r = 用户ef(空);
常量 handle复制 = 异步 (文本, 键) => {
尝试 {
等待 navigator.clip董事会.写入文本(文本);
} 捕获 {
常量 ta = document.创建元素('文本面积');
ta.值 = 文本;
document.正文.应用结束Child(ta);
ta.选择();
document.exec命令('复制');
document.正文.移除Child(ta);
}
清楚超时(复制时间r.当前);

```
设置复制(键);
复制时间r.当前 = 设置超时(() => 设置复制(空), 1800);
};
常量 card正文Cmd = (cmd, 及时) => {
常量 on = 复制 === 'term';
回报
{及时 || '$'}
{cmd}
handle复制(cmd, 'term')}>
{on ? 图标检查(13) : 图标复制(13)}
{on ? '复制' : '复制'}
```

```
    ;
```

```
};
常量 isWin安装er = pkg === 'win';
常量 isWin及时 = pkg === 'win' || pkg === 'WinGet';
常量 终端Cmd = isWin安装er ? WIN_VARIANTS[winCmd ? 'cmd' : 'ps'] :
TERM[pkg].cmd;
常量 alt = ALT_目标S[目标];
常量 显示通知 = 团队 && 提供者 !== 'anthropic';
常量 STYLES = `
.cc-ic {
--ic-板岩: #141413;
--ic-粘土: #d97757;
--ic-粘土-深: #c6613f;
--ic-gray-000: #ffffff;
--ic-gray-150: #f0eee6;
--ic-gray-550: #73726c;
--ic-gray-700: #3d3d3a;
--ic-b顺序-微妙: rgba(31, 30, 29, 0.08);
--ic-b顺序-默认: rgba(31, 30, 29, 0.15);
--ic-b顺序-强: rgba(31, 30, 29, 0.3);
--ic-font-mono: ui-monos步伐, SFMono-常规, Menlo, Monaco, 'Courier 新', monos步
伐;
font-家庭: 'Anthropic Sans', -应用le-系统, B链接Mac系统Font, 'Se前往e UI', sans-serif;
font-尺寸: 14px; 行-身高: 1.5; color: var(--ic-板岩);
```

```
margin: 8px 0 32px;
}
.暗 .cc-ic {
--ic-板岩: #f0eee6;
--ic-gray-000: #262624;
--ic-gray-150: #1f1e1d;
--ic-gray-550: #91908a;
--ic-gray-700: #bfbdb4;
--ic-b顺序-微妙: rgba(240, 238, 230, 0.08);
--ic-b顺序-默认: rgba(240, 238, 230, 0.14);
--ic-b顺序-强: rgba(240, 238, 230, 0.28);
}
.暗 .cc-ic-检查 { 背景: 透明; }
.暗 .cc-ic-card { b顺序: 0.5px 固体 var(--ic-b顺序-微妙); }
.暗 .cc-ic-p-pill.cc-ic-活跃 { box-shadow: 0 1px 2px rgba(0, 0, 0, 0.3); }
.cc-ic , .cc-ic ::之前, .cc-ic *::之后 { box-sizing: b顺序-box; }
.cc-ic a { 文本-装饰: 无; }
.cc-ic a:not([阶级]) { color: inherit; }
.cc-ic button { font-家庭: inherit; cursor: pointer; }

.cc-ic-tab-s旅行 {
显示: in行-flex; gap: 2px;
p添加ing: 4px; 背景: var(--ic-gray-150);
b顺序-半径: 10px; 结束f低-x: auto;
max-宽度: 100%;
}
.cc-ic-tab {
应用earance: 无; 背景: 无; b顺序: 无;
p添加ing: 10px 18px; font-尺寸: 15px; font-重量: 430;
color: var(--ic-gray-550); b顺序-半径: 7px;
white-s步伐: nowrap;
过渡: color 0.12s, 背景-color 0.12s;
}
.cc-ic-tab:h结束 { color: var(--ic-gray-700); }
.cc-ic-tab.cc-ic-活跃 {
color: var(--ic-板岩); font-重量: 500;
背景: var(--ic-gray-000);
box-shadow: 0 1px 3px rgba(0, 0, 0, 0.08);
```

```
}
.暗 .cc-ic-tab.cc-ic-活跃 { box-shadow: 0 1px 3px rgba(0, 0, 0, 0.4); }

.cc-ic-团队-wrap { p添加ing: 16px 0 20px; }
.cc-ic-团队-toggle {
显示: flex; align-项s: c进入; gap: 12px; font-家庭: inherit;
p添加ing: 12px 16px; font-尺寸: 14px; font-重量: 430;
color: var(--ic-gray-700); cursor: pointer; 用户-选择: 无;
宽度: fit-满意; 背景: var(--ic-gray-150);
b顺序: 0.5px 固体 var(--ic-b顺序-微妙); b顺序-半径: 8px;
过渡: b顺序-color 0.15s;
}
.cc-ic-团队-toggle:h结束 { b顺序-color: var(--ic-b顺序-默认); }
.cc-ic-团队-toggle.cc-ic-检查ed {
背景: rgba(217, 119, 87, 0.08);
b顺序-color: rgba(217, 119, 87, 0.25);
}
.cc-ic-检查 {
宽度: 16px; 身高: 16px;
b顺序: 1px 固体 var(--ic-b顺序-强); b顺序-半径: 4px;
背景: var(--ic-gray-000);
显示: flex; align-项s: c进入; 公正ify-满意: c进入;
flex-shrink: 0;
}
.cc-ic-检查 svg { color: #fff; 显示: 无; }
.cc-ic-团队-toggle.cc-ic-检查ed .cc-ic-检查 { 背景: var(--ic-粘土-深); b顺序-color: var(--
ic-粘土-深); }
.cc-ic-团队-toggle.cc-ic-检查ed .cc-ic-检查 svg { 显示: 块; }

.cc-ic-团队-reveal { 显示: flex; flex-指导: 列; gap: 12px; margin-底部: 16px; }
.cc-ic-销售 {
显示: flex; align-项s: c进入; 公正ify-满意: s步伐-between;
gap: 16px; p添加ing: 14px 16px;
背景: var(--ic-gray-000); b顺序: 0.5px 固体 var(--ic-b顺序-默认);
b顺序-半径: 8px; flex-wrap: wrap;
}
.cc-ic-销售-文本 { font-尺寸: 13px; color: var(--ic-gray-700); 行-身高: 1.5; flex: 1; min-宽
度: 200px; }
```

```
.cc-ic-销售-文本 强 { font-重量: 550; color: var(--ic-板岩); }
.cc-ic-销售-行动s { 显示: flex; align-项s: c进入; gap: 8px; flex-shrink: 0; }
.cc-ic-btn-粘土 {
显示: in行-flex; align-项s: c进入; gap: 8px;
背景: var(--ic-粘土-深); color: #fff; b顺序: 无;
b顺序-半径: 8px; p添加ing: 8px 14px;
font-尺寸: 13px; font-重量: 500;
过渡: 背景-color 0.15s; white-s步伐: nowrap;
}
.cc-ic-btn-粘土:h结束 { 背景: var(--ic-粘土); }
.cc-ic-btn-ghost {
显示: in行-flex; align-项s: c进入; gap: 8px;
背景: 透明; color: var(--ic-gray-700);
b顺序: 0.5px 固体 var(--ic-b顺序-默认);
b顺序-半径: 8px; p添加ing: 8px 14px;
font-尺寸: 13px; font-重量: 500;
}
.cc-ic-btn-ghost:h结束 { 背景: rgba(0, 0, 0, 0.04); }

.cc-ic-提供者-bar {
显示: flex; align-项s: c进入; gap: 12px;
p添加ing: 14px 16px; 背景: var(--ic-gray-150);
b顺序-半径: 8px; font-尺寸: 13px; flex-wrap: wrap;
}
.cc-ic-提供者-bar .cc-ic-标签 { color: var(--ic-gray-550); flex-shrink: 0; }
.cc-ic-提供者-pills { 显示: flex; gap: 4px; flex-wrap: wrap; }
.cc-ic-p-pill {
应用earance: 无; b顺序: 无; 背景: 透明;
p添加ing: 6px 12px; b顺序-半径: 6px;
font-尺寸: 13px; font-重量: 430; color: var(--ic-gray-700);
white-s步伐: nowrap;
}
.cc-ic-p-pill:h结束 { 背景: rgba(0, 0, 0, 0.04); }
.cc-ic-p-pill.cc-ic-活跃 {
背景: var(--ic-gray-000); color: var(--ic-板岩);
font-重量: 500; box-shadow: 0 1px 2px rgba(0, 0, 0, 0.05);
}
.cc-ic-提供者-通知 {
```

显示: flex; p添加ing: 16px 18px;
背景: var(--ic-gray-000); b顺序: 0.5px 固体 var(--ic-b顺序-默认);
b顺序-半径: 8px; gap: 14px; align-项s: flex-启动;
}
.cc-ic-提供者-通知 > svg { color: var(--ic-gray-550); margin-顶部: 2px; flex-shrink: 0; }
.cc-ic-提供者-通知-正文 { font-尺寸: 14px; 行-身高: 1.55; color: var(--ic-gray-700); }
.cc-ic-提供者-通知-正文 强 { font-重量: 550; color: var(--ic-板岩); }
.cc-ic-提供者-通知-正文 a { color: var(--ic-粘土-深); font-重量: 500; }
.cc-ic-提供者-通知-正文 a:h结束 { 文本-装饰: under行; }

.cc-ic-card { 背景: #141413; b顺序-半径: 12px; 结束f低: 隐藏; }
.cc-ic-subtabs {
显示: flex; align-项s: c进入;
背景: #1a1918;
b顺序-底部: 0.5px 固体 rgba(255, 255, 255, 0.08);
p添加ing: 0 8px; 结束f低-x: auto;
}
.cc-ic-subtab-s步伐r { flex: 1; }
.cc-ic-subtab {
应用earance: 无; 背景: 无; b顺序: 无;
p添加ing: 12px 16px; font-尺寸: 12px;
color: rgba(255, 255, 255, 0.5);
位置: relative; white-s步伐: nowrap;
}
.cc-ic-subtab:h结束 { color: rgba(255, 255, 255, 0.75); }
.cc-ic-subtab.cc-ic-活跃 { color: #fff; }
.cc-ic-subtab.cc-ic-活跃::之后 {
满意: ''; 位置: absolute;
左: 12px; 正确: 12px; 底部: -0.5px;
身高: 2px; 背景: var(--ic-粘土);
}
.cc-ic-cmd-toggle {
显示: flex; align-项s: c进入; gap: 8px; font-家庭: inherit;
背景: 无; b顺序: 无;
p添加ing: 0 12px; font-尺寸: 11px;
color: rgba(255, 255, 255, 0.5);
cursor: pointer; 用户-选择: 无; white-s步伐: nowrap;
}

```
.cc-ic-cmd-toggle:h结束 { color: rgba(255, 255, 255, 0.75); }
.cc-ic-mini-检查 {
宽度: 12px; 身高: 12px;
b顺序: 1px 固体 rgba(255, 255, 255, 0.3); b顺序-半径: 3px;
显示: flex; align-项s: c进入; 公正ify-满意: c进入;
flex-shrink: 0;
}
.cc-ic-mini-检查 svg { color: #fff; 显示: 无; }
.cc-ic-cmd-toggle.cc-ic-检查ed .cc-ic-mini-检查 { 背景: var(--ic-粘土-深); b顺序-color: var(--ic-粘土-深); }
.cc-ic-cmd-toggle.cc-ic-检查ed .cc-ic-mini-检查 svg { 显示: 块; }

.cc-ic-card-正文 { p添加ing: 24px 26px; 显示: flex; align-项s: flex-启动; gap: 14px; }
.cc-ic-及时 {
color: var(--ic-粘土); font-家庭: var(--ic-font-mono);
font-尺寸: 17px; 用户-选择: 无; p添加ing-顶部: 2px;
}
.cc-ic-cmd {
flex: 1; font-家庭: var(--ic-font-mono);
font-尺寸: 17px; color: #f0eee6;
行-身高: 1.55; white-s步伐: pre-wrap; 词-break: break-词;
}
.cc-ic-复制 {
显示: in行-flex; align-项s: c进入; gap: 6px;
背景: rgba(255, 255, 255, 0.08);
b顺序: 0.5px 固体 rgba(255, 255, 255, 0.12);
color: rgba(255, 255, 255, 0.85);
p添加ing: 7px 13px; b顺序-半径: 8px;
font-尺寸: 13px; font-重量: 500; flex-shrink: 0;
}
.cc-ic-复制:h结束 { 背景: rgba(255, 255, 255, 0.14); }
.cc-ic-复制.cc-ic-复制 { 背景: var(--ic-粘土-深); b顺序-color: var(--ic-粘土-深); color: #fff; }

.cc-ic-be低 {
margin-顶部: 12px; font-尺寸: 13px; color: var(--ic-gray-550);
显示: flex; gap: 16px; flex-wrap: wrap; align-项s: 基础行;
}
```

```
.cc-ic-be低 a { color: var(--ic-gray-700); b顺序-底部: 0.5px 固体 var(--ic-b顺序-默认); }
.cc-ic-be低 a:h结束 { color: var(--ic-粘土-深); b顺序-底部-color: var(--ic-粘土-深); }
.cc-ic-handoff {
p添加ing: 20px 22px;
背景: var(--ic-gray-000);
b顺序: 0.5px 固体 var(--ic-b顺序-默认);
b顺序-半径: 12px;
}
.cc-ic-handoff-负责人 {
font-尺寸: 14px; 行-身高: 1.55; color: var(--ic-gray-700);
margin-底部: 14px;
}
.cc-ic-handoff-负责人 强 { font-重量: 550; color: var(--ic-板岩); }
.cc-ic-handoff-行动s { 显示: flex; gap: 10px; flex-wrap: wrap; }
.cc-ic-handoff-alt {
margin-顶部: 12px; font-尺寸: 12px; color: var(--ic-gray-550);
}
.cc-ic-handoff-alt 代码 {
font-家庭: var(--ic-font-mono); font-尺寸: 11px;
背景: var(--ic-gray-150); p添加ing: 2px 6px;
b顺序-半径: 4px; color: var(--ic-gray-700);
}
.cc-ic-复制-sm {
应用earance: 无; b顺序: 无;
显示: in行-flex; align-项s: c进入; 公正ify-满意: c进入;
宽度: 22px; 身高: 22px;
margin-左: 4px; 垂直-align: middle;
背景: var(--ic-gray-150); color: var(--ic-gray-550);
b顺序-半径: 4px;
过渡: color 0.1s, 背景-color 0.1s;
}
.cc-ic-复制-sm:h结束 { color: var(--ic-gray-700); 背景: var(--ic-b顺序-默认); }
.cc-ic-复制-sm.cc-ic-复制 { 背景: var(--ic-粘土-深); color: #fff; }

@media (max-宽度: 720px) {
.cc-ic-tab { p添加ing: 12px 14px; font-尺寸: 14px; }
.cc-ic-销售-行动s { 宽度: 100%; }
.cc-ic-card-正文 { p添加ing: 20px; }
```

```
.cc-ic-cmd { font-尺寸: 15px; }
}
`;
回报
{STYLES}
```

```
 {}

   {TABS.映射(t =>  设置目标(t.键)}>
       {t.标签}
     )}


{}

   设置团队(!团队)}>
     {图标检查(11)}

       I’m buying for a 团队 or 公司 (SSO, AWS/Azure/GCP, 中央计费)




{}
{团队 &&


       设置 上 your 团队: self-serve or talk to 销售.


         开始使用


           Contact 销售 {图标Ar行正确()}



      运行 on

          {提供者S.映射(p =>  设置提供者(p.键)}>
```

```
              {p.标签}
            )}



    {显示通知 &&
        {图标信息()}

          {提供者_通知[提供者]}

      }
  }

{}
{目标 === '终端' &&

      {对象.键s(TERM).映射(k =>  设置Pkg(k)}>
          {TERM[k].标签}
        )}

      {isWin安装er &&  设置WinCmd(!winCmd)}>
          {图标检查(9)}
          CMD instead of 权力命令行
        }

    {card正文Cmd(终端Cmd, isWin及时 ? '>' : '$')}
  }

{}
{目标 === '终端' &&
    {isWin安装er &&
        Requ怒s{' '}

          Git for 风ows
        .
      }
    {(pkg === 'brew' || pkg === 'WinGet') &&
```

```
          Does not auto-更新. 运行{' '}
          {pkg === 'brew' ? 'brew 升级 claude-代码' : 'WinGet 升级 Anthropi
        期间ic所有y.
        }
      故障排除
    }

  {alt &&

        The步骤s be低 使用 the 命令 行.{' '}
        Prefer {alt.名称}? 安装 here, then fol低 the {alt.名称} 指南 instead



          {alt.安装标签} {图标Ar行上正确(13)}


          {alt.名称} 指南 {图标Ar行正确(12)}


      {alt.altCmd &&
          or 运行 {alt.altCmd}
           handle复制(alt.altCmd, 'alt')} aria-标签="复制 命令">
            {复制 === 'alt' ? 图标检查(11) : 图标复制(11)}

        }
    }
;
```

};

出口 常量 实验 = ({flag, treatment, children}) => {
常量 VID_键 = 'exp_vid';
常量 CON发送_COUNTRIES = 新 设置(['AT', 'BE', 'BG', 'HR', 'CY', 'CZ', 'DK', 'EE', 'FI', 'FR', 'DE', 'GR', 'HU', 'IE', 'IT', 'LV', 'LT', 'LU', 'MT', 'NL', 'PL', 'PT', 'RO', 'SK', 'SI', 'ES', 'SE', 'RE', 'GP', 'MQ', 'GF', 'YT', 'BL', 'MF', 'PM', 'WF', 'PF', 'NC', 'AW', 'CW', 'SX', 'FO', 'GL', 'AX', 'GB', 'UK', 'AI', 'BM', 'IO', 'VG', 'KY', 'FK', 'GI', 'MS', 'PN', 'SH', 'TC', 'GG', 'JE', 'IM', 'CA', 'BR', 'IN']);

```
常量 fnv1a = s => {
let h = 0x811c9dc5;
for (let i = 0; i >> 0;
};
常量 bucket = (种子, vid) => fnv1a(fnv1a(种子 + vid) + '') % 10000 {
常量 params = 新 URL搜索Params(location.搜索);
常量 强制 = params.get('gb-强制');
if (强制) {
for (常量 p of 强制.split(',')) {
常量 [k, v] = p.split(':');
if (k === flag) 回报 {
variant: v || 'treatment',
跟踪: 虚假
};
}
}
if (navigator.全球隐私控制) {
回报 {
variant: '控制',
跟踪: 虚假
};
}
常量 prefs匹配 = document.首席运营官kie.匹配(/(?:^|; )anthropic-con发送-偏好设置
=([^;]+)/);
if (prefs匹配) {
尝试 {
if (JSON.parse(解码URI组件(prefs匹配[1])).analytics !== 真实) {
回报 {
variant: '控制',
跟踪: 虚假
};
}
} 捕获 {
回报 {
variant: '控制',
跟踪: 虚假
};
```

```
}
} else {
常量 coun尝试 = params.get('coun尝试')?.to上per案例() || (document.首席运营官kie.匹配(/(?:^|; )cf_geo=([A-Z]{2})/) || [])[1];
if (!coun尝试 || CON发送_COUNTRIES.has(coun尝试)) {
回报 {
variant: '控制',
跟踪: 虚假
};
}
}
let vid;
尝试 {
常量 ajs匹配 = document.首席运营官kie.匹配(/(?:^|; )ajs_anonymous_id=([^;]+)/);
if (ajs匹配) {
vid = 解码URI组件(ajs匹配[1]).替换(/^"|"$/g, '');
} else {
vid = 本地Sto狂怒.get项(VID_键);
if (!vid) {
vid = crypto.随机UUID();
}
document.首席运营官kie = `ajs_anonymous_id=${vid}; do主=.claude.com; 路径=/; 安全; 相同Site=Lax; max-age=31536000`;
}
尝试 {
本地Sto狂怒.设置项(VID_键, vid);
} 捕获 {}
} 捕获 {
回报 {
variant: '控制',
跟踪: 虚假
};
}
回报 {
variant: bucket(flag, vid),
跟踪: 真实,
vid
```

```
};
});
使用效果(() => {
if (!决定.跟踪) 回报;
获取('https://api.anthropic.com/api/事件_日志ging/v2/batch', {
方法: 'POST',
页眉s: {
'满意-类型': '应用程序/json',
'x-服务-名称': 'claude_代码_docs'
},
正文: JSON.字符串ify({
事件s: [{
事件_类型: '增长book实验事件',
事件_数据: {
device_id: 决定.vid,
anonymous_id: 决定.vid,
时间戳: 新 日期().toISO字符串(),
实验_id: flag,
变化_id: 决定.variant === 'treatment' ? 1 : 0,
环境: '生产'
}
}]
}),
keep活着: 真实
}).捕获(() => {});
}, []);
回报 决定.variant === 'treatment' ? treatment : children;
};
```

This 快速入门 指南 will have you using AI-权力ed coding 协助 in a 几个 微小s. By the 结束, you'll understand how to 使用 Claude 代码 for 常见 开发 任务s.

} />

之前 you begin

Make 确定 you have:

- A 终端 or 命令 及时 打开

- If you've never 使用d the 终端 之前, 检查 out the 终端 指南
- A 代码 项目 to 工作 with
- A Claude 订阅 (Pro, Max, 团队, or 企业), Claude 控制台 说明, or access th粗糙 a 支持ed 云 提供者

This 指南 c结束s the 终端 CLI. Claude 代码 is also 可用 on the 网页, as a 桌面 应用, in VS 代码 and JetB雨s IDEs, in 松弛, and in CI/CD with GitHub 行动s and GitLab.看见 所有 接口s.

步骤 1: 安装 Claude 代码

To 安装 Claude 代码, 使用 one of the跟随 方法:

macOS、Linux、WSL:

```bash  主题={空}
curl -fsSL https://claude.ai/安装.sh | bash
```

风ows 权力命令行:

```权力命令行  主题={空}
irm https://claude.ai/安装.ps1 | iex
```

风ows CMD:

```batch  主题={空}
curl -fsSL https://claude.ai/安装.cmd -o 安装.cmd && 安装.cmd && del 安装.c
```

If you看见 `The 令牌 '&&' is not a 有效 州ment separator`, you're in 权力命令

风ows requ怒s [Git for 风ows](https://Git-scm.com/下载s/win). 安装 it 第

原生安装ations 自动所有y 更新 in the 背景 to keep you on the 最新 版本.

```bash  主题={空}
brew 安装 --cask claude-代码
```

Homebrew 安装s do not auto-更新. 运行 `brew 升级 claude-代码` 期间ic所有y t

```
```权力命令行  主题={空}
WinGet 安装 Anthropic.Clau解码
```


  WinGet 安装s do not auto-更新. 运行 `WinGet 升级 Anthropic.Clau解码` 期间ic
```

步骤 2: 日志 in to your 说明

Claude 代码 requ怒s an 说明 to 使用. When you 启动 an 交互 会话 with the `claude` 命令, you'll 需要 to 日志 in:

```bash 主题={空}
claude

# You'll be 及时ed to 日志 in on 第一个 使用
```

```bash  主题={空}
/日志in
# Fol低 the 及时s to 日志 in with your 说明
```

You can 日志 in using 任何 of these 说明 类型s:

- Claude Pro, Max, 团队, or 企业 (推荐)
- Claude 控制台 (API access with pre-支付 鸣谢). On 第一个 日志in, a "Claude 代码" 工作区 is 自动所有y 创建d in the 控制台 for 集中 成本 跟踪ing.
- Amazon 基岩, 前往ogle Vertex AI, or Micro软 Foun干 (企业 云 提供者s)

Once 日志ged in, your credentials are 存储 and you 获胜't 需要 to 日志 in a收益. To switch 说明s 更晚, 使用 the `/日志in` 命令.

步骤 3: 启动 your 第一个 会话

打开 your 终端 in 任何 项目 目录 and 启动 Claude 代码:

```bash 主题={空}
cd /路径/to/your/项目
claude
```

```
You'll看见 the Claude 代码 welcome screen with your 会话 信息, 最近 对话s, an

  之后日志 in (步骤 2), your credentials are 存储 on your 系统. Learn 更多 in

##步骤 4: Ask your 第一个 问题

Let's 启动 with 理解 your 代码基础. 尝试 one of these 命令:

```文本  主题={空}
what does this 项目 do?
```

Claude will分析 your 文件 and provide a 摘要. You can also ask 更多 特定 问题s:

```文本 主题={空}
what techno日志ies does this 项目 使用?
```

```
```文本  主题={空}
where is the 主 条目 point?
```

```文本 主题={空}
ex简单 the 文件夹 结构
```

```
You can also ask Claude about its own 能力:

```文本  主题={空}
what can Claude 代码 do?
```
```

```文本 主题={空}
how do I 创建 习俗 技能 in Claude 代码?
```

```文本  主题={空}
can Claude 代码 工作 with Docker?
```

Claude 代码 读取s your 项目 文件 as 需要ed. You don't have to 手册ly 添加 上下文.

步骤 5: Make your 第一个 代码 更改

Now let's make Claude 代码 do 一些 实际 coding. 尝试 a 简单 任务:

```文本 主题={空}
添加 a hello world 功能 to the 主 文件
```

```
Claude 代码 will:

1. 查找 the 恰当 文件
2. 显示 you the 支撑osed 更改s
3. Ask for your 批准
4. Make the 编辑

  Claude 代码 al方式s asks for 许可 之前修改 文件. You can 批准 个人 更改s or 启

##步骤 6: 使用 Git with Claude 代码

Claude 代码 makes Git 运营 对话al:

```文本  主题={空}
what 文件 have I 更改d?
```

```文本 主题={空}
提交 my 更改s with a de脚本ive 消息
```
```

```
You can also 及时 for 更多 复杂 Git 运营:

```文本  主题={空}
创建 a 新 分支 c所有ed feature/快速入门
```

```文本 主题={空}
显示 me the 最后一个 5 提交s
```

```
```文本  主题={空}
帮助 me resolve 合并 conflicts
```

步骤 7: 修复 a 缺陷 or 添加 a feature

Claude is 精通 at 调试ging and feature 实施.

Describe what you 想要 in 自然 language:

```文本 主题={空}
添加 输入 验证 to the 用户 regist比率n 形式
```

```
Or 修复 现有 问题s:

```文本  主题={空}
there's a 缺陷 where 用户s can submit 空 形式s - 修复 it
```

Claude 代码 will:

- Locate the 相关 代码
- Understand the 上下文
- 实现 a 解决
- 运行 测试s if 可用
```

步骤 8: 测试 out other 通用工作流

There are a 数字 of 方式s to 工作 with Claude:

Re事实or 代码

```文本 主题={空}
re事实or the 认证 模块 to 使用 异步/等待 instead of 回调s
```

```
**写入 测试s**

```文本  主题={空}
写入 单位 测试s for the calculator 功能s
```

更新 文档

```文本 主题={空}
更新 the 读取ME with 安装 说明
```

```
**代码 re视图**

```文本  主题={空}
re视图 my 更改s and suggest 改善s
```

Talk to Claude 像 you would a 有帮助 colleague. Describe what you 想要 to achieve, and it will 帮助 you get there.

必要 命令

Here are the 最多 重要 命令 for daily 使用:

命令	What it does	示例
`claude`	启动 交互 模式	`claude`
`claude "任务"`	运行 a one-时间 任务	

命令	What it does	示例
		`claude "修复 the 构建 错误"`
`claude -p "查询"`	运行 one-off 查询, then 退出	`claude -p "ex简单 this 功能"`
`claude -c`	继续 最多 最近 对话 in 当前 目录	`claude -c`
`claude -r`	恢复 a 之前 对话	`claude -r`
`/清楚`	清楚 对话 history	`/清楚`
`/帮助`	显示 可用 命令	`/帮助`
`退出` or Ctrl+D	退出 Claude 代码	`退出`

看见 the CLI 参考 for a 完成 列表 of 命令.

Pro 提示 for 初学者s

For 更多,看见 最佳实践 and 通用工作流.

```
 Instead of: "修复 the 缺陷"

尝试: "修复 the 日志in 缺陷 where 用户s看见 a 空白 screen 之后进入 错误 credenti



Break 复杂 任务s into步骤s:

```文本  主题={空}
1. 创建 a 新 数据库 表 for 用户 pro文件
2. 创建 an API 结束point to get and 更新 用户 pro文件
3. 构建 a 网页页 that 允许s 用户s to看见 and 编辑 their 信息
```


之前制造 更改s, let Claude understand your 代码:

```文本  主题={空}
```

分析 the 数据库 模式
```

```
 ```文本  主题={空}
构建 a dash董事会显示 产品s that are 最多 频繁ly 返回 by our UK 习俗ers
```


* Press `?` to看见 所有 可用 键董事会 短剪切s
* 使用 Tab for 命令 completion
* Press ↑ for 命令 history
* 类型 `/` to看见 所有 命令 and 技能
```

What's 下一个?

Now that you've 学习 the 基础s,探索 更多 先进 features:

```
Understand the 代理ic loop, built-in 工具, and how Claude 代码 interacts w
```

```
Get 更好 结果s with 有效 及时ing and 项目 设置
```

步骤-by-步骤 指南s for 常见 任务s

```
习俗ize with CLAUDE.md, 技能, 钩子, MCP, and 更多
```

Getting 帮助

- **In Claude 代码**: 类型 `/帮助` or ask "how do I..."
- **文档**: You're here! 浏览 other 指南s
- **社区**: Join our 不和 for 提示 and 支持

记忆与指令

文档 索引

获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.

Claude 如何记住您的项目

> Give Claude 持续 说明 with CLAUDE.md 文件, and let Claude accumu晚 learnings 自动所有y with auto 记忆.

每个 Claude 代码 会话 begins with a 新鲜 上下文 风ow. Two mechanisms carry 知识 across 会话s:

- **CLAUDE.md 文件**: 说明 you 写入 to give Claude 持续 上下文
- **Auto 记忆**: 注意s Claude 写入s itself 基础d on your 正确ions and 偏好设置

This 页 c结束s how to:

- 写入 and organize CLAUDE.md 文件
- 范围 规则 to 特定 文件 类型s with `.claude/规则/`
- Con图 auto 记忆 so Claude takes 注意s 自动所有y
- 麻烦shoot when 说明 aren't being foll欠

CLAUDE.md vs auto 记忆

Claude 代码 has two complementary 记忆 系统s. 机器人h are 加载 at the 启动 of 每个 对话. Claude treats them as 上下文, not en强迫 配置. The 更多 特定 and concise your 说明, the 更多 一致ly Claude fol低s them.

	CLAUDE.md 文件	Auto 记忆
Who 写入s it	You	Claude
What it contains	说明 and 规则	Learnings and 模式s
范围	项目, 用户, or org	Per工作 树
加载 into	每个 会话	每个 会话 (第一个 200 行s or 25KB)
使用 for	Coding 标准, 工作流s, 项目 架构	构建 命令, 调试ging insights, 偏好设置 Claude disc结束s

使用 CLAUDE.md 文件 when you 想要 to 指南 Claude's behavior. Auto 记忆 lets Claude learn from your 正确ions without 手册 effort.

子代理 can also 主tain their own auto 记忆.看见 sub代理 配置 for 详情s.

CLAUDE.md 文件

CLAUDE.md 文件 are mark下 文件 that give Claude 持续 说明 for a 项目, your 个人 工作流, or your 整个 组织. You 写入 these 文件 in 简单 文本; Claude 读取s them at the 启动 of 每个 会话.

Choose where to put CLAUDE.md 文件

CLAUDE.md 文件 can live in 几个 locations, 每个 with a 不同 范围. 更多 特定 locations take precedence 结束 broader ones.

范围	Location	目的	使用 案例 示例	共享 with
管理政策	• macOS: `/库/应用程序 支持/Clau解码/CLAUDE.md` • Linux and WSL: `/etc/claude-代码/CLAUDE.md` • 风ows: `C:\计划文件\Clau解码\CLAUDE.md`	组织-wide 说明 管理 by IT/DevOps	公司 coding 标准, 安全 政策, 合规 要求	所有 用户s in 组织
项目说明	`./CLAUDE.md` or `./.claude/CLAUDE.md`	团队-共享 说明 for the 项目	项目 架构, coding 标准, 通用工作流	团队 成员s via 来源 控制
用户说明	`~/.claude/CLAUDE.md`	个人 偏好设置 for 所有 项目s	代码 styling 偏好设置, 个人 工具ing 短剪切s	公正 you (所有 项目s)
本地	`./CLAUDE.本地.md`	个人 项目-特定 偏好设置; 添加 to `.Gitignore`	Your 沙box URLs,	公正 you

范围	Location	目的	使用 案例 示例	共享 with
说明			preferred 测试 数据	(当前项目)

CLAUDE.md and CLAUDE.本地.md 文件 in the 目录 hierarchy above the工作 目录 are 加载 in 满 at 启动. 文件 in sub总监ies 加载 on 需求 when Claude 读取s 文件 in those 总监ies.看见 How CLAUDE.md 文件 加载 for the 满 决议 顺序.

For large 项目s, you can break 说明 into 话题-特定 文件 using 项目 规则. 规则 let you 范围 说明 to 特定 文件 类型s or sub总监ies.

设置 上 a 项目 CLAUDE.md

A 项目 CLAUDE.md can be 存储 in either `./CLAUDE.md` or `./.claude/CLAUDE.md`. 创建 this 文件 and 添加 说明 that 应用ly to 任何one工作 on the 项目: 构建 and 测试 命令, coding 标准, 架构师ural 决定s, naming 惯例, and 通用工作流. These 说明 are 共享 with your 团队 th粗糙 版本 控制, so 焦点 on 项目-级别 标准 rather than 个人 偏好设置.

运行 `/init` to gene速率 a 启动ing CLAUDE.md 自动所有y. Claude分析s your 代码基础 and 创建s a 文件 with 构建 命令, 测试 说明, and 项目 惯例 it disc结束s. If a CLAUDE.md al就绪 exists, `/init` suggests 改善s rather than 结束writing it. Re好 from there with 说明 Claude wouldn't disc结束 on its own.

设置 `CLAUDE_代码_新_INIT=1` to 启用 an 交互 multi-阶段 f低. `/init` asks which 艺术i事实s to 设置 上: CLAUDE.md 文件, 技能, and 钩子. It then探索s your 代码基础 with a sub代理, fills in gaps via fol低-上 问题s, and 现在s a re视图能够 支撑osal 之前写作 任何 文件.

写入 有效 说明

CLAUDE.md 文件 are 加载 into the 上下文 风ow at the 启动 of 每个 会话, con总和ing 令牌s a长side your 对话. The 上下文 风ow 可视化 显示s where CLAUDE.md 加载s relative to the rest of the 启动上 上下文. Be原因 they're 上下文 rather than en强迫 配置, how you 写入 说明 affects how reliably Claude fol低s them. 特定, concise, well-结构化 说明 工作 最佳.

尺寸: 目标 under 200 行s per CLAUDE.md 文件. 长er 文件 con总和e 更多 上下文 and reduce adherence. If your 说明 are生长 large, split them using 进口s or `.claude/规则/` 文件.

结构: 使用 mark下 页眉s and bullets to 组 r兴高采烈 说明. Claude扫描s 结构 the 相同 方式 读取器s do: 组织 节s are easier to fol低 than 密集 段落s.

特定: 写入 说明 that are 具体 enough to 验证. For 示例:

- "使用 2-s步伐 indentation" instead of "格式 代码 适当ly"
- "运行 `npm 测试` 之前 提交ting" instead of "测试 your 更改s"
- "API 处理器s live in `src/api/处理器s/`" instead of "Keep 文件 组织"

一致: if two 规则 contradict 每个 other, Claude may pick one arbitrarily. Re视图 your CLAUDE.md 文件, nested CLAUDE.md 文件 in sub总监ies, and `.claude/规则/` 期间 ic所有y to 移除 过时 or conflicting 说明. In monorepos, 使用 `claudeMdExcludes` to 跳 CLAUDE.md 文件 from other 团队 that aren't 相关 to your 工作.

进口 添加itional 文件

CLAUDE.md 文件 can 进口 添加itional 文件 using `@路径/to/进口` syn税. 导入 文件 are 扩展 and 加载 into 上下文 at 启动 a长side the CLAUDE.md that 参考文献 them.

机器人h relative and absolute 路径s are 允许. Relative 路径s resolve relative to the 文件包含 the 进口, not the工作 目录. 导入 文件 can 递归ly 进口 other 文件, with a 最大 深度 of five跳s.

To 拉取 in a 读取ME, 包.json, and a 工作流 指南, 参考 them with `@` syn税 任何where in your CLAUDE.md:

```文本 主题={空}
看见 @读取ME for 项目 概述 and @包.json for 可用 npm 命令 for this 项目.
```

添加itional 说明

- Git 工作流 @docs/Git-说明.md

```
For 私人 per-项目 偏好设置 that shouldn't be 检查ed into 版本 控制, 创建 a `CLA

If you 工作 across mul提示le Git 工作树 of the 相同 仓库, a Git忽略 `CLAUDE.本

```文本  主题={空}
# 个人 偏好设置
- @~/.claude/my-项目-说明.md
```

The 第一个 时间 Claude 代码 encounters 外部 进口s in a 项目, it 显示s an 批准 dia日志 列表 the 文件. If you 下降, the 进口s stay 禁用 and the dia日志 does not 应用ear a收益.

For a 更多 结构化 方法 to组织 说明,看见 `.claude/规则/`.

代理S.md

Claude 代码 读取s `CLAUDE.md`, not `代理S.md`. If your 仓库 al就绪 使用s `代理S.md` for other coding 代理s, 创建 a `CLAUDE.md` that 进口s it so 机器人h 工具 读取 the 相同 说明 without复制 them. You can also 添加 Claude-特定 说明 be低 the 进口. Claude 加载s the 导入 文件 at 会话 启动, then 应用结束s the rest:

```mark下 CLAUDE.md 主题={空}
@代理S.md
```

Claude 代码

使用 计划 模式 for 更改s under `src/billing/`.
```

```
### How CLAUDE.md 文件 加载

Claude 代码 读取s CLAUDE.md 文件 by走ing 上 the 目录 树 from your 当前工作 目录

所有 disc结束ed 文件 are concatenated into 上下文 rather than 结束riding 每个

Claude also disc结束s `CLAUDE.md` and `CLAUDE.本地.md` 文件 in sub总监ies u

If you 工作 in a large monorepo where other 团队' CLAUDE.md 文件 get picked

块-级别 HTML comments (``) in CLAUDE.md 文件 are s旅行ped 之前 the 满意 is in

#### 加载 from 添加itional 总监ies

The `--添加-dir` flag gives Claude access to 添加itional 总监ies 外部 your

To also 加载 CLAUDE.md 文件 from 添加itional 总监ies,包括 `CLAUDE.md`, `.cla

```bash  主题={空}
CLAUDE_代码_添加ITIONAL_总监IES_CLAUDE_MD=1 claude --添加-dir ../共享-config
```

`CLAUDE.本地.md` 文件 in 添加itional 总监ies are not 加载.

Organize 规则 with `.claude/规则/`

For larger 项目s, you can organize 说明 into mul提示le 文件 using the `.claude/规则/` 目录. This keeps 说明 模块化 and easier for 团队 to 主tain. 规则 can also be 范围d to 特定 文件 路径s, so they only 加载 into 上下文 when Claude 工作s with 匹配 文件, reducing 噪音 and保存 上下文 s步伐.

规则 加载 into 上下文 每个 会话 or when 匹配 文件 are 打开. For 任务-特定 说明 that don't 需要 to be in 上下文 所有 the 时间, 使用 技能 instead, which only 加载 when you invoke them or when Claude determines they're 相关 to your 及时.
```

设置 上 规则

Place mark下 文件 in your 项目's `.claude/规则/` 目录. 每个 文件 should c结束 one 话题, with a de脚本ive 文件名称 像 `测试.md` or `api-设计.md`. 所有 `.md` 文件 are disc结束ed 递归ly, so you can organize 规则 into sub总监ies 像 `front结束/` or `返回结束/`:

```文本 主题={空}
your-项目/
├── .claude/
│   ├── CLAUDE.md # 主 项目 说明
│   └── 规则/
│       ├── 代码-style.md # 代码 style 准则
│       ├── 测试.md # 测试 惯例
│       └── 安全.md # 安全 要求
```

```
规则 without [`路径s` front事情](#路径-特定-规则) are 加载 at 启动 with the 相同

#### 路径-特定 规则

规则 can be 范围d to 特定 文件 using YAML front事情 with the `路径s` 字段. The

```mark下  主题={空}
---
路径s:
  - "src/api/**/*.ts"
---

# API 开发 规则

- 所有 API 结束points must include 输入 验证
- 使用 the 标准 错误 响应 格式
- Include 打开API 文档 comments
```

规则 without a `路径s` 字段 are 加载 un条件ly and 应用ly to 所有 文件. 路径-范围d 规则 trigger when Claude 读取s 文件 匹配 the 模式, not on 每个 工具 使用.
```

使用 glob 模式s in the `路径s` 字段 to 匹配 文件 by 扩展, 目录, or 任何 组合:

模式	匹配es
`**/*.ts`	所有 类型脚本 文件 in 任何 目录
`src/**/*`	所有 文件 under `src/` 目录
`*.md`	Mark下 文件 in the 项目 根
`src/组件s/*.tsx`	React 组件s in a 特定 目录

You can 规格ify mul提示le 模式s and 使用 b种族 expansion to 匹配 mul提示le 扩展s in one 模式:

```mark下 主题={空}

路径s:
- "src//**.{ts,tsx}"
- "lib//.ts"
- "测试s//.测试.ts"

```
#### Share 规则 across 项目s with sym链接s

The `.claude/规则/` 目录 支持s sym链接s, so you can 主tain a 共享 设置 of 规则

This 示例 链接s 机器人h a 共享 目录 and an 个人 文件:

```bash  主题={空}
ln -s ~/共享-claude-规则 .claude/规则/共享
ln -s ~/公司-标准/安全.md .claude/规则/安全.md
```

用户-级别 规则

个人 规则 in `~/.claude/规则/` 应用ly to 每个 项目 on your machine. 使用 them for 偏好设置 that aren't 项目-特定:
```

```文本 主题={空}
~/.claude/规则/
 ├── 偏好设置.md # Your 个人 coding 偏好设置
 └── 工作流s.md # Your preferred 工作流s
```

```
用户-级别 规则 are 加载 之前 项目 规则,给 项目 规则 高er 优先级.

### Manage CLAUDE.md for large 团队

For 组织s 部署ing Claude 代码 across 团队, you can 中央ize 说明 and 控制 which

#### 部署 组织-wide CLAUDE.md

组织s can 部署 a 中央ly 管理 CLAUDE.md that 应用lies to 所有 用户s on a machin


    * macOS: `/库/应用程序 支持/Clau解码/CLAUDE.md`
    * Linux and WSL: `/etc/claude-代码/CLAUDE.md`
    * 风ows: `C:\计划 文件\Clau解码\CLAUDE.md`



   使用 MDM, 组 政策, Ansible, or 相似 工具 to distribute the 文件 across 开发


A 管理 CLAUDE.md and [管理 设置](/en/设置#设置-文件) serve 不同 目的s. 使用 设置

| 关心                                          | Con图 in
| :-------------------------------------------- | :--------------------
| 块 特定 工具, 命令, or 文件 路径s  | 管理 设置: `权限.拒绝`
| En强制 沙box isolation                        | 管理 设置: `沙box.启用`
| 环境变量 and API 提供者 r郊游 | 管理 设置: `env`
| 认证 方法 and 组织 锁定     | 管理 设置: `强制日志in方法`, `强制日志inOrgUUID`
| 代码 style and quality 准则                | 管理 CLAUDE.md
| 数据处理 and 合规 reminders          | 管理 CLAUDE.md
| 行为 说明 for Claude              | 管理 CLAUDE.md

设置 规则 are en强迫 by the 客户端 注意较少 of what Claude decides to do. CLAU

#### Exclude 特定 CLAUDE.md 文件
```

```
In large monorepos, ancestor CLAUDE.md 文件 may contain 说明 that aren't 相

This 示例 excludes a 顶部-级别 CLAUDE.md and a 规则 目录 from a parent 文件夹

```json  主题={空}
{
  "claudeMdExcludes": [
    "**/monorepo/CLAUDE.md",
    "/home/用户/monorepo/other-团队/.claude/规则/**"
  ]
}
```

模式s are 匹配ed a收益st absolute 文件 路径s using glob syn税. You can con图 `claudeMdExcludes` at 任何 设置 layer: 用户, 项目, 本地, or 管理 政策. 数组s 合并 across layers.

管理 政策 CLAUDE.md 文件 cannot be 排除. This en确定s 组织-wide 说明 al方式s 应用 ly 注意较少 of 个人 设置.

Auto 记忆

Auto 记忆 lets Claude accumu晚 知识 across 会话s without you写作 任何薄g. Claude 保存s 注意s for itself as it 工作s: 构建 命令, 调试ging insights, 架构 注意s, 代码 style 偏好设置, and 工作流 习惯s. Claude doesn't 保存 一些薄g 每个 会话. It decides what's worth re成员ing 基础d on whether the 信息 would be 有用 in a 未来 对话.

Auto 记忆 requ怒s Claude 代码 v2.1.59 or 更晚. 检查 your 版本 with `claude --版本`.

启用 or 禁用 auto 记忆

Auto 记忆 is on by 默认. To toggle it, 打开 `/记忆` in a 会话 and 使用 the auto 记忆 toggle, or 设置 `auto记忆启用` in your 项目 设置:

```json 主题={空}
{
"auto记忆启用": 虚假
}
```
```

```
To 禁用 auto 记忆 via 环境 可变, 设置 `CLAUDE_代码_禁用_AUTO_记忆=1`.

### Sto狂怒 location

每个 项目 gets its own 记忆 目录 at `~/.claude/项目s//记忆/`. The `` 路径 is 派

To store auto 记忆 in a 不同 location, 设置 `auto记忆目录` in your 用户 or 本地

```json  主题={空}
{
  "auto记忆目录": "~/my-习俗-记忆-dir"
}
```

This设置 is 接受 from 政策, 本地, and 用户 设置. It is not 接受 from 项目 设置 (`.claude/设置.json`) to pr事件 a 共享 项目 from 重定向ing auto 记忆 写入s to 敏感 locations.

The 目录 contains a `记忆.md` 条目point and 可选 话题 文件:

```文本 主题={空}
~/.claude/项目s//记忆/
├── 记忆.md # Concise 索引, 加载 into 每个 会话
├── 调试ging.md # 详情ed 注意s on 调试ging 模式s
├── api-惯例.md # API 设计 决定s
└── ... # 任何 other 话题 文件 Claude 创建s
```
```

```
`记忆.md` acts as an 索引 of the 记忆 目录. Claude 读取s and 写入s 文件 in thi

Auto 记忆 is machine-本地. 所有 工作树 and sub总监ies 在...内 the 相同 Git 仓库

### How it 工作s

The 第一个 200 行s of `记忆.md`, or the 第一个 25KB, whichever comes 第一个,

This 限制 应用lies only to `记忆.md`. CLAUDE.md 文件 are 加载 in 满 注意较少 o

话题 文件 像 `调试ging.md` or `模式s.md` are not 加载 at 启动上. Claude 读取s

Claude 读取s and 写入s 记忆 文件 期间 your 会话. When you看见 "Writing 记忆" o

### Audit and 编辑 your 记忆

Auto 记忆 文件 are 简单 mark下 you can 编辑 or 删除 at 任何 时间. 运行 [`/记忆`]

##视图 and 编辑 with `/记忆`

The `/记忆` 命令 列表s 所有 CLAUDE.md, CLAUDE.本地.md, and 规则 文件 加载 in yo

When you ask Claude to re成员 一些薄g, 像 "al方式s 使用 pnpm, not npm" or "r

## 麻烦shoot 记忆 问题s

These are the 最多 常见问题 with CLAUDE.md and auto 记忆, a长 with步骤s to 调

### Claude isn't跟随 my CLAUDE.md

CLAUDE.md 满意 is 交付 as a 用户 消息 之后 the 系统 及时, not as 部分 of the 系

To 调试:

* 运行 `/记忆` to 验证 your CLAUDE.md and CLAUDE.本地.md 文件 are being 加载.
```

```
* 检查 that the 相关 CLAUDE.md is in a location that gets 加载 for your 会话
* Make 说明 更多 特定. "使用 2-s步伐 indentation" 工作s 更好 than "格式 代码 nic
*寻找 conflicting 说明 across CLAUDE.md 文件. If two 文件 give 不同 指导 for

For 说明 you 想要 at the 系统 及时 级别, 使用 [`--应用结束-系统-及时`](/en/cli-参

  使用 the [`说明加载` hook](/en/钩子#说明加载) to 日志 精确ly which instructio

### I don't know what auto 记忆 保存

运行 `/记忆` and 选择 the auto 记忆 文件夹 to 浏览 what Claude has 保存. 每个薄

### My CLAUDE.md is too large

文件 结束 200 行s con总和e 更多 上下文 and may reduce adherence. 移动 详情ed 满

### 说明看见m 失败 之后 `/紧凑`

CLAUDE.md 满y survives 紧凑ion. 之后 `/紧凑`, Claude re-读取s your CLAUDE.md

看见 [写入 有效 说明](#写入-有效-说明) for 指导 on 尺寸, 结构, and 特定.

## R兴高采烈 资源

* [技能](/en/技能): 包 repea表 工作流s that 加载 on 需求
* [设置](/en/设置): con图 Claude 代码 behavior with 设置 文件
* [Manage 会话s](/en/会话s): manage 上下文, 恢复 对话s, and 运行 平行 会话s
* [Sub代理 记忆](/en/sub-代理s#启用-持续-记忆): let 子代理 主tain their own au

---

# 通用工作流

> ## 文档 索引
> 获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
> 使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.
```

```
# 通用工作流

>步骤-by-步骤 指南s for探索 代码基础s,固定 缺陷s, re事实oring, 测试, and other

This 页 c结束s 实践 工作流s for 日常 开发:探索 unfamiliar 代码, 调试ging, re事实

## Understand 新 代码基础s

### Get a 快 代码基础 概述

S上pose you've 公正 连接 a 新 项目 and 需要 to understand its 结构 快ly.


    ```bash  主题={空}
    cd /路径/to/项目
    ```



    ```bash  主题={空}
    claude
    ```



    ```文本  主题={空}
    give me an 概述 of this 代码基础
    ```



    ```文本  主题={空}
    ex简单 the 主 架构 模式s 使用d here
    ```

    ```文本  主题={空}
    what are the 键 数据 模型s?
```
```

```
    ```

    ```文本  主题={空}
    how is 认证 已处理?
    ```


  提示:

   * 启动 with broad 问题s, then nar行 下 to 特定 面积s
   * Ask about coding 惯例 and 模式s 使用d in the 项目
   * 请求 a 词汇表 of 项目-特定 条款

查找 相关 代码

S上pose you 需要 to locate 代码 r兴高采烈 to a 特定 feature or 函数式ity.


    ```文本  主题={空}
    查找 the 文件 that handle 用户 认证
    ```



    ```文本  主题={空}
    how do these 认证 文件 工作 together?
    ```



    ```文本  主题={空}
    跟踪 the 日志in 流程 from front-结束 to 数据库
    ```


  提示:
```
```

```
  * Be 特定 about what you're 看 for
  * 使用 do主 language from the 项目
  * 安装 a [代码 intelligence 插件](/en/disc结束-插件s#代码-intelligence) for

***

## 修复 缺陷s 高效ly

S上pose you've encountered an 错误 消息 and 需要 to 查找 and 修复 its 来源.


    ```文本  主题={空}
    I'm看见 an 错误 when I 运行 npm 测试
    ```



    ```文本  主题={空}
    suggest a 几个 方式s to 修复 the @ts-ignore in 用户.ts
    ```



    ```文本  主题={空}
   更新 用户.ts to 添加 the 空 检查 you 建议
    ```


  提示:

  * Tell Claude the 命令 to reproduce the 问题 and get a 栈 跟踪
  * Mention 任何步骤s to reproduce the 错误
  * Let Claude know if the 错误 is intermittent or 一致

***

Re事实or 代码
```

```
S上pose you 需要 to 更新 旧 代码 to 使用 现代 模式s and 实践.


    ```文本  主题={空}
   查找 deprecated API 使用方法 in our 代码基础
    ```



    ```文本  主题={空}
    suggest how to re事实or utils.js to 使用 现代 Java脚本 features
    ```



    ```文本  主题={空}
    re事实or utils.js to 使用 ES2024 features while维护 the 相同 behavior
    ```



    ```文本  主题={空}
   运行 测试s for the re事实ored 代码
    ```


  提示:

  * Ask Claude to ex简单 the 益处s of the 现代 方法
  * 请求 that 更改s 主tain 落后 兼容性 when 需要ed
  * Do re事实oring in 小, 可测试 increments

***

使用 专业 子代理
```

S上pose you 想要 to 使用 专业 AI 子代理 to handle 特定 任务s 更多 有效ly.

    ```文本  主题={空}
    /代理s
    ```

    This 显示s 所有 可用 子代理 and lets you 创建 新 ones.

    Claude 代码 自动所有y delegates 恰当 任务s to 专业 子代理:

    ```文本  主题={空}
    re视图 my 最近 代码 更改s for 安全 问题s
    ```

    ```文本  主题={空}
   运行 所有 测试s and 修复 任何 失败s
    ```

    ```文本  主题={空}
   使用 the 代码-审查员 sub代理 to 检查 the auth 模块
    ```

    ```文本  主题={空}
    have the 调试ger sub代理调查 why 用户s can't 日志 in
    ```

    ```文本  主题={空}
    /代理s
    ```

```
    Then 选择 "创建 新 sub代理" and fol低 the 及时s to de好:

    * A 唯一 标识符 that describes the sub代理's 目的 (for 示例, `代码-审查员`
    * When Claude should 使用 this 代理
    * Which 工具 it can access
    * A 系统 及时描述 the 代理's 角色 and behavior


  提示:

  * 创建 项目-特定 子代理 in `.claude/代理s/` for 团队分享
  * 使用 de脚本ive `描述` 字段s to 启用 自动 delegation
  * 限制 工具 access to what 每个 sub代理 实际ly 需要s
  * 检查 the [子代理 文档](/en/sub-代理s) for 详情ed 示例

***

## 使用 计划 模式 for 安全 代码 分析

计划 模式 instructs Claude to 创建 a 计划 by分析 the 代码基础 with 读取-only 运

### When to 使用 计划 模式

* **Multi-步骤 实施**: When your feature requ怒s制造 编辑s to 许多 文件
* **代码 探索**: When you 想要 to研究 the 代码基础 tho粗糙ly 之前改变 任何薄g
* **交互 开发**: When you 想要 to ite速率 on the 指导 with Claude

### How to 使用 计划 模式

**Turn on 计划 模式 期间 a 会话**

You can switch into 计划 模式 期间 a 会话 using **Shift+Tab** to cycle th粗糙

If you are in 正常 模式, **Shift+Tab** 第一个 switches into Auto-接受 模式, i

**启动 a 新 会话 in 计划 模式**
```

```
To 启动 a 新 会话 in 计划 模式, 使用 the `--许可-模式 计划` flag:

```bash  主题={空}
claude --许可-模式 计划
```

运行 "负责人较少" queries in 计划 模式

You can also 运行 a 查询 in 计划 模式 直接ly with `-p` (that is, in "负责人较少 模式"):

```bash 主题={空}
claude --许可-模式 计划 -p "分析 the 认证 系统 and suggest 改善s"
```

```
### 示例: 规划 a 复杂 re事实or

```bash  主题={空}
claude --许可-模式 计划
```

```文本 主题={空}
I 需要 to re事实or our 认证 系统 to 使用 OAuth2. 创建 a 详情ed mig比率n 计划.
```

```
Claude分析s the 当前 实施 and 创建 a comprehensive 计划. Re好 with fol低-上s:

```文本  主题={空}
What about 落后 兼容性?
```

```文本 主题={空}
How should we handle 数据库 mig比率n?
```
```

````
Press `Ctrl+G` to 打开 the 计划 in your 默认 文本 编辑or, where you can 编辑

When you 接受 a 计划, Claude 自动所有y 名称s the 会话 from the 计划 满意. The

### Con图 计划 模式 as 默认

```json  主题={空}
// .claude/设置.json
{
  "权限": {
    "默认模式": "计划"
  }
}
````

看见 设置 文档 for 更多 配置 选项.

工作 with 测试s

S上pose you 需要 to 添加 测试s for 揭开 代码.
````

```
```文本  主题={空}
查找 功能s in 通知s服务.快速 that are not c结束ed by 测试s
```



```文本  主题={空}
添加 测试s for the 通知 服务
```



```文本  主题={空}
添加 测试 案例s for 边 条件 in the 通知 服务
```



```文本  主题={空}
运行 the 新 测试s and 修复 任何 失败s
```
```

Claude can gene速率 测试s that fol低 your 项目's 现有 模式s and 惯例. When询问 for 测试s, be 特定 about what behavior you 想要 to 验证. Claude检查s your 现有 测试 文件 to 匹配 the style, 框架s, and 断言ion 模式s al就绪 in 使用.

For comprehensive cove狂怒, ask Claude to identify 边 案例s you might have missed. Claude can分析 your 代码 路径s and suggest 测试s for 错误 条件, boundary 值s, and 意外 输入s that are 容易 to 结束看.

创建 拉取请求s

You can 创建 拉取请求s by询问 Claude 直接ly ("创建 a pr for my 更改s"), or 指南 Claude th粗糙 it步骤-by-步骤:

```
```文本  主题={空}
总和marize the 更改s I've made to the 认证 模块
```



```文本  主题={空}
创建 a pr
```



```文本  主题={空}
enhance the PR 描述 with 更多 上下文 about the 安全 改善s
```
```

When you 创建 a PR using `gh pr 创建`, the 会话 is 自动所有y 链接 to that PR. You can 恢复 it 更晚 with `claude --from-pr`.

Re视图 Claude's gene速率d PR 之前提交 and ask Claude to 高轻 潜力 风险s or conside比率ns.

Handle 文档

S上pose you 需要 to 添加 or 更新 文档 for your 代码.

```文本  主题={空}
查找 功能s without 适当 JSDoc comments in the auth 模块
```

```文本  主题={空}
添加 JSDoc comments to the 撤销cumented 功能s in auth.js
```

```文本  主题={空}
improve the gene速率d 文档 with 更多 上下文 and 示例
```

```文本  主题={空}
检查 if the 文档 fol低s our 项目 标准
```

提示:

- 规格ify the 文档 style you 想要 (JSDoc, doc字符串s, etc.)
- Ask for 示例 in the 文档
- 请求 文档 for 公共 APIs, 接口s, and 复杂 日志ic

工作 with 图像s

S上pose you 需要 to 工作 with 图像s in your 代码基础, and you 想要 Claude's 帮助分析 图像 满意.

```
 You can 使用 任何 of these 方法:

1. Drag and 下降 an 图像 into the Claude 代码 风ow
2. 复制 an 图像 and 粘贴 it into the CLI with ctrl+v (Do not 使用 cmd+v)
3. Provide an 图像 路径 to Claude. E.g., "分析 this 图像: /路径/to/your/图像.
```

```
```文本  主题={空}
What does this 图像 显示?
```
```

```
```文本  主题={空}
Describe the UI 元素s in this 截图
```
```

```
```文本  主题={空}
Are there 任何 问题atic 元素s in this 图表?
```
```

```
```文本  主题={空}
Here's a 截图 of the 错误. What's causing it?
```
```

```
```文本  主题={空}
This is our 当前 数据库 模式. How should we 修改 it for the 新 feature?
```
```

```
```文本  主题={空}
Gene速率 CSS to 匹配 this 设计 模型
```
```

```文本  主题={空}
What HTML 结构 would re创建 this 组件?
```

提示:

- 使用 图像s when 文本 描述s would be 不清楚 or 笨重
- Include 截图s of 错误, UI 设计s, or 图表s for 更好 上下文
- You can 工作 with mul提示le 图像s in a 对话
- 图像 分析 工作s with 图表s, 截图s, 模型s, and 更多
- When Claude 参考文献 图像s (for 示例, `[图像 #1]`), `Cmd+Click` (Mac) or `Ctrl+Click` (风ows/Linux) the 链接 to 打开 the 图像 in your 默认视图er

---

参考 文件 and 总监ies

使用 @ to 快ly include 文件 or 总监ies without 等待ing for Claude to 读取 them.

```
```文本  主题={空}
Ex简单 the 日志ic in @src/utils/auth.js
```

This includes the 满 满意 of the 文件 in the 对话.


```文本  主题={空}
What's the 结构 of @src/组件s?
```

This provides a 目录列表 with 文件 信息.


```文本  主题={空}
显示 me the 数据 from @GitHub:repos/所有者/repo/问题s
```

This 获取es 数据 from 连接 MCP 服务器 using the 格式 @服务器:资源.看见 [MCP 资源
```

提示:

- 文件 路径s can be relative or absolute
- @ 文件 参考文献 添加 `CLAUDE.md` in the 文件's 目录 and parent 总监ies to 上下文
- 目录 参考文献 显示 文件列表s, not 满意s
- You can 参考 mul提示le 文件 in a single 消息 (for 示例, "@文件1.js and @文件2.js")

使用 延长思考 (薄king 模式)

延长思考 is 启用 by 默认,给 Claude s步伐 to 原因 th粗糙 复杂 问题s步骤-by-步骤 之前 responding. This推理 is 可见 in 详细 模式, which you can toggle on with `Ctrl+O`.

添加ition所有y, Opus 4.6 and Sonnet 4.6 支持 适应推理: instead of a 固定思考 令牌 预算, the 模型 动态所有y 所有ocates思考 基础d on your effort 级别设置. 延长思考 and 适应推理 工作 together to give you 控制 结束 how 深ly Claude 原因s 之前 responding.

延长思考 is 特定ly 有价值 for 复杂 架构师ural 决定s, 挑战 缺陷s, multi-步骤 实施 规划, and评估 贸易offs between 不同 方法es.

pH值rases 像 "薄k", "薄k 硬", and "薄k 更多" are 解释ed as 常规 及时 说明 and don't 所有ocate思考 令牌s.

Con图思考 模式

薄king is 启用 by 默认, but you can ad公正 or 禁用 it.

范围	How to con图	详情s
Effort 级别	运行 `/effort`, ad公正 in `/模型`, or 设置 `CLAUDE_代码_EFFORT_级别`	控制思考 深度 for Opus 4.6 and Sonnet 4.6. 看见 Ad公正 effort 级别
`ultra薄k` 键词	Include "ultra薄k" 任何 where in your 及时	设置s effort to 高 for that turn on Opus 4.6 and Sonnet 4.6. 有用 for one-off 任务s要求 深推理 without 永久ly改变 your effort设置
Toggle 短剪切	Press `选项+T` (macOS) or `Alt+T` (风ows/ Linux)	Toggle思考 on/off for the 当前 会话 (所有 模型s). May requ怒 终端 配置 to 启用 选项 键 短剪切s
全球 默认	使用 `/config` to toggle思考 模式	设置s your 默认 across 所有 项目s (所有 模型s).保存 as `al方式s薄king启用` in `~/.claude/设置.json`
限制 令牌 预算	设置 `MAX_薄KING_令牌S` 环境 可变	限制 the思考 预算 to a 特定 数字 of 令牌s. On Opus 4.6 and Sonnet 4.6, only `0` 应用lies un 较少 适应推理 is 禁用. 示例: `出口 MAX_薄KING_令牌S=10000`

To视图 Claude's思考 流程, press `Ctrl+O` to toggle 详细 模式 and看见 the 内部推理 显示ed as gray italic 文本.

How 延长思考 工作s

延长思考 控制s how 很多 内部推理 Claude per形式s 之前 responding. 更多思考 provides 更多 s步伐 to探索 解决方案,分析 边 案例s, and self-正确 薄雾akes.

With Opus 4.6 and Sonnet 4.6,思考 使用s 适应推理: the 模型 动态所有y 所有ocates思考 令牌s 基础d on the effort 级别 you 选择. This is the 推荐 方式 to tune the 贸易off between 速度 and推理 深度.

With 更旧 模型s,思考 使用s a 固定 令牌 预算 drawn from your 输出 所有ocation. The 预算 varies by 模型;看见 `MAX_薄KING_令牌S` for per-模型 ceilings. You can 限制 the 预算 with that 环境 可变, or 禁用思考 整个ly via `/config` or the `选项+T` / `Alt+T` toggle.

On Opus 4.6 and Sonnet 4.6, 适应推理 控制s思考 深度, so `MAX_薄KING_令牌S` only 应用lies when 设置 to `0` to 禁用思考, or when `CLAUDE_代码_禁用_适应_薄KING=1` 撤销s these 模型s to the 固定 预算.看见 环境变量.

You're 收费 for 所有思考 令牌s 使用d 偶数 when思考 总和maries are redacted. In 交互 模式,思考 应用ears as a collapsed 桩 by 默认. 设置 `显示薄king总和maries: 真实` in `设置.json` to 显示 满 总和maries.

恢复 之前 对话s

When 启动ing Claude 代码, you can 恢复 a 之前 会话:

- `claude --继续` 继续s the 最多 最近 对话 in the 当前 目录
- `claude --恢复` 打开s a 对话 picker or 恢复s by 名称
- `claude --from-pr 123` 恢复s 会话s 链接 to a 特定 拉取请求

From 内部 an 活跃 会话, 使用 `/恢复` to switch to a 不同 对话.

会话s are 存储 per 项目 目录. The `/恢复` picker 显示s 交互 会话s from the 相同 Git 仓库,包括 工作树. 会话s 创建d by `claude -p` or SDK invocations do not 应用ear in the picker, but you can 静止 恢复 one by通过 its 会话 ID 直接ly to `claude --恢复`.

名称 your 会话s

Give 会话s de脚本ive 名称s to 查找 them 更晚. This is a 最佳 实践 when工作 on mul提示le 任务s or features.

```
 名称 a 会话 at 启动上 with `-n`:

```bash  主题={空}
claude -n auth-re事实or
```

Or 使用 `/重命名` 期间 a 会话, which also 显示s the 名称 on the 及时 bar:

```文本  主题={空}
/重命名 auth-re事实or
```

You can also 重命名 任何 会话 from the picker: 运行 `/恢复`, 导航 to a 会话, a

From the 命令 行:

```bash  主题={空}
claude --恢复 auth-re事实or
```

Or from 内部 an 活跃 会话:

```文本  主题={空}
/恢复 auth-re事实or
```
```

使用 the 会话 picker

The `/恢复` 命令 (or `claude --恢复` without 参数) 打开s an 交互 会话 picker with these features:

键董事会 短剪切s in the picker:

短剪切	行动
`↑` / `↓`	导航 between 会话s
`→` / `←`	Expand or collapse 分组 会话s
`进入`	选择 and 恢复 the 高轻ed 会话
`P`	Pre视图 the 会话 满意
`R`	重命名 the 高轻ed 会话
`/`	搜索 to 过滤 会话s
`A`	Toggle between 当前 目录 and 所有 项目s
`B`	过滤 to 会话s from your 当前 Git 分支
`Esc`	退出 the picker or 搜索 模式

会话 组织:

The picker 显示s 会话s with 有帮助 meta数据:

- 会话 名称 or initial 及时
- 时间 elapsed since 最后一个 活动
- 消息 count
- Git 分支 (if 应用lic能够)

分叉ed 会话s (创建d with `/分支`, `/re风`, or `--分叉-会话`) are 分组 together under their 根 会话,制造 it easier to 查找 r兴高采烈 对话s.

提示:

- **名称 会话s 早**: 使用 `/重命名` when 启动ing 工作 on a 不同 任务: it's 很多 easier to 查找 "payment-集成" than "ex简单 this 功能" 更晚
- 使用 `--继续` for 快 access to your 最多 最近 对话 in the 当前 目录
- 使用 `--恢复 会话-名称` when you know which 会话 you 需要
- 使用 `--恢复` (without a 名称) when you 需要 to 浏览 and 选择
- For 脚本s, 使用 `claude --继续 --print "及时"` to 恢复 in non-交互 模式

- Press `P` in the picker to pre视图 a 会话 之前 re总和ing it
- The 恢复d 对话 启动s with the 相同 模型 and 配置 as the 原始

How it 工作s:

1. **对话 Sto狂怒**: 所有 对话s are 自动所有y 保存 本地ly with their 满 消息 history
2. **消息 De串行ization**: When re总和ing, the 整个 消息 history is 恢复 to 主tain 上下文
3. **工具 州**: 工具 使用方法 and 结果s from the 之前 对话 are 保存
4. **上下文 Resto比率n**: The 对话 恢复s with 所有 之前 上下文 完整

运行 平行 Claude 代码 会话s with Git 工作树

When工作 on mul提示le 任务s at once, you 需要 每个 Claude 会话 to have its own 复制 of the 代码基础 so 更改s don't collide. Git 工作树 solve this by创建 单独工作 总监ies that 每个 have their own 文件 and 分支, while分享 the 相同 仓库 history and 远程 连接s. This 手段 you can have Claude工作 on a feature in one 工作树 while固定 a 缺陷 in another, without either 会话 干涉 with the other.

使用 the `--工作树` (`-w`) flag to 创建 an iso晚d 工作树 and 启动 Claude in it. The 值 you pass becomes the 工作树 目录 名称 and 分支 名称:

```bash 主题={空}
# 启动 Claude in a 工作树 名称d "feature-auth"

# 创建s .claude/工作树/feature-auth/ with a 新 分支

claude --工作树 feature-auth
```

启动 another 会话 in a 单独 工作树

claude --工作树 缺陷修复-123

```
If you omit the 名称, Claude gene速率s a 随机 one 自动所有y:

```bash  主题={空}
# Auto-gene速率s a 名称 像 "亮-运行中-fox"
claude --工作树
```

工作树 are 创建d at `/.claude/工作树/` and 分支 from the 默认 远程 分支, which is where `起源/负责人` points. The 工作树 分支 is 名称d `工作树-`.

The 基础 分支 is not configur能够 th粗糙 a Claude 代码 flag or设置. `起源/负责人` is a 参考 存储 in your 本地 `.Git` 目录 that Git 设置 once when you 克隆. If the 仓库's 默认 分支 更晚 更改s on GitHub or GitLab, your 本地 `起源/负责人` keeps pointing at the 旧 one, and 工作树 will 分支 from there. To re-同步 your 本地 参考 with whatever the 远程 当前ly considers its 默认:

```bash 主题={空}
Git 远程 设置-负责人 起源 -a
```
```

```
This is a 标准 Git 命令 that only 更新s your 本地 `.Git` 目录. No薄g on the

For 满 控制 结束 how 工作树 are 创建d,包括选择 a 不同 基础 per invocation, con图

You can also ask Claude to "工作 in a 工作树" or "启动 a 工作树" 期间 a 会话,

### Sub代理 工作树

子代理 can also 使用 工作树 isolation to 工作 in 平行 without conflicts. Ask

### 工作树 干净上

When you 退出 a 工作树 会话, Claude handles 干净上 基础d on whether you made

* **No 更改s**: the 工作树 and its 分支 are 移除 自动所有y
* **更改s or 提交s exist**: Claude 及时s you to keep or 移除 the 工作树.保持

Sub代理 工作树 orpH值aned by a crash or an 中断ed 平行 运行 are 移除 自动所有y

To 干净 上 工作树 外部 of a Claude 会话, 使用 [手册 工作树 管理](#manage-工作树-手

  添加 `.claude/工作树/` to your `.Gitignore` to pr事件 工作树 满意s from 应用

### 复制 Git忽略 文件 to 工作树

Git 工作树 are 新鲜 检查outs, so they don't include un跟踪ed 文件 像 `.env` o

The 文件 使用s `.Gitignore` syn税 to 列表 which 文件 to 复制. Only 文件 that

```文本 .工作树include 主题={空}
.env
.env.本地
config/秘密s.json
```
```

This 应用lies to 工作树 创建d with `--工作树`, sub代理 工作树, and 平行 会话s in the 桌面 应用.

Manage 工作树 手册ly

For 更多 控制 结束 工作树 location and 分支 配置, 创建 工作树 with Git 直接ly. This is 有用 when you 需要 to 检查 out a 特定 现有 分支 or place the 工作树 外部 the 仓库.

```bash 主题={空}
# 创建 a 工作树 with a 新 分支
Git 工作树 添加 ../项目-feature-a -b feature-a

# 创建 a 工作树 with an 现有 分支
Git 工作树 添加 ../项目-缺陷修复 缺陷修复-123

# 启动 Claude in the 工作树
cd ../项目-feature-a && claude

# 干净 上 when 完成
Git 工作树 列表
Git 工作树 移除 ../项目-feature-a
```

````
Learn 更多 in the [official Git 工作树 文档](https://Git-scm.com/docs/Git-工

  Re成员 to initialize your 开发 环境 in 每个 新 工作树 一致ing to your 项目's

### Non-Git 版本 控制

工作树 isolation 工作s with Git by 默认. For other 版本 控制 系统s 像 SVN, Per

For automated 协调 of 平行 会话s with 共享 任务s and 混乱aging,看见 [代理 团队]

***

## Get notified when Claude 需要s your 注意

When you kick off a 长-运行中 任务 and switch to another 风ow, you can 设置

   打开 `~/.claude/设置.json` and 添加 a `通知` hook that c所有s your 平台's


      ```json  主题={空}
      {
        "钩子": {
          "通知": [
            {
              "匹配er": "",
              "钩子": [
                {
                  "类型": "命令",
                  "命令": "osa脚本 -e '显示 通知 \"Claude 代码 需要s your 注
                }
              ]
            }
          ]
````
````

```
        }
      }
      ```



      ```json  主题={空}
      {
        "钩子": {
          "通知": [
            {
              "匹配er": "",
              "钩子": [
                {
                  "类型": "命令",
                  "命令": "notify-发送 'Claude 代码' 'Claude 代码 需要s yo
                }
              ]
            }
          ]
        }
      }
      ```



      ```json  主题={空}
      {
        "钩子": {
          "通知": [
            {
              "匹配er": "",
              "钩子": [
                {
                  "类型": "命令",
                  "命令": "权力命令行.exe -命令 \"[系统.Reflection.装配]::加
                }
```

```
                ]
              }
            ]
          }
        }
        ```


    If your 设置 文件 al就绪 has a `钩子` 键, 合并 the `通知` 条目 into it rat

    By 默认 the hook f怒s on 所有 通知 类型s. To f怒 only for 特定 事件s, 设置

匹配er	F怒s when
`许可_及时`	Claude 需要s you to 批准 a 工具 使用
`空闲_及时`	Claude is 完成 and 等待ing for your 下一个 及时
`auth_成功`	认证 完成s
`e合法ation_dia日志`	Claude is询问 you a 问题

   类型 `/钩子` and 选择 `通知` to 确认 the hook 应用ears.选择 it 显示s the 命

For the 完成 事件 模式 and 通知 类型s,看见 the [通知 参考](/en/钩子#通知).

***

使用 Claude as a unix-style 实用

添加 Claude to your 验证 流程

S上pose you 想要 to 使用 Claude 代码 as a linter or 代码 审查员.

```
**添加 Claude to your 构建 脚本:**

```json  主题={空}
// 包.json
{
    ...
    "脚本s": {
        ...
        "lint:claude": "claude -p 'you are a linter. pl容易看 at the 更改s
    }
}
```

提示:

- 使用 Claude for automated 代码 re视图 in your CI/CD 管道
- 习俗ize the 及时 to 检查 for 特定 问题s 相关 to your 项目
- Consider创建 mul提示le 脚本s for 不同 类型s of 验证

管道 in, 管道 out

S上pose you 想要 to 管道 数据 into Claude, and get 返回 数据 in a 结构化 格式.

管道 数据 th粗糙 Claude:

```bash 主题={空}
cat 构建-错误.txt | claude -p 'concisely ex简单 the 根 原因 of this 构建 错误' > 输出.txt
```
```

提示:

* 使用 管道s to integ速率 Claude into 现有 命令行 脚本s
* Combine with other Unix 工具 for 强大 工作流s
* Consider using `--输出-格式` for 结构化 输出

控制 输出 格式

S上pose you 需要 Claude's 输出 in a 特定 格式, e特殊ly when集成 Claude 代码 in

```bash 主题={空}
cat 数据.txt | claude -p '总和marize this 数据' --输出-格式 文本 > 摘要.t
```

This 输出s 公正 Claude's 简单 文本 响应 (默认 behavior).

```bash 主题={空}
cat 代码.py | claude -p '分析 this 代码 for 缺陷s' --输出-格式 json > 分析
```

This 输出s a JSON 数组 of 消息s with meta数据包括 成本 and 持续时间.

```bash 主题={空}
cat 日志.txt | claude -p 'parse this 日志 文件 for 错误' --输出-格式 流-j
```

This 输出s a series of JSON 对象s in 真实-时间 as Claude 流程es the 请求.

提示:

```
  * 使用 `--输出-格式 文本` for 简单 集成s where you 公正 需要 Claude's 响应
  * 使用 `--输出-格式 json` when you 需要 the 满 对话 日志
  * 使用 `--输出-格式 流-json` for 真实-时间 输出 of 每个 对话 turn

***

## 运行 Claude on a 时间表

S上pose you 想要 Claude to handle a 任务 自动所有y on a recurring 基础, 像回顾

Pick a scheduling 选项 基础d on where you 想要 the 任务 to 运行:

| 选项                                                                | Where i
| :------------------------------------------------------------------ | :---
| [云 计划 任务s](/en/网页-计划-任务s)                                | Anthropic-管理 基础设施
| [桌面 计划 任务s](/en/桌面#时间表-recurring-任务s) | Your machine, via the
| [GitHub 行动s](/en/GitHub-行动s)                                    | Your CI 管
| [`/loop`](/en/计划-任务s)                                           | The 当前 CLI

  When写作 及时s for 计划 任务s, be 明确 about what 成功看s 像 and what to do

***

## Ask Claude about its 能力

Claude has built-in access to its 文档 and can 答案 问题s about its own fea

### 示例 问题s

```文本  主题={空}
can Claude 代码 创建 拉取请求s?
```

```文本 主题={空}
how does Claude 代码 handle 权限?
```
```

```
```文本  主题={空}
what 技能 are 可用?
```

```文本 主题={空}
how do I 使用 MCP with Claude 代码?
```

```
```文本  主题={空}
how do I con图 Claude 代码 for Amazon 基岩?
```

```文本 主题={空}
what are the 限制 of Claude 代码?
```

```
  Claude provides 文档-基础d 答案s to these 问题s. For hands-on 演示s, 运行

  提示:

  * Claude al方式s has access to the 最新 Claude 代码 文档, 注意较少 of the 版
  * Ask 特定 问题s to get 详情ed 答案s
  * Claude can ex简单 复杂 features 像 MCP 集成, 企业 配置s, and 先进 工作流s

***

## 下一步


   模式s for getting the 最多 out of Claude 代码



    Understand the 代理ic loop and 上下文 管理



   添加 技能, 钩子, MCP, 子代理, and 插件s



   克隆 the 开发 container 参考 实施

---

# 最佳实践

> ## 文档 索引
> 获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
> 使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.
```

```
# Claude 代码 最佳实践

> 提示 and 模式s for getting the 最多 out of Claude 代码, from configuring y

Claude 代码 is an 代理ic coding 环境. 不像 a chat机器人 that 答案s 问题s and 等

This 更改s how you 工作. Instead of写作 代码 yourself and询问 Claude to re视图

But this autonomy 静止 comes with a learning curve. Claude 工作s 在...内 确

This 指南 c结束s 模式s that have 证实 有效 across Anthropic's 内部 团队 and fo

***

最多 最佳实践 are 基础d on one 常量雨t: Claude's 上下文 风ow fills 上 快, and 性

Claude's 上下文 风ow h旧s your 整个 对话,包括 每个 消息, 每个 文件 Claude 读取s,

This 事情s since LLM 性能 de年级s as 上下文 fills. When the 上下文 风ow is ge

***

## Give Claude a 方式 to 验证 its 工作

  Include 测试s, 截图s, or 预期 输出s so Claude can 检查 itself. This is the

Claude per形式s dramatic所有y 更好 when it can 验证 its own 工作, 像 运行 测试

Without 清楚 成功 标准, it might produce 一些薄g that看s 正确 but 实际ly doesn

| 战略                                  | 之前
| ------------------------------------ | ------------------------------
| **Provide 验证 标准**     | *"实现 a 功能 that 验证s email 添加resses"* | *
| **验证 UI 更改s visu所有y**         | *"make the dash董事会看 更好"*
| **添加ress 根 原因s, not 症状s** | *"the 构建 is failing"*

UI 更改s can be 验证 using the [Claude in Chrome 扩展](/en/chrome). It 打开
```

Your 验证 can also be a 测试 suite, a linter, or a Bash 命令 that 检查s 输出

***

##探索 第一个, then 计划, then 代码

  单独研究 and 规划 from 实施 to a空白 solving the 错误 问题.

Letting Claude跳 直 to coding can produce 代码 that solves the 错误 问题. 使

The 推荐 工作流 has four 阶段s:

   进入 计划 模式. Claude 读取s 文件 and 答案s 问题s without制造 更改s.

    ```txt claude (计划 模式) 主题={空}
   读取 /src/auth and understand how we handle 会话s and 日志in.
    also看 at how we manage 环境变量 for 秘密s.
    ```

    Ask Claude to 创建 a 详情ed 实施 计划.

    ```txt claude (计划 模式) 主题={空}
    I 想要 to 添加 前往ogle OAuth. What 文件 需要 to 更改?
    What's the 会话 f低? 创建 a 计划.
    ```

    Press `Ctrl+G` to 打开 the 计划 in your 文本 编辑or for 直接 编辑ing 之前

    Switch 返回 to 正常 模式 and let Claude 代码,验证 a收益st its 计划.

    ```txt claude (正常 模式) 主题={空}
    ```

```
   实现 the OAuth f低 from your 计划. 写入 测试s for the
   回调 处理器, 运行 the 测试 suite and 修复 任何 失败s.
    ```


    Ask Claude to 提交 with a de脚本ive 消息 and 创建 a PR.

    ```txt claude (正常 模式) 主题={空}
   提交 with a de脚本ive 消息 and 打开 a PR
    ```


  计划 模式 is 有用, but also 添加s 结束负责人.

  For 任务s where the 范围 is 清楚 and the 修复 is 小 (像固定 a typo,添加 a 日

  规划 is 最多 有用 when you're 不确定 about the 方法, when the 更改 modifies

***

Provide 特定 上下文 in your 及时s

  The 更多 精确 your 说明, the 几个er 正确ions you'll 需要.

Claude can infer 意图, but it can't 读取 your mind. 参考 特定 文件, mention

| 战略
| --
| **范围 the 任务.** 规格ify which 文件, what scenario, and 测试 偏好设置.
| **Point to 来源s.** 直接 Claude to the 来源 that can 答案 a 问题.
| **参考 现有 模式s.** Point Claude to 模式s in your 代码基础.
| **Describe the 症状.** Provide the 症状, the 可能 location, and what "固定

模糊 及时s can be 有用 when you're探索 and can afford to course-正确. A 及时

Provide rich 满意
```
```

```
  使用 `@` to 参考 文件, 粘贴 截图s/图像s, or 管道 数据 直接ly.

You can provide rich 数据 to Claude in 几个 方式s:

* **参考 文件 with `@`** instead of描述 where 代码 lives. Claude 读取s the 文
* **粘贴 图像s 直接ly**. 复制/粘贴 or drag and 下降 图像s into the 及时.
* **Give URLs** for 文档 and API 参考s. 使用 `/权限` to 允许列表 频繁ly-使用d
* **管道 in 数据** by 运行中 `cat 错误.日志 | claude` to 发送 文件 满意s 直接ly
* **Let Claude 获取 what it 需要s**. Tell Claude to 拉取 上下文 itself using

***

## Con图 your 环境

A 几个 设置步骤s make Claude 代码 重要ly 更多 有效 across 所有 your 会话s. For

### 写入 an 有效 CLAUDE.md

  运行 `/init` to gene速率 a 启动er CLAUDE.md 文件 基础d on your 当前 项目 结构

CLAUDE.md is a 特殊 文件 that Claude 读取s at the 启动 of 每个 对话. Include

The `/init` 命令分析s your 代码基础 to detect 构建 系统s, 测试 框架s, and 代码

There's no 必需 格式 for CLAUDE.md 文件, but keep it 短 and 人类-读取能够. Fo

```mark下 CLAUDE.md 主题={空}
# 代码 style
- 使用 ES 模块s (进口/出口) syn税, not 常见JS (requ怒)
- De结构 进口s when 可能 (eg. 进口 { foo } from 'bar')

# 工作流
- Be 确定 to 类型检查 when you're 完成制造 a series of 代码 更改s
- Prefer 运行中 single 测试s, and not the 整体 测试 suite, for 性能
```
```

CLAUDE.md is 加载 每个 会话, so only include 薄gs that 应用ly broadly. For do主 知识 or 工作流s that are only 相关 一些时间s, 使用 技能 instead. Claude 加载s them on 需求 without膨胀 每个 对话.

Keep it concise. For 每个 行, ask: "Would removing this 原因 Claude to make 薄雾 akes?" If not, 剪切 it. Bloated CLAUDE.md 文件 原因 Claude to ignore your 实际 说明!

☒nclude	× Exclude
Bash 命令 Claude can't guess	任何薄g Claude can 图 out by 读取ing 代码
代码 style 规则 that differ from 默认s	标准 language 惯例 Claude al就绪 knows
测试 说明 and preferred 测试 运行ners	详情ed API 文档 (链接 to docs instead)
仓库 礼仪 (分支 naming, PR 惯例)	信息 that 更改s 频繁ly
架构师ural 决定s 特定 to your 项目	长 解释s or 教程s
开发者 环境 quirks (必需 env vars)	文件-by-文件 描述s of the 代码基础
常见 前往tchas or non-明显 behaviors	Self-明显 实践 像 "写入 干净 代码"

If Claude keeps做 一些薄g you don't 想要 despite having a 规则 a收益st it, the 文件 is probably too 长 and the 规则 is getting 失败. If Claude asks you 问题s that are 答案ed in CLAUDE.md, the pH值rasing might be 模糊. Treat CLAUDE.md 像 代码: re视图 it when 薄gs 前往 错误, p运行e it 常规ly, and 测试 更改s by观察 whether Claude's behavior 实际ly shifts.

You can tune 说明 by添加 empH值asis (e.g., "重要" or "YOU MUST") to improve adherence. 检查 CLAUDE.md into Git so your 团队 can contribute. The 文件 compounds in 值 结束 时间.

CLAUDE.md 文件 can 进口 添加itional 文件 using `@路径/to/进口` syn税:

```mark下 CLAUDE.md 主题={空}
看见 @读取ME.md for 项目 概述 and @包.json for 可用 npm 命令.
```

添加itional 说明

- Git 工作流: @docs/Git-说明.md
- 个人 结束rides: @~/.claude/my-项目-说明.md

```
You can place CLAUDE.md 文件 in 几个 locations:

* **Home 文件夹 (`~/.claude/CLAUDE.md`)**: 应用lies to 所有 Claude 会话s
* **项目 根 (`./CLAUDE.md`)**: 检查 into Git to share with your 团队
* **项目 根 (`./CLAUDE.本地.md`)**: 个人 项目-特定 注意s; 添加 this 文件 to you
* **Parent 总监ies**: 有用 for monorepos where 机器人h `根/CLAUDE.md` and `
* **Child 总监ies**: Claude 拉取s in child CLAUDE.md 文件 on 需求 when工作 w

### Con图 权限

  使用 [auto 模式](/en/许可-模式s#eliminate-及时s-with-auto-模式) to let a 阶

By 默认, Claude 代码 请求s 许可 for 行动s that might 修改 your 系统: 文件 写入s

* **Auto 模式**: a 单独 阶级ifier 模型 re视图s 命令 and 块s only what看s 风险y
* **许可 允许列表s**: permit 特定 工具 you know are 安全, 像 `npm 运行 lint` o
* **沙boxing**: 启用 OS-级别 isolation that re严格s 文件系统 and net工作 acces

读取 更多 about [许可 模式s](/en/许可-模式s), [许可 规则](/en/权限), and [沙box

### 使用 CLI 工具

  Tell Claude 代码 to 使用 CLI 工具 像 `gh`, `aws`, `g云`, and `s条目-cli` w

CLI 工具 are the 最多 上下文-高效 方式 to interact with 外部 服务s. If you 使用

Claude is also 有效 at learning CLI 工具 it doesn't al就绪 know. 尝试 及时s

### Connect MCP 服务器

  运行 `claude mcp 添加` to connect 外部 工具 像 概念, Figma, or your 数据库.

With [MCP 服务器](/en/mcp), you can ask Claude to 实现 features from 问题 跟

### 设置 上 钩子
```

```
  使用 钩子 for 行动s that must h应用en 每个 时间 with zero 异常.

[钩子](/en/钩子-指南) 运行 脚本s 自动所有y at 特定 points in Claude's 工作流. 不

Claude can 写入 钩子 for you. 尝试 及时s 像 *"写入 a hook that 运行s eslint 之

### 创建技能

  创建 `技能.md` 文件 in `.claude/技能/` to give Claude do主 知识 and reus能够

[技能](/en/技能) ext结束 Claude's 知识 with 信息 特定 to your 项目, 团队, or do

创建 a 技能 by添加 a 目录 with a `技能.md` to `.claude/技能/`:

```mark下 .claude/技能/api-惯例/技能.md 主题={空}
---
名称: api-惯例
描述: REST API 设计 惯例 for our 服务s
---
# API 惯例
- 使用 kebab-案例 for URL 路径s
- 使用 camel案例 for JSON 适当ties
- Al方式s include pagination for 列表 结束points
- 版本 APIs in the URL 路径 (/v1/, /v2/)
```

技能 can also de好 repea表 工作流s you invoke 直接ly:

```mark下 .claude/技能/修复-问题/技能.md 主题={空}

名称: 修复-问题
描述: 修复 a GitHub 问题
禁用-模型-invocation: 真实

分析 and 修复 the GitHub 问题: $参数.

1. 使用 `gh 问题视图` to get the 问题 详情s
```

2. Understand the 问题 described in the 问题
3. 搜索 the 代码基础 for 相关 文件
4. 实现 the 必要 更改s to 修复 the 问题
5. 写入 and 运行 测试s to 验证 the 修复
6. En确定 代码 passes linting and 类型检查
7. 创建 a de脚本ive 提交 消息
8. 推送 and 创建 a PR

```
运行 `/修复-问题 1234` to invoke it. 使用 `禁用-模型-invocation: 真实` for 工

### 创建 习俗 子代理

  De好 专业 assistants in `.claude/代理s/` that Claude can delegate to for

[子代理](/en/sub-代理s) 运行 in their own 上下文 with their own 设置 of 允许 工

```mark下 .claude/代理s/安全-审查员.md 主题={空}
---
名称: 安全-审查员
描述: Re视图s 代码 for 安全 vulner能力
工具: 读取, Grep, Glob, Bash
模型: opus
---
You are a 高级 安全 工程师. Re视图 代码 for:
- Injection vulner能力 (SQL, XSS, 命令 injection)
- 认证 and 授权 flaws
- 秘密s or credentials in 代码
- In安全 数据处理

Provide 特定 行 参考文献 and 建议 修复es.
```

Tell Claude to 使用 子代理 明确ly: "使用 a sub代理 to re视图 this 代码 for 安全 问题s."

安装 插件s

运行 `/插件` to 浏览 the 市场place. 插件s 添加 技能, 工具, and 集成s without 配置.
```

插件s 捆绑 技能, 钩子, 子代理, and MCP 服务器 into a single 安装能够 单位 from the 社区 and Anthropic. If you 工作 with a 类型化 language, 安装 a 代码 intelligence 插件 to give Claude 精确 符号 navigation and 自动 错误 检测 之后 编辑s.

For 指导 on选择 between 技能, 子代理, 钩子, and MCP,看见 Ext结束 Claude 代码.

Communicate 有效ly

The 方式 you communicate with Claude 代码 重要ly 影响s the quality of 结果s.

Ask 代码基础 问题s

Ask Claude 问题s you'd ask a 高级 工程师.

When on董事会ing to a 新 代码基础, 使用 Claude 代码 for learning and 探索. You can ask Claude the 相同 类别s of 问题s you would ask another 工程师:

- How does日志 工作?
- How do I make a 新 API 结束point?
- What does `异步 移动 { ... }` do on 行 134 of `foo.rs`?
- What 边 案例s does `习俗erOn董事会ingF低Impl` handle?
- Why does this 代码 c所有 `foo()` instead of `bar()` on 行 333?

Using Claude 代码 this 方式 is an 有效 on董事会ing 工作流,改善 ramp-上 时间 and reducing 加载 on other 工程师s. No 特殊 及时ing 必需: ask 问题s 直接ly.

Let Claude inter视图 you

For larger features, have Claude inter视图 you 第一个. 启动 with a 最小 及时 and ask Claude to inter视图 you using the `Ask用户问题` 工具.

Claude asks about 薄gs you might not have considered yet,包括 技术 实施, UI/UX, 边案例s, and 贸易offs.

```文本 主题={空}
I 想要 to 构建 [简短 描述]. Inter视图 me in 详情 using the Ask用户问题 工具.

Ask about 技术 实施, UI/UX, 边 案例s, 关心s, and 贸易offs. Don't ask 明显 问题s, dig into the 硬 部分s I might not have considered.
```

Keep inter视图ing until we've c结束ed 每个薄g, then 写入 a 完成 规格 to 规格.md.

```
Once the 规格 is 完成, 启动 a 新鲜 会话 to 执行 it. The 新 会话 has 干净 上下文 角

***

## Manage your 会话

对话s are 持续 and reversible. 使用 this to your 优势!

### Course-正确 早 and often

  正确 Claude as soon as you 通知 it 前往ing off 跟踪.

The 最佳 结果s come from 紧 反馈 loops. Though Claude 偶尔ly solves 问题s 完美

* **`Esc`**: 停止 Claude mid-行动 with the `Esc` 键. 上下文 is 保存, so you
* **`Esc + Esc` or `/re风`**: press `Esc` twice or 运行 `/re风` to 打开 the
* **`"撤销 that"`**: have Claude 撤销 its 更改s.
* **`/清楚`**: 重置 上下文 between unr兴高采烈 任务s. 长 会话s with 不相关 上下文

If you've 正确ed Claude 更多 than twice on the 相同 问题 in one 会话, the 上下

### Manage 上下文 aggressively

  运行 `/清楚` between unr兴高采烈 任务s to 重置 上下文.

Claude 代码 自动所有y 紧凑s 对话 history when you 方法 上下文 限制s, which pres

期间 长 会话s, Claude's 上下文 风ow can fill with 不相关 对话, 文件 满意s, and

* 使用 `/清楚` 频繁ly between 任务s to 重置 the 上下文 风ow 整个ly
* When auto 紧凑ion triggers, Claude 总和marizes what 事情s 最多,包括 代码 模
* For 更多 控制, 运行 `/紧凑 `, 像 `/紧凑 焦点 on the API 更改s`
* To 紧凑 only 部分 of the 对话, 使用 `Esc + Esc` or `/re风`, 选择 a 消息 检查
* 习俗ize 紧凑ion behavior in CLAUDE.md with 说明 像 `"When 紧凑ing, al方式s
* For 快 问题s that don't 需要 to stay in 上下文, 使用 [`/btw`](/en/交互-模式#
```

```
### 使用 子代理 for 调查

  Delegate研究 with `"使用 子代理 to调查 X"`. They探索 in a 单独 上下文,保持 you

Since 上下文 is your 资金a心理 常量雨t, 子代理 are one of the 最多 强大 工具 可用

```文本  主题={空}
使用 子代理 to调查 how our 认证 系统 handles 令牌
刷新, and whether we have 任何 现有 OAuth utilities I should re使用.
```

The sub代理探索s the 代码基础, 读取s 相关 文件, and 报告s 返回 with发现s, 所有 without cluttering your 主 对话.

You can also 使用 子代理 for 验证 之后 Claude 实现s 一些薄g:

```文本 主题={空}
使用 a sub代理 to re视图 this 代码 for 边 案例s
```
```

```
### Re风 with 检查points

  每个 行动 Claude makes 创建s a 检查point. You can 恢复 对话, 代码, or 机器人h

Claude 自动所有y 检查points 之前 更改s. 双精度-tap `Escape` or 运行 `/re风` to

Instead of 小心ly 规划 每个 移动, you can tell Claude to 尝试 一些薄g 风险y. I

  检查points only 跟踪 更改s made *by Claude*, not 外部 流程es. This isn't a

### 恢复 对话s

  运行 `claude --继续` to pick 上 where you 左 off, or `--恢复` to choose fr

Claude 代码 保存s 对话s 本地ly. When a 任务 跨度s mul提示le 会话s, you don't h

```bash  主题={空}
claude --继续    # 恢复 the 最多 最近 对话
claude --恢复      # 选择 from 最近 对话s
```

使用 `/重命名` to give 会话s de脚本ive 名称s 像 `"oauth-mig比率n"` or `"调试ging-记忆-leak"` so you can 查找 them 更晚. Treat 会话s 像 分支es: 不同 工作流s can have 单独, 持续 上下文s.

Automate and scale

Once you're 有效 with one Claude, mul提示ly your 输出 with 平行 会话s, non-交互 模式, and fan-out 模式s.

每个薄g so far as总和es one 人类, one Claude, and one 对话. But Claude 代码 scales 水平ly. The 技术 in this 节 显示 how you can get 更多 完成.
```

运行 non-交互 模式

使用 `claude -p "及时"` in CI, pre-提交 钩子, or 脚本s. 添加 `--输出-格式 流-json` for 流ing JSON 输出.

With `claude -p "your 及时"`, you can 运行 Claude non-交互ly, without a 会话. Non-交互 模式 is how you integ速率 Claude into CI 管道s, pre-提交 钩子, or 任何 automated 工作流. The 输出 格式s let you parse 结果s 计划matic所有y: 简单 文本, JSON, or 流ing JSON.

```bash 主题={空}

# One-off queries

claude -p "Ex简单 what this 项目 does"

# 结构化 输出 for 脚本s

claude -p "列表 所有 API 结束points" --输出-格式 json

# 流ing for 真实-时间 处理中

claude -p "分析 this 日志 文件" --输出-格式 流-json
```

```
### 运行 mul提示le Claude 会话s

  运行 mul提示le Claude 会话s in 平行 to 速度 上 开发, 运行 iso晚d 实验s, or 启动

There are three 主 方式s to 运行 平行 会话s:

* [Claude 代码 桌面 应用](/en/桌面#工作-in-平行-with-会话s): Manage mul提示le
* [Claude 代码 on the 网页](/en/claude-代码-on-the-网页): 运行 on Anthropic'
* [代理 团队](/en/代理-团队): Automated 协调 of mul提示le 会话s with 共享 任务s

超出 平行izing 工作, mul提示le 会话s 启用 quality-焦点ed 工作流s. A 新鲜 上下文

For 示例, 使用 a 写入器/审查员 模式:

| 会话 A (写入器)                                                          | 会
| ------------------------------------------------------------------------
| `实现 a 速率 限制er for our API 结束points`                              |
|
| `Here's the re视图 反馈: [会话 B 输出]. 添加ress these 问题s.` |

You can do 一些薄g 相似 with 测试s: have one Claude 写入 测试s, then another

### Fan out across 文件

  Loop th粗糙 任务s c所有ing `claude -p` for 每个. 使用 `--允许工具` to 范围 权

For large mig比率ns or analyses, you can distribute 工作 across 许多 平行 Cl


    Have Claude 列表 所有 文件 that 需要 mig评级 (e.g., `列表 所有 2,000 Pytho



    ```bash  主题={空}
    for 文件 in $(cat 文件.txt); do
```
```

```
      claude -p "Mig速率 $文件 from React to Vue. 回报 OK or FAIL." \
        --允许工具 "编辑,Bash(Git 提交 *)"
   完成
    ```


    Re好 your 及时 基础d on what 前往es 错误 with the 第一个 2-3 文件, then 运

You can also integ速率 Claude into 现有 数据/处理中 管道s:

```bash  主题={空}
claude -p "" --输出-格式 json | your_命令
```

使用 `--详细` for 调试ging 期间 开发, and turn it off in 生产.

运行 自治ly with auto 模式

For un中断ed 执行 with 背景 安全 检查s, 使用 auto 模式. A 阶级ifier 模型 re视图s 命令 之前 they 运行, 阻塞 范围 escalation, unknown 基础设施, and hos瓦-满意-driven 行动s while让 常规 工作 proceed without 及时s.

```bash 主题={空}
claude --许可-模式 auto -p "修复 所有 lint 错误"
```

```
For non-交互 运行s with the `-p` flag, auto 模式 中止s if the 阶级ifier repe

***

## A空白 常见 失败 模式s

These are 常见 薄雾akes.认可 them 早 保存s 时间:

* **The kitchen sink 会话.** You 启动 with one 任务, then ask Claude 一些薄g
  > **修复**: `/清楚` between unr兴高采烈 任务s.
* **正确ing 结束 and 结束.** Claude does 一些薄g 错误, you 正确 it, it's 静止
  > **修复**: 之后 two 失败 正确ions, `/清楚` and 写入 a 更好 initial 及时合并 w
* **The 结束-指定 CLAUDE.md.** If your CLAUDE.md is too 长, Claude ignores
  > **修复**: 无情ly p运行e. If Claude al就绪 does 一些薄g 正确ly without the
* **The 信任-then-验证 gap.** Claude produces a plausible-看 实施 that does
  > **修复**: Al方式s provide 验证 (测试s, 脚本s, 截图s). If you can't 验证 i
* **The infinite 探索.** You ask Claude to "调查" 一些薄g without scoping i
  > **修复**: 范围 调查s nar行ly or 使用 子代理 so the 探索 doesn't con总和e yo

***

## Develop your 直觉

The 模式s in this 指南 aren't 设置 in 石头. They're 启动ing points that 工作

一些时间s you *should* let 上下文 accumu晚 be原因 you're 深 in one 复杂 问题 an

Pay 注意 to what 工作s. When Claude produces great 输出, 通知 what you did:

结束 时间, you'll develop 直觉 that no 指南 can capture. You'll know when to

## R兴高采烈 资源

* [How Claude 代码 工作s](/en/how-claude-代码-工作s): the 代理ic loop, 工具,
* [Ext结束 Claude 代码](/en/features-概述): 技能, 钩子, MCP, 子代理, and 插件s
```

```
* [通用工作流](/en/常见-工作流s):步骤-by-步骤 recipes for 调试ging, 测试, PRs,
* [CLAUDE.md](/en/记忆): store 项目 惯例 and 持续 上下文

---

# 设置

> ## 文档 索引
> 获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
> 使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.

# Claude 代码 设置

> Con图 Claude 代码 with 全球 and 项目-级别 设置, and 环境变量.

Claude 代码 offers a 多样 of 设置 to con图 its behavior to meet your 需要s.

## 配置 范围s

Claude 代码 使用s a **范围 系统** to determine where 配置s 应用ly and who the

### 可用 范围s

| 范围        | Location
| :---------- | :-------------------------------------------------------
| **管理** | 服务器-管理 设置, p列表 / regis尝试, or 系统-级别 `管理-设置.json`
| **用户**    | `~/.claude/` 目录
| **项目** | `.claude/` in 仓库
| **本地**   | `.claude/设置.本地.json`

### When to 使用 每个 范围

**管理 范围** is for:

* 安全 政策 that must be en强迫 组织-wide
* 合规 要求 that can't be 覆盖
* 标准ized 配置s 部署ed by IT/DevOps
```

```
**用户 范围** is 最佳 for:

* 个人 偏好设置 you 想要 每个where (主题s, 编辑or 设置)
* 工具 and 插件s you 使用 across 所有 项目s
* API 键s and 认证 (存储 安全ly)

**项目 范围** is 最佳 for:

* 团队-共享 设置 (权限, 钩子, MCP 服务器)
* 插件s the 整体 团队 should have
* 标准izing 工具ing across collaborators

**本地 范围** is 最佳 for:

* 个人 结束rides for a 特定 项目
* 测试 配置s 之前分享 with the 团队
* Machine-特定 设置 that 获胜't 工作 for others

### How 范围s interact

When the 相同设置 is con图d in mul提示le 范围s, 更多 特定 范围s take precedenc

1. **管理** (高est) - can't be 覆盖 by 任何薄g
2. **命令 行 参数** - 临时 会话 结束rides
3. **本地** - 结束rides 项目 and 用户 设置
4. **项目** - 结束rides 用户 设置
5. **用户** (低est) - 应用lies when no薄g else 规格ifies the设置

For 示例, if a 许可 is 允许 in 用户 设置 but 拒绝 in 项目 设置, the 项目设置 tak

### What 使用s 范围s

范围s 应用ly to 许多 Claude 代码 features:

| Feature          | 用户 location              | 项目 location
| :--------------- | :------------------------- | :-----------------------
```

```
| **设置**    | `~/.claude/设置.json` | `.claude/设置.json`            | `
| **子代理**   | `~/.claude/代理s/`       | `.claude/代理s/`
| **MCP 服务器** | `~/.claude.json`          | `.mcp.json`
| **插件s**     | `~/.claude/设置.json` | `.claude/设置.json`            |
| **CLAUDE.md**   | `~/.claude/CLAUDE.md`     | `CLAUDE.md` or `.claude/C

***

## 设置 文件

The `设置.json` 文件 is the official mechanism for configuring Claude
代码 th粗糙 分层 设置:

* **用户 设置** are 定义 in `~/.claude/设置.json` and 应用ly to 所有
  项目s.
* **项目 设置** are 保存 in your 项目 目录:
  * `.claude/设置.json` for 设置 that are 检查ed into 来源 控制 and 共享 with
  * `.claude/设置.本地.json` for 设置 that are not 检查ed in, 有用 for 个人 偏
* **管理 设置**: For 组织s that 需要 集中 控制, Claude 代码 支持s mul提示le del

  * **服务器-管理 设置**: 交付 from Anthropic's 服务器s via the Claude.ai 管理
  * **MDM/OS-级别 政策**: 交付 th粗糙 本地 device 管理 on macOS and 风ows:
    * macOS: `com.anthropic.clau解码` 管理 偏好设置 do主 (部署ed via 配置 pro
    * 风ows: `HKLM\软件\政策\Clau解码` regis尝试 键 with a `设置` 值 (REG\_SZ
    * 风ows (用户-级别): `HKCU\软件\政策\Clau解码` (低est 政策 优先级, only 使用
  * **文件-基础d**: `管理-设置.json` and `管理-mcp.json` 部署ed to 系统 总监ie

    * macOS: `/库/应用程序 支持/Clau解码/`
    * Linux and WSL: `/etc/claude-代码/`
    * 风ows: `C:\计划 文件\Clau解码\`


      The legacy 风ows 路径 `C:\计划数据\Clau解码\管理-设置.json` is no 长er


   文件-基础d 管理 设置 also 支持 a 下降-in 目录 at `管理-设置.d/` in the 相同 系
```

```
  跟随 the 系统d 惯例, `管理-设置.json` is 合并 第一个 as the 基础, then 所有 `

   使用 numeric pre修复es to 控制 合并 顺序, for 示例 `10-teleme尝试.json` an

看见 [管理 设置](/en/权限#管理-only-设置) and [管理 MCP 配置](/en/mcp#管理-mcp


   管理 部署ments can also re严格 **插件 市场place 添加itions** using
    `严格Known市场places`. For 更多 信息,看见 [管理 市场place re严格ions](/en/

* **Other 配置** is 存储 in `~/.claude.json`. This 文件 contains your 偏好设

  Claude 代码 自动所有y 创建s 时间戳ed 备份s of 配置 文件 and retains the five

```JSON 示例 设置.json 主题={空}
{
  "$模式": "https://json.模式store.org/claude-代码-设置.json",
  "权限": {
    "允许": [
      "Bash(npm 运行 lint)",
      "Bash(npm 运行 测试 *)",
      "读取(~/.zshrc)"
    ],
    "拒绝": [
      "Bash(curl *)",
      "读取(./.env)",
      "读取(./.env.*)",
      "读取(./秘密s/**)"
    ]
  },
  "env": {
    "CLAUDE_代码_启用_TELEME尝试": "1",
    "OTEL_指标S_出口ER": "otlp"
  },
  "公司Announ水泥s": [
```
```

```
    "Welcome to Acme Corp! Re视图 our 代码 准则 at docs.acme.com",
    "Reminder: 代码 re视图s 必需 for 所有 PRs",
    "新 安全 政策 in 效果"
  ]
}
```

The `$模式` 行 in the 示例 above points to the official JSON 模式 for Claude 代码 设置. 添加 it to your `设置.json` 启用s auto完成 and in行 验证 in VS 代码, Cursor, and 任何 other 编辑or that 支持s JSON 模式 验证.

可用 设置

`设置.json` 支持s a 数字 of 选项:

键	描述	示例
`代理`	运行 the 主 th读取 as a 名称d sub代理. 应用lies that sub代理's 系统 及时, 工具 re严格ions, and 模型.看见 Invoke 子代理 明确ly	`"代码-审查员"`
`允许通道插件s`	(管理 设置 only) 允许列表 of 通道 插件s that may 推送 消息s. 替换s the 默认 Anthropic 允许列表 when 设置. 未定义 = 下降 返回 to the 默认, 空 数组 = 块 所有 通道 插件s. Requ怒s `通道s启用: 真实` .看见 Re严格 which 通道 插件s can 运行	`[{ "市场place": "claude-插件s-official", "插件": "telegram" }]`
`允许HttpH值ookUrls`	允许列表 of URL 模式s that HTTP 钩子 may 目标. 支持s `*` as a wildcard. When 设置, 钩子 with non-匹配 URLs are 阻塞. 未定义 = no re严格ion, 空 数组 = 块 所有 HTTP 钩子. 数组s 合并 across 设置 来源s.看见 Hook 配置	`["https://钩子.示例.com/*"]`

键	描述	示例
`允许Mcp服务器s`	When 设置 in 管理-设置.json, 允许列表 of MCP 服务器 用户s can con图. 未定义 = no re严格ions, 空 数组 = 锁定下. 应用lies to 所有 范围s. 拒绝列表 takes precedence.看见 管理 MCP 配置	`[{ "服务器名称": "GitHub" }]`
`允许管理钩子Only`	(管理 设置 only) Pr事件 加载ing of 用户, 项目, and 插件 钩子. Only 允许s 管理 钩子 and SDK 钩子.看见 Hook 配置	`真实`
`允许管理Mcp服务器sOnly`	(管理 设置 only) Only `允许Mcp服务器s` from 管理 设置 are 尊敬. `拒绝Mcp服务器s` 静止 合并s from 所有 来源s. 用户s can 静止 添加 MCP 服务器, but only the 管理员-定义 允许列表 应用lies.看见 管理 MCP 配置	`真实`
`允许管理许可规则Only`	(管理 设置 only) Pr事件 用户 and 项目 设置 from定义 `允许`, `ask`, or `拒绝` 许可 规则. Only 规则 in 管理 设置 应用ly.看见 管理-only 设置	`真实`
`al方式s薄king启用`	启用 延长思考 by 默认 for 所有 会话s. 典型ly con图d via the `/config` 命令 rather than 编辑ing 直接ly	`真实`
`api键帮助er`	习俗 脚本, to be 执行d in `/bin/sh`, to gene速率 an auth 值. This 值 will be 发送 as `X-Api-键` and `授权: Bearer` 页眉s for 模型 请求s	`/bin/gene速率_temp_api_键.sh`

键	描述	示例
`attribution`	习俗ize attribution for Git 提交s and 拉取请求s.看见 Attribution 设置	`{"提交": "🤖 Gene速率d with Claude 代码", "pr": ""}`
`auto记忆目录`	习俗 目录 for auto 记忆 sto狂怒. 接受s `~/`-扩展 路径s. Not 接受 in 项目 设置 (`.claude/设置.json`) to pr事件 共享 repos from 重定向ing 记忆 写入s to 敏感 locations. 接受 from 政策, 本地, and 用户 设置	`"~/my-记忆-dir"`
`auto模式`	习俗ize what the auto 模式 阶级ifier 块s and 允许s. Contains `环境` , `允许` , and `软_拒绝` 数组s of prose 规则.看见 Con图 the auto 模式 阶级ifier. Not 读取 from 共享 项目 设置	`{"环境": ["信任ed repo: GitHub.示例.com/acme"]}`
`auto更新s通道`	释放 通道 to fol低 for 更新s. 使用 `"稳定"` for a 版本 that is 典型ly about one week 旧 and跳s 版本s with 主要 退步ions, or `"最新"` (默认) for the 最多 最近 释放	`"稳定"`
`可用模型s`	Re严格 which 模型s 用户s can 选择 via `/模型` , `--模型` , Config 工具, or `ANTHROPIC_模型` . Does not affect the 默认 选项.看见 Re严格 模型 选择	`["sonnet", "haiku"]`
`awsAuth刷新`	习俗 脚本 that modifies the `.aws` 目录 (看见 先进 credential 配置)	`aws sso 日志in --pro文件 mypro文件`

键	描述	示例
`awsCredential出口`	习俗 脚本 that 输出s JSON with AWS credentials (看见 先进 credential 配置)	`/bin/gene速率_aws_授予.sh`
`阻塞市场places`	(管理 设置 only) 块列表 of 市场 place 来源s. 阻塞 来源s are 检查ed 之前 下载ing, so they never touch the 文件系统.看见 管理 市场place re严格ions	`[{ "来源": "GitHub", "repo": "un信任ed/插件s" }]`
`通道s启用`	(管理 设置 only) 允许 通道s for 团队 and 企业 用户s. 未设置 or `虚假` 块s 通道 消息 delivery 注意较少 of what 用户s pass to `--通道s`	`真实`
`干净上期间Days`	会话s 不活跃 for 长er than this 期间 are 删除 at 启动上 (默认: 30 days, 最小 1).设置 to `0` is 拒绝 with a 验证 错误. Also 控制s the age 剪切off for 自动 rem椭圆形 of orpH值aned sub代理 工作树 at 启动上. To 禁用 tran脚本 写入s 整个ly in non-交互 模式 (`-p`), 使用 the `--no-会话-persistence` flag or the `persist会话: 虚假` SDK 选项; there is no 交互-模式 等价.	`20`
`公司Announ水泥s`	Announ水泥 to 显示 to 用户s at 启动上. If mul提示le announ水泥s are provided, they will be cyc 领导 th粗糙 at 随机.	`["Welcome to Acme Corp! Re视图 our 代码 准则 at docs.acme.com"]`
`默认命令行`	默认 命令行 for 输入-box `!` 命令. 接受s `"bash"` (默认) or	`"权力命令行"`

键	描述	示例
	`"权力命令行"`.设置 `"权力命令行"` 路由s 交互 `！` 命令 th粗糙 权力命令行 on 风ows. Requ怒s `CLAUDE_代码_使用_权力命令行_工具=1` .看见 权力命令行 工具	
`拒绝Mcp服务器s`	When 设置 in 管理-设置.json, 拒绝列表 of MCP 服务器 that are 明确ly 阻塞. 应用lies to 所有 范围s 包括 管理 服务器s. 拒绝列表 takes precedence 结束 允许列表. 看见 管理 MCP 配置	`[{ "服务器名称": "文件系统" }]`
`禁用所有钩子`	禁用 所有 钩子 and 任何 习俗 状态 行	`真实`
`禁用Auto模式`	设置 to `"禁用"` to pr事件 auto 模式 from being 激活d. 移除s `auto` from the `Shift+Tab` cycle and 拒绝s `--许可-模式 auto` at 启动上. 最多 有用 in 管理 设置 where 用户s cannot 结束 ride it	`"禁用"`
`禁用eep链接 Regist比率n`	设置 to `"禁用"` to pr事件 Claude 代码 from注册 the `claude-cli://` 协议 处理器 with the操作 系统 on 启动上. 深链接s let 外部 工具 打开 a Claude 代码 会话 with a pre-满 及时 via `claude-cli://打开?q=...`. The `q` 参数 支持s multi-行 及时s using URL-编码 新行s (`%0A`). 有用 in 环境s where 协议 处理器 regist比率n is 受限 or 管理 单独ly	`"禁用"`

键	描述	示例
`禁用Mcpjson服务器s`	列表 of 特定 MCP 服务器 from `.mcp.json` 文件 to 拒绝	`["文件系统"]`
`effort级别`	Persist the effort 级别 across 会话s. 接受s `"低"`, `"中"`, or `"高"`. Written 自动所有y when you 运行 `/effort 低`, `/effort 中`, or `/effort 高`. 支持ed on Opus 4.6 and Sonnet 4.6	`"中"`
`启用所有项目Mcp服务器s`	自动所有y 批准 所有 MCP 服务器 定义 in 项目 `.mcp.json` 文件	`真实`
`启用Mcpjson服务器s`	列表 of 特定 MCP 服务器 from `.mcp.json` 文件 to 批准	`["记忆", "GitHub"]`
`env`	环境变量 that will be 应用 to 每个 会话	`{"FOO": "bar"}`
`快模式Per会话OptIn`	When `真实`, 快 模式 does not persist across 会话s. 每个 会话 启动s with 快 模式 off,要求 用户s to 启用 it with `/快`. The 用户's 快 模式 偏好 is 静止 保存.看见 Requ怒 per-会话 opt-in	`真实`
`反馈调查速率`	概率 (0–1) that the 会话 quality 调查 应用ears when eligible. 设置 to `0` to s上press 整个ly. 有用 when using 基岩, Vertex, or Foun干 where the 默认 样本 速率 does not 应用ly	`0.05`
`文件建议`	Con图 a 习俗 脚本 for `@` 文件 auto完成.看见 文件 建议 设置	`{"类型": "命令", "命令": "~/.claude/文件-建议.sh"}`

键	描述	示例
`强制日志in方法`	使用 `claudeai` to re严格 日志in to Claude.ai 说明s, `控制台` to re严格 日志in to Claude 控制台 (API 使用方法计费) 说明s	`claudeai`
`强制日志inOrgUUID`	Requ怒 日志in to be长 to a 特定 组织. 接受s a single UUID 字符串, which also pre-选择s that 组织 期间 日志in, or an 数组 of UUIDs where 任何 列出 组织 is 接受 without pre-选择. When 设置 in 管理 设置, 日志in fails if the 认证 说明 does not be长 to a 列出 组织; an 空 数组 fails 关闭 and 块s 日志in with a mis配置 消息	`"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"` or `["xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx", "yyyyyyyy-yyyy-yyyy-yyyy-yyyyyyyyyyyy"]`
`钩子`	Con图 习俗 命令 to 运行 at lifecycle 事件s.看见 钩子 文档 for 格式	看见 钩子
`httpH值ook允许EnvVars`	允许列表 of 环境 可变 名称s HTTP 钩子 may interpo晚 into 页眉s. When 设置, 每个 hook's 有效 `允许EnvVars` is the inter节 with this 列表. 未定义 = no re严格ion. 数组s 合并 across 设置 来源s.看见 Hook 配置	`["MY_令牌", "HOOK_秘密"]`
`includeCo作者edBy`	**Deprecated**: 使用 `attribution` instead. Whether to include the `co-作者ed-by Claude` by行 in Git 提交s and 拉取请求s (默认: `真实`)	`虚假`
`includeGit说明`	Include built-in 提交 and PR 工作流 说明 and the Git 状态 snaps热 in Claude's 系统 及时 (默认:	`虚假`

键	描述	示例
	真实). 设置 to 虚假 to 移除 机器人h, for 示例 when using your own Git 工作流 技能. The CLAUDE_代码_禁用_Git_说明 环境 可变 takes precedence 结束 this设置 when 设置	
language	Con图 Claude's preferred 响应 language (e.g., "japanese", "跨度ish", "french"). Claude will respond in this language by 默认. Also 设置s the 语音 dictation language	"japanese"
模型	结束ride the 默认 模型 to 使用 for Claude 代码	"claude-sonnet-4-6"
模型结束rides	映射 Anthropic 模型 IDs to 提供者-特定 模型 IDs such as 基岩 inference pro文件 ARNs. 每个 模型 picker 条目 使用s its m应用ed 值 when c所有ing the 提供者 API.看见 结束ride 模型 IDs per 版本	{"claude-opus-4-6": "arn:aws:基岩:..."}
otel页眉s帮助er	脚本 to gene速率 动态 打开 Teleme尝试 页眉s. 运行s at 启动 上 and 期间ic所有y (看见 动态 页眉s)	/bin/gene速率_otel_页眉s.sh
输出Style	Con图 an 输出 style to ad公正 the 系统 及时.看见 输出 styles 文档	"Ex计划atory"
权限	看见 表 be低 for 结构 of 权限.	
计划s目录		"./计划s"

键	描述	示例
	习俗ize where 计划 文件 are 存储. 路径 is relative to 项目 根. 默认: `~/.claude/计划s`	
`插件信任消息`	(管理 设置 only) 习俗 消息 应用结束 to the 插件 信任警告 显示n 之前 安装. 使用 this to 添加 组织-特定 上下文, for 示例 to 确认 that 插件s from your 内部 市场place are vetted.	`"所有 插件s from our 市场place are 批准 by IT"`
`prefers减少 Motion`	Reduce or 禁用 UI 动画s (sp内部s, shimmer, flash 效果s) for accessibility	`真实`
`re规格 tGitignore`	控制 whether the `@` 文件 picker re规格ts `.Gitignore` 模式s. When `真实` (默认), 文件 匹配 `.Gitignore` 模式s are 排除 from 建议s	`虚假`
`显示清楚上下文On 计划接受`	显示 the "清楚 上下文" 选项 on the 计划 接受 screen. 默认s to `虚假`. 设置 to `真实` to 恢复 the 选项	`真实`
`显示薄king总和 maries`	显示 延长思考 总和maries in 交互 会话s. When 未设置 or `虚假` (默认 in 交互 模式),思考 块s are redacted by the API and 显示n as a collapsed 桩. Red行动 only 更改s what you看见, not what the 模型 gene速率s: to reduce思考 sp结束, 低er the 预算 or 禁用 思考 instead. Non-交互 模式 (`-p`) and SDK c所有ers al方式s	`真实`

键	描述	示例
	接收 总和maries 注意较少 of this 设置	
`sp内部提示启用`	显示 提示 in the sp内部 while Claude is工作. 设置 to `虚假` to 禁用 提示 (默认: `真实`)	`虚假`
`sp内部提示结束ride`	结束ride sp内部 提示 with 习俗 字符串s. `提示`: 数组 of 提示 字符串s. `排除efault`: if `真实`, only 显示 习俗 提示; if `虚假` or ab发送, 习俗 提示 are 合并 with built-in 提示	`{ "排除efault": 真实, "提示": ["使用 our 内部工具 X"] }`
`sp内部Verbs`	习俗ize the 行动 verbs 显示n in the sp内部 and turn 持续时间 消息s. 设置 `模式` to `"替换"` to 使用 only your verbs, or `"应用结束"` to 添加 them to the 默认s	`{"模式": "应用结束", "verbs": ["Pondering", "工艺ing"]}`
`状态行`	Con图 a 习俗 状态 行 to 显示 上下文.看见 `状态行` 文档	`{"类型": "命令", "命令": "~/.claude/状态行.sh"}`
`严格Known市场places`	(管理 设置 only) 允许列表 of 插件 市场places 用户s can 添加. 未定义 = no re严格ions, 空 数组 = 锁定下. 应用lies to 市场place 添加itions only.看见 管理 市场place re严格ions	`[{ "来源": "GitHub", "repo": "acme-corp/插件s" }]`
`使用Auto模式期间计划`	Whether 计划 模式 使用s auto 模式 semantics when auto 模式 is 可用. 默认: `真实`. Not 读取 from 共享 项目 设置. 应用ears in `/config` as "使用 auto 模式 期间计划"	`虚假`

键	描述	示例
`语音启用`	启用 推送-to-talk 语音 dictation. Written 自动所有y when you 运行 `/语音` . Requ怒s a Claude.ai 说明	`真实`

全球 config 设置

These 设置 are 存储 in `~/.claude.json` rather than `设置.json` .添加 them to `设置.json` will trigger a 模式 验证 错误.

键	描述	示例
`autoConnectIde`	自动所有y connect to a 运行中 IDE when Claude 代码 启动s from an 外部 终端. 默认: `虚假` . 应用ears in `/config` as **Auto-connect to IDE (外部终端)** when 运行中 外部 a VS 代码 or JetB雨s 终端	`真实`
`auto安装Ide扩展`	自动所有y 安装 the Claude 代码 IDE 扩展 when 运行中 from a VS 代码 终端. 默认: `真实` . 应用ears in `/config` as **Auto-安装 IDE 扩展** when 运行中 内部 a VS 代码 or JetB雨s 终端. You can also 设置 the `CLAUDE_代码_IDE_跳_AUTO_安装` 环境 可变	`虚假`
`编辑or模式`	键绑定 模式 for the 输入 及时: `"正常"` or `"vim"` . 默认: `"正常"` . Written 自动所有y when you 运行 `/vim` . 应用ears in `/config` as **键绑定 模式**	`"vim"`
`显示Turn持续时间`	显示 turn 持续时间 消息s 之后 响应s, e.g. "首席运营官ked for 1m 6s". 默认: `真实` . 应用ears in `/config` as **显示 turn 持续时间**	`虚假`
`终端进步Bar启用`	显示 the 终端 进步 bar in 支持ed 终端s: ConEmu, Ghostty 1.2.0+, and iTerm2 3.6.6+. 默认: `真实` . 应用ears in `/config` as **终端 进步 bar**	`虚假`
`团队mate模式`		

键	描述	示例
	How 代理 团队 团队mates 显示: `auto` (picks split panes in tmux or iTerm2, in-流程 other明智), `in-流程` , or `tmux` .看见 choose a 显示 模式	`"in-流程"`

工作树 设置

Con图 how `--工作树` 创建s and manages Git 工作树. 使用 these 设置 to reduce disk 使用方法 and 启动上 时间 in large monorepos.

键	描述	示例
`工作树.sym链接总监ies`	总监ies to sym链接 from the 主 仓库 into 每个 工作树 to a空白复制 large 总监ies on disk. No 总监ies are sym链接 by 默认	`["节点_模块s", ".缓存"]`
`工作树.稀疏路径s`	总监ies to 检查 out in 每个 工作树 via Git 稀疏-检查out (cone 模式). Only the 列出 路径s are written to disk, which is 快er in large monorepos	`["包s/my-应用", "共享/utils"]`

To 复制 Git忽略 文件 像 `.env` into 新 工作树, 使用 a `.工作树include` 文件 in your 项目 根 instead of a设置.

许可 设置

键s	描述	示例
`允许`	数组 of 许可 规则 to 允许 工具 使用.看见 许可 规则 syn税 be低 for 模式 匹配 详情s	`[ "Bash(Git diff *)" ]`
`ask`	数组 of 许可 规则 to ask for 确认 上on 工具 使用.看见 许可 规则 syn税 be低	`[ "Bash(Git 推送 *)" ]`
`拒绝`	数组 of 许可 规则 to 拒绝 工具 使用. 使用 this to exclude 敏感 文件 from Claude 代码 access.看见 许可 规则 syn税 and Bash 许可 限制	`[ "网页获取", "Bash(curl *)", "读取(./.env)", "读取(./秘密s/**)" ]`

键s	描述	示例
`添加itional总监ies`	添加itional 工作 总监ies for 文件 access. 最多 `.claude/` 配置 is not disc结束ed from these 总监ies	`[ "../docs/" ]`
`默认模式`	默认 许可 模式 when 打开ing Claude 代码. 有效 值s: `默认`, `接受its`, `计划`, `auto`, `don任务`, `bypass权限`. The `--许可-模式` CLI flag 结束rides this设置 for a single 会话	`"接受its"`
`禁用Bypass权限模式`	设置 to `"禁用"` to pr事件 `bypass权限` 模式 from being 激活d. This 禁用s the `--d愤怒ously-跳-权限` 命令-行 flag. 典型ly placed in 管理 设置 to en强制 组织al 政策, but 工作s from 任何 范围	`"禁用"`
`跳D愤怒ous模式许可及时`	跳 the 确认 及时 显示n 之前进入 bypass 权限 模式 via `--d愤怒ously-跳-权限` or `默认模式: "bypass权限"`. 忽略 when 设置 in 项目 设置 (`.claude/设置.json`) to pr事件 un信任ed repositories from auto-bypassing the 及时	`真实`

许可 规则 syn税

许可 规则 fol低 the 格式 `工具` or `工具(规格ifier)`. 规则 are evaluated in 顺序: 拒绝 规则 第一个, then ask, then 允许. The 第一个 匹配 规则 wins.

快 示例:

规则	效果
`Bash`	匹配es 所有 Bash 命令
`Bash(npm 运行 *)`	匹配es 命令 启动ing with `npm 运行`
`读取(./.env)`	匹配es 读取ing the `.env` 文件

规则	效果
`网页获取(do主:示例.com)`	匹配es 获取 请求s to 示例.com

For the 完成 规则 syn税 参考,包括 wildcard behavior, 工具-特定 模式s for 读取, 编辑, 网页获取, MCP, and 代理 规则, and 安全 限制 of Bash 模式s,看见 许可 规则 syn税.

沙box 设置

Con图 先进 沙boxing behavior. 沙boxing iso晚s bash 命令 from your 文件系统 and net 工作.看见 沙boxing for 详情s.

键s	描述	示例
`启用`	启用 bash 沙boxing (macOS, Linux, and WSL2). 默认: 虚假	`真实`
`failIf不可用`	退出 with an 错误 at 启动上 if `沙box.启用` is 真实 but the 沙box cannot 启动 (missing 依赖, un支持ed 平台, or 平台 re严格ions). When 虚假 (默认), a警告 is 显示n and 命令 运行 un沙boxed. Int结束 for 管理 设置 部署ments that requ怒 沙boxing as a 硬 gate	`真实`
`auto允许BashIf沙boxed`	Auto-批准 bash 命令 when 沙boxed. 默认: 真实	`真实`
`排除命令`	命令 that should 运行 外部 of the 沙box	`["docker *"]`
`允许Un沙boxed命令`	允许 命令 to 运行 外部 the 沙box via the `d愤怒ously禁用沙box` 参数. When 设置 to `虚假`, the `d愤怒ously禁用沙box` escape hatch is 完成ly 禁用 and 所有 命令 must 运行 沙boxed (or be in `排除命令`). 有用 for 企业 政策 that requ怒 严格 沙boxing. 默认: 真实	`虚假`

键s	描述	示例
`文件系统.允许写入`	添加itional 路径s where 沙boxed 命令 can 写入. 数组s are 合并 across 所有 设置 范围s: 用户, 项目, and 管理 路径s are 组合, not 替换d. Also 合并 with 路径s from `编辑(...)` 允许 许可 规则.看见 路径 pre修复es be低.	`["/tmp/构建", "~/.kube"]`
`文件系统.拒绝写入`	路径s where 沙boxed 命令 cannot 写入. 数组s are 合并 across 所有 设置 范围s. Also 合并 with 路径s from `编辑(...)` 拒绝 许可 规则.	`["/etc", "/usr/本地/bin"]`
`文件系统.拒绝读取`	路径s where 沙boxed 命令 cannot 读取. 数组s are 合并 across 所有 设置 范围s. Also 合并 with 路径s from `读取(...)` 拒绝 许可 规则.	`["~/.aws/credentials"]`
`文件系统.允许读取`	路径s to re-允许 读取ing 在...内 `拒绝读取` regions. Takes precedence 结束 `拒绝读取`. 数组s are 合并 across 所有 设置 范围s. 使用 this to 创建 工作区-only 读取 access 模式s.	`["."]`
`文件系统.允许管理读取路径sOnly`	(管理 设置 only) Only `文件系统.允许读取` 路径s from 管理 设置 are 尊敬. `拒绝读取` 静止 合并s from 所有 来源s. 默认: 虚假	`真实`
`net工作.允许Unix套接字s`	Unix 套接字 路径s 可访问 in 沙box (for SSH 代理s, etc.)	`["~/.ssh/代理-套接字"]`
`net工作.允许所有Unix套接字s`	允许 所有 Unix 套接字 连接s in 沙box. 默认: 虚假	`真实`
`net工作.允许本地Binding`	允许绑定 to 本地host 端口s (macOS only). 默认: 虚假	`真实`

键s	描述	示例
`net工作.允许Do主s`	数组 of do主s to 允许 for outbound net 工作 traffic. 支持s wildcards (e.g., `*.示例.com`).	`["GitHub.com", "*.npmjs.org"]`
`net工作.允许管理Do主sOnly`	(管理 设置 only) Only `允许Do主s` and `网页获取(do主:...)` 允许 规则 from 管理 设置 are 尊敬. Do主s from 用户, 项目, and 本地 设置 are 忽略. Non-允许 do主s are 阻塞 自动所有y without 及时 ing the 用户. 拒绝 do主s are 静止 尊敬 from 所有 来源s. 默认: 虚假	`真实`
`net工作.httpProxy端口`	HTTP proxy 端口 使用d if you 希望 to bring your own proxy. If not 指定, Claude will 运行 its own proxy.	`8080`
`net工作.socksProxy端口`	SOCKS5 proxy 端口 使用d if you 希望 to bring your own proxy. If not 指定, Claude will 运行 its own proxy.	`8081`
`启用弱erNested沙box`	启用 弱er 沙box for unprivileged Docker 环境s (Linux and WSL2 only). **Reduces 安全.** 默认: 虚假	`真实`
`启用弱erNet工作Isolation`	(macOS only) 允许 access to the 系统 TLS 信任 服务 (`com.应用le.信任d.代理`) in the 沙box. 必需 for 前往-基础d 工具 像 `gh`, `g云`, and `terra形式` to 验证 TLS certificates when using `httpProxy端口` with a MITM proxy and 习俗 CA. **Reduces 安全** by 打开ing a 潜力 数据 exfilt比率n 路径. 默认: 虚假	`真实`

沙box 路径 pre修复es

路径s in `文件系统.允许写入`, `文件系统.拒绝写入`, `文件系统.拒绝读取`, and `文件系统.允许读取` 支持 these pre修复es:

Pre修复	Meaning	示例
`/`	Absolute 路径 from 文件系统 根	`/tmp/构建` stays `/tmp/构建`
`~/`	Relative to home 目录	`~/.kube` becomes `$HOME/.kube`
`./` or no pre修复	Relative to the 项目 根 for 项目 设置, or to `~/.claude` for 用户 设置	`./输出` in `.claude/设置.json` resolves to `/输出`

The 更旧 `//路径` pre修复 for absolute 路径s 静止 工作s. If you 之前ly 使用d single-slash `/路径` expecting 项目-relative 决议, switch to `./路径`. This syn税 differs from 读取 and 编辑 许可 规则, which 使用 `//路径` for absolute and `/路径` for 项目-relative. 沙box 文件系统 路径s 使用 标准 惯例: `/tmp/构建` is an absolute 路径.

配置 示例:

```json 主题={空}
{
"沙box": {
"启用": 真实,
"auto允许BashIf沙boxed": 真实,
"排除命令": ["docker "],
"文件系统": {
"允许写入": ["/tmp/构建", "~/.kube"],
"拒绝读取": ["~/.aws/credentials"]
},
"net工作": {
"允许Do主s": ["GitHub.com", ".npmjs.org", "regis尝试.yarnpkg.com"],
"允许Unix套接字s": [
"/var/运行/docker.sock"
],
"允许本地Binding": 真实
}
```

```
}
}
```

```
**文件系统 and net工作 re严格ions** can be con图d in two 方式s that are 合并

* **`沙box.文件系统` 设置** (显示n above): 控制 路径s at the OS-级别 沙box bou
* **许可 规则**: 使用 `编辑` 允许/拒绝 规则 to 控制 Claude's 文件 工具 access, `

### Attribution 设置

Claude 代码 添加s attribution to Git 提交s and 拉取请求s. These are con图d 单

* 提交s 使用 [Git trailers](https://Git-scm.com/docs/Git-解释-trailers) (像
* 拉取请求 描述s are 简单 文本

| 键s      | 描述
| :------- | :-----------------------------------------------------
| `提交` | Attribution for Git 提交s,包括 任何 trailers. 空 字符串 隐藏s 提交 a
| `pr`     | Attribution for 拉取请求 描述s. 空 字符串 隐藏s 拉取请求 attributi

**默认 提交 attribution:**

```文本  主题={空}
 Gene速率d with [Claude 代码](https://claude.com/claude-代码)

   Co-作者ed-By: Claude Sonnet 4.6
```

默认 拉取请求 attribution:

```文本 主题={空}
 Gene速率d with Claude 代码
```
```

示例:

```json 主题={空}
{
  "attribution": {
    "提交": "Gene速率d with AI\n\nCo-作者ed-By: AI ",
    "pr": ""
  }
}
```

The `attribution` 设置 takes precedence 结束 the deprecated `includeCo作者edBy` 设置. To 隐藏 所有 attribution, 设置 `提交` and `pr` to 空 字符串s.

文件 建议 设置

Con图 a 习俗 命令 for `@` 文件 路径 auto完成. The built-in 文件 建议 使用s 快 文件系统 traversal, but large monorepos may 益处 from 项目-特定 索引ing such as a pre-built 文件 索引 or 习俗 工具ing.

```json 主题={空}
{
"文件建议": {
"类型": "命令",
"命令": "~/.claude/文件-建议.sh"
}
}
```

The 命令 运行s with the 相同 环境变量 as [钩子](/en/钩子),包括 `CLAUDE_项目_DIR

```json 主题={空}
{"查询": "src/comp"}
```

输出 新行-分离 文件 路径s to stdout (当前ly 有限 to 15):

```文本 主题={空}
src/组件s/Button.tsx
```

src/组件s/Modal.tsx
src/组件s/形式.tsx

> **示例:**
>
> ```bash  主题={空}
> #!/bin/bash
> 查询=$(cat | jq -r '.查询')
> your-repo-文件-索引 --查询 "$查询" | 负责人 -20
> ```

Hook 配置

These 设置 控制 which 钩子 are 允许 to 运行 and what HTTP 钩子 can access. The `允许管理钩子Only` 设置 can only be con图d in 管理 设置. The URL and env var 允许列表s can be 设置 at 任何 设置 级别 and 合并 across 来源s.

Behavior when `允许管理钩子Only` is `真实`:

- 管理 钩子 and SDK 钩子 are 加载
- 用户 钩子, 项目 钩子, and 插件 钩子 are 阻塞

Re严格 HTTP hook URLs:

限制 which URLs HTTP 钩子 can 目标. 支持s `*` as a wildcard for 匹配. When the 数组 is 定义, HTTP 钩子 目标ing non-匹配 URLs are 沉默ly 阻塞.

```json 主题={空}
{
"允许HttpH值ookUrls": ["https://钩子.示例.com/", "http://本地host:"]
}
```

```
**Re严格 HTTP hook 环境变量:**

限制 which 环境 可变 名称s HTTP 钩子 can interpo晚 into 页眉 值s. 每个 hook's 有

```json  主题={空}
{
  "httpH值ook允许EnvVars": ["MY_令牌", "HOOK_秘密"]
}
```

设置 precedence

设置 应用ly in 顺序 of precedence. From 高est to 低est:

1. **管理 设置** (服务器-管理, MDM/OS-级别 政策, or 管理 设置)
2. 政策 部署ed by IT th粗糙 服务器 delivery, MDM 配置 pro文件, regis尝试 政策, or 管理 设置 文件
3. Cannot be 覆盖 by 任何 other 级别,包括 命令 行 参数
4. 在...内 the 管理 tier, precedence is: 服务器-管理 > MDM/OS-级别 政策 > 文件-基础d (`管理-设置.d/*.json` + `管理-设置.json`) > HKCU regis尝试 (风ows only). Only one 管理 来源 is 使用d; 来源s do not 合并 across tiers. 在...内 the 文件-基础d tier, 下降-in 文件 and the 基础 文件 are 合并 together.
5. **命令 行 参数**
6. 临时 结束rides for a 特定 会话
7. **本地 项目 设置** (`.claude/设置.本地.json`)
8. 个人 项目-特定 设置
9. **共享 项目 设置** (`.claude/设置.json`)
10. 团队-共享 项目 设置 in 来源 控制
11. **用户 设置** (`~/.claude/设置.json`)
12. 个人 全球 设置
```

This hierarchy en确定s that 组织al 政策 are al方式s en强迫 while 静止允许 团队 and 个人s to 习俗ize their experience. The 相同 precedence 应用lies whether you 运行 Claude 代码 from the CLI, the VS 代码 扩展, or a JetB雨s IDE.

For 示例, if your 用户 设置 允许 `Bash(npm 运行 *)` but a 项目's 共享 设置 拒绝 it, the 项目设置 takes precedence and the 命令 is 阻塞.

数组 设置 合并 across 范围s. When the 相同 数组-重视设置 (such as `沙box.文件系统.允许写入` or `权限.允许`) 应用ears in mul提示le 范围s, the 数组s are **concatenated and de重复d**, not 替换d. This 手段 低er-优先级 范围s can 添加 entries without 结束riding those 设置 by 高er-优先级 范围s, and vice versa. For 示例, if 管理 设置 设置 `允许写入` to `["/opt/公司-工具"]` and a 用户 添加s `["~/.kube"]`, 机器人h 路径s are 包含 in the 最终 配置.

验证 活跃 设置

运行 `/状态` 内部 Claude 代码 to看见 which 设置 来源s are 活跃 and where they come from. The 输出 显示s 每个 配置 layer (管理, 用户, 项目) a长 with its 起源, such as `企业 管理 设置 (远程)`, `企业 管理 设置 (p列表)`, `企业 管理 设置 (HKLM)`, or `企业 管理 设置 (文件)`. If a 设置 文件 contains 错误, `/状态` 报告s the 问题 so you can 修复 it.

键 points about the 配置 系统

- **记忆 文件 (`CLAUDE.md`)**: Contain 说明 and 上下文 that Claude 加载s at 启动上
- **设置 文件 (JSON)**: Con图 权限, 环境变量, and 工具 behavior
- **技能**: 习俗 及时s that can be invoked with `/技能-名称` or 加载 by Claude 自动所有y
- **MCP 服务器**: Ext结束 Claude 代码 with 添加itional 工具 and 集成s
- **Precedence**: 高er-级别 配置s (管理) 结束ride 低er-级别 ones (用户/项目)
- **Inheritance**: 设置 are 合并, with 更多 特定 设置添加 to or 结束riding broader ones

系统 及时

Claude 代码's 内部 系统 及时 is not published. To 添加 习俗 说明, 使用 `CLAUDE.md` 文件 or the `--应用结束-系统-及时` flag.

Excluding 敏感 文件

To pr事件 Claude 代码 from accessing 文件包含 敏感 信息 像 API 键s, 秘密s, and 环境 文件, 使用 the `权限.拒绝` 设置 in your `.claude/设置.json` 文件:

```json 主题={空}
{
"权限": {
"拒绝": [
"读取(./.env)",
"读取(./.env.)",
"读取(./秘密s/*)",
"读取(./config/credentials.json)",
"读取(./构建)"
]
}
}
```

```
This 替换s the deprecated `ignore模式s` 配置. 文件 匹配 these 模式s are 排除 f

## Sub代理 配置

Claude 代码 支持s 习俗 AI 子代理 that can be con图d at 机器人h 用户 and 项目 级

* **用户 子代理**: `~/.claude/代理s/` - 可用 across 所有 your 项目s
* **项目 子代理**: `.claude/代理s/` - 特定 to your 项目 and can be 共享 with y

Sub代理 文件 de好 专业 AI assistants with 习俗 及时s and 工具 权限. Learn 更多 a

## 插件 配置

Claude 代码 支持s a 插件 系统 that lets you ext结束 函数式ity with 技能, 代理s,

### 插件 设置

插件-r兴高采烈 设置 in `设置.json`:

```json  主题={空}
{
  "启用插件s": {
    "for事情@acme-工具": 真实,
    "部署er@acme-工具": 真实,
    "分析r@安全-插件s": 虚假
  },
  "extraKnown市场places": {
    "acme-工具": {
      "来源": "GitHub",
      "repo": "acme-corp/claude-插件s"
    }
  }
}
```
```

`启用插件s`

控制s which 插件s are 启用. 格式: `"插件-名称@市场place-名称": 真实/虚假`

范围s:

- **用户 设置** (`~/.claude/设置.json`): 个人 插件 偏好设置
- **项目 设置** (`.claude/设置.json`): 项目-特定 插件s 共享 with 团队
- **本地 设置** (`.claude/设置.本地.json`): Per-machine 结束rides (not 提交ted)
- **管理 设置** (`管理-设置.json`): 组织-wide 政策 结束rides that 块 安装 at 所有 范围s and 隐藏 the 插件 from the 市场place

示例:

```json 主题={空}
{
"启用插件s": {
"代码-for事情@团队-工具": 真实,
"部署ment-工具@团队-工具": 真实,
"实验-features@个人": 虚假
}
}
```

```
#### `extraKnown市场places`

De好s 添加itional 市场places that should be made 可用 for the 仓库. 典型ly 使

**When a 仓库 includes `extraKnown市场places`**:

1. 团队 成员s are 及时ed to 安装 the 市场place when they 信任 the 文件夹
2. 团队 成员s are then 及时ed to 安装 插件s from that 市场place
3. 用户s can跳 un想要ed 市场places or 插件s (存储 in 用户 设置)
4. 安装 re规格ts 信任 boundaries and requ怒s 明确 con发送

**示例**:

```json  主题={空}
{
  "extraKnown市场places": {
    "acme-工具": {
      "来源": {
        "来源": "GitHub",
        "repo": "acme-corp/claude-插件s"
      }
    },
    "安全-插件s": {
      "来源": {
        "来源": "Git",
        "url": "https://Git.示例.com/安全/插件s.Git"
      }
    }
  }
}
```

市场place 来源 类型s:

- `GitHub`: GitHub 仓库 (使用s `repo`)
- `Git`: 任何 Git URL (使用s `url`)
- `目录`: 本地 文件系统 路径 (使用s `路径`, for 开发 only)
```

- `host模式`: regex 模式 to 匹配 市场place hosts (使用s `host模式`)
- `设置`: in行 市场place declared 直接ly in 设置.json without a 单独 hosted 仓库 (使用s `名称` and `插件s`)

使用 `来源: '设置'` to declare a 小 设置 of 插件s in行 without建立 a hosted 市场place 仓库. 插件s 列出 here must 参考 外部 来源s such as GitHub or npm. You 静止 需要 to 启用 每个 插件 单独ly in `启用插件s`.

```json 主题={空}
{
"extraKnown市场places": {
"团队-工具": {
"来源": {
"来源": "设置",
"名称": "团队-工具",
"插件s": [
{
"名称": "代码-for事情",
"来源": {
"来源": "GitHub",
"repo": "acme-corp/代码-for事情"
}
}
]
}
}
}
}
```

```
#### `严格Known市场places`

**管理 设置 only**: 控制s which 插件 市场places 用户s are 允许 to 添加. This设置

**管理 设置 文件 locations**:

* **macOS**: `/库/应用程序 支持/Clau解码/管理-设置.json`
* **Linux and WSL**: `/etc/claude-代码/管理-设置.json`
* **风ows**: `C:\计划 文件\Clau解码\管理-设置.json`

**键 字符istics**:

* Only 可用 in 管理 设置 (`管理-设置.json`)
* Cannot be 覆盖 by 用户 or 项目 设置 (高est precedence)
* En强迫 之前 net工作/文件系统 运营 (阻塞 来源s never 执行)
* 使用s 精确 匹配 for 来源 规范s (包括 `ref`, `路径` for Git 来源s), except `h

**允许列表 behavior**:

* `未定义` (默认): No re严格ions - 用户s can 添加 任何 市场place
* 空 数组 `[]`: 完成 锁定下 - 用户s cannot 添加 任何 新 市场places
* 列表 of 来源s: 用户s can only 添加 市场places that 匹配 精确ly

**所有 支持ed 来源 类型s**:

The 允许列表 支持s mul提示le 市场place 来源 类型s. 最多 来源s 使用 精确 匹配, whi

1. **GitHub repositories**:

```json  主题={空}
{ "来源": "GitHub", "repo": "acme-corp/批准-插件s" }
{ "来源": "GitHub", "repo": "acme-corp/安全-工具", "ref": "v2.0" }
{ "来源": "GitHub", "repo": "acme-corp/插件s", "ref": "主", "路径": "市场pla
```
```

字段s: `repo` (必需), `ref` (可选: 分支/标签/SHA), `路径` (可选: sub目录)

1. **Git repositories**:

```json 主题={空}
{ "来源": "Git", "url": "https://Gitlab.示例.com/工具/插件s.Git" }
{ "来源": "Git", "url": "https://bitbucket.org/acme-corp/插件s.Git", "ref": "生产" }
{ "来源": "Git", "url": "ssh://Git@Git.示例.com/插件s.Git", "ref": "v3.1", "路径": "批准" }
```

```
字段s: `url` (必需), `ref` (可选: 分支/标签/SHA), `路径` (可选: sub目录)

3. **URL-基础d 市场places**:

```json  主题={空}
{ "来源": "url", "url": "https://插件s.示例.com/市场place.json" }
{ "来源": "url", "url": "https://cdn.示例.com/市场place.json", "页眉s": { "
```

字段s: `url` (必需), `页眉s` (可选: HTTP 页眉s for 认证 access)

URL-基础d 市场places only 下载 the `市场place.json` 文件. They do not 下载 插件 文件 from the 服务器. 插件s in URL-基础d 市场places must 使用 外部 来源s (GitHub, npm, or Git URLs) rather than relative 路径s. For 插件s with relative 路径s, 使用 a Git-基础d 市场place instead.看见 故障排除 for 详情s.

1. **NPM 包s**:

```json 主题={空}
{ "来源": "npm", "包": "@acme-corp/claude-插件s" }
{ "来源": "npm", "包": "@acme-corp/批准-市场place" }
```
```

```
字段s: `包` (必需, 支持s 范围d 包s)

5. **文件 路径s**:

```json  主题={空}
{ "来源": "文件", "路径": "/usr/本地/share/claude/acme-市场place.json" }
{ "来源": "文件", "路径": "/opt/acme-corp/插件s/市场place.json" }
```

字段s: `路径` (必需: absolute 路径 to 市场place.json 文件)

1. **目录 路径s**:

```json 主题={空}
{"来源": "目录", "路径": "/usr/本地/share/claude/acme-插件s"}
{"来源": "目录", "路径": "/opt/acme-corp/批准-市场places"}
```

```
字段s: `路径` (必需: absolute 路径 to 目录包含 `.claude-插件/市场place.json`)

7. **Host 模式 匹配**:

```json  主题={空}
{ "来源": "host模式", "host模式": "^GitHub\\.示例\\.com$" }
{ "来源": "host模式", "host模式": "^Gitlab\\.内部\\.示例\\.com$" }
```

字段s: `host模式` (必需: regex 模式 to 匹配 a收益st the 市场place host)

使用 host 模式 匹配 when you 想要 to 允许 所有 市场places from a 特定 host without枚举 每个 仓库 个人ly. This is 有用 for 组织s with 内部 GitHub 企业 or GitLab 服务器s where 开发者s 创建 their own 市场places.

Host 提取 by 来源 类型:

- `GitHub`: al方式s 匹配es a收益st `GitHub.com`
- `Git`: 提取s host名称 from the URL (支持s 机器人h HTTPS and SSH 格式s)
- `url`: 提取s host名称 from the URL
- `npm`, `文件`, `目录`: not 支持ed for host 模式 匹配

配置 示例:

示例: 允许 特定 市场places only:

```json 主题={空}
{
"严格Known市场places": [
{
"来源": "GitHub",
"repo": "acme-corp/批准-插件s"
},
{
"来源": "GitHub",
"repo": "acme-corp/安全-工具",
"ref": "v2.0"
},
{
"来源": "url",
"url": "https://插件s.示例.com/市场place.json"
},
{
"来源": "npm",
"包": "@acme-corp/合规-插件s"
}
]
}
```

> 示例 - 禁用 所有 市场place 添加itions:
>
> ```json 主题={空}
> {
>   "严格Known市场places": []
> }
> ```

示例: 允许 所有 市场places from an 内部 Git 服务器:

```json 主题={空}
{
"严格Known市场places": [
{
"来源": "host模式",
"host模式": "^GitHub\.示例\.com$"
}
]
}
```

```
**精确 匹配 要求**:

市场place 来源s must 匹配 **精确ly** for a 用户's 添加ition to be 允许. For G

* The `repo` or `url` must 匹配 精确ly
* The `ref` 字段 must 匹配 精确ly (or 机器人h be 未定义)
* The `路径` 字段 must 匹配 精确ly (or 机器人h be 未定义)

示例 of 来源s that **do NOT 匹配**:

```json  主题={空}
// These are 不同 来源s:
{ "来源": "GitHub", "repo": "acme-corp/插件s" }
{ "来源": "GitHub", "repo": "acme-corp/插件s", "ref": "主" }

// These are also 不同:
{ "来源": "GitHub", "repo": "acme-corp/插件s", "路径": "市场place" }
{ "来源": "GitHub", "repo": "acme-corp/插件s" }
```
```

比较 with `extraKnown市场places`:

A规格t	**`严格Known市场places`**	**`extraKnown市场places`**
目的	组织al 政策 enfor水泥	团队 方便
设置 文件	`管理-设置.json` only	任何 设置 文件

A规格t	`严格Known市场places`	`extraKnown市场places`
Behavior	块s non-允许列出 添加itions	Auto-安装s错过 市场places
When en强迫	之前 net工作/文件系统 运营	之后 用户 信任 及时
Can be 覆盖	No (高est precedence)	Yes (by 高er precedence 设置)
来源 格式	直接 来源 对象	名称d 市场place with nested 来源
使用 案例	合规, 安全 re严格ions	On董事会ing, 标准ization

格式 差异:

`严格Known市场places` 使用s 直接 来源 对象s:

```json 主题={空}
{
"严格Known市场places": [
{ "来源": "GitHub", "repo": "acme-corp/插件s" }
]
}
```

> `extraKnown市场places` requ怒s 名称d 市场places:
>
> ```json 主题={空}
> {
>   "extraKnown市场places": {
>     "acme-工具": {
>       "来源": { "来源": "GitHub", "repo": "acme-corp/插件s" }
>     }
>   }
> }
> ```

Using 机器人h together:

`严格Known市场places` is a 政策 gate: it 控制s what 用户s may 添加 but does not register 任何 市场places. To 机器人h re严格 and pre-register a 市场place for 所有 用户s, 设置 机器人h in `管理-设置.json`:

```json 主题={空}
{
"严格Known市场places": [
{ "来源": "GitHub", "repo": "acme-corp/插件s" }
],
"extraKnown市场places": {
"acme-工具": {
"来源": { "来源": "GitHub", "repo": "acme-corp/插件s" }
}
}
}
```

```
With only `严格Known市场places` 设置, 用户s can 静止 添加 the 允许 市场place 手

**重要 注意s**:

* Re严格ions are 检查ed 之前 任何 net工作 请求s or 文件系统 运营
* When 阻塞, 用户s看见 清楚 错误 消息s indicating the 来源 is 阻塞 by 管理 政策
* The re严格ion 应用lies only to添加 新 市场places; 之前ly 安装ed 市场places r
* 管理 设置 have the 高est precedence and cannot be 覆盖

看见 [管理 市场place re严格ions](/en/插件-市场places#管理-市场place-re严格ions)

###管理 插件s

使用 the `/插件` 命令 to manage 插件s 交互ly:

* 浏览 可用 插件s from 市场places
* 安装/卸载 插件s
* 启用/禁用 插件s
*视图 插件 详情s (命令, 代理s, 钩子 provided)
* 添加/移除 市场places

Learn 更多 about the 插件 系统 in the [插件s 文档](/en/插件s).

## 环境变量

环境变量 let you 控制 Claude 代码 behavior without 编辑ing 设置 文件. 任何 可变

看见 the [环境变量 参考](/en/env-vars) for the 满 列表.

## 工具 可用 to Claude

Claude 代码 has access to a 设置 of 工具 for 读取ing, 编辑ing, 搜索ing, 运行中

看见 the [工具 参考](/en/工具-参考) for the 满 列表 and Bash 工具 behavior 详情
```

````
## 另见

* [权限](/en/权限): 许可 系统, 规则 syn税, 工具-特定 模式s, and 管理 政策
* [认证](/en/认证): 设置 上 用户 access to Claude 代码
* [故障排除](/en/故障排除): 解决方案 for 常见 配置 问题s

---

# 故障排除

> ## 文档 索引
> 获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
> 使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.

# 故障排除

> Disc结束 解决方案 to 常见问题 with Claude 代码 安装 and 使用方法.

## 麻烦shoot 安装 问题s

  If you'd rather跳 the 终端 整个ly, the [Claude 代码 桌面 应用](/en/桌面-快速

查找 the 错误 消息 or 症状 you're看见:

| What you看见                                                  | 解决
| :------------------------------------------------------------ | :-------
| `命令 not found: claude` or `'claude' is not 认可` | [修复 your 路径](#命令
| `syn税 错误 near 意外 令牌 '

    ```bash  主题={空}
    echo $路径 | tr ':' '\n' | grep 本地/bin
    ```

    If there's no 输出, the 目录 is错过. 添加 it to your 命令行 配置:

    ```bash  主题={空}
    # Zsh (macOS 默认)
````
````

```
 echo '出口 路径="$HOME/.本地/bin:$路径"' >> ~/.zshrc
来源 ~/.zshrc

 # Bash (Linux 默认)
 echo '出口 路径="$HOME/.本地/bin:$路径"' >> ~/.bashrc
来源 ~/.bashrc
 ```


替代方案ly, 关闭 and re打开 your 终端.

验证 the 修复 工作ed:

 ```bash  主题={空}
 claude --版本
 ```



 ```权力命令行  主题={空}
 $env:路径 -split ';' | 选择-字符串 '本地\\bin'
 ```

 If there's no 输出, 添加 the 安装 目录 to your 用户 路径:

 ```权力命令行  主题={空}
 $当前路径 = [环境]::Get环境可变('路径', '用户')
 [环境]::设置环境可变('路径', "$当前路径;$env:用户PRO文件\.本地\bin", '用户')
 ```


重启 your 终端 for the 更改 to take 效果.

验证 the 修复 工作ed:

 ```权力命令行  主题={空}
 claude --版本
 ```
```
```

```batch  主题={空}
echo %路径% | 查找str /i "本地\bin"
```

If there's no 输出, 打开 系统 设置, 前往 to 环境变量, and 添加 `%用户PRO文件

验证 the 修复 工作ed:

```batch  主题={空}
claude --版本
```

检查 for conflicting 安装s

Mul提示le Claude 代码 安装s can 原因 版本 mis匹配es or 意外 behavior. 检查 what

列表 所有 `claude` binaries found in your 路径:

```bash  主题={空}
which -a claude
```

检查 whether the 原生安装er and npm 版本s are 现在:

```bash  主题={空}
ls -la ~/.本地/bin/claude
```

```bash  主题={空}
ls -la ~/.claude/本地/
```

```bash  主题={空}
```

```
    npm -g ls @anthropic-ai/claude-代码 2>/dev/空
    ```



    ```权力命令行  主题={空}
    where.exe claude
    测试-路径 "$env:本地应用数据\Claude 代码\claude.exe"
    ```


If you 查找 mul提示le 安装s, keep only one. The 原生安装 at `~/.本地/bin/clau

卸载 an npm 全球 安装:

```bash  主题={空}
npm 卸载 -g @anthropic-ai/claude-代码
```

移除 a Homebrew 安装 on macOS:

```bash 主题={空}
brew 卸载 --cask claude-代码
```

```
### 检查 目录 权限

The 安装er 需要s 写入 access to `~/.本地/bin/` and `~/.claude/`. If 安装 fai

```bash  主题={空}
测试 -w ~/.本地/bin && echo "wri表" || echo "not wri表"
测试 -w ~/.claude && echo "wri表" || echo "not wri表"
```

If either 目录 isn't wri表, 创建 the 安装 目录 and 设置 your 用户 as the 所有者:

```bash 主题={空}
sudo mkdir -p ~/.本地/bin
sudo chown -R $(whoami) ~/.本地
```
```

```
### 验证 the 二进制 工作s

If `claude` is 安装ed but crashes or hangs on 启动上, 运行 these 检查s to na

确认 the 二进制 exists and is 可执行文件:

```bash  主题={空}
ls -la $(which claude)
```

On Linux, 检查 for错过 共享 libraries. If `ldd` 显示s错过 libraries, you may 需要 to 安装 系统 包s. On Alpine Linux and other musl-基础d distributions,看见 Alpine Linux 设置.

```bash 主题={空}
ldd $(which claude) | grep "not found"
```

```
运行 a 快 sanity 检查 that the 二进制 can 执行:

```bash  主题={空}
claude --版本
```

常见 安装 问题s

These are the 最多 频繁ly encountered 安装 问题s and their 解决方案.

安装 脚本 回报s HTML instead of a 命令行 脚本

When 运行中 the 安装 命令, you may看见 one of these 错误:

``文本 主题={空} bash: 行 1: syn税 错误 near 意外 令牌'

```
On 权力命令行, the 相同 问题 应用ears as:

```文本  主题={空}
Invoke-表达:错过 论点 in 参数 列表.
```
```

This 手段 the 安装 URL 返回 an HTML 页 instead of the 安装 脚本. If the HTML 页 says "应用 不可用 in region," Claude 代码 is not 可用 in your coun尝试.看见 支持ed countries.

Other明智, this can h应用en due to net工作 问题s, 区域 r郊游, or a 临时 服务 disr上tion.

解决方案:

1. **使用 an alter原生安装 方法**:

On macOS or Linux, 安装 via Homebrew:

```
bash 主题={空} brew 安装 --cask claude-代码
```

On 风ows, 安装 via WinGet:

```
权力命令行 主题={空} WinGet 安装 Anthropic.Clau解码
```

1. **重试 之后 a 几个 微小s**: the 问题 is often 临时. 等待 and 尝试 the 原始 命令 a收益.

`命令 not found: claude` 之后 安装

The 安装 完成 but `claude` doesn't 工作. The 精确 错误 varies by 平台:

平台	错误 消息
macOS	`zsh: 命令 not found: claude`
Linux	`bash: claude: 命令 not found`
风ows CMD	`'claude' is not 认可 as an 内部 or 外部 命令`
权力命令行	`claude : The term 'claude' is not 认可 as the 名称 of a cmdlet`

This 手段 the 安装 目录 isn't in your 命令行's 搜索 路径.看见 验证 your 路径 for the 修复 on 每个 平台.

`curl: (56) 失败写作 输出 to destination`

The `curl ... | bash` 命令 下载s the 脚本 and passes it 直接ly to Bash for 执行 using a 管道 (`|`). This 错误 手段 the 连接 broke 之前 the 脚本 完成 下载ing. 常见 原因s include net工作 中断ions, the 下载 being 阻塞 mid-流, or 系统 资源 限制s.

解决方案:

1. **检查 net工作 稳定性**: Claude 代码 binaries are hosted on 前往ogle 云 Sto狂怒. 测试 that you can 范围 it:
   `bash 主题={空} curl -fsSL https://sto狂怒.前往og跳is.com -o /dev/空`
   If the 命令 完成s 沉默ly, your 连接 is 好 and the 问题 is 可能 intermittent. 重试 the 安装 命令. If you看见 an 错误, your net工作 may be 阻塞 the 下载.

2. **尝试 an alter原生安装 方法**:

On macOS or Linux:

`bash 主题={空} brew 安装 --cask claude-代码`

On 风ows:

`权力命令行 主题={空} WinGet 安装 Anthropic.Clau解码`

TLS or SSL 连接 错误

错误 像 `curl: (35) TLS connect 错误` , `s通道: 下一个 Initialize安全上下文失败` , or 权力命令行's `Could not establish 信任 relationship for the SSL/TLS 安全 通道` indicate TLS handshake 失败s.

解决方案:

1. **更新 your 系统 CA certificates**:

On Ubuntu/Debian:

`bash 主题={空} sudo apt-get 更新 && sudo apt-get 安装 ca-certificates`

On macOS via Homebrew:

```
bash 主题={空} brew 安装 ca-certificates
```

1. **On 风ows, 启用 TLS 1.2** in 权力命令行 之前 运行中 the 安装er:
   ```
   权力命令行 主题={空} [Net.服务Point经理]::安全协议 = [Net.安全协议类型]::Tls12 irm https://claude.ai/安装.ps1 | iex
   ```

2. **检查 for proxy or f怒w所有 interference**: corpo速率 proxies that per形式 TLS 审查 can 原因 these 错误,包括 `不能 to get 本地 问题r certificate`. 设置 `节点_EXTRA_CA_CERTS` to your corpo速率 CA certificate 捆绑:
   ```
   bash 主题={空} 出口 节点_EXTRA_CA_CERTS=/路径/to/corpo速率-ca.pem
   ```
   Ask your IT 团队 for the certificate 文件 if you don't have it. You can also 尝试 on a 直接 连接 to 确认 the proxy is the 原因.

`失败 to 获取 版本 from sto狂怒.前往og跳is.com`

The 安装er couldn't 范围 the 下载 服务器. This 典型ly 手段 `sto狂怒.前往og跳is.com` is 阻塞 on your net工作.

解决方案:

1. **测试 connectivity 直接ly**:
   ```
   bash 主题={空} curl -sI https://sto狂怒.前往og跳is.com
   ```

2. **If 落后 a proxy**, 设置 `HTTPS_PROXY` so the 安装er can 路由 th粗糙 it.看见 proxy 配置 for 详情s.
   ```
   bash 主题={空} 出口 HTTPS_PROXY=http://proxy.示例.com:8080 curl -fsSL https://claude.ai/安装.sh | bash
   ```

3. **If on a 受限 net工作**, 尝试 a 不同 net工作 or VPN, or 使用 an alter原生安装 方法:

On macOS or Linux:

```
bash 主题={空} brew 安装 --cask claude-代码
```

On 风ows:

```
权力命令行 主题={空} WinGet 安装 Anthropic.Clau解码
```

风ows: `irm` or `&&` not 认可

If you看见 `'irm' is not 认可` or `The 令牌 '&&' is not 有效`, you're 运行中 the 错误 命令 for your 命令行.

- **`irm` not 认可**: you're in CMD, not 权力命令行. You have two 选项:

打开 权力命令行 by 搜索ing for "权力命令行" in the 启动 menu, then 运行 the 原始 安装 命令:

```
权力命令行 主题={空} irm https://claude.ai/安装.ps1 | iex
```

Or stay in CMD and 使用 the CMD 安装er instead:

```
batch 主题={空} curl -fsSL https://claude.ai/安装.cmd -o 安装.cmd && 安装.cmd && del 安装.cmd
```

- **`&&` not 有效**: you're in 权力命令行 but ran the CMD 安装er 命令. 使用 the 权力命令行 安装er:
  `权力命令行 主题={空} irm https://claude.ai/安装.ps1 | iex`

安装 kil领导 on 低-记忆 Linux 服务器s

If you看见 `Kil领导` 期间 安装 on a VPS or 云 实例:

```文本 主题={空}
设置ting 上 Claude 代码...
安装ing Claude 代码 本地 构建 最新...
bash: 行 142: 34803 Kil领导 "$二进制_路径" 安装 ${目标:+"$目标"}
```

```
The Linux OOM killer 终止 the 流程 be原因 the 系统 ran out of 记忆. Claude 代

**解决方案:**

1. **添加 swap s步伐** if your 服务器 has 有限 RAM. Swap 使用s disk s步伐 as 

   创建 a 2 GB swap 文件 and 启用 it:

   ```bash  主题={空}
   sudo 下降ocate -l 2G /swap文件
   sudo chmod 600 /swap文件
   sudo mkswap /swap文件
   sudo swapon /swap文件
   ```

   Then 重试 the 安装:

   ```bash  主题={空}
   curl -fsSL https://claude.ai/安装.sh | bash
   ```

2. **关闭 other 流程es** to free 记忆 之前安装.

3. **使用 a larger 实例** if 可能. Claude 代码 requ怒s at least 4 GB of RAM.

安装 hangs in Docker

When安装 Claude 代码 in a Docker container,安装 as 根 into `/` can 原因 hang

解决方案:

1. **设置 a工作 目录** 之前 运行中 the 安装er. When 运行 from `/`, the 安装er扫
   ```docker文件  主题={空}
   工作DIR /tmp
   运行 curl -fsSL https://claude.ai/安装.sh | bash
```
```

```
   ```

2. **增加 Docker 记忆 限制s** if using Docker 桌面:
   ```bash  主题={空}
   docker 构建 --记忆=4g .
   ```

风ows: Claude 桌面 结束rides `claude` CLI 命令

If you 安装ed an 更旧 版本 of Claude 桌面, it may register a `Claude.exe` in

更新 Claude 桌面 to the 最新 版本 to 修复 this 问题.

风ows: "Claude 代码 on 风ows requ怒s Git-bash"

Claude 代码 on 本地 风ows 需要s [Git for 风ows](https://Git-scm.com/下载s/wi

If Git is not 安装ed, 下载 and 安装 it from [Git-scm.com/下载s/win](http

If Git is al就绪 安装ed but Claude 代码 静止 can't 查找 it, 设置 the 路径

```json  主题={空}
{
  "env": {
    "CLAUDE_代码_Git_BASH_路径": "C:\\计划 文件\\Git\\bin\\bash.exe"
  }
}
```

If your Git is 安装ed 一些where else, 查找 the 路径 by 运行中 `where.exe Git` in 权力命令行 and 使用 the `bin\bash.exe` 路径 from that 目录.

Linux: 错误 二进制 variant 安装ed (musl/glibc mis匹配)

If you看见 错误 about错过 共享 libraries 像 `libstdc++.so.6` or `libgcc_s.so.1` 之后 安装, the 安装er may have 下载ed the 错误 二进制 variant for your 系统.

```文本 主题={空}
错误 加载ing 共享 库 libstdc++.so.6: No such 文件 or 目录
```

```
This can h应用en on glibc-基础d 系统s that have musl cross-compilation 包s

**解决方案:**

1. **检查 which libc your 系统 使用s**:
   ```bash  主题={空}
   ldd /bin/ls | 负责人 -1
   ```
   If it 显示s `linux-vdso.so` or 参考文献 to `/lib/x86_64-linux-gnu/`, you

2. **If you're on glibc but 前往t the musl 二进制**, 移除 the 安装 and re安装

3. **If you're 实际ly on musl** (Alpine Linux), 安装 the 必需 包s:
   ```bash  主题={空}
   apk 添加 libgcc libstdc++ ripgrep
   ```

`非法 instruction` on Linux

If the 安装er prints `非法 instruction` instead of the OOM `Kil领导` 消息,

```文本  主题={空}
bash: 行 142: 2238232 非法 instruction    "$二进制_路径" 安装 ${目标:+"$目标"
```

解决方案:

1. **验证 your 架构**:
   `bash 主题={空} u名称 -m`
   `x86_64` 手段 64-bit Intel/AMD, `aarch64` 手段 ARM64. If the 二进制 doesn't 匹配, 文件 a GitHub 问题 with the 输出.

2. **尝试 an alter原生安装 方法** while the 架构 问题 is 解决:
   `bash 主题={空} brew 安装 --cask claude-代码`
```

`dyld: cannot 加载` on macOS

If you看见 `dyld: cannot 加载` or `中止 trap: 6` 期间 安装, the 二进制 is incompatible with your macOS 版本 or 硬件.

```文本 主题={空}
dyld: cannot 加载 'claude-2.1.42-darwin-x64' (加载 命令 0x80000034 is unknown)
中止 trap: 6
```

解决方案:

1. **检查 your macOS 版本**: Claude 代码 requ怒s macOS 13.0 or 更晚. 打开 the

2. **更新 macOS** if you're on an 更旧 版本. The 二进制 使用s 加载 命令 that 更

3. **尝试 Homebrew** as an alter原生安装 方法:
   ```bash  主题={空}
   brew 安装 --cask claude-代码
   ```

风ows 安装 问题s: 错误 in WSL

You might encounter the跟随 问题s in WSL:

OS/平台 检测 问题s: if you 接收 an 错误 期间 安装, WSL may be using 风ows

* 运行 `npm config 设置 os linux` 之前 安装
* 安装 with `npm 安装 -g @anthropic-ai/claude-代码 --强制 --no-os-检查`. Do

节点 not found 错误: if you看见 `exec: 节点: not found` when 运行中 `cla

nvm 版本 conflicts: if you have nvm 安装ed in 机器人h WSL and 风ows, yo

You can identify this 问题 by:

* 运行中 `which npm` and `which 节点` - if they point to 风ows 路径s (启动in
*体验 破碎 函数式ity 之后 switching 节点 版本s with nvm in WSL

To resolve this 问题, 修复 your Linux 路径 to en确定 the Linux 节点/npm 版本s

主要 解决: En确定 nvm is 适当ly 加载 in your 命令行

The 最多 常见 原因 is that nvm isn't 加载 in non-交互 命令行s. 添加 the跟随 to

```bash  主题={空}
# 加载 nvm if it exists
出口 NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

Or 运行 直接ly in your 当前 会话:

```bash 主题={空}
来源 ~/.nvm/nvm.sh
```

```
**替代方案: Ad公正 路径 顺序**

If nvm is 适当ly 加载 but 风ows 路径s 静止 take 优先级, you can 明确ly prep结束

```bash  主题={空}
出口 路径="$HOME/.nvm/版本s/节点/$(节点 -v)/bin:$路径"
```

A空白 disabling 风ows 路径 进口ing via `应用结束风ows路径 = 虚假` as this breaks the ability to c所有 风ows 可执行文件s from WSL. 相似ly, a空白 卸载ing 节点.js from 风ows if you 使用 it for 风ows 开发.

WSL2 沙box 设置

沙boxing is 支持ed on WSL2 but requ怒s安装 添加itional 包s. If you看见 an 错误 about 错过 `bubblewrap` or `socat` when 运行中 `/沙box`, 安装 the 依赖:

```
 ```bash  主题={空}
sudo apt-get 安装 bubblewrap socat
```



```bash  主题={空}
sudo dnf 安装 bubblewrap socat
```
```
```

WSL1 does not 支持 沙boxing. If you看见 "沙boxing requ怒s WSL2", you 需要 to 升级 to WSL2 or 运行 Claude 代码 without 沙boxing.

许可 错误 期间 安装

If the 原生安装er fails with 许可 错误, the 目标 目录 may not be wri表.看见 检查 目录 权限.

If you 之前ly 安装ed with npm and are hitting npm-特定 许可 错误, switch to the 原生安装er:

```bash 主题={空}
curl -fsSL https://claude.ai/安装.sh | bash
```

```
## 权限 and 认证

These 节s 添加ress 日志in 失败s, 令牌 问题s, and 许可 及时 behavior.

### Repeated 许可 及时s

If you 查找 yourself repeatedly批准 the 相同 命令, you can 允许 特定 工具
to 运行 without 批准 using the `/权限` 命令.看见 [权限 docs](/en/权限#manage-权

### 认证 问题s

If you're体验 认证 问题s:

1. 运行 `/标志ut` to 标志 out 完成ly
2. 关闭 Claude 代码
3. 重启 with `claude` and 完成 the 认证 流程 a收益

If the 浏览器 doesn't 打开 自动所有y 期间 日志in, press `c` to 复制 the OAuth

### OAuth 错误: 无效 代码

If you看见 `OAuth 错误: 无效 代码. Pl容易 make 确定 the 满 代码 was 复制`, the

**解决方案:**

* Press 进入 to 重试 and 完成 the 日志in 快ly 之后 the 浏览器 打开s
* 类型 `c` to 复制 the 满 URL if the 浏览器 doesn't 打开 自动所有y
* If using a 远程/SSH 会话, the 浏览器 may 打开 on the 错误 machine. 复制 the

### 403 禁止 之后 日志in

If you看见 `API 错误: 403 {"错误":{"类型":"禁止","消息":"请求 not 允许"}}` 之后

* **Claude Pro/Max 用户s**: 验证 your sub脚本ion is 活跃 at [claude.ai/设置]
* **控制台 用户s**: 确认 your 说明 has the "Claude 代码" or "开发者" 角色 分配
```

```
* **落后 a proxy**: corpo速率 proxies can interfere with API 请求s.看见 [ne

### "This 组织 has been 禁用" with an 活跃 sub脚本ion

If you看见 `API 错误: 400 ... "This 组织 has been 禁用"` despite having an

When `ANTHROPIC_API_键` is 现在 and you have 批准 it, Claude 代码 使用s that

To 使用 your sub脚本ion instead, 未设置 the 环境 可变 and 移除 it from your 命

```bash  主题={空}
未设置 ANTHROPIC_API_键
claude
```

检查 `~/.zshrc`, `~/.bashrc`, or `~/.pro文件` for 出口 `ANTHROPIC_API_键=...` 行s and 移除 them to make the 更改 永久. 运行 `/状态` 内部 Claude 代码 to 确认 which 认证 方法 is 活跃.

OAuth 日志in fails in WSL2

浏览器-基础d 日志in in WSL2 may fail if WSL can't 打开 your 风ows 浏览器. 设置 the `浏览器` 环境 可变:

```bash 主题={空}
出口 浏览器="/mnt/c/计划 文件/前往ogle/Chrome/应用程序/chrome.exe"
claude
```
```

```
Or 复制 the URL 手册ly: when the 日志in 及时 应用ears, press `c` to 复制 the

### "Not 日志ged in" or 令牌 过期

If Claude 代码 及时s you to 日志 in a收益 之后 a 会话, your OAuth 令牌 may hav

运行 `/日志in` to re-authenticate. If this h应用ens 频繁ly, 检查 that your 系

## 配置 文件 locations

Claude 代码 stores 配置 in 几个 locations:

| 文件                              | 目的
| :-------------------------------- | :-----------------------------------
| `~/.claude/设置.json`      | 用户 设置 (权限, 钩子, 模型 结束rides)
| `.claude/设置.json`        | 项目 设置 (检查ed into 来源 控制)
| `.claude/设置.本地.json` | 本地 项目 设置 (not 提交ted)
| `~/.claude.json`                | 全球 州 (主题, OAuth, MCP 服务器)
| `.mcp.json`                     | 项目 MCP 服务器 (检查ed into 来源 控制)
| `管理-mcp.json`             | [管理 MCP 服务器](/en/mcp#管理-mcp-配置)
| 管理 设置                | [管理 设置](/en/设置#设置-文件) (服务器-管理, MDM/OS

On 风ows, `~` refers to your 用户 home 目录, such as `C:\用户s\Your名称`.

For 详情s on configuring these 文件,看见 [设置](/en/设置) and [MCP](/en/mcp)

### 重置ting 配置

To 重置 Claude 代码 to 默认 设置, you can 移除 the 配置 文件:

```bash  主题={空}
# 重置 所有 用户 设置 and 州
rm ~/.claude.json
rm -rf ~/.claude/
```
```

```
# 重置 项目-特定 设置
rm -rf .claude/
rm .mcp.json
```

This will 移除 所有 your 设置, MCP 服务器 配置s, and 会话 history.

性能 and 稳定性

These 节s c结束 问题s r兴高采烈 to 资源 使用方法, 响应ness, and 搜索 behavior.

高 CPU or 记忆 使用方法

Claude 代码 is 设计ed to 工作 with 最多 开发 环境s, but may con总和e 重要 资源 when 处理中 large 代码基础s. If you're体验 性能 问题s:

1. 使用 `/紧凑` 常规ly to reduce 上下文 尺寸
2. 关闭 and 重启 Claude 代码 between 主要 任务s
3. Consider添加 large 构建 总监ies to your `.Gitignore` 文件

命令 hangs or freezes

If Claude 代码看见ms un响应:

1. Press Ctrl+C to attempt to 取消 the 当前 运营
2. If un响应, you may 需要 to 关闭 the 终端 and 重启

搜索 and 发现 问题s

If 搜索 工具, `@文件` mentions, 习俗 代理s, and 习俗 技能 aren't工作, 安装 系统 `ripgrep`:

```bash 主题={空}

# macOS (Homebrew)

brew 安装 ripgrep
```

风ows (WinGet)

WinGet 安装 BurntSushi.ripgrep.MSVC

Ubuntu/Debian

sudo apt 安装 ripgrep

Alpine Linux

apk 添加 ripgrep

Arch Linux

pacman -S ripgrep

````
Then 设置 `使用_BUILTIN_RIPGREP=0` in your [环境](/en/env-vars).

### 慢 or 不完整 搜索 结果s on WSL

Disk 读取 性能 penalties when [工作 across 文件 系统s on WSL](https://learn.m

  `/do首席技术官r` will 显示 搜索 as OK in this 案例.

**解决方案:**

1. **Submit 更多 特定 搜索es**: reduce the 数字 of 文件 搜索ed by指定 总监ies c

2. **移动 项目 to Linux 文件系统**: if 可能, en确定 your 项目 is located on th

3. **使用 本地 风ows instead**: consider 运行中 Claude 代码 本地ly on 风ows in

## IDE 集成 问题s

If Claude 代码 does not connect to your IDE or behaves 意外ly 在...内 an ID

### JetB雨s IDE not detected on WSL2

If you're using Claude 代码 on WSL2 with JetB雨s IDEs and getting "No 可用

#### WSL2 net工作 模式s

WSL2 使用s NAT net工作 by 默认, which can pr事件 IDE 检测. You have two 选项:

**选项 1: Con图 风ows F怒w所有** (推荐)

1. 查找 your WSL2 IP 添加ress:
   ```bash  主题={空}
   wsl host名称 -I
   # 示例 输出: 172.21.123.45
   ```
````

```
2. 打开 权力命令行 as 管理员 and 创建 a f怒w所有 规则:
   ```权力命令行  主题={空}
   新-NetF怒w所有规则 -显示名称 "允许 WSL2 内部 Traffic" -指导 Inbound -协议 TCP
   ```
   Ad公正 the IP 范围 基础d on your WSL2 subnet from步骤 1.

3. 重启 机器人h your IDE and Claude 代码

选项 2: Switch to 镜像 net工作

添加 to `.wslconfig` in your 风ows 用户 目录:

```ini  主题={空}
[wsl2]
net工作模式=镜像
```

Then 重启 WSL with `wsl --关闭` from 权力命令行.

These net工作 问题s only affect WSL2. WSL1 使用s the host's net工作 直接ly and doesn't requ怒 these 配置s.

For 添加itional JetB雨s 配置 提示,看见 the JetB雨s IDE 指南.

报告 风ows IDE 集成 问题s

If you're体验 IDE 集成 问题s on 风ows, 创建 an 问题 with the跟随 信息:

- 环境 类型: 本地 风ows (Git Bash) or WSL1/WSL2
- WSL net工作 模式, if 应用lic能够: NAT or 镜像
- IDE 名称 and 版本
- Claude 代码 扩展/插件 版本
- 命令行 类型: Bash, Zsh, 权力命令行, etc.

Escape 键 not工作 in JetB雨s IDE 终端s

If you're using Claude 代码 in JetB雨s 终端s and the `Esc` 键 doesn't 中断 the 代理 as 预期, this is 可能 due to a 键binding clash with JetB雨s' 默认 短剪切s.
```

To 修复 this 问题:

1. 前往 to 设置 → 工具 → 终端
2. Either:
3. 取消检查 "移动 焦点 to the 编辑or with Escape", or
4. Click "Con图 终端 键bindings" and 删除 the "Switch 焦点 to 编辑or" 短剪切
5. 应用ly the 更改s

This 允许s the `Esc` 键 to 适当ly 中断 Claude 代码 运营.

Mark下 格式ting 问题s

Claude 代码 一些时间s gene速率s mark下 文件 with错过 language 标签s on 代码 fences, which can affect syn税 高轻ing and 可读性 in GitHub, 编辑ors, and 文档 工具.

错过 language 标签s in 代码 块s

If you 通知 代码 块s 像 this in gene速率d mark下:

````
```mark下 主题={空}
功能 示例() {
  回报 "hello";
}
```
````

````
Instead of 适当ly 标记 块s 像:

```mark下  主题={空}
```java脚本
功能 示例() {
  回报 "hello";
}
```
````
````

解决方案:

1. **Ask Claude to 添加 language 标签s**: 请求 "添加 恰当 language 标签s to 所有 代码 块s in this mark下 文件."
2. **使用 post-处理中 钩子**: 设置 上 自动 格式ting 钩子 to detect and 添加错过 language 标签s.看见 Auto-格式 代码 之后 编辑s for an 示例 of a 工具使用后 格式ting hook.
3. **手册 验证**: 之后生成 mark下 文件, re视图 them for 适当 代码 块 格式ting and 请求 正确ions if 需要ed.

In一致 spacing and 格式ting

If gene速率d mark下 has 过度 空白 行s or in一致 spacing:

解决方案:

1. **请求 格式ting 正确ions**: ask Claude to "修复 spacing and 格式ting 问题s in this mark下 文件."
2. **使用 格式ting 工具**: 设置 上 钩子 to 运行 mark下 for事情s 像 `prettier` or 习俗 格式ting 脚本s on gene速率d mark下 文件.
3. **规格ify 格式ting 偏好设置**: include 格式ting 要求 in your 及时s or 项目 记忆 文件.

Reduce mark下 格式ting 问题s

To minimize 格式ting 问题s:

- **Be 明确 in 请求s**: ask for "适当ly 格式ted mark下 with language-标记 代码 块s"
- **使用 项目 惯例**: document your preferred mark下 style in `CLAUDE.md`
- **设置 上 验证 钩子**: 使用 post-处理中 钩子 to 自动所有y 验证 and 修复 常见 格式ting 问题s

Get 更多 帮助

If you're体验 问题s not c结束ed here:

1. 使用 the `/反馈` 命令 在...内 Claude 代码 to 报告 问题s 直接ly to Anthropic
2. 检查 the GitHub 仓库 for 已知问题

3. 运行 `/do首席技术官r` to diagnose 问题s. It 检查s:
4. 安装 类型, 版本, and 搜索 函数式ity
5. Auto-更新 状态 and 可用 版本s
6. 无效 设置 文件 (mal形式ed JSON, 错误 类型s)
7. MCP 服务器 配置 错误
8. 键binding 配置 问题s
9. 上下文 使用方法警告s (large CLAUDE.md 文件, 高 MCP 令牌 使用方法, 达不到 许可规则)
10. 插件 and 代理 加载ing 错误
11. Ask Claude 直接ly about its 能力 and features - Claude has built-in access to its 文档

CLI 参考

> ## 文档 索引
>
> 获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
> 使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.

CLI 参考

> 完成 参考 for Claude 代码 命令-行 接口,包括 命令 and 标志.

CLI 命令

You can 启动 会话s, 管道 满意, 恢复 对话s, and manage 更新s with these 命令:

命令	描述	示例
`claude`	启动 交互 会话	`claude`

命令	描述	示例
`claude "查询"`	启动 交互 会话 with initial 及时	`claude "ex简单 this 项目"`
`claude -p "查询"`	查询 via SDK, then 退出	`claude -p "ex简单 this 功能"`
`cat 文件 | claude -p "查询"`	流程 管道d 满意	`cat 日志s.txt | claude -p "ex简单"`
`claude -c`	继续 最多 最近 对话 in 当前 目录	`claude -c`
`claude -c -p "查询"`	继续 via SDK	`claude -c -p "检查 for 类型 错误"`
`claude -r "" "查询"`	恢复 会话 by ID or 名称	`claude -r "auth-re事实or" "Finish this PR"`
`claude 更新`	更新 to 最新 版本	`claude 更新`
`claude auth 日志 in`	标志 in to your Anthropic 说明. 使用 `--email` to pre-fill your email 添加ress, `--sso` to 强制 SSO 认证, and `--控制台` to 标志 in with Anthropic 控制台 for API 使用方法 计费 instead of a Claude 订阅	`claude auth 日志 in --控制台`
`claude auth 标志 ut`	日志 out from your Anthropic 说明	`claude auth 标志 ut`
`claude auth 状态`	显示 认证 状态 as JSON. 使用 `--文本` for 人类-读取能够 输出. 退出s with 代码 0 if 日志ged in, 1 if not	`claude auth 状态`

命令	描述	示例
`claude 代理s`	列表 所有 con图d 子代理, 分组 by 来源	`claude 代理s`
`claude auto-模式 默认s`	Print the built-in auto 模式 阶级ifier 规则 as JSON. 使用 `claude auto-模式 config` to看见 your 有效 config with 设置 应用	`claude auto-模式 默认s > 规则.json`
`claude mcp`	Con图 模型上下文协议 (MCP) 服务器s	看见 the Claude 代码 MCP 文档.
`claude 插件`	Manage Claude 代码 插件s. Alias: `claude 插件s` .看见 插件 参考 for sub命令	`claude 插件 安装 代码-re视图 @claude-插件s-official`
`claude 远程-控制`	启动 a 远程 控制 服务器 to 控制 Claude 代码 from Claude.ai or the Claude 应用. 运行s in 服务器 模式 (no 本地 交互 会话).看见 服务器 模式 标志	`claude 远程-控制 --名称 "My 项目"`

CLI 标志

习俗ize Claude 代码's behavior with these 命令-行 标志. `claude --帮助` does not 列表 每个 flag, so a flag's absence from `--帮助` does not mean it is 不可用.

Flag	描述	示例
`--添加-dir`	添加 添加itional工作 总监ies for Claude to 读取 and 编辑 文件. 授予s 文件 access; 最多 `.claude/` 配置 is not disc结束ed from these 总监ies. 验证s 每个 路径 exists as a 目录	`claude --添加-dir ../应用s ../lib`
`--代理`	规格ify an 代理 for the 当前 会话 (结束rides the `代理` 设置)	`claude --代理 my-习俗-代理`

Flag	描述	示例
`--代理s`	De好 习俗 子代理 动态所有y via JSON. 使用s the 相同 字段名称s as sub代理 front事情, plus a `及时` 字段 for the 代理's 说明	`claude --代理s '{"审查员":{"描述":"Re视图s 代码","及时":"You are a 代码 审查员"}}'`
`--允许-d愤怒ously-跳-权限`	添加 `bypass权限` to the `Shift+Tab` 模式 cycle without 启动ing in it. Lets you begin in a 不同 模式 像 `计划` and switch to `bypass权限` 更晚.看见 许可 模式s	`claude --许可-模式 计划 --允许-d愤怒ously-跳-权限`
`--允许工具`	工具 that 执行 without 及时ing for 许可.看见 许可 规则 syn税 for 模式 匹配. To re严格 which 工具 are 可用, 使用 `--工具` instead	`"Bash(Git 日志 *)" "Bash(Git diff *)" "读取"`
`--应用结束-系统-及时`	应用结束 习俗 文本 to the 结束 of the 默认 系统 及时	`claude --应用结束-系统-及时 "Al方式s 使用 类型脚本"`
`--应用结束-系统-及时-文件`	加载 添加itional 系统 及时 文本 from a 文件 and 应用结束 to the 默认 及时	`claude --应用结束-系统-及时-文件 ./extra-规则.txt`
`--赤裸`	最小 模式:跳 auto-发现 of 钩子, 技能, 插件s, MCP 服务器, auto 记忆, and CLAUDE.md so 脚本ed c所有s 启动 快er. Claude has access to Bash, 文件 读取, and 文件 编辑 工具. 设置s `CLAUDE_代码_简单` .看见 赤裸模式	`claude --赤裸 -p "查询"`
`--betas`	Beta 页眉s to include in API 请求s (API 键 用户s only)	

Flag	描述	示例
		`claude --betas interleaved-薄king`
`--通道s`	(研究 pre视图) MCP 服务器 whose 通道 通知s Claude should 列表en for in this 会话. S步伐-分离 列表 of `插件:@` entries. Requ怒s Claude.ai 认证	`claude --通道s 插件:my-notifier@my-市场place`
`--chrome`	启用 Chrome 浏览器 集成 for 网页 automation and 测试	`claude --chrome`
`--继续`, `-c`	加载 the 最多 最近 对话 in the 当前 目录	`claude --继续`
`--d愤怒ously-加载-开发-通道s`	启用 通道s that are not on the 批准 允许列表, for 本地 开发. 接受s `插件:@` and `服务器:` entries. 及时s for 确认	`claude --d愤怒ously-加载-开发-通道s 服务器:网页hook`
`--d愤怒ously-跳-权限`	跳 许可 及时s. 等价 to `--许可-模式 bypass权限`. 看见 许可 模式s for what this does and does not跳	`claude --d愤怒ously-跳-权限`
`--调试`	启用 调试 模式 with 可选 类别 过滤 (for 示例, `"api,钩子"` or `"!statsig,!文件"`)	`claude --调试 "api,mcp"`
`--调试-文件`	写入 调试 日志s to a 特定 文件 路径. 隐含ly 启用s 调试 模式. Takes precedence 结束 `CLAUDE_代码_调试_日志S_DIR`	`claude --调试-文件 /tmp/claude-调试.日志`
	禁用 所有 技能 and 命令 for this 会话	`claude --禁用-slash-命令`

Flag	描述	示例
`--禁用-slash-命令`		
`--dis允许工具`	工具 that are 移除 from the 模型's 上下文 and cannot be 使用d	`"Bash(Git 日志 *)" "Bash(Git diff *)" "编辑"`
`--effort`	设置 the effort 级别 for the 当前 会话. 选项: `低`, `中`, `高`, `max` (Opus 4.6 only). 会话-范围d and does not persist to 设置	`claude --effort 高`
`--下降返回-模型`	启用 自动 下降返回 to 指定 模型 when 默认 模型 is 结束加载 (print 模式 only)	`claude -p --下降返回-模型 sonnet "查询"`
`--分叉-会话`	When re总和ing, 创建 a 新 会话 ID instead of reusing the 原始 (使用 with `--恢复` or `--继续`)	`claude --恢复 abc123 --分叉-会话`
`--from-pr`	恢复 会话s 链接 to a 特定 GitHub PR. 接受s a PR 数字 or URL. 会话s are 自动所有y 链接 when 创建d via `gh pr 创建`	`claude --from-pr 123`
`--ide`	自动所有y connect to IDE on 启动上 if 精确ly one 有效 IDE is 可用	`claude --ide`
`--init`	运行 initialization 钩子 and 启动 交互 模式	`claude --init`
`--init-only`	运行 initialization 钩子 and 退出 (no 交互 会话)	`claude --init-only`

Flag	描述	示例
`--include-hook-事件s`	Include 所有 hook lifecycle 事件s in the 输出 流. Requ怒s `--输出-格式 流-json`	`claude -p --输出-格式 流-json --include-hook-事件s "查询"`
`--include-偏袒-消息s`	Include 偏袒 流ing 事件s in 输出. Requ怒s `--print` and `--输出-格式 流-json`	`claude -p --输出-格式 流-json --include-偏袒-消息s "查询"`
`--输入-格式`	规格ify 输入 格式 for print 模式 (选项: `文本`, `流-json`)	`claude -p --输出-格式 json --输入-格式 流-json`
`--json-模式`	Get 验证 JSON 输出 匹配 a JSON 模式 之后 代理 完成s its 工作流 (print 模式 only,看见 结构化 输出s)	`claude -p --json-模式 '{"类型":"对象","适当ties":{...}}' "查询"`
`--主tenance`	运行 主tenance 钩子 and 启动 交互 模式	`claude --主tenance`
`--max-预算-usd`	最大 dollar 数量 to sp结束 on API c所有s 之前停止 (print 模式 only)	`claude -p --max-预算-usd 5.00 "查询"`
`--max-turns`	限制 the 数字 of 代理ic turns (print 模式 only). 退出s with an 错误 when the 限制 is 范围ed. No 限制 by 默认	`claude -p --max-turns 3 "查询"`
`--mcp-config`	加载 MCP 服务器 from JSON 文件 or 字符串s (s步伐-分离)	`claude --mcp-config ./mcp.json`
`--模型`	设置s the 模型 for the 当前 会话 with an alias for the 最新 模型 (`sonnet` or `opus`) or a 模型's 满 名称	`claude --模型 claude-sonnet-4-6`
`--名称`, `-n`	设置 a 显示 名称 for the 会话, 显示n in `/恢复` and the 终端	

Flag	描述	示例
	标题. You can 恢复 a 名称d 会话 with `claude --恢复` . `/重命名` 更改s the 名称 mid-会话 and also 显示s it on the 及时 bar	`claude -n "my-feature-工作"`
`--no-chrome`	禁用 Chrome 浏览器 集成 for this 会话	`claude --no-chrome`
`--no-会话-persistence`	禁用 会话 persistence so 会话s are not 保存 to disk and cannot be 恢复d (print 模式 only)	`claude -p --no-会话-persistence "查询"`
`--输出-格式`	规格ify 输出 格式 for print 模式 (选项: `文本` , `json` , `流-json`)	`claude -p "查询" --输出-格式 json`
`--启用-auto-模式`	解锁 auto 模式 in the `Shift+Tab` cycle. Requ怒s a 团队, 企业, or API 计划 and Claude Sonnet 4.6 or Opus 4.6	`claude --启用-auto-模式`
`--许可-模式`	Begin in a 指定 许可 模式. 接受s `默认` , `接受its` , `计划` , `auto` , `don任务` , or `bypass权限` . 结束rides `默认模式` from 设置 文件	`claude --许可-模式 计划`
`--许可-及时-工具`	规格ify an MCP 工具 to handle 许可 及时s in non-交互 模式	`claude -p --许可-及时-工具 mcp_auth_工具 "查询"`
`--插件-dir`	加载 插件s from a 目录 for this 会话 only. 每个 flag takes one 路径. Repeat the flag for mul 提示le 总监ies:	`claude --插件-dir ./my-插件s`

Flag	描述	示例
	`--插件-dir A --插件-dir B`	
`--print`, `-p`	Print 响应 without 交互 模式 (看见 代理 SDK 文档 for 计划 matic 使用方法 详情s)	`claude -p "查询"`
`--远程`	创建 a 新 网页 会话 on claude.ai with the provided 任务 描述	`claude --远程 "修复 the 日志in 缺陷"`
`--远程-控制`, `--rc`	启动 an 交互 会话 with 远程 控制 启用 so you can also 控制 it from claude.ai or the Claude 应用. 可选ly pass a 名称 for the 会话	`claude --远程-控制 "My 项目"`
`--replay-用户-消息s`	Re-emit 用户 消息s from stdin 返回 on stdout for 确认. Requ 怒s `--输入-格式 流-json` and `--输出-格式 流-json`	`claude -p --输入-格式 流-json --输出-格式 流-json --replay-用户-消息s`
`--恢复`, `-r`	恢复 a 特定 会话 by ID or 名称, or 显示 an 交互 picker to choose a 会话	`claude --恢复 auth-re事实or`
`--会话-id`	使用 a 特定 会话 ID for the 对话 (must be a 有效 UUID)	`claude --会话-id "550e8400-e29b-41d4-a716-446655440000"`
`--设置ting-来源s`	Comma-分离 列表 of设置 来源s to 加载 (`用户`, `项目`, `本地`)	`claude --设置ting-来源s 用户,项目`
`--设置`	路径 to a 设置 JSON 文件 or a JSON 字符串 to 加载 添加itional 设置 from	`claude --设置 ./设置.json`

Flag	描述	示例
`--严格-mcp-config`	Only 使用 MCP 服务器 from `--mcp-config`, ignoring 所有 other MCP 配置s	`claude --严格-mcp-config --mcp-config ./mcp.json`
`--系统-及时`	替换 the 整个 系统 及时 with 习俗 文本	`claude --系统-及时 "You are a Python 专家"`
`--系统-及时-文件`	加载 系统 及时 from a 文件, replacing the 默认 及时	`claude --系统-及时-文件 ./习俗-及时.txt`
`--tele端口`	恢复 a 网页 会话 in your 本地 终端	`claude --tele端口`
`--团队mate-模式`	设置 how 代理 团队 团队mates 显示: `auto` (默认), `in-流程`, or `tmux`.看见 Choose a 显示 模式	`claude --团队mate-模式 in-流程`
`--tmux`	创建 a tmux 会话 for the 工作树. Requ怒s `--工作树`. 使用s iTerm2 本地 panes when 可用; pass `--tmux=经典` for 传统 tmux	`claude -w feature-auth --tmux`
`--工具`	Re严格 which built-in 工具 Claude can 使用. 使用 `""` to 禁用 所有, `"默认"` for 所有, or 工具 名称s 像 `"Bash,编辑,读取"`	`claude --工具 "Bash,编辑,读取"`
`--详细`	启用 详细日志, 显示s 满 turn-by-turn 输出	`claude --详细`
`--版本`, `-v`	输出 the 版本 数字	`claude -v`
`--工作树`, `-w`	启动 Claude in an iso晚d Git 工作树 at `/.claude/工作树/`. If no 名	`claude -w feature-auth`

Flag	描述	示例
	称 is given, one is auto-gene速率d	

系统 及时 标志

Claude 代码 provides four 标志 for 习俗izing the 系统 及时. 所有 four 工作 in 机器人h 交互 and non-交互 模式s.

Flag	Behavior	示例
`--系统-及时`	替换s the 整个 默认 及时	`claude --系统-及时 "You are a Python 专家"`
`--系统-及时-文件`	替换s with 文件 满意s	`claude --系统-及时-文件 ./及时s/re视图.txt`
`--应用结束-系统-及时`	应用结束s to the 默认 及时	`claude --应用结束-系统-及时 "Al方式s 使用 类型脚本"`
`--应用结束-系统-及时-文件`	应用结束s 文件 满意s to the 默认 及时	`claude --应用结束-系统-及时-文件 ./style-规则.txt`

`--系统-及时` and `--系统-及时-文件` are 互相ly exclusive. The 应用结束 标志 can be 组合 with either 替换ment flag.

For 最多 使用 案例s, 使用 an 应用结束 flag. Ap待处理 preserves Claude 代码's built-in 能力 while添加 your 要求. 使用 a 替换ment flag only when you 需要 完成 控制 结束 the 系统 及时.

另见

- Chrome 扩展 - 浏览器 automation and 网页 测试
- 交互 模式 - 短剪切s, 输入 模式s, and 交互 features
- 快速入门 指南 - 入门 with Claude 代码
- 通用工作流 - 先进 工作流s and 模式s
- 设置 - 配置 选项
- 代理 SDK 文档 - 计划matic 使用方法 and 集成s

技能扩展

> **文档 索引**
>
> 获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
> 使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.

使用技能扩展 Claude

> 创建, manage, and 分享技能 to ext结束 Claude's 能力 in Claude 代码. Includes 习俗 命令 and 捆绑d 技能.

技能 ext结束 what Claude can do. 创建 a `技能.md` 文件 with 说明, and Claude 添加s it to its 工具包. Claude 使用s 技能 when 相关, or you can invoke one 直接ly with `/技能-名称`.

For built-in 命令 像 `/帮助` and `/紧凑`,看见 the built-in 命令 参考.

习俗 命令 have been 合并 into 技能. A 文件 at `.claude/命令/部署.md` and a 技能 at `.claude/技能/部署/技能.md` 机器人h 创建 `/部署` and 工作 the 相同 方式. Your 现有 `.claude/命令/` 文件 keep工作. 技能 添加 可选 features: a 目录 for 支持ing 文件, front事情 to 控制 whether you or Claude invokes them, and the ability for Claude to 加载 them 自动所有y when 相关.

Claude 代码 技能 fol低 the 代理 技能 打开 标准, which 工作s across mul提示le AI 工具. Claude 代码 ext结束s the 标准 with 添加itional features 像 invocation 控制, sub代理 执行, and 动态 上下文 injection.

捆绑d 技能

捆绑d 技能 ship with Claude 代码 and are 可用 in 每个 会话. 不像 built-in 命令, which 执行 固定 日志ic 直接ly, 捆绑d 技能 are 及时-基础d: they give Claude a 详情ed playbook and let it orchest速率 the 工作 using its 工具. This 手段 捆绑d 技能 can spawn 平行 代理s, 读取 文件, and adapt to your 代码基础.

You invoke 捆绑d 技能 the 相同 方式 as 任何 other 技能: 类型 `/` foll欠 by the 技能 名称. In the 表 be低, `` ` indicates a 必需 论点 and [arg]` `` indicates an 可选 one.

技能	目的
`/batch`	Orchest速率 large-scale 更改s across a 代码基础 in 平行.研究es the 代码基础,分解s the 工作 into 5 to 30 独立 单位s, and 现在s a 计划. Once 批准, spawns one 背景 代理 per 单位 in an iso晚d Git 工作树. 每个 代理 实现s its 单位, 运行s 测试s, and 打开s a 拉取请求. Requ怒s a Git 仓库. 示例: `/batch mig速率 src/ from 固体 to React`
`/claude-api`	加载 Claude API 参考 物质 for your 项目's language (Python, 类型脚本, Java, 前往, Ruby, C#, pH值P, or cURL) and 代理 SDK 参考 for Python and 类型脚本. C结束s 工具 使用, 流ing, batches, 结构化 输出s, and 常见 pit下降s. Also 激活s 自动所有y when your 代码 进口s `anthropic`, `@anthropic-ai/sdk`, or `claude_代理_sdk`
`/调试 [描述]`	启用 调试日志 for the 当前 会话 and 麻烦shoot 问题s by 读取ing the 会话 调试 日志. 调试日志 is off by 默认 un较少 you 开始 with `claude --调试`, so 运行中 `/调试` mid-会话 启动s capturing 日志s from that point 前进. 可选ly describe the 问题 to 焦点 the 分析
`/loop [间隔]`	运行 a 及时 repeatedly on an 间隔 while the 会话 stays 打开. 有用 for polling a 部署ment, baby坐 a PR, or 期间ic所有y re-运行中 another 技能. 示例: `/loop 5m 检查 if the 部署 完成`.看见 运行 及时s on a 时间表
`/simplify [焦点]`	Re视图 your 最近ly 更改d 文件 for 代码 re使用, quality, and efficiency 问题s, then 修复 them. Spawns three re视图 代理s in 平行, aggregates their发现s, and 应用lies 修复es. Pass 文本 to 焦点 on 特定 关心s: `/simplify 焦点 on 记忆 efficiency`

入门

创建 your 第一个 技能

This 示例 创建s a 技能 that t每个es Claude to ex简单 代码 using visual 图表s and ana日志ies. Since it 使用s 默认 front事情, Claude can 加载 it 自动所有y when you ask how 一些薄g 工作s, or you can invoke it 直接ly with `/ex简单-代码`.

创建 a 目录 for the 技能 in your 个人 技能 文件夹. 个人 技能 are 可用 across 所有

```bash  主题={空}
mkdir -p ~/.claude/技能/ex简单-代码
```

每个 技能 需要s a `技能.md` 文件 with two 部分s: YAML front事情 (between `---`

创建 `~/.claude/技能/ex简单-代码/技能.md`:

```yaml  主题={空}
---
名称: ex简单-代码
描述: Ex简单s 代码 with visual 图表s and ana日志ies. 使用 when ex简单ing how
---

When ex简单ing 代码, al方式s include:

1. **启动 with an ana日志y**: 比较 the 代码 to 一些薄g from 日常 life
2. **Draw a 图表**: 使用 ASCII 艺术 to 显示 the f低, 结构, or relationships
3. **走 th粗糙 the 代码**: Ex简单步骤-by-步骤 what h应用ens
4. **高轻 a 前往tcha**: What's a 常见 薄雾ake or mis概念ion?

Keep 解释s 对话al. For 复杂 概念s, 使用 mul提示le ana日志ies.
```

You can 测试 it two 方式s:

Let Claude invoke it 自动所有y by询问 一些薄g that 匹配es the 描述:

```文本  主题={空}
How does this 代码 工作?
```

```
```

Or invoke it 直接ly with the 技能 名称:

```文本  主题={空}
/ex简单-代码 src/auth/日志in.ts
```

Either 方式, Claude should include an ana日志y and ASCII 图表 in its 解释.
```

## Where 技能 live

Where you store a 技能 determines who can 使用 it:

| Location | 路径 | 应用lies to |
| --- | --- | --- |
| 企业 | 看见 管理 设置 | 所有 用户s in your 组织 |
| 个人 | `~/.claude/技能//技能.md` | 所有 your 项目s |
| 项目 | `.claude/技能//技能.md` | This 项目 only |
| 插件 | `/技能//技能.md` | Where 插件 is 启用 |

When 技能 share the 相同 名称 across 级别s, 高er-优先级 locations win: 企业 > 个人 > 项目. 插件 技能 使用 a `插件-名称:技能-名称` 名称s步伐, so they cannot conflict with other 级别s. If you have 文件 in `.claude/命令/`, those 工作 the 相同 方式, but if a 技能 and a 命令 share the 相同 名称, the 技能 takes precedence.

### 自动 发现 from nested 总监ies

When you 工作 with 文件 in sub总监ies, Claude 代码 自动所有y disc结束s 技能 from nested `.claude/技能/` 总监ies. For 示例, if you're 编辑ing a 文件 in `包s/front结束/`, Claude 代码 also看s for 技能 in `包s/front结束/.claude/技能/`. This 支持s monorepo 设置s where 包s have their own 技能.

每个 技能 is a 目录 with `技能.md` as the 条目point:
```

```文本 主题={空}
my-技能/
├── 技能.md # 主 说明 (必需)
├── 模板.md # 模板 for Claude to fill in
├── 示例/
│   └── 样本.md # 示例 输出显示 预期 格式
└── 脚本s/
└── 验证.sh # 脚本 Claude can 执行
```

```
The `技能.md` contains the 主 说明 and is 必需. Other 文件 are 可选 and let y

  文件 in `.claude/命令/` 静止 工作 and 支持 the 相同 [front事情](#front事情-参

#### 技能 from 添加itional 总监ies

The `--添加-dir` flag [授予s 文件 access](/en/权限#添加itional-总监ies-授予-文

Other `.claude/` 配置 such as 子代理, 命令, and 输出 styles is not 加载 from

  CLAUDE.md 文件 from `--添加-dir` 总监ies are not 加载 by 默认. To 加载 them

## Con图 技能

技能 are con图d th粗糙 YAML front事情 at the 顶部 of `技能.md` and the mark下

### 类型s of 技能 满意

技能 文件 can contain 任何 说明, but思考 about how you 想要 to invoke them 帮助

**参考 满意** 添加s 知识 Claude 应用lies to your 当前 工作. 惯例, 模式s, style

```yaml  主题={空}
---
名称: api-惯例
描述: API 设计模式s for this 代码基础
---

When写作 API 结束points:
- 使用 安静 naming 惯例
- 回报 一致 错误 格式s
- Include 请求 验证
```

任务 满意 gives Claude步骤-by-步骤 说明 for a 特定 行动, 像 部署ments, 提交s, or 代码生成. These are often 行动s you 想要 to invoke 直接ly with `/技能-名称` rather than让
```

Claude decide when to 运行 them. 添加 `禁用-模型-invocation: 真实` to pr事件 Claude from triggering it 自动所有y.

```yaml 主题={空}

名称: 部署
描述: 部署 the 应用程序 to 生产
上下文: 分叉
禁用-模型-invocation: 真实

部署 the 应用程序:
1. 运行 the 测试 suite
2. 构建 the 应用程序
3. 推送 to the 部署ment 目标

```
Your `技能.md` can contain 任何薄g, but思考 th粗糙 how you 想要 the 技能 invol

### Front事情 参考

超出 the mark下 满意, you can con图 技能 behavior using YAML front事情 字段s

```yaml  主题={空}
---
名称: my-技能
描述: What this 技能 does
禁用-模型-invocation: 真实
允许-工具: 读取 Grep
---

Your 技能 说明 here...
```

所有 字段s are 可选. Only `描述` is 推荐 so Claude knows when to 使用 the 技能.
```

字段	必需	描述
`名称`	No	显示 名称 for the 技能. If omitted, 使用s the 目录 名称. 低er 案例 letters, 数字s, and hypH值ens only (max 64 字符s).
`描述`	推荐	What the 技能 does and when to 使用 it. Claude 使用s this to decide when to 应用ly the 技能. If omitted, 使用s the 第一个 段落 of mark下 满意. Front-加载 the 键 使用 案例: 描述s 长er than 250 字符s are t运行cated in the 技能列表 to reduce 上下文 使用方法.
`论点-暗示`	No	暗示 显示n 期间 auto完成 to indicate 预期 参数. 示例: `[问题-数字]` or `[文件名称] [格式]`.
`禁用-模型-invocation`	No	设置 to `真实` to pr事件 Claude from 自动所有y 加载ing this 技能. 使用 for 工作流s you 想要 to trigger 手册ly with `/名称`. 默认: `虚假`.
`用户-invoc能够`	No	设置 to `虚假` to 隐藏 from the `/` menu. 使用 for 背景 知识 用户s shouldn't invoke 直接ly. 默认: `真实`.
`允许-工具`	No	工具 Claude can 使用 without询问 许可 when this 技能 is 活跃. 接受s a s步伐-分离 字符串 or a YAML 列表.
`模型`	No	模型 to 使用 when this 技能 is 活跃.
`effort`	No	Effort 级别 when this 技能 is 活跃. 结束rides the 会话 effort 级别. 默认: inherits from 会话. 选项: `低`, `中`, `高`, `max` (Opus 4.6 only).
`上下文`	No	设置 to `分叉` to 运行 in a 分叉ed sub代理 上下文.
`代理`	No	Which sub代理 类型 to 使用 when `上下文: 分叉` is 设置.
`钩子`	No	钩子 范围d to this 技能's lifecycle.看见 钩子 in 技能 and 代理s for 配置 格式.
`路径s`	No	Glob 模式s that 限制 when this 技能 is 激活d. 接受s a comma-分离 字符串 or a YAML 列表. When 设置, Claude 加

字段	必需	描述
		载s the 技能 自动所有y only when工作 with 文件 匹配 the 模式s. 使用s the 相同 格式 as 路径-特定 规则.
`命令行`	No	命令行 to 使用 for `!`命令`` 块s in this 技能. 接受s `bash` (默认) or `权力命令行` .设置 `权力命令行` 运行s in行 命令行 命令 via 权力命令行 on 风ows. Requ怒s `CLAUDE_代码_使用_权力命令行_工具=1` .

可用 字符串 substitutions

技能 支持 字符串 substitution for 动态 值s in the 技能 满意:

可变	描述
`$参数`	所有 参数 passed when invoking the 技能. If `$参数` is not 现在 in the 满意, 参数 are 应用结束 as `参数:` .
`$参数[N]`	Access a 特定 论点 by 0-基础d 索引, such as `$参数[0]` for the 第一个 论点.
`$N`	短hand for `$参数[N]` , such as `$0` for the 第一个 论点 or `$1` for the second.
`${CLAUDE_会话_ID}`	The 当前 会话 ID. 有用 for日志,创建 会话-特定 文件, or关联 技能 输出 with 会话s.
`${CLAUDE_技能_DIR}`	The 目录包含 the 技能's `技能.md` 文件. For 插件 技能, this is the 技能's sub目录 在...内 the 插件, not the 插件 根. 使用 this in bash injection 命令 to 参考 脚本s or 文件 捆绑d with the 技能, 注意较少 of the 当前工作 目录.

示例 using substitutions:

```yaml 主题={空}

名称: 会话-日志ger
描述: 日志 活动 for this 会话

日志 the跟随 to 日志s/${CLAUDE_会话_ID}.日志:

$参数

```
### 添加 支持ing 文件

技能 can include mul提示le 文件 in their 目录. This keeps `技能.md` 焦点ed on

```文本  主题={空}
my-技能/
├── 技能.md (必需 - 概述 and navigation)
├── 参考.md (详情ed API docs - 加载 when 需要ed)
├── 示例.md (使用方法 示例 - 加载 when 需要ed)
└── 脚本s/
    └── 帮助er.py (实用 脚本 - 执行d, not 加载)
```

参考 支持ing 文件 from `技能.md` so Claude knows what 每个 文件 contains and when to 加载 it:

```mark下 主题={空}

## 添加itional 资源

- For 完成 API 详情s,看见 参考.md
- For 使用方法 示例,看见 示例.md

```
Keep `技能.md` under 500 行s. 移动 详情ed 参考 物质 to 单独 文件.

控制 who invokes a 技能

By 默认, 机器人h you and Claude can invoke 任何 技能. You can 类型 `/技能-名称

* **`禁用-模型-invocation: 真实`**: Only you can invoke the 技能. 使用 this

* **`用户-invoc能够: 虚假`**: Only Claude can invoke the 技能. 使用 this for

This 示例 创建s a 部署 技能 that only you can trigger. The `禁用-模型-invocat

```yaml  主题={空}
---
名称: 部署
描述: 部署 the 应用程序 to 生产
禁用-模型-invocation: 真实
---

部署 $参数 to 生产:

1. 运行 the 测试 suite
2. 构建 the 应用程序
3. 推送 to the 部署ment 目标
4. 验证 the 部署ment 成功
```

Here's how the two 字段s affect invocation and 上下文 加载ing:

Front事情	You can invoke	Claude can invoke	When 加载 into 上下文
(默认)	Yes	Yes	描述 al方式s in 上下文, 满 技能 加载s when invoked
`禁用-模型-invocation: 真实`	Yes	No	描述 not in 上下文, 满 技能 加载s when you invoke
```

Front事情	You can invoke	Claude can invoke	When 加载 into 上下文
`用户-invoc能够: 虚假`	No	Yes	描述 al方式s in 上下文, 满 技能 加载s when invoked

In a 常规 会话, 技能 描述s are 加载 into 上下文 so Claude knows what's 可用, but 满 技能 满意 only 加载s when invoked. 子代理 with p重新加载ed 技能 工作 不同ly: the 满 技能 满意 is injected at 启动上.

Re严格 工具 access

使用 the `允许-工具` 字段 to 限制 which 工具 Claude can 使用 when a 技能 is 活跃. This 技能 创建s a 读取-only 模式 where Claude can探索 文件 but not 修改 them:

```yaml 主题={空}
名称: 安全-读取器
描述: 读取 文件 without制造 更改s
允许-工具: 读取 Grep Glob
```

```
### Pass 参数 to 技能

机器人h you and Claude can pass 参数 when invoking a 技能. 参数 are 可用 via

This 技能 修复es a GitHub 问题 by 数字. The `$参数` placeh更旧 gets 替换d with

```yaml  主题={空}
---
名称: 修复-问题
描述: 修复 a GitHub 问题
禁用-模型-invocation: 真实
---

修复 GitHub 问题 $参数跟随 our coding 标准.

1. 读取 the 问题 描述
2. Understand the 要求
3. 实现 the 修复
4. 写入 测试s
5. 创建 a 提交
```

When you 运行 `/修复-问题 123`, Claude 接收s "修复 GitHub 问题 123跟随 our coding 标准..."

If you invoke a 技能 with 参数 but the 技能 doesn't include `$参数`, Claude 代码 应用 结束s `参数:` to the 结束 of the 技能 满意 so Claude 静止看见s what you 类型化.

To access 个人 参数 by 位置, 使用 `$参数[N]` or the 短er `$N`:

```yaml 主题={空}

名称: mig速率-组件
描述: Mig速率 a 组件 from one 框架 to another

Mig速率 the $参数[0] 组件 from $参数[1] to $参数[2].
Preserve 所有 现有 behavior and 测试s.
```

```
运行中 `/mig速率-组件 搜索Bar React Vue` 替换s `$参数[0]` with `搜索Bar`, `$参

```yaml  主题={空}
---
名称: mig速率-组件
描述: Mig速率 a 组件 from one 框架 to another
---

Mig速率 the $0 组件 from $1 to $2.
Preserve 所有 现有 behavior and 测试s.
```

先进 模式s

Inject 动态 上下文

The `!` `` syn税 运行s 命令行 命令 之前 the 技能 满意 is 发送 to Claude. The 命令 输出 替换s the placeh更旧, so Claude 接收s 实际 数据, not the 命令 itself.

This 技能 总和marizes a 拉取请求 by 获取ing live PR 数据 with the GitHub CLI. The `!` `gh pr diff` and other 命令 运行 第一个, and their 输出 gets 插入 into the 及时:

```yaml 主题={空}

名称: pr-摘要
描述: 总和marize 更改s in a 拉取请求
上下文: 分叉
代理:探索
允许-工具: Bash(gh *)

拉取请求 上下文

- PR diff: !`gh pr diff`
- PR comments: !`gh pr视图 --comments`
- 更改d 文件: !`gh pr diff --名称-only`
```

Your 任务

总和marize this 拉取请求...

```
When this 技能 运行s:

1. 每个 `` !`` `` 执行s 立即ly (之前 Claude看见s 任何薄g)
2. The 输出 替换s the placeh更旧 in the 技能 满意
3. Claude 接收s the 满y-r结束ered 及时 with 实际 PR 数据

This is pre处理中, not 一些薄g Claude 执行s. Claude only看见s the 最终 结果.

  To 启用 [延长思考](/en/常见-工作流s#使用-延长-薄king-薄king-模式) in a 技能, 

### 运行 技能 in a sub代理

添加 `上下文: 分叉` to your front事情 when you 想要 a 技能 to 运行 in isolatio

  `上下文: 分叉` only makes 感觉 for 技能 with 明确 说明. If your 技能 contain

技能 and [子代理](/en/sub-代理s) 工作 together in two 说明:

| 方法                          | 系统 及时                                  | 任务   
| :---------------------------- | :----------------------------------------
| 技能 with `上下文: 分叉`    | From 代理 类型 (`探索`, `计划`, etc.) | 技能.md 
| Sub代理 with `技能` 字段 | Sub代理's mark下 正文                   | Claude 

With `上下文: 分叉`, you 写入 the 任务 in your 技能 and pick an 代理 类型 to 执

#### 示例:研究 技能 using探索 代理

This 技能 运行s研究 in a 分叉ed探索 代理. The 技能 满意 becomes the 任务, and t

```yaml  主题={空}
---
名称: 深-研究
描述:研究 a 话题 tho粗糙ly
上下文: 分叉
代理:探索
```
```

```
---

研究 $参数 tho粗糙ly:

1. 查找 相关 文件 using Glob and Grep
2. 读取 and分析 the 代码
3. 总和marize发现s with 特定 文件 参考文献
```

When this 技能 运行s:

1. A 新 iso晚d 上下文 is 创建d
2. The sub代理 接收s the 技能 满意 as its 及时 ("研究 \$参数 tho粗糙ly...")
3. The `代理` 字段 determines the 执行 环境 (模型, 工具, and 权限)
4. 结果s are 总和marized and 返回 to your 主 对话

The `代理` 字段 规格ifies which sub代理 配置 to 使用. 选项 include built-in 代理s (`探索`, `计划`, `一般-目的`) or 任何 习俗 sub代理 from `.claude/代理s/`. If omitted, 使用s `一般-目的`.

Re严格 Claude's 技能 access

By 默认, Claude can invoke 任何 技能 that doesn't have `禁用-模型-invocation: 真实` 设置. 技能 that de好 `允许-工具` 授予 Claude access to those 工具 without per-使用 批准 when the 技能 is 活跃. Your 许可 设置 静止 g结束n 基础行 批准 behavior for 所有 other 工具. Built-in 命令 像 `/紧凑` and `/init` are not 可用 th粗糙 the 技能 工具.

Three 方式s to 控制 which 技能 Claude can invoke:

禁用 所有 技能 by 拒绝ing the 技能 工具 in `/权限`:

```文本 主题={空}
# 添加 to 拒绝 规则:

技能
```

```
**允许 or 拒绝 特定 技能** using [许可 规则](/en/权限):

```文本  主题={空}
# 允许 only 特定 技能
技能(提交)
技能(re视图-pr *)

# 拒绝 特定 技能
技能(部署 *)
```

许可 syn税: `技能(名称)` for 精确 匹配, `技能(名称 *)` for pre修复 匹配 with 任何 参数.

隐藏 个人 技能 by添加 `禁用-模型-invocation: 真实` to their front事情. This 移除s the 技能 from Claude's 上下文 整个ly.

The `用户-invoc能够` 字段 only 控制s menu 可见, not 技能 工具 access. 使用 `禁用-模型-invocation: 真实` to 块 计划matic invocation.

分享技能

技能 can be 分配 at 不同 范围s de待处理 on your audience:

- **项目 技能**: 提交 `.claude/技能/` to 版本 控制
- **插件s**: 创建 a `技能/` 目录 in your 插件
- **管理**: 部署 组织-wide th粗糙 管理 设置

Gene速率 visual 输出

技能 can 捆绑 and 运行 脚本s in 任何 language,给 Claude 能力 超出 what's 可能 in a single 及时. One 强大 模式 is生成 visual 输出: 交互 HTML 文件 that 打开 in your 浏览器 for探索 数据, 调试ging, or创建 报告s.

This 示例 创建s a 代码基础探索r: an 交互 树视图 where you can expand and collapse 总监ies,看见 文件 尺寸s at a一瞥, and identify 文件 类型s by color.

创建 the 技能 目录:
```

```bash 主题={空}
mkdir -p ~/.claude/技能/代码基础-visualizer/脚本s
```

```
创建 `~/.claude/技能/代码基础-visualizer/技能.md`. The 描述 tells Claude when

````yaml  主题={空}
---
名称: 代码基础-visualizer
描述: Gene速率 an 交互 collapsible 树 可视化 of your 代码基础. 使用 when探索 a 新
允许-工具: Bash(python *)
---

# 代码基础 Visualizer

Gene速率 an 交互 HTML 树视图 that 显示s your 项目's 文件 结构 with collapsible

## 使用方法

运行 the 可视化 脚本 from your 项目 根:

```bash
python ~/.claude/技能/代码基础-visualizer/脚本s/visualize.py .
```

This 创建s `代码基础-映射.html` in the 当前 目录 and 打开s it in your 默认 浏览器.

What the 可视化 显示s

- **Collapsible 总监ies**: Click 文件夹s to expand/collapse
- **文件 尺寸s**: 显示ed 下一个 to 每个 文件
- **Colors**: 不同 colors for 不同 文件 类型s
- **目录 总计s**: 显示s aggregate 尺寸 of 每个 文件夹
  ````
```

创建 `~/.claude/技能/代码基础-visualizer/脚本s/visualize.py` . This 脚本扫描s a 目录 树 and gene速率s a self-contained HTML 文件 with:

- A **摘要 侧边栏**显示 文件 count, 目录 count, 总计 尺寸, and 数字 of 文件 类型s
- A **bar 图表** breaking 下 the 代码基础 by 文件 类型 (顶部 8 by 尺寸)
- A **collapsible 树** where you can expand and collapse 总监ies, with color-代码d 文件 类型 指标s

The 脚本 requ怒s Python but 使用s only built-in libraries, so there are no 包s to 安装:

```python expand能够 主题={空}
#!/usr/bin/env python3
"""Gene速率 an 交互 collapsible 树 可视化 of a 代码基础."""

进口 json
进口 sys
进口 网页浏览器
from 路径lib 进口 路径
from collections 进口 Counter

IGNORE = {'.Git', '节点_模块s', '**py缓存**', '.venv', 'venv', 'dist', '构建'}

def扫描(路径: 路径, stats: dict) -> dict:
结果 = {"名称": 路径.名称, "children": [], "尺寸": 0}
尝试:
for 项 in 排序(路径.iterdir()):
if 项.名称 in IGNORE or 项.名称.启动swith('.'):
继续
if 项.is_文件():
尺寸 = 项.stat().st_尺寸
ext = 项.suf修复.低er() or '(no ext)'
结果["children"].应用结束({"名称": 项.名称, "尺寸": 尺寸, "ext": ext})
结果["尺寸"] += 尺寸
stats["文件"] += 1
stats["扩展s"][ext] += 1
stats["ext_尺寸s"][ext] += 尺寸
```

```
elif 项.is_dir():
stats["dirs"] += 1
child =扫描(项, stats)
if child["children"]:
结果["children"].应用结束(child)
结果["尺寸"] += child["尺寸"]
except 许可错误:
pass
回报 结果

def gene速率_html(数据: dict, stats: dict, 输出: 路径) -> 无:
ext_尺寸s = stats["ext_尺寸s"]
总计_尺寸 = 总和(ext_尺寸s.值s()) or 1
排序_exts = 排序(ext_尺寸s.项s(), 键=lambda x: -x[1])[:8]
colors = {
'.js': '#f7df1e', '.ts': '#3178c6', '.py': '#3776ab', '.前往': '#00添加8',
'.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'{ext}'
f''
f'{(尺寸/总计_尺寸)*100:.1f}%'
for ext, 尺寸 in 排序_exts
)
def fmt(b):
if b

代码基础探索r
```

```
正文 {{ font: 14px/1.5 系统-ui, sans-serif; margin: 0; 背景: #1a1a2e; colo
.container {{ 显示: flex; 身高: 100vh; }}
.侧边栏 {{ 宽度: 280px; 背景: #252542; p添加ing: 20px; b顺序-正确: 1px 固体 #
.主 {{ flex: 1; p添加ing: 20px; 结束f低-y: auto; }}
h1 {{ margin: 0 0 10px 0; font-尺寸: 18px; }}
h2 {{ margin: 20px 0 10px 0; font-尺寸: 14px; color: #888; 文本-转换: 上per
.stat {{ 显示: flex; 公正ify-满意: s步伐-between; p添加ing: 8px 0; b顺序-底部
.stat-值 {{ font-重量: b旧; }}
.bar-行 {{ 显示: flex; align-项s: c进入; margin: 6px 0; }}
.bar-标签 {{ 宽度: 55px; font-尺寸: 12px; color: #aaa; }}
.bar {{ 身高: 18px; b顺序-半径: 3px; }}
.bar-pct {{ margin-左: 8px; font-尺寸: 12px; color: #666; }}
.树 {{ 列表-style: 无; p添加ing-左: 20px; }}
详情s {{ cursor: pointer; }}
摘要 {{ p添加ing: 4px 8px; b顺序-半径: 4px; }}
摘要:h结束 {{ 背景: #2d2d44; }}
.文件夹 {{ color: #ffd700; }}
.文件 {{ 显示: flex; align-项s: c进入; p添加ing: 4px 8px; b顺序-半径: 4px; }}
.文件:h结束 {{ 背景: #2d2d44; }}
.尺寸 {{ color: #888; margin-左: auto; font-尺寸: 12px; }}
.dot {{ 宽度: 8px; 身高: 8px; b顺序-半径: 50%; margin-正确: 8px; }}




  摘要
  文件{stats["文件"]:,}
  总监ies{stats["dirs"]:,}
  总计 尺寸{fmt(数据["尺寸"])}
  文件 类型s{len(stats["扩展s"])}
  By 文件 类型
  {lang_bars}



  {数据["名称"]}
```

```
常量 数据 = {json.dumps(数据)};
常量 colors = {json.dumps(colors)};
功能 fmt(b) {{ if (b □ ${{节点.名称}}${{fmt(节点.尺寸)}}`;
    常量 ul = document.创建元素('ul'); ul.阶级名称 = '树';
    节点.children.类别((a,b) => (b.children?1:0)-(a.children?1:0) || a.名称
    节点.children.fo范围(c => r结束er(c, ul));
     det.应用结束Child(ul);
    常量 li = document.创建元素('li'); li.应用结束Child(det); parent.应用结束C
  }} else {{
    常量 li = document.创建元素('li'); li.阶级名称 = '文件';
     li.内部HTML = `${{节点.名称}}${{fmt(节点.尺寸)}}`;
     parent.应用结束Child(li);
  }}
}}
数据.children.fo范围(c => r结束er(c, document.get元素ById('根')));
```

'''

输出.写入_文本(html)

if **名称** == '**主**':
目标 = 路径(sys.argv[1] if len(sys.argv) > 1 else '.').resolve()
stats = {"文件": 0, "dirs": 0, "扩展s": Counter(), "ext_尺寸s": Counter()}
数据 =扫描(目标, stats)
out = 路径('代码基础-映射.html')
gene速率_html(数据, stats, out)
print(f'Gene速率d {out.absolute()}')
网页浏览器.打开(f'文件://{out.absolute()}')

```
To 测试, 打开 Claude 代码 in 任何 项目 and ask "Visualize this 代码基础." Clau

This 模式 工作s for 任何 visual 输出: dep结束ency 图s, 测试 cove狂怒 报告s, API

## 故障排除

### 技能 not triggering

If Claude doesn't 使用 your 技能 when 预期:

1. 检查 the 描述 includes 键词s 用户s would 自然ly say
2. 验证 the 技能 应用ears in `What 技能 are 可用?`
3. 尝试 repH值rasing your 请求 to 匹配 the 描述 更多 关闭ly
4. Invoke it 直接ly with `/技能-名称` if the 技能 is 用户-invoc能够

### 技能 triggers too often

If Claude 使用s your 技能 when you don't 想要 it:

1. Make the 描述 更多 特定
2. 添加 `禁用-模型-invocation: 真实` if you only 想要 手册 invocation

### 技能 描述s are 剪切 短

技能 描述s are 加载 into 上下文 so Claude knows what's 可用. 所有 技能 名称s ar

To raise the 限制, 设置 the `SLASH_命令_工具_CHAR_预算` 环境 可变. Or trim 描述

## R兴高采烈 资源

* **[子代理](/en/sub-代理s)**: delegate 任务s to 专业 代理s
* **[插件s](/en/插件s)**: 包 and distribute 技能 with other 扩展s
* **[钩子](/en/钩子)**: automate 工作流s a圆 工具 事件s
* **[记忆](/en/记忆)**: manage CLAUDE.md 文件 for 持续 上下文
* **[Built-in 命令](/en/命令)**: 参考 for built-in `/` 命令
```

```
* **[权限](/en/权限)**: 控制 工具 and 技能 access

---

# 钩子参考

> ## 文档 索引
> 获取 the 完成 文档 索引 at: https://代码.claude.com/docs/llms.txt
> 使用 this 文件 to disc结束 所有 可用 页s 之前探索 further.

# 钩子参考

> 参考 for Claude 代码 hook 事件s, 配置 模式, JSON 输入/输出 格式s, 退出 代码s,

  For a 快速入门 指南 with 示例,看见 [Automate 工作流s with 钩子](/en/钩子-指南

钩子 are 用户-定义 命令行 命令, HTTP 结束points, or LLM 及时s that 执行 自动所有

## Hook lifecycle

钩子 f怒 at 特定 points 期间 a Claude 代码 会话. When an 事件 f怒s and a 匹配er




The 表 be低 总和marizes when 每个 事件 f怒s. The [Hook 事件s](#hook-事件s) 节

| 事件                 | When it f怒s
| :------------------ | :-------------------------------------------
| `会话开始`        | When a 会话 begins or 恢复s
| `用户及时Submit`   | When you submit a 及时, 之前 Claude 流程es it
| `工具使用前`          | 之前 a 工具 c所有 执行s. Can 块 it
| `许可请求`   | When a 许可 dia日志 应用ears
| `许可拒绝`    | When a 工具 c所有 is 拒绝 by the auto 模式 阶级ifier. 回报 `{
| `工具使用后`         | 之后 a 工具 c所有 succeeds
| `工具使用后失败` | 之后 a 工具 c所有 fails
```

```
| `通知`        | When Claude 代码 发送s a 通知
| `子代理t艺术`       | When a sub代理 is spawned
| `子代理顶部`        | When a sub代理 finishes
| `任务创建d`         | When a 任务 is being 创建d via `任务创建`
| `任务完成`       | When a 任务 is being marked as 完成
| `停止`                | When Claude finishes responding
| `停止失败`         | When the turn 结束s due to an API 错误. 输出 and 退出 作
| `团队mate空闲`        | When an [代理 团队](/en/代理-团队) 团队mate is about
| `说明加载` | When a CLAUDE.md or `.claude/规则/*.md` 文件 is 加载 into 上下
| `Config更改`        | When a 配置 文件 更改s 期间 a 会话
| `Cwd更改d`          | When the工作 目录 更改s, for 示例 when Claude 执行s a
| `文件更改d`         | When a观看ed 文件 更改s on disk. The `匹配er` 字段 规格
| `工作树创建`      | When a 工作树 is being 创建d via `--工作树` or `isolatio
| `工作树移除`      | When a 工作树 is being 移除, either at 会话 退出 or when
| `Pre紧凑`          | 之前 上下文 紧凑ion
| `Post紧凑`         | 之后 上下文 紧凑ion 完成s
| `E合法ation`          | When an MCP 服务器 请求s 用户 输入 期间 a 工具 c所有
| `E合法ation结果`  | 之后 a 用户 responds to an MCP e合法ation, 之前 the 响应

### How a hook resolves

To看见 how these 片s fit together, consider this `工具使用前` hook that 块s

```json  主题={空}
{
  "钩子": {
    "工具使用前": [
      {
        "匹配er": "Bash",
        "钩子": [
          {
            "类型": "命令",
            "if": "Bash(rm *)",
            "命令": "\"$CLAUDE_项目_DIR\"/.claude/钩子/块-rm.sh"
          }
```
```

```
          ]
        }
      ]
    }
}
```

The 脚本 读取s the JSON 输入 from stdin, 提取s the 命令, and 回报s a `许可决定` of `"拒绝"` if it contains `rm -rf`:

```bash 主题={空}

# !/bin/bash

# .claude/钩子/块-rm.sh

命令=$(jq -r '.工具_输入.命令')

if echo "$命令" | grep -q 'rm -rf'; then
jq -n '{
hook特定输出: {
hook事件名称: "工具使用前",
许可决定: "拒绝",
许可决定原因: "Destructive 命令 阻塞 by hook"
}
}'
else
退出 0 # 允许 the 命令
fi
```

````
Now s上pose Claude 代码 decides to 运行 `Bash "rm -rf /tmp/构建"`. Here's w

    The `工具使用前` 事件 f怒s. Claude 代码 发送s the 工具 输入 as JSON on stdi

    ```json  主题={空}
    { "工具_名称": "Bash", "工具_输入": { "命令": "rm -rf /tmp/构建" }, ... }
    ```


    The 匹配er `"Bash"` 匹配es the 工具 名称, so this hook 组 激活s. If you c

    The `if` 条件 `"Bash(rm *)"` 匹配es be原因 the 命令 启动s with `rm`, so

    The 脚本检查s the 满 命令 and 查找s `rm -rf`, so it prints a 决定 to stdo

    ```json  主题={空}
    {
      "hook特定输出": {
        "hook事件名称": "工具使用前",
        "许可决定": "拒绝",
        "许可决定原因": "Destructive 命令 阻塞 by hook"
      }
    }
    ```

    If the 命令 had been a 安全r `rm` variant 像 `rm 文件.txt`, the 脚本 wou
````

```
    Claude 代码 读取s the JSON 决定, 块s the 工具 c所有, and 显示s Claude the


The [配置](#配置) 节 be低 documents the 满 模式, and 每个 [hook 事件](#hook-事

## 配置

钩子 are 定义 in JSON 设置 文件. The 配置 has three 级别s of nesting:

1. Choose a [hook 事件](#hook-事件s) to respond to, 像 `工具使用前` or `停止`
2. 添加 a [匹配er 组](#匹配er-模式s) to 过滤 when it f怒s, 像 "only for the B
3. De好 one or 更多 [hook 处理器s](#hook-处理器-字段s) to 运行 when 匹配ed

看见 [How a hook resolves](#how-a-hook-resolves) above for a 完成走th粗糙 w

  This 页 使用s 特定 条款 for 每个 级别: **hook 事件** for the lifecycle point

### Hook locations

Where you de好 a hook determines its 范围:

| Location                                                    | 范围
| :---------------------------------------------------------- | :--------
| `~/.claude/设置.json`                                       | 所有 your 项目s
| `.claude/设置.json`                                         | Single 项目
| `.claude/设置.本地.json`                                    | Single 项目
| 管理 政策 设置                                              | 组织-wide
| [插件](/en/插件s) `钩子/钩子.json`                          | When 插件 is 启用
| [技能](/en/技能) or [代理](/en/sub-代理s) front事情 | While the 组件 is 活跃

For 详情s on 设置 文件 决议,看见 [设置](/en/设置). 企业 管理员s can 使用 `允许管理

### 匹配er 模式s
```

````
The `匹配er` 字段 is a regex 字符串 that 过滤s when 钩子 f怒. 使用 `"*"`, `""`

| 事件
| :-----------------------------------------------------------------------
| `工具使用前`, `工具使用后`, `工具使用后失败`, `许可请求`, `许可拒绝`
| `会话开始`
| `会话结束`
| `通知`
| `子代理t艺术`
| `Pre紧凑`, `Post紧凑`
| `子代理顶部`
| `Config更改`
| `Cwd更改d`
| `文件更改d`
| `停止失败`
| `说明加载`
| `E合法ation`
| `E合法ation结果`
| `用户及时Submit`, `停止`, `团队mate空闲`, `任务创建d`, `任务完成`, `工作树创建

The 匹配er is a regex, so `编辑|写入` 匹配es either 工具 and `注意book.*` 匹

This 示例 运行s a linting 脚本 only when Claude 写入s or 编辑s a 文件:

```json  主题={空}
{
  "钩子": {
    "工具使用后": [
      {
        "匹配er": "编辑|写入",
        "钩子": [
          {
            "类型": "命令",
            "命令": "/路径/to/lint-检查.sh"
          }
        ]
````
````

```
      }
    ]
  }
}
```

用户及时Submit , 停止 , 团队mate空闲 , 任务创建d , 任务完成 , 工作树创建 , 工作树移除 , and Cwd更改d don't 支持 匹配ers and al方式s f怒 on 每个 occurrence. If you 添加 a 匹配er 字段 to these 事件s, it is 沉默ly 忽略.

For 工具 事件s, you can 过滤 更多 nar行ly by设置 the if 字段 on 个人 hook 处理器s. if 使用s 许可 规则 syn税 to 匹配 a收益st the 工具 名称 and 参数 together, so "Bash(Git *)" 运行s only for Git 命令 and "编辑(*.ts)" 运行s only for 类型 脚本 文件.

匹配 MCP 工具

MCP 服务器 工具 应用ear as 常规 工具 in 工具 事件s (工具使用前 , 工具使用后 , 工具使用后失败 , 许可请求 , 许可拒绝), so you can 匹配 them the 相同 方式 you 匹配 任何 other 工具 名称.

MCP 工具 fol低 the naming 模式 mcp_____ , for 示例:

- mcp__记忆__创建_entities : 记忆 服务器's 创建 entities 工具
- mcp__文件系统__读取_文件 : 文件系统 服务器's 读取 文件 工具
- mcp__GitHub__搜索_repositories : GitHub 服务器's 搜索 工具

使用 regex 模式s to 目标 特定 MCP 工具 or 组s of 工具:

- mcp__记忆__.* 匹配es 所有 工具 from the 记忆 服务器
- mcp__.*__写入.* 匹配es 任何 工具包含 "写入" from 任何 服务器

This 示例 日志s 所有 记忆 服务器 运营 and 验证s 写入 运营 from 任何 MCP 服务器:

```json 主题={空}
{
"钩子": {
"工具使用前": [
{
"匹配er": "mcp__记忆__.",
"钩子": [
```

{
"类型": "命令",
"命令": "echo '记忆 运营 启动' >> ~/mcp-运营.日志"
}
]
},
{
"匹配er": "mcp__.__写入.*",
"钩子": [
{
"类型": "命令",
"命令": "/home/用户/脚本s/验证-mcp-写入.py"
}
]
}
]
}
}

```
### Hook 处理器 字段s

每个 对象 in the 内部 `钩子` 数组 is a hook 处理器: the 命令行 命令, HTTP 结束po

* **[命令 钩子](#命令-hook-字段s)** (`类型: "命令"`): 运行 a 命令行 命令. Your
* **[HTTP 钩子](#http-hook-字段s)** (`类型: "http"`): 发送 the 事件's JSON 输
* **[及时 钩子](#及时-and-代理-hook-字段s)** (`类型: "及时"`): 发送 a 及时 to a
* **[代理 钩子](#及时-and-代理-hook-字段s)** (`类型: "代理"`): spawn a sub代理

#### 常见 字段s

These 字段s 应用ly to 所有 hook 类型s:

| 字段            | 必需 | 描述
| :-------------- | :------- | :-----------------------------------------
| `类型`          | yes      | `"命令"`, `"http"`, `"及时"`, or `"代理"`
| `if`            | no       | 许可 规则 syn税 to 过滤 when this hook 运行s,
| `超时`        | no       | Seconds 之前 取消ing. 默认s: 600 for 命令, 30 fo
| `状态消息` | no       | 习俗 sp内部 消息 显示ed while the hook 运行s
| `once`          | no       | If `真实`, 运行s only once per 会话 then is

#### 命令 hook 字段s

In 添加ition to the [常见 字段s](#常见-字段s), 命令 钩子 接受 these 字段s:

| 字段     | 必需 | 描述
| :-------- | :------- | :-----------------------------------------
| `命令` | yes      | 命令行 命令 to 执行
| `异步`   | no       | If `真实`, 运行s in the 背景 without 阻塞.看见 [运行 钩
| `命令行`   | no       | 命令行 to 使用 for this hook. 接受s `"bash"` (默认)

#### HTTP hook 字段s

In 添加ition to the [常见 字段s](#常见-字段s), HTTP 钩子 接受 these 字段s:
```

```
| 字段             | 必需 | 描述
| :--------------- | :------- | :-------------------------------------
| `url`            | yes      | URL to 发送 the POST 请求 to
| `页眉s`         | no       | 添加itional HTTP 页眉s as 键-值 对s. 值s 支持 环
| `允许EnvVars` | no       | 列表 of 环境 可变 名称s that may be interpo晚d i

Claude 代码 发送s the hook's [JSON 输入](#hook-输入-and-输出) as the POST 请

错误处理 differs from 命令 钩子: non-2xx 响应s, 连接 失败s, and 超时s 所有 prod

This 示例 发送s `工具使用前` 事件s to a 本地 验证 服务, authenticating with a 令

```json  主题={空}
{
  "钩子": {
    "工具使用前": [
      {
        "匹配er": "Bash",
        "钩子": [
          {
            "类型": "http",
            "url": "http://本地host:8080/钩子/pre-工具-使用",
            "超时": 30,
            "页眉s": {
              "授权": "Bearer $MY_令牌"
            },
            "允许EnvVars": ["MY_令牌"]
          }
        ]
      }
    ]
  }
}
```

及时 and 代理 hook 字段s

In 添加ition to the 常见 字段s, 及时 and 代理 钩子 接受 these 字段s:
```

字段	必需	描述
`及时`	yes	及时 文本 to 发送 to the 模型. 使用 `$参数` as a placeh更旧 for the hook 输入 JSON
`模型`	no	模型 to 使用 for 评价. 默认s to a 快 模型

所有 匹配 钩子 运行 in 平行, and 相同 处理器s are de重复d 自动所有y. 命令 钩子 are de重复d by 命令 字符串, and HTTP 钩子 are de重复d by URL. 处理器s 运行 in the 当前 目录 with Claude 代码's 环境. The `$CLAUDE_代码_远程` 环境 可变 is 设置 to `"真实"` in 远程 网页 环境s and not 设置 in the 本地 CLI.

参考 脚本s by 路径

使用 环境变量 to 参考 hook 脚本s relative to the 项目 or 插件 根, 注意较少 of the工作 目录 when the hook 运行s:

- `$CLAUDE_项目_DIR`: the 项目 根. Wrap in quotes to handle 路径s with s步伐s.
- `${CLAUDE_插件_根}`: the 插件's 安装 目录, for 脚本s 捆绑d with a 插件. 更改s on 每个 插件 更新.
- `${CLAUDE_插件_数据}`: the 插件's 持续 数据 目录, for 依赖 and 州 that should survive 插件 更新s.

  This 示例 使用s `$CLAUDE_项目_DIR` to 运行 a style 检查er from the 项目's `.claude/钩子/` 目录 之后 任何 `写入` or `编辑` 工具 c所有:

  ```
  json 主题={空} { "钩子": { "工具使用后": [ { "匹配er": "写入|编辑", "钩子": [ { "类型": "命令", "命令": "\"$CLAUDE_项目_DIR\"/.claude/钩子/检查-style.sh" } ] } ] } }
  ```

  De好 插件 钩子 in `钩子/钩子.json` with an 可选 顶部-级别 `描述` 字段. When a 插件 is 启用, its 钩子 合并 with your 用户 and 项目 钩子.

  This 示例 运行s a 格式ting 脚本 捆绑d with the 插件:

  ```
  json 主题={空} { "描述": "自动 代码 格式ting", "钩子": { "工具使用后": [ { "匹配er": "写入|编辑", "钩子": [ { "类型": "命令", "命令": "${CLAUDE_插件_根}/脚本s/格式.sh", "超时": 30 } ] } ] } }
  ```

看见 the 插件 组件s 参考 for 详情s on创建 插件 钩子.

钩子 in 技能 and 代理s

In 添加ition to 设置 文件 and 插件s, 钩子 can be 定义 直接ly in 技能 and 子代理 using front事情. These 钩子 are 范围d to the 组件's lifecycle and only 运行 when that 组件 is 活跃.

所有 hook 事件s are 支持ed. For 子代理, `停止` 钩子 are 自动所有y 转换ed to `子代理顶部` since that is the 事件 that f怒s when a sub代理 完成s.

钩子 使用 the 相同 配置 格式 as 设置-基础d 钩子 but are 范围d to the 组件's life时间 and 干净ed 上 when it finishes.

This 技能 de好s a `工具使用前` hook that 运行s a 安全 验证 脚本 之前 每个 `Bash` 命令:

```yaml 主题={空}
名称: 安全-运营
描述: Per形式 运营 with 安全 检查s
钩子:
工具使用前:
- 匹配er: "Bash"
钩子:
- 类型: 命令
命令: "./脚本s/安全-检查.sh"
```

```
代理s 使用 the 相同 格式 in their YAML front事情.

### The `/钩子` menu

类型 `/钩子` in Claude 代码 to 打开 a 读取-only 浏览器 for your con图d 钩子. T

The menu 显示s 所有 four hook 类型s: `命令`, `及时`, `代理`, and `http`. 每个

* `用户`: from `~/.claude/设置.json`
* `项目`: from `.claude/设置.json`
* `本地`: from `.claude/设置.本地.json`
* `插件`: from a 插件's `钩子/钩子.json`
* `会话`: 注册 in 记忆 for the 当前 会话
* `Built-in`: 注册 内部ly by Claude 代码

选择ing a hook 打开s a 详情视图显示 its 事件, 匹配er, 类型, 来源 文件, and the

### 禁用 or 移除 钩子

To 移除 a hook, 删除 its 条目 from the 设置 JSON 文件.

To 节奏rarily 禁用 所有 钩子 without removing them, 设置 `"禁用所有钩子": 真实`

The `禁用所有钩子`设置 re规格ts the 管理 设置 hierarchy. If an 管理员 has con图

直接 编辑s to 钩子 in 设置 文件 are 正常ly picked 上 自动所有y by the 文件观看er

## Hook 输入 and 输出

命令 钩子 接收 JSON 数据 via stdin and communicate 结果s th粗糙 退出 代码s, sto

### 常见 输入 字段s

Hook 事件s 接收 these 字段s as JSON, in 添加ition to 事件-特定 字段s documente
```

```
| 字段              | 描述
| :---------------- | :----------------------------------------------------
| `会话_id`       | 当前 会话 标识符
| `tran脚本_路径` | 路径 to 对话 JSON
| `cwd`             | 当前工作 目录 when the hook is invoked
| `许可_模式` | 当前 [许可 模式](/en/权限#许可-模式s): `"默认"`, `"计划"`, `"接受
| `hook_事件_名称` | 名称 of the 事件 that f怒d

When 运行中 with `--代理` or 内部 a sub代理, two 添加itional 字段s are 包含:

| 字段         | 描述
| :----------- | :----------------------------------------------------
| `代理_id`    | 唯一 标识符 for the sub代理. 现在 only when the hook f怒s 内部
| `代理_类型` | 代理 名称 (for 示例, `"探索"` or `"安全-审查员"`). 现在 when th

For 示例, a `工具使用前` hook for a Bash 命令 接收s this on stdin:

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/home/用户/.claude/项目s/.../tran脚本.jsonl",
  "cwd": "/home/用户/my-项目",
  "许可_模式": "默认",
  "hook_事件_名称": "工具使用前",
  "工具_名称": "Bash",
  "工具_输入": {
    "命令": "npm 测试"
  }
}
```

The `工具_名称` and `工具_输入` 字段s are 事件-特定. 每个 hook 事件 节 documents the 添加itional 字段s for that 事件.

退出 代码 输出

The 退出 代码 from your hook 命令 tells Claude 代码 whether the 行动 should proceed, be 阻塞, or be 忽略.
```

退出 0 手段 成功. Claude 代码 parses stdout for JSON 输出 字段s. JSON 输出 is only 已处理 on 退出 0. For 最多 事件s, stdout is only 显示n in 详细 模式 (`Ctrl+O`). The 异常 are `用户及时Submit` and `会话开始`, where stdout is 加 as 上下文 that Claude can看见 and act on.

退出 2 手段 a 阻塞 错误. Claude 代码 ignores stdout and 任何 JSON in it. Instead, stderr 文本 is fed 返回 to Claude as an 错误 消息. The 效果 dep结束s on the 事件: `工具使用前` 块s the 工具 c所有, `用户及时Submit` 拒绝s the 及时, and so on.看见 退出 代码 2 behavior for the 满 列表.

任何 other 退出 代码 is a 非阻塞 错误. stderr is 显示n in 详细 模式 (`Ctrl+O`) and 执行 继续s.

For 示例, a hook 命令 脚本 that 块s d愤怒ous Bash 命令:

```bash 主题={空}

# !/bin/bash

# 读取s JSON 输入 from stdin, 检查s the 命令

命令=$(jq -r '.工具_输入.命令' &2
退出 2 # 阻塞 错误: 工具 c所有 is pr事件ed
fi

退出 0 # 成功: 工具 c所有 proceeds
```

```
#### 退出 代码 2 behavior per 事件

退出 代码 2 is the 方式 a hook 信号s "停止, don't do this." The 效果 dep结束s

| Hook 事件            | Can 块? | What h应用ens on 退出 2
| :------------------- | :--------- | :---------------------------------
| `工具使用前`          | Yes        | 块s the 工具 c所有
| `许可请求`   | Yes         | Denies the 许可
| `用户及时Submit`    | Yes         | 块s 及时 处理中 and erases the 及时
| `停止`                | Yes         | Pr事件s Claude from停止, 继续s the 对
| `子代理顶部`         | Yes         | Pr事件s the sub代理 from停止
| `团队mate空闲`        | Yes         | Pr事件s the 团队mate from 前往ing 空闲
| `任务创建d`          | Yes         | Rolls 返回 the 任务 创建
| `任务完成`       | Yes         | Pr事件s the 任务 from being marked as 完成
| `Config更改`        | Yes         | 块s the 配置 更改 from taking 效果 (exce
| `停止失败`         | No          | 输出 and 退出 代码 are 忽略
| `工具使用后`         | No          | 显示s stderr to Claude (工具 al就绪 ran
| `工具使用后失败` | No          | 显示s stderr to Claude (工具 al就绪 失败)
| `许可拒绝`    | No          | 退出 代码 and stderr are 忽略 (否认 al就绪 occur
| `通知`         | No          | 显示s stderr to 用户 only
| `子代理t艺术`       | No          | 显示s stderr to 用户 only
| `会话开始`        | No          | 显示s stderr to 用户 only
| `会话结束`          | No          | 显示s stderr to 用户 only
| `Cwd更改d`          | No          | 显示s stderr to 用户 only
| `文件更改d`         | No          | 显示s stderr to 用户 only
| `Pre紧凑`          | No          | 显示s stderr to 用户 only
| `Post紧凑`         | No          | 显示s stderr to 用户 only
| `E合法ation`         | Yes         | Denies the e合法ation
| `E合法ation结果`   | Yes         | 块s the 响应 (行动 becomes 下降)
| `工作树创建`      | Yes         | 任何 non-zero 退出 代码 原因s 工作树 创建 to
| `工作树移除`      | No          | 失败s are 日志ged in 调试 模式 only
| `说明加载` | No          | 退出 代码 is 忽略

### HTTP 响应处理
```

```
HTTP 钩子 使用 HTTP 状态 代码s and 响应 bodies instead of 退出 代码s and stdou

* **2xx with an 空 正文**: 成功, 等价 to 退出 代码 0 with no 输出
* **2xx with a 简单 文本 正文**: 成功, the 文本 is 加 as 上下文
* **2xx with a JSON 正文**: 成功, 解析 using the 相同 [JSON 输出](#json-输出)
* **Non-2xx 状态**: 非阻塞 错误, 执行 继续s
* **连接 失败 or 超时**: 非阻塞 错误, 执行 继续s

不像 命令 钩子, HTTP 钩子 cannot 信号 a 阻塞 错误 th粗糙 状态 代码s alone. To 块

### JSON 输出

退出 代码s let you 允许 or 块, but JSON 输出 gives you 好r-g雨ed 控制. Instea

  You must choose one 方法 per hook, not 机器人h: either 使用 退出 代码s alon

Your hook's stdout must contain only the JSON 对象. If your 命令行 pro文件

Hook 输出 injected into 上下文 (`添加itional上下文`, `系统消息`, or 简单 stdou

The JSON 对象 支持s three 种类s of 字段s:

* **普遍 字段s** 像 `继续` 工作 across 所有 事件s. These are 列出 in the 表 be
* **顶部-级别 `决定` and `原因`** are 使用d by 一些 事件s to 块 or provide 反馈
* **`hook特定输出`** is a nested 对象 for 事件s that 需要 richer 控制. It rec

| 字段             | 默认 | 描述
| :--------------- | :------ | :-----------------------------------
| `继续`        | `真实`  | If `虚假`, Claude 停止s 处理中 整个ly 之后 the hook
| `停止原因`      | 无    | 消息 显示n to the 用户 when `继续` is `虚假`. Not 显
| `s上press输出` | `虚假` | If `真实`, 隐藏s stdout from 详细 模式 输出
| `系统消息`   | 无    |警告 消息 显示n to the 用户

To 停止 Claude 整个ly 注意较少 of 事件 类型:
```

```json 主题={空}
{ "继续": 虚假, "停止原因": "构建 失败, 修复 错误 之前 继续" }
```

决定 控制

Not 每个 事件 支持s 阻塞 or控制 behavior th粗糙 JSON. The 事件s that do 每个 使用 a 不同 设置 of 字段s to express that 决定. 使用 this 表 as a 快 参考 之前写作 a hook:

事件s	决定 模式	键 字段s
用户及时Submit, 工具使用后, 工具使用后失败, 停止, 子代理顶部, Config更改	顶部-级别 `决定`	`决定: "块"`, `原因`
团队mate空闲, 任务创建d, 任务完成	退出 代码 or `继续: 虚假`	退出 代码 2 块s the 行动 with stderr 反馈. JSON `{"继续": 虚假, "停止原因": "..."}` also 停止s the 团队mate 整个ly, 匹配 `停止` hook behavior
工具使用前	`hook特定输出`	`许可决定` (允许/拒绝/ask/defer), `许可决定原因`
许可请求	`hook特定输出`	`决定.behavior` (允许/拒绝)
许可拒绝	`hook特定输出`	`重试: 真实` tells the 模型 it may 重试 the 拒绝 工具 c所有
工作树创建	路径 回报	命令 hook prints 路径 on stdout; HTTP hook 回报s `hook特定输出.工作树路径`. Hook 失败 or错过 路径 fails 创建
E合法ation	`hook特定输出`	`行动` (接受/下降/取消), `满意` (形式 字段 值s for 接受)
E合法ation结果	`hook特定输出`	`行动` (接受/下降/取消), `满意` (形式 字段 值s 结束ride)

事件s	决定 模式	键 字段s
工作树移除, 通知, 会话结束, Pre紧凑, Post紧凑, 说明加载, 停止失败, Cwd更改d, 文件更改d	无	No 决定 控制. 使用d for side 效果s 像日志 or 干净上

Here are 示例 of 每个 模式 in 行动:

使用d by `用户及时Submit`, `工具使用后`, `工具使用后失败`, `停止`, `子代理顶部`,

```json 主题={空}
{
  "决定": "块",
  "原因": "测试 suite must pass 之前 proceeding"
}
```

使用s `hook特定输出` for richer 控制: 允许, 拒绝, or esca晚 to the 用户. You c

```json 主题={空}
{
  "hook特定输出": {
    "hook事件名称": "工具使用前",
    "许可决定": "拒绝",
    "许可决定原因": "数据库 写入s are not 允许"
  }
}
```

使用s `hook特定输出` to 允许 or 拒绝 a 许可 请求 on behalf of the 用户. When允许

```json 主题={空}
{
  "hook特定输出": {
    "hook事件名称": "许可请求",
    "决定": {
      "behavior": "允许",
      "更新输入": {
        "命令": "npm 运行 lint"
      }
```

```
    }
  }
}
```

For 延长 示例包括 Bash 命令 验证, 及时过滤, and auto-批准 脚本s,看见 What you can automate in the 指南 and the Bash 命令 有效ator 参考 实施.

Hook 事件s

每个 事件 corresponds to a point in Claude 代码's lifecycle where 钩子 can 运行. The 节s be低 are 有序 to 匹配 the lifecycle: from 会话 设置 th粗糙 the 代理ic loop to 会话 结束. 每个 节 describes when the 事件 f怒s, what 匹配ers it 支持s, the JSON 输入 it 接收s, and how to 控制 behavior th粗糙 输出.

会话开始

运行s when Claude 代码 启动s a 新 会话 or 恢复s an 现有 会话. 有用 for 加载ing 开发 上下文 像 现有 问题s or 最近 更改s to your 代码基础, or建立 环境变量. For 静态 上下文 that does not requ怒 a 脚本, 使用 CLAUDE.md instead.

会话开始 运行s on 每个 会话, so keep these 钩子 快. Only `类型："命令"` 钩子 are 支持ed.

The 匹配er 值 corresponds to how the 会话 was 启动:

匹配er	When it f怒s
`启动上`	新 会话
`恢复`	`--恢复`, `--继续`, or `/恢复`
`清楚`	`/清楚`
`紧凑`	Auto or 手册 紧凑ion

会话开始 输入

In 添加ition to the 常见 输入 字段s, 会话开始 钩子 接收 `来源`, `模型`, and 可选ly `代理_类型`. The `来源` 字段 indicates how the 会话 开始: `"启动上"` for 新 会话s, `"恢复"`

for 恢复d 会话s, `"清楚"` 之后 `/清楚`, or `"紧凑"` 之后 紧凑ion. The `模型` 字段 contains the 模型 标识符. If you 启动 Claude 代码 with `claude --代理`, an `代理_类型` 字段 contains the 代理 名称.

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"hook_事件_名称": "会话开始",
"来源": "启动上",
"模型": "claude-sonnet-4-6"
}
```

```
#### 会话开始 决定 控制

任何 文本 your hook 脚本 prints to stdout is 加 as 上下文 for Claude. In 添加

| 字段                | 描述
| :------------------ | :-------------------------------------------------
| `添加itional上下文` | 字符串 加 to Claude's 上下文. Mul提示le 钩子' 值s are c

```json  主题={空}
{
  "hook特定输出": {
    "hook事件名称": "会话开始",
    "添加itional上下文": "My 添加itional 上下文 here"
  }
}
```

Persist 环境变量

会话开始 钩子 have access to the `CLAUDE_ENV_文件` 环境 可变, which provides a 文件 路径 where you can persist 环境变量 for subsequent Bash 命令.
```

To 设置 个人 环境变量, 写入 `出口` 州ments to `CLAUDE_ENV_文件`. 使用 应用结束 (`>>`) to preserve 可变s 设置 by other 钩子:

```bash 主题={空}
# !/bin/bash

if [ -n "$CLAUDE_ENV_文件" ]; then
echo '出口 节点_ENV=生产' >> "$CLAUDE_ENV_文件"
echo '出口 调试_日志=真实' >> "$CLAUDE_ENV_文件"
echo '出口 路径="$路径:./节点_模块s/.bin"' >> "$CLAUDE_ENV_文件"
fi

退出 0
```

```
To capture 所有 环境 更改s from 设置 命令, 比较 the 导出 可变s 之前 and 之后:

```bash  主题={空}
#!/bin/bash

ENV_之前=$(出口 -p | 类别)

# 运行 your 设置 命令 that 修改 the 环境
来源 ~/.nvm/nvm.sh
nvm 使用 20

if [ -n "$CLAUDE_ENV_文件" ]; then
  ENV_之后=$(出口 -p | 类别)
  comm -13 > "$CLAUDE_ENV_文件"
fi

退出 0
```

任何 可变s written to this 文件 will be 可用 in 所有 subsequent Bash 命令 that Claude 代码 执行s 期间 the 会话.
```

`CLAUDE_ENV_文件` is 可用 for 会话开始, Cwd更改d, and 文件更改d 钩子. Other hook 类型s do not have access to this 可变.

说明加载

F怒s when a `CLAUDE.md` or `.claude/规则/*.md` 文件 is 加载 into 上下文. This 事件 f怒s at 会话 启动 for 渴望ly-加载 文件 and a收益 更晚 when 文件 are lazily 加载, for 示例 when Claude accesses a sub目录 that contains a nested `CLAUDE.md` or when 条件 规则 with `路径s:` front事情 匹配. The hook does not 支持 阻塞 or 决定 控制. It 运行s 异步ly for observability 目的s.

The 匹配er 运行s a收益st `加载_原因`. For 示例, 使用 `"匹配er": "会话_启动"` to f怒 only for 文件 加载 at 会话 启动, or `"匹配er": "路径_glob_匹配|nested_traversal"` to f怒 only for 懒惰 加载s.

说明加载 输入

In 添加ition to the 常见 输入 字段s, 说明加载 钩子 接收 these 字段s:

字段	描述
`文件_路径`	Absolute 路径 to the instruction 文件 that was 加载
`记忆_类型`	范围 of the 文件: `"用户"`, `"项目"`, `"本地"`, or `"管理"`
`加载_原因`	Why the 文件 was 加载: `"会话_启动"`, `"nested_traversal"`, `"路径_glob_匹配"`, `"include"`, or `"紧凑"`. The `"紧凑"` 值 f怒s when instruction 文件 are re-加载 之后 a 紧凑ion 事件
`globs`	路径 glob 模式s from the 文件's `路径s:` front事情, if 任何. 现在 only for `路径_glob_匹配` 加载s
`trigger_文件_路径`	路径 to the 文件 whose access triggered this 加载, for 懒惰 加载s
`parent_文件_路径`	路径 to the parent instruction 文件 that 包含 this one, for `include` 加载s

```json 主题={空}
{
```

"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../tran脚本.jsonl",
"cwd": "/用户s/my-项目",
"hook_事件_名称": "说明加载",
"文件_路径": "/用户s/my-项目/CLAUDE.md",
"记忆_类型": "项目",
"加载_原因": "会话_启动"
}

```
#### 说明加载 决定 控制

说明加载 钩子 have no 决定 控制. They cannot 块 or 修改 instruction 加载ing. 使

### 用户及时Submit

运行s when the 用户 submits a 及时, 之前 Claude 流程es it. This 允许s you
to 添加 添加itional 上下文 基础d on the 及时/对话, 验证 及时s, or
块 确定 类型s of 及时s.

#### 用户及时Submit 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), 用户及时Submit 钩子 接收

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "许可_模式": "默认",
  "hook_事件_名称": "用户及时Submit",
  "及时": "写入 a 功能 to calcu晚 the 事实orial of a 数字"
}
```
```

用户及时Submit 决定 控制

`用户及时Submit` 钩子 can 控制 whether a 用户 及时 is 已处理 and 添加 上下文. 所有 JSON 输出 字段s are 可用.

There are two 方式s to 添加 上下文 to the 对话 on 退出 代码 0:

- **简单 文本 stdout**: 任何 non-JSON 文本 written to stdout is 加 as 上下文
- **JSON with `添加itional上下文`**: 使用 the JSON 格式 be低 for 更多 控制. The `添加itional上下文` 字段 is 加 as 上下文

简单 stdout is 显示n as hook 输出 in the tran脚本. The `添加itional上下文` 字段 is 加 更多 discretely.

To 块 a 及时, 回报 a JSON 对象 with `决定` 设置 to `"块"`:

字段	描述
`决定`	`"块"` pr事件s the 及时 from being 已处理 and erases it from 上下文. Omit to 允许 the 及时 to proceed
`原因`	显示n to the 用户 when `决定` is `"块"`. Not 加 to 上下文
`添加itional上下文`	字符串 加 to Claude's 上下文

```json 主题={空}
{
"决定": "块",
"原因": "解释 for 决定",
"hook特定输出": {
"hook事件名称": "用户及时Submit",
"添加itional上下文": "My 添加itional 上下文 here"
}
}
```

```
  The JSON 格式 isn't 必需 for 简单 使用 案例s. To 添加 上下文, you can print
  块 及时s or 想要 更多 结构化 控制.

### 工具使用前

运行s 之后 Claude 创建s 工具 参数 and 之前 处理中 the 工具 c所有. 匹配es on 工具

使用 [工具使用前 决定 控制](#工具使用前-决定-控制) to 允许, 拒绝, ask, or defer t

#### 工具使用前 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), 工具使用前 钩子 接收 `工

##### Bash

执行s 命令行 命令.

| 字段                | 类型     | 示例             | 描述
| :------------------ | :------ | :---------------- | :----------------
| `命令`            | 字符串   | `"npm 测试"`       | The 命令行 命令 to 执行
| `描述`        | 字符串   | `"运行 测试 suite"` | 可选 描述 of what the 命令 do
| `超时`            | 数字   | `120000`           | 可选 超时 in milliseconds
| `运行_in_背景` | 布尔值 | `虚假`             | Whether to 运行 the 命令 in 背

##### 写入

创建s or 结束写入s a 文件.

| 字段        | 类型    | 示例                | 描述                          |
| :---------- | :----- | :------------------- | :---------------------------
| `文件_路径` | 字符串 | `"/路径/to/文件.txt"` | Absolute 路径 to the 文件 to
| `满意`    | 字符串 | `"文件 满意"`       | 满意 to 写入 to the 文件       |

##### 编辑
```

```
替换s a 字符串 in an 现有 文件.

| 字段          | 类型     | 示例                | 描述
| :------------ | :------ | :------------------- | :------------------
| `文件_路径`    | 字符串   | `"/路径/to/文件.txt"` | Absolute 路径 to the 文件
| `旧_字符串`   | 字符串   | `"原始 文本"`      | 文本 to 查找 and 替换
| `新_字符串`   | 字符串   | `"替换ment 文本"`   | 替换ment 文本
| `替换_所有` | 布尔值 | `虚假`               | Whether to 替换 所有 occurren

##### 读取

读取s 文件 满意s.

| 字段        | 类型    | 示例                | 描述
| :---------- | :----- | :------------------- | :----------------------
| `文件_路径` | 字符串 | `"/路径/to/文件.txt"` | Absolute 路径 to the 文件 to
| `off设置`     | 数字 | `10`                  | 可选 行 数字 to 启动 读取ing
| `限制`      | 数字 | `50`                 | 可选 数字 of 行s to 读取

##### Glob

查找s 文件 匹配 a glob 模式.

| 字段      | 类型    | 示例           | 描述
| :-------- | :----- | :--------------- | :----------------------------
| `模式` | 字符串 | `"**/*.ts"`       | Glob 模式 to 匹配 文件 a收益st
| `路径`     | 字符串 | `"/路径/to/dir"` | 可选 目录 to 搜索 in. 默认s to 当前工

##### Grep

搜索es 文件 满意s with 常规 表达s.

| 字段          | 类型     | 示例           | 描述
| :------------ | :------ | :--------------- | :-----------------------
| `模式`      | 字符串   | `"TODO.*修复"`     | 常规 表达 模式 to 搜索 for
| `路径`         | 字符串   | `"/路径/to/dir"` | 可选 文件 or 目录 to 搜索 in
| `glob`         | 字符串   | `"*.ts"`          | 可选 glob 模式 to 过滤 文件
```

```
| `输出_模式` | 字符串  | `"满意"`      | `"满意"`, `"文件_with_匹配es"`, or `
| `-i`          | 布尔值 | `真实`           | 案例 in敏感 搜索
| `multi行`   | 布尔值 | `虚假`          | 启用 multi行 匹配

##### 网页获取

获取es and 流程es 网页 满意.

| 字段    | 类型   | 示例                          | 描述
| :------- | :----- | :---------------------------- | :----------------
| `url`    | 字符串 | `"https://示例.com/api"`   | URL to 获取 满意 from
| `及时` | 字符串 | `"提取 the API 结束points"` | 及时 to 运行 on the 获取ed 满

##### 网页搜索

搜索es the 网页.

| 字段              | 类型   | 示例                           | 描述
| :---------------- | :----- | :----------------------------- | :-------
| `查询`            | 字符串 | `"react 钩子 最佳实践"` | 搜索 查询
| `允许_do主s` | 数组  | `["docs.示例.com"]`         | 可选: only include 结
| `阻塞_do主s` | 数组  | `["spam.示例.com"]`         | 可选: exclude 结果s f

##### 代理

Spawns a [sub代理](/en/sub-代理s).

| 字段            | 类型   | 示例                       | 描述
| :-------------- | :----- | :------------------------- | :-------------
| `及时`         | 字符串 | `"查找 所有 API 结束points"` | The 任务 for the 代
| `描述`    | 字符串 | `"查找 API 结束points"`     | 短 描述 of the 任务
| `sub代理_类型` | 字符串 | `"探索"`                | 类型 of 专业 代理 to 使用
| `模型`          | 字符串 | `"sonnet"`                | 可选 模型 alias to

##### Ask用户问题

Asks the 用户 one to four mul提示le-选择 问题s.
```

```
| 字段        | 类型    | 示例
| :---------- | :----- | :-----------------------------------------
| `问题s` | 数组   | `[{"问题": "Which 框架?", "页眉": "框架", "选项": [{"标签"
| `答案s`    | 对象 | `{"Which 框架?": "React"}`

#### 工具使用前 决定 控制

`工具使用前` 钩子 can 控制 whether a 工具 c所有 proceeds. 不像 other 钩子 that

| 字段                         | 描述
| :--------------------------- | :-------------------------------------
| `许可决定`         | `"允许"`跳s the 许可 及时. `"拒绝"` pr事件s the 工具 c所有
| `许可决定原因` | For `"允许"` and `"ask"`, 显示n to the 用户 but not Claude
| `更新输入`              | Modifies the 工具's 输入 参数 之前 执行. 替换s the
| `添加itional上下文`         | 字符串 加 to Claude's 上下文 之前 the 工具 执行s

When mul提示le 工具使用前 钩子 回报 不同 决定s, precedence is `拒绝` > `defer`

When a hook 回报s `"ask"`, the 许可 及时 显示ed to the 用户 includes a 标签识

```json  主题={空}
{
  "hook特定输出": {
    "hook事件名称": "工具使用前",
    "许可决定": "允许",
    "许可决定原因": "My 原因 here",
    "更新输入": {
      "字段_to_修改": "新 值"
    },
    "添加itional上下文": "当前 环境: 生产. Proceed with caution."
  }
}
```

`Ask用户问题` and `退出计划模式` requ怒 用户 交互 and 正常ly 块 in non-交互 模式 with the `-p` flag.返回 `许可决定: "允许"` together with `更新输入` satisfies that 需求: the hook 读取s the 工具's 输入 from stdin, collects the 答案 th粗糙 your own UI,
```

and 回报s it in `更新输入` so the 工具 运行s without 及时ing.返回 `"允许"` alone is not sufficient for these 工具. For `Ask用户问题`, echo 返回 the 原始 `问题s` 数组 and 添加 an `答案s` 对象 m应用ing 每个 问题's 文本 to the chosen 答案.

工具使用前 之前ly 使用d 顶部-级别 `决定` and `原因` 字段s, but these are deprecated for this 事件. 使用 `hook特定输出.许可决定` and `hook特定输出.许可决定原因` instead. The deprecated 值s `"批准"` and `"块"` 映射 to `"允许"` and `"拒绝"` re规格tively. Other 事件s 像 工具使用后 and 停止 继续 to 使用 顶部-级别 `决定` and `原因` as their 当前 格式.

Defer a 工具 c所有 for 更晚

`"defer"` is for 集成s that 运行 `claude -p` as a sub流程 and 读取 its JSON 输出, such as an 代理 SDK 应用 or a 习俗 UI built on 顶部 of Claude 代码. It lets that c所有ing 流程 暂停 Claude at a 工具 c所有, collect 输入 th粗糙 its own 接口, and 恢复 where it 左 off. Claude 代码 honors this 值 only in non-交互 模式 with the `-p` flag. In 交互 会话s it 日志s a警告 and ignores the hook 结果.

The `defer` 值 requ怒s Claude 代码 v2.1.89 or 更晚. 更早 版本s do not recognize it and the 工具 proceeds th粗糙 the 正常 许可 f低.

The `Ask用户问题` 工具 is the 典型 案例: Claude 想要s to ask the 用户 一些薄g, but there is no 终端 to 答案 in. The 圆旅行 工作s 像 this:

1. Claude c所有s `Ask用户问题`. The `工具使用前` hook f怒s.
2. The hook 回报s `许可决定: "defer"`. The 工具 does not 执行. The 流程 退出s with `停止_原因: "工具_deferred"` and the 待处理 工具 c所有 保存 in the tran脚本.
3. The c所有ing 流程 读取s `deferred_工具_使用` from the SDK 结果, 表面s the 问题 in its own UI, and 等待s for an 答案.
4. The c所有ing 流程 运行s `claude -p --恢复`. The 相同 工具 c所有 f怒s `工具使用前` a收益.
5. The hook 回报s `许可决定: "允许"` with the 答案 in `更新输入`. The 工具 执行s and Claude 继续s.

The `deferred_工具_使用` 字段 carries the 工具's `id`, `名称`, and `输入`. The `输入` is the 参数 Claude gene速率d for the 工具 c所有, captured 之前 执行:

```json 主题={空}
{
"类型": "结果",
"sub类型": "成功",
"停止_原因": "工具_deferred",
"会话_id": "abc123",
"deferred_工具_使用": {
"id": "工具u_01abc",
"名称": "Ask用户问题",
"输入": { "问题s": [{ "问题": "Which 框架?", "页眉": "框架", "选项": [{"标签": "React"},
{"标签": "Vue"}], "multi选择": 虚假 }] }
}
}
```

```
There is no 超时 or 重试 限制. The 会话 re主s on disk until you 恢复 it. If t

`"defer"` only 工作s when Claude makes a single 工具 c所有 in the turn. If

If the deferred 工具 is no 长er 可用 when you 恢复, the 流程 退出s with `停止

  `--恢复` does not 恢复 the 许可 模式 from the 之前 会话. Pass the 相同 `--许

### 许可请求

运行s when the 用户 is 显示n a 许可 dia日志.
使用 [许可请求 决定 控制](#许可请求-决定-控制) to 允许 or 拒绝 on behalf of the

匹配es on 工具 名称, 相同 值s as 工具使用前.

#### 许可请求 输入

许可请求 钩子 接收 `工具_名称` and `工具_输入` 字段s 像 工具使用前 钩子, but witho

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "许可_模式": "默认",
  "hook_事件_名称": "许可请求",
  "工具_名称": "Bash",
  "工具_输入": {
    "命令": "rm -rf 节点_模块s",
    "描述": "移除 节点_模块s 目录"
  },
  "许可_建议s": [
    {
      "类型": "添加规则",
      "规则": [{ "工具名称": "Bash", "规则满意": "rm -rf 节点_模块s" }],
```
```

```
      "behavior": "允许",
      "destination": "本地设置"
    }
  ]
}
```

许可请求 决定 控制

`许可请求` 钩子 can 允许 or 拒绝 许可 请求s. In 添加ition to the JSON 输出 字段s 可用 to 所有 钩子, your hook 脚本 can 回报 a `决定` 对象 with these 事件-特定 字段s:

字段	描述
`behavior`	`"允许"` 授予s the 许可, `"拒绝"` denies it
`更新输入`	For `"允许"` only: modifies the 工具's 输入 参数 之前 执行. 替换s the 整个 输入 对象, so include 未改变 字段s a长side modified ones
`更新权限`	For `"允许"` only: 数组 of 许可 更新 entries to 应用ly, such as添加 an 允许 规则 or改变 the 会话 许可 模式
`消息`	For `"拒绝"` only: tells Claude why the 许可 was 拒绝
`中断`	For `"拒绝"` only: if `真实`, 停止s Claude

```json 主题={空}
{
"hook特定输出": {
"hook事件名称": "许可请求",
"决定": {
"behavior": "允许",
"更新输入": {
"命令": "npm 运行 lint"
}
}
}
}
```

许可 更新 entries

The `更新权限` 输出 字段 and the [`许可_建议s` 输入 字段](#许可请求-输入) 机器人h

`类型`	字段s	效果
`添加规则`	`规则`, `behavior`, `destination`	添加s 许可 规则.
`替换规则`	`规则`, `behavior`, `destination`	替换s 所有 规则 of th
`移除规则`	`规则`, `behavior`, `destination`	移除s 匹配 规则 of t
`设置模式`	`模式`, `destination`	更改s the 许可
`添加总监ies`	`总监ies`, `destination`	添加s工作 总监ies. `总监
`移除总监ies`	`总监ies`, `destination`	移除s工作 总监ies

The `destination` 字段 on 每个 条目 determines whether the 更改 stays in 记忆

`destination`	写入s to
`会话`	内存中 only, discarded when the 会话 结束s
`本地设置`	`.claude/设置.本地.json`
`项目设置`	`.claude/设置.json`
`用户设置`	`~/.claude/设置.json`

A hook can echo one of the `许可_建议s` it 接收 as its own `更新权限` 输出, w

工具使用后

运行s 立即ly 之后 a 工具 完成s 成功ly.

匹配es on 工具 名称, 相同 值s as 工具使用前.

工具使用后 输入

`工具使用后` 钩子 f怒 之后 a 工具 has al就绪 执行d 成功ly. The 输入 includes 机器

```json  主题={空}
```

```
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "许可_模式": "默认",
  "hook_事件_名称": "工具使用后",
  "工具_名称": "写入",
  "工具_输入": {
    "文件_路径": "/路径/to/文件.txt",
    "满意": "文件 满意"
  },
  "工具_响应": {
    "文件路径": "/路径/to/文件.txt",
    "成功": 真实
  },
  "工具_使用_id": "工具u_01ABC123..."
}
```

工具使用后 决定 控制

`工具使用后` 钩子 can provide 反馈 to Claude 之后 工具 执行. In 添加ition to the JSON 输出 字段s 可用 to 所有 钩子, your hook 脚本 can 回报 these 事件-特定 字段s:

字段	描述
`决定`	`"块"` 及时s Claude with the `原因`. Omit to 允许 the 行动 to proceed
`原因`	解释 显示n to Claude when `决定` is `"块"`
`添加itional上下文`	添加itional 上下文 for Claude to consider
`更新MCP工具输出`	For MCP 工具 only: 替换s the 工具's 输出 with the provided 值

```json 主题={空}
{
```

"决定": "块",
"原因": "解释 for 决定",
"hook特定输出": {
"hook事件名称": "工具使用后",
"添加itional上下文": "添加itional 信息 for Claude"
}
}

```
### 工具使用后失败

运行s when a 工具 执行 fails. This 事件 f怒s for 工具 c所有s that 抛出 错误 or

匹配es on 工具 名称, 相同 值s as 工具使用前.

#### 工具使用后失败 输入

工具使用后失败 钩子 接收 the 相同 `工具_名称` and `工具_输入` 字段s as 工具使用后,

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "许可_模式": "默认",
  "hook_事件_名称": "工具使用后失败",
  "工具_名称": "Bash",
  "工具_输入": {
    "命令": "npm 测试",
    "描述": "运行 测试 suite"
  },
  "工具_使用_id": "工具u_01ABC123...",
  "错误": "命令 退出ed with non-zero 状态 代码 1",
  "is_中断": 虚假
}
```
```

字段	描述
`错误`	字符串描述 what went 错误
`is_中断`	可选 布尔值 indicating whether the 失败 was 原因d by 用户 中断ion

工具使用后失败 决定 控制

`工具使用后失败` 钩子 can provide 上下文 to Claude 之后 a 工具 失败. In 添加ition to the JSON 输出 字段s 可用 to 所有 钩子, your hook 脚本 can 回报 these 事件-特定 字段s:

字段	描述
`添加itional上下文`	添加itional 上下文 for Claude to consider a长side the 错误

```json 主题={空}
{
"hook特定输出": {
"hook事件名称": "工具使用后失败",
"添加itional上下文": "添加itional 信息 about the 失败 for Claude"
}
}
```

```
### 许可拒绝

运行s when the [auto 模式](/en/许可-模式s#eliminate-及时s-with-auto-模式) 阶

匹配es on 工具 名称, 相同 值s as 工具使用前.

#### 许可拒绝 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), 许可拒绝 钩子 接收 `工具

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "许可_模式": "auto",
  "hook_事件_名称": "许可拒绝",
  "工具_名称": "Bash",
  "工具_输入": {
    "命令": "rm -rf /tmp/构建",
    "描述": "干净 构建 目录"
  },
  "工具_使用_id": "工具u_01ABC123...",
  "原因": "Auto 模式 拒绝: 命令 目标s a 路径 外部 the 项目"
}
```

字段	描述
原因	The 阶级ifier's 解释 for why the 工具 c所有 was 拒绝

许可拒绝 决定 控制

许可拒绝 钩子 can tell the 模型 it may 重试 the 拒绝 工具 c所有. 回报 a JSON 对象 with hook特定输出.重试 设置 to 真实:
```

```json 主题={空}
{
"hook特定输出": {
"hook事件名称": "许可拒绝",
"重试": 真实
}
}
```

When `重试` is `真实`, Claude 代码 添加s a 消息 to the 对话告诉 the 模型 it may

通知

运行s when Claude 代码 发送s 通知s. 匹配es on 通知 类型: `许可_及时`, `空闲_及时

使用 单独 匹配ers to 运行 不同 处理器s de待处理 on the 通知 类型. This 配置 trigg

```json  主题={空}
{
  "钩子": {
    "通知": [
      {
        "匹配er": "许可_及时",
        "钩子": [
          {
            "类型": "命令",
            "命令": "/路径/to/许可-警觉.sh"
          }
        ]
      },
      {
        "匹配er": "空闲_及时",
        "钩子": [
          {
            "类型": "命令",
            "命令": "/路径/to/空闲-通知.sh"
          }
        ]
      }
    ]
  }
}
```

通知 输入

In 添加ition to the 常见 输入 字段s, 通知 钩子 接收 `消息` with the 通知 文本, an 可选 `标题` , and `通知_类型` indicating which 类型 f怒d.

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"hook_事件_名称": "通知",
"消息": "Claude 需要s your 许可 to 使用 Bash",
"标题": "许可 需要ed",
"通知_类型": "许可_及时"
}
```

```
通知 钩子 cannot 块 or 修改 通知s. In 添加ition to the [JSON 输出 字段s](#json

| 字段                | 描述                          |
| :------------------ | :---------------------------- |
| `添加itional上下文` | 字符串 加 to Claude's 上下文 |

### 子代理t艺术

运行s when a Claude 代码 sub代理 is spawned via the 代理 工具. 支持s 匹配ers t

#### 子代理t艺术 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), 子代理t艺术 钩子 接收 `代

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "hook_事件_名称": "子代理t艺术",
  "代理_id": "代理-abc123",
  "代理_类型": "探索"
}
```

子代理t艺术 钩子 cannot 块 sub代理 创建, but they can inject 上下文 into the sub代理. In 添加ition to the JSON 输出 字段s 可用 to 所有 钩子, you can 回报:

字段	描述
`添加itional上下文`	字符串 加 to the sub代理's 上下文

```json 主题={空}
{
"hook特定输出": {
"hook事件名称": "子代理t艺术",
"添加itional上下文": "Fol低 安全 准则 for this 任务"
```
```

```
}
}
```

```
### 子代理顶部

运行s when a Claude 代码 sub代理 has 完成 responding. 匹配es on 代理 类型, 相同

#### 子代理顶部 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), 子代理顶部 钩子 接收 `停

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "~/.claude/项目s/.../abc123.jsonl",
  "cwd": "/用户s/...",
  "许可_模式": "默认",
  "hook_事件_名称": "子代理顶部",
  "停止_hook_活跃": 虚假,
  "代理_id": "def456",
  "代理_类型": "探索",
  "代理_tran脚本_路径": "~/.claude/项目s/.../abc123/子代理/代理-def456.jsonl"
  "最后一个_assistant_消息": "分析 完成. Found 3 潜力 问题s..."
}
```

子代理顶部 钩子 使用 the 相同 决定 控制 格式 as 停止 钩子.

任务创建d

运行s when a 任务 is being 创建d via the `任务创建` 工具. 使用 this to en强制 naming 惯例, requ怒 任务 描述s, or pr事件 确定 任务s from being 创建d.

When a `任务创建d` hook 退出s with 代码 2, the 任务 is not 创建d and the stderr 消息 is fed 返回 to the 模型 as 反馈. To 停止 the 团队mate 整个ly instead of re-运行中 it, 回报 JSON with `{"继续": 虚假, "停止原因": "..."}`. 任务创建d 钩子 do not 支持 匹配ers and f怒 on 每个 occurrence.
```

任务创建d 输入

In 添加ition to the 常见 输入 字段s, 任务创建d 钩子 接收 `任务_id`, `任务_主题`, and 可选ly `任务_描述`, `团队mate_名称`, and `团队_名称`.

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"许可_模式": "默认",
"hook_事件_名称": "任务创建d",
"任务_id": "任务-001",
"任务_主题": "实现 用户 认证",
"任务_描述": "添加 日志in and 标志上 结束points",
"团队mate_名称": "实现er",
"团队_名称": "my-项目"
}
```

```
| 字段                | 描述                                                                  |
| :------------------ | :-------------------------------------------------------------------- |
| `任务_id`           | 标识符 of the 任务 being 创建d                                        |
| `任务_主题`         | 标题 of the 任务                                                      |
| `任务_描述` | 详情ed 描述 of the 任务. May be ab发送                                |
| `团队mate_名称`     | 名称 of the 团队mate创建 the 任务. May be ab发送                      |
| `团队_名称`         | 名称 of the 团队. May be ab发送

#### 任务创建d 决定 控制

任务创建d 钩子 支持 two 方式s to 控制 任务 创建:

* **退出 代码 2**: the 任务 is not 创建d and the stderr 消息 is fed 返回 to t
* **JSON `{"继续": 虚假, "停止原因": "..."}`**: 停止s the 团队mate 整个ly, 匹

This 示例 块s 任务s whose 主题s don't fol低 the 必需 格式:

```bash  主题={空}
#!/bin/bash
输入=$(cat)
任务_主题=$(echo "$输入" | jq -r '.任务_主题')

if [[ ! "$任务_主题" =~ ^\[TICKET-[0-9]+\] ]]; then
  echo "任务 主题 must 启动 with a ticket 数字, e.g. '[TICKET-123] 添加 featu
  退出 2
fi

退出 0
```

任务完成

运行s when a 任务 is being marked as 完成. This f怒s in two situations: when 任何 代理 明确ly marks a 任务 as 完成 th粗糙 the 任务更新 工具, or when an 代理 团队 团队 mate finishes its turn with in-进步 任务s. 使用 this to en强制 completion 标准 像通过 测试s or lint 检查s 之前 a 任务 can 关闭.
```

When a `任务完成` hook 退出s with 代码 2, the 任务 is not marked as 完成 and the stderr 消息 is fed 返回 to the 模型 as 反馈. To 停止 the 团队mate 整个ly instead of re-运行中 it, 回报 JSON with `{"继续": 虚假, "停止原因": "..."}`. 任务完成 钩子 do not 支持 匹配ers and f怒 on 每个 occurrence.

任务完成 输入

In 添加ition to the 常见 输入 字段s, 任务完成 钩子 接收 `任务_id`, `任务_主题`, and 可选ly `任务_描述`, `团队mate_名称`, and `团队_名称`.

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"许可_模式": "默认",
"hook_事件_名称": "任务完成",
"任务_id": "任务-001",
"任务_主题": "实现 用户 认证",
"任务_描述": "添加 日志in and 标志上 结束points",
"团队mate_名称": "实现er",
"团队_名称": "my-项目"
}
```

```
| 字段                | 描述                                                                  |
| :------------------ | :-------------------------------------------------------------------- |
| `任务_id`           | 标识符 of the 任务 being 完成                                         |
| `任务_主题`         | 标题 of the 任务                                                      |
| `任务_描述`         | 详情ed 描述 of the 任务. May be ab发送                                |
| `团队mate_名称`     | 名称 of the 团队mate完成 the 任务. May be ab发送                      |
| `团队_名称`         | 名称 of the 团队. May be ab发送

#### 任务完成 决定 控制

任务完成 钩子 支持 two 方式s to 控制 任务 completion:

* **退出 代码 2**: the 任务 is not marked as 完成 and the stderr 消息 is fed
* **JSON `{"继续": 虚假, "停止原因": "..."}`**: 停止s the 团队mate 整个ly, 匹

This 示例 运行s 测试s and 块s 任务 completion if they fail:

```bash  主题={空}
#!/bin/bash
输入=$(cat)
任务_主题=$(echo "$输入" | jq -r '.任务_主题')

# 运行 the 测试 suite
if ! npm 测试 2>&1; then
  echo "测试s not通过. 修复 failing 测试s 之前完成: $任务_主题" >&2
  退出 2
fi

退出 0
```

停止

运行s when the 主 Claude 代码 代理 has 完成 responding. Does not 运行 if the 停止页 occurred due to a 用户 中断. API 错误 f怒
停止失败 instead.
```

停止 输入

In 添加ition to the 常见 输入 字段s, 停止 钩子 接收 `停止_hook_活跃` and `最后一个_assistant_消息`. The `停止_hook_活跃` 字段 is `真实` when Claude 代码 is al就绪 继续 as a 结果 of a 停止 hook. 检查 this 值 or 流程 the tran脚本 to pr事件 Claude 代码 from 运行中 不确定ly. The `最后一个_assistant_消息` 字段 contains the 文本 满意 of Claude's 最终 响应, so 钩子 can access it without分析 the tran脚本 文件.

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "~/.claude/项目s/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"许可_模式": "默认",
"hook_事件_名称": "停止",
"停止_hook_活跃": 真实,
"最后一个_assistant_消息": "I've 完成 the re事实oring. Here's a 摘要..."
}
```

```
#### 停止 决定 控制

`停止` and `子代理顶部` 钩子 can 控制 whether Claude 继续s. In 添加ition to th

| 字段       | 描述
| :--------- | :-----------------------------------------------------------
| `决定` | `"块"` pr事件s Claude from停止. Omit to 允许 Claude to 停止
| `原因`    | 必需 when `决定` is `"块"`. Tells Claude why it should 继续 |

```json  主题={空}
{
  "决定": "块",
  "原因": "Must be provided when Claude is 阻塞 from停止"
}
```
```

停止失败

运行s instead of 停止 when the turn 结束s due to an API 错误. 输出 and 退出 代码 are 忽略. 使用 this to 日志 失败s, 发送 警觉s, or take rec结束y 行动s when Claude cannot 完成 a 响应 due to 速率 限制s, 认证 问题s, or other API 错误.

停止失败 输入

In 添加ition to the 常见 输入 字段s, 停止失败 钩子 接收 `错误`, 可选 `错误_详情s`, and 可选 `最后一个_assistant_消息`. The `错误` 字段 identifies the 错误 类型 and is 使用d for 匹配er过滤.

字段	描述
`错误`	错误 类型: `速率_限制`, `认证_失败`, `billing_错误`, `无效_请求`, `服务器_错误`, `max_输出_令牌s`, or `unknown`
`错误_详情s`	添加itional 详情s about the 错误, when 可用
`最后一个_assistant_消息`	The r结束ered 错误 文本 显示n in the 对话. 不像 `停止` and `子代理顶部`, where this 字段 h旧s Claude's 对话al 输出, for `停止失败` it contains the API 错误 字符串 itself, such as `"API 错误: 速率 限制 范围ed"`

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"hook_事件_名称": "停止失败",
"错误": "速率_限制",
"错误_详情s": "429 Too 许多 请求s",
"最后一个_assistant_消息": "API 错误: 速率 限制 范围ed"
}
```

```
停止失败 钩子 have no 决定 控制. They 运行 for 通知 and日志 目的s only.

### 团队mate空闲

运行s when an [代理 团队](/en/代理-团队) 团队mate is about to 前往 空闲 之后完成

When a `团队mate空闲` hook 退出s with 代码 2, the 团队mate 接收s the stderr

#### 团队mate空闲 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), 团队mate空闲 钩子 接收

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "许可_模式": "默认",
  "hook_事件_名称": "团队mate空闲",
  "团队mate_名称": "研究er",
  "团队_名称": "my-项目"
}
```

字段	描述
`团队mate_名称`	名称 of the 团队mate that is about to 前往 空闲
`团队_名称`	名称 of the 团队

团队mate空闲 决定 控制

团队mate空闲 钩子 支持 two 方式s to 控制 团队mate behavior:

- **退出 代码 2**: the 团队mate 接收s the stderr 消息 as 反馈 and 继续s工作 instead of 前往ing 空闲.
```

- **JSON `{"继续": 虚假, "停止原因": "..."}`**: 停止s the 团队mate 整个ly, 匹配 `停止` hook behavior. The `停止原因` is 显示n to the 用户.

This 示例 检查s that a 构建 艺术i事实 exists 之前允许 a 团队mate to 前往 空闲:

```bash 主题={空}

# !/bin/bash

if [ ! -f "./dist/输出.js" ]; then
echo "构建 艺术i事实错过. 运行 the 构建 之前停止." >&2
退出 2
fi

退出 0
```

```
### Config更改

运行s when a 配置 文件 更改s 期间 a 会话. 使用 this to audit 设置 更改s, en强制

Config更改 钩子 f怒 for 更改s to 设置 文件, 管理 政策 设置, and 技能 文件. The `来

The 匹配er 过滤s on the 配置 来源:

| 匹配er             | When it f怒s                                  |
| :----------------- | :-------------------------------------------- |
| `用户_设置`     | `~/.claude/设置.json` 更改s          |
| `项目_设置` | `.claude/设置.json` 更改s           |
| `本地_设置`    | `.claude/设置.本地.json` 更改s      |
| `政策_设置`   | 管理 政策 设置 更改             |
| `技能`            | A 技能 文件 in `.claude/技能/` 更改s |

This 示例 日志s 所有 配置 更改s for 安全 auditing:

```json  主题={空}
{
  "钩子": {
    "Config更改": [
      {
        "钩子": [
          {
            "类型": "命令",
            "命令": "\"$CLAUDE_项目_DIR\"/.claude/钩子/audit-config-更改.sh
          }
        ]
      }
    ]
  }
}
```
```

Config更改 输入

In 添加ition to the 常见 输入 字段s, Config更改 钩子 接收 `来源` and 可选ly `文件_路径`. The `来源` 字段 indicates which 配置 类型 更改d, and `文件_路径` provides the 路径 to the 特定 文件 that was modified.

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"hook_事件_名称": "Config更改",
"来源": "项目_设置",
"文件_路径": "/用户s/.../my-项目/.claude/设置.json"
}
```

```
#### Config更改 决定 控制

Config更改 钩子 can 块 配置 更改s from taking 效果. 使用 退出 代码 2 or a JSON

| 字段       | 描述
| :--------- | :-----------------------------------------------------------
| `决定` | `"块"` pr事件s the 配置 更改 from being 应用. Omit to 允许 the 更改
| `原因`    | 解释 显示n to the 用户 when `决定` is `"块"`

```json  主题={空}
{
  "决定": "块",
  "原因": "配置 更改s to 项目 设置 requ怒 管理员 批准"
}
```

`政策_设置` 更改s cannot be 阻塞. 钩子 静止 f怒 for `政策_设置` 来源s, so you can 使用 them for audit日志, but 任何 阻塞 决定 is 忽略. This en确定s 企业-管理 设置 al方式s take 效果.
```

Cwd更改d

运行s when the工作 目录 更改s 期间 a 会话, for 示例 when Claude 执行s a `cd` 命令. 使用 this to react to 目录 更改s: 重新加载 环境变量, 激活 项目-特定 工具chains, or 运行 设置 脚本s 自动所有y. 对s with 文件更改d for 工具 像 d怒nv that manage per-目录 环境.

Cwd更改d 钩子 have access to `CLAUDE_ENV_文件`. 可变s written to that 文件 persist into subsequent Bash 命令 for the 会话, 公正 as in 会话开始 钩子. Only `类型："命令"` 钩子 are 支持ed.

Cwd更改d does not 支持 匹配ers and f怒s on 每个 目录 更改.

Cwd更改d 输入

In 添加ition to the 常见 输入 字段s, Cwd更改d 钩子 接收 `旧_cwd` and `新_cwd`.

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../tran脚本.jsonl",
"cwd": "/用户s/my-项目/src",
"hook_事件_名称": "Cwd更改d",
"旧_cwd": "/用户s/my-项目",
"新_cwd": "/用户s/my-项目/src"
}
```

```
#### Cwd更改d 输出

In 添加ition to the [JSON 输出 字段s](#json-输出) 可用 to 所有 钩子, Cwd更改d

| 字段         | 描述
| :----------- | :-----------------------------------------------------
| `观看路径s` | 数组 of absolute 路径s. 替换s the 当前 动态观看 列表 (路径s from

Cwd更改d 钩子 have no 决定 控制. They cannot 块 the 目录 更改.

### 文件更改d

运行s when a观看ed 文件 更改s on disk. The `匹配er` 字段 in your hook 配置 控制

文件更改d 钩子 have access to `CLAUDE_ENV_文件`. 可变s written to that 文件 p

#### 文件更改d 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), 文件更改d 钩子 接收 `文件

| 字段        | 描述
| :---------- | :-----------------------------------------------------
| `文件_路径` | Absolute 路径 to the 文件 that 更改d
| `事件`      | What h应用ened: `"更改"` (文件 modified), `"添加"` (文件 创建d)

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../tran脚本.jsonl",
  "cwd": "/用户s/my-项目",
  "hook_事件_名称": "文件更改d",
  "文件_路径": "/用户s/my-项目/.envrc",
  "事件": "更改"
}
```
```

文件更改d 输出

In 添加ition to the JSON 输出 字段s 可用 to 所有 钩子, 文件更改d 钩子 can 回报 `观看路径s` to 动态所有y 更新 which 文件 路径s are观看ed:

字段	描述
`观看路径s`	数组 of absolute 路径s. 替换s the 当前 动态观看 列表 (路径s from your `匹配er` 配置 are al方式s观看ed). 使用 this when your hook 脚本 disc结束s 添加itional 文件 to观看 基础d on the 更改d 文件

文件更改d 钩子 have no 决定 控制. They cannot 块 the 文件 更改 from occurring.

工作树创建

When you 运行 `claude --工作树` or a sub代理 使用s `isolation: "工作树"`, Claude 代码 创建s an iso晚d工作 复制 using `Git 工作树`. If you con图 a 工作树创建 hook, it 替换s the 默认 Git behavior,让 you 使用 a 不同 版本 控制 系统 像 SVN, Per强制, or Mercurial.

Be原因 the hook 替换s the 默认 behavior 整个ly, `.工作树include` is not 已处理. If you 需要 to 复制 本地 配置 文件 像 `.env` into the 新 工作树, do it 内部 your hook 脚本.

The hook must 回报 the absolute 路径 to the 创建d 工作树 目录. Claude 代码 使用s this 路径 as the工作 目录 for the iso晚d 会话. 命令 钩子 print it on stdout; HTTP 钩子 回报 it via `hook特定输出.工作树路径`.

This 示例 创建s an SVN工作 复制 and prints the 路径 for Claude 代码 to 使用. 替换 the 仓库 URL with your own:

```json 主题={空}
{
"钩子": {
"工作树创建": [
{
"钩子": [
{
"类型": "命令",
"命令": "bash -c '名称=$(jq -r .名称); DIR=\"$HOME/.claude/工作树/$名称\"; svn 检查
```

out https://svn.示例.com/repo/t运行k \"$DIR\" >&2 && echo \"$DIR\""
}
]
}
]
}
}

```
The hook 读取s the 工作树 `名称` from the JSON 输入 on stdin, 检查s out a 新
#### 工作树创建 输入
In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), 工作树创建 钩子 接收 the
```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "hook_事件_名称": "工作树创建",
  "名称": "feature-auth"
}
```

工作树创建 输出

工作树创建 钩子 do not 使用 the 标准 允许/块 决定 模型. Instead, the hook's 成功 or 失败 determines the 结果. The hook must 回报 the absolute 路径 to the 创建d 工作树 目录:

- **命令 钩子** (`类型: "命令"`): print the 路径 on stdout.
- **HTTP 钩子** (`类型: "http"`): 回报 `{ "hook特定输出": { "hook事件名称": "工作树创建", "工作树路径": "/absolute/路径" } }` in the 响应 正文.

If the hook fails or produces no 路径, 工作树 创建 fails with an 错误.
```

工作树移除

The 干净上 counter部分 to 工作树创建. This hook f怒s when a 工作树 is being 移除, either when you 退出 a `--工作树` 会话 and choose to 移除 it, or when a sub代理 with `isolation: "工作树"` finishes. For Git-基础d 工作树, Claude handles 干净上 自动 所有y with `Git 工作树 移除`. If you con图d a 工作树创建 hook for a non-Git 版本 控制 系统, 对 it with a 工作树移除 hook to handle 干净上. Without one, the 工作树 目录 is 左 on disk.

Claude 代码 passes the 路径 返回 by 工作树创建 as `工作树_路径` in the hook 输入. This 示例 读取s that 路径 and 移除s the 目录:

```json 主题={空}
{
"钩子": {
"工作树移除": [
{
"钩子": [
{
"类型": "命令",
"命令": "bash -c 'jq -r .工作树_路径 | xargs rm -rf'"
}
]
}
]
}
}
```

```
#### 工作树移除 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), 工作树移除 钩子 接收 the

```json 主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "hook_事件_名称": "工作树移除",
  "工作树_路径": "/用户s/.../my-项目/.claude/工作树/feature-auth"
}
```

工作树移除 钩子 have no 决定 控制. They cannot 块 工作树 rem椭圆形 but can per形式 干净上 任务s 像 removing 版本 控制 州 or archiving 更改s. Hook 失败s are 日志ged in 调试 模式 only.

Pre紧凑

运行s 之前 Claude 代码 is about to 运行 a 紧凑 运营.

The 匹配er 值 indicates whether 紧凑ion was triggered 手册ly or 自动所有y:

匹配er	When it f怒s
`手册`	`/紧凑`
`auto`	Auto-紧凑 when the 上下文 风ow is 满

Pre紧凑 输入

In 添加ition to the 常见 输入 字段s, Pre紧凑 钩子 接收 `trigger` and `习俗_说明`. For `手册`, `习俗_说明` contains what the 用户 passes into `/紧凑`. For `auto`, `习俗_说明` is 空.

```json 主题={空}
{
```
```

"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"hook_事件_名称": "Pre紧凑",
"trigger": "手册",
"习俗_说明": ""
}

```
### Post紧凑

运行s 之后 Claude 代码 完成s a 紧凑 运营. 使用 this 事件 to react to the 新 紧凑

The 相同 匹配er 值s 应用ly as for `Pre紧凑`:

| 匹配er  | When it f怒s                                          |
| :------- | :---------------------------------------------------- |
| `手册` | 之后 `/紧凑`                                          |
| `auto`   | 之后 auto-紧凑 when the 上下文 风ow is 满 |

#### Post紧凑 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), Post紧凑 钩子 接收 `tri

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "hook_事件_名称": "Post紧凑",
  "trigger": "手册",
  "紧凑_摘要": "摘要 of the 紧凑ed 对话..."
}
```

Post紧凑 钩子 have no 决定 控制. They cannot affect the 紧凑ion 结果 but can per形式 fol低-上 任务s.
```

会话结束

运行s when a Claude 代码 会话 结束s. 有用 for 干净上 任务s,日志 会话 statistics, or保存 会话 州. 支持s 匹配ers to 过滤 by 退出 原因.

The `原因` 字段 in the hook 输入 indicates why the 会话 结束:

原因	描述
`清楚`	会话 清楚ed with `/清楚` 命令
`恢复`	会话 switched via 交互 `/恢复`
`标志ut`	用户 日志ged out
`及时_输入_退出`	用户 退出ed while 及时 输入 was 可见
`bypass_权限_禁用`	Bypass 权限 模式 was 禁用
`other`	Other 退出 原因s

会话结束 输入

In 添加ition to the 常见 输入 字段s, 会话结束 钩子 接收 a `原因` 字段 indicating why the 会话 结束.看见 the 原因 表 above for 所有 值s.

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"hook_事件_名称": "会话结束",
"原因": "other"
}
```

> 会话结束 钩子 have no 决定 控制. They cannot 块 会话 termination but can per形
>
> 会话结束 钩子 have a 默认 超时 of 1.5 seconds. This 应用lies to 会话 退出, `/清
>
> ```bash 主题={空}
> CLAUDE_代码_会话结束_钩子_超时_MS=5000 claude
> ```

E合法ation

运行s when an MCP 服务器 请求s 用户 输入 mid-任务. By 默认, Claude 代码 显示s an 交互 dia日志 for the 用户 to respond. 钩子 can intercept this 请求 and respond 计划matic所有y,跳ping the dia日志 整个ly.

The 匹配er 字段 匹配es a收益st the MCP 服务器 名称.

E合法ation 输入

In 添加ition to the 常见 输入 字段s, E合法ation 钩子 接收 `mcp_服务器_名称`, `消息`, and 可选 `模式`, `url`, `e合法ation_id`, and `请求ed_模式` 字段s.

For 形式-模式 e合法ation (the 最多 常见 案例):

```json 主题={空}
{
"会话_id": "abc123",
"tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
"cwd": "/用户s/...",
"许可_模式": "默认",
"hook_事件_名称": "E合法ation",
"mcp_服务器_名称": "my-mcp-服务器",
"消息": "Pl容易 provide your credentials",
"模式": "形式",
"请求ed_模式": {
"类型": "对象",
"适当ties": {
"用户名称": { "类型": "字符串", "标题": "用户名称" }
```

```
}
}
}
```

```
For URL-模式 e合法ation (浏览器-基础d 认证):

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "许可_模式": "默认",
  "hook_事件_名称": "E合法ation",
  "mcp_服务器_名称": "my-mcp-服务器",
  "消息": "Pl容易 authenticate",
  "模式": "url",
  "url": "https://auth.示例.com/日志in"
}
```

E合法ation 输出

To respond 计划matic所有y without显示 the dia日志, 回报 a JSON 对象 with `hook特定输出`:

```
```json 主题={空}
{
"hook特定输出": {
"hook事件名称": "E合法ation",
"行动": "接受",
"满意": {
"用户名称": "alice"
}
}
}
```

```
| 字段      | 值s                            | 描述
| :-------- | :---------------------------- | :-------------------------
| `行动`   | `接受`, `下降`, `取消` | Whether to 接受, 下降, or 取消 the 请求
| `满意` | 对象                           | 形式 字段 值s to submit. Only 使用d

退出 代码 2 denies the e合法ation and 显示s stderr to the 用户.

### E合法ation结果

运行s 之后 a 用户 responds to an MCP e合法ation. 钩子 can观察, 修改, or 块 the

The 匹配er 字段 匹配es a收益st the MCP 服务器 名称.

#### E合法ation结果 输入

In 添加ition to the [常见 输入 字段s](#常见-输入-字段s), E合法ation结果 钩子 接收

```json  主题={空}
{
  "会话_id": "abc123",
  "tran脚本_路径": "/用户s/.../.claude/项目s/.../00893aaf-19fa-41d2-8238-13
  "cwd": "/用户s/...",
  "许可_模式": "默认",
  "hook_事件_名称": "E合法ation结果",
  "mcp_服务器_名称": "my-mcp-服务器",
  "行动": "接受",
  "满意": { "用户名称": "alice" },
  "模式": "形式",
  "e合法ation_id": "e合法-123"
}
```

E合法ation结果 输出

To 结束ride the 用户's 响应, 回报 a JSON 对象 with `hook特定输出`:
```

```json 主题={空}
{
"hook特定输出": {
"hook事件名称": "E合法ation结果",
"行动": "下降",
"满意": {}
}
}
```

```
| 字段      | 值s                            | 描述
| :-------- | :---------------------------- | :------------------------
| `行动`   | `接受`, `下降`, `取消` | 结束rides the 用户's 行动
| `满意` | 对象                          | 结束rides 形式 字段 值s. Only 有意义

退出 代码 2 块s the 响应,改变 the 有效 行动 to `下降`.

## 及时-基础d 钩子

In 添加ition to 命令 and HTTP 钩子, Claude 代码 支持s 及时-基础d 钩子 (`类型: "

事件s that 支持 所有 four hook 类型s (`命令`, `http`, `及时`, and `代理`):

* `许可请求`
* `工具使用后`
* `工具使用后失败`
* `工具使用前`
* `停止`
* `子代理顶部`
* `任务完成`
* `任务创建d`
* `用户及时Submit`

事件s that 支持 `命令` and `http` 钩子 but not `及时` or `代理`:

* `Config更改`
* `Cwd更改d`
* `E合法ation`
* `E合法ation结果`
* `文件更改d`
* `说明加载`
* `通知`
* `许可拒绝`
* `Post紧凑`
* `Pre紧凑`
```

```
* `会话结束`
* `停止失败`
* `子代理t艺术`
* `团队mate空闲`
* `工作树创建`
* `工作树移除`

`会话开始` 支持s only `命令` 钩子.

### How 及时-基础d 钩子 工作

Instead of执行 a Bash 命令, 及时-基础d 钩子:

1. 发送 the hook 输入 and your 及时 to a Claude 模型, Haiku by 默认
2. The LLM responds with 结构化 JSON包含 a 决定
3. Claude 代码 流程es the 决定 自动所有y

### 及时 hook 配置

设置 `类型` to `"及时"` and provide a `及时` 字符串 instead of a `命令`. 使用

This `停止` hook asks the LLM to evaluate whether 所有 任务s are 完成 之前允

```json  主题={空}
{
  "钩子": {
    "停止": [
      {
        "钩子": [
          {
            "类型": "及时",
            "及时": "Evaluate if Claude should 停止: $参数. 检查 if 所有 任务
          }
        ]
      }
    ]
```
```

```
  }
}
```

字段	必需	描述
`类型`	yes	Must be `"及时"`
`及时`	yes	The 及时 文本 to 发送 to the LLM. 使用 `$参数` as a placeh更旧 for the hook 输入 JSON. If `$参数` is not 现在, 输入 JSON is 应用结束 to the 及时
`模型`	no	模型 to 使用 for 评价. 默认s to a 快 模型
`超时`	no	超时 in seconds. 默认: 30

响应 模式

The LLM must respond with JSON包含:

```json 主题={空}
{
"ok": 真实 | 虚假,
"原因": "解释 for the 决定"
}
```

```
| 字段    | 描述                                                          |
| :------- | :------------------------------------------------------------ |
| `ok`     | `真实` 允许s the 行动, `虚假` pr事件s it                |
| `原因` | 必需 when `ok` is `虚假`. 解释 显示n to Claude |

### 示例: Multi-标准 停止 hook

This `停止` hook 使用s a 详情ed 及时 to 检查 three 条件 之前允许 Claude to 停止

```json  主题={空}
{
  "钩子": {
    "停止": [
      {
        "钩子": [
          {
            "类型": "及时",
            "及时": "You are评估 whether Claude should 停止工作. 上下文: $参数
            "超时": 30
          }
        ]
      }
    ]
  }
}
```

代理-基础d 钩子

代理-基础d 钩子 (类型: "代理") are 像 及时-基础d 钩子 but with multi-turn 工具 access. Instead of a single LLM c所有, an 代理 hook spawns a sub代理 that can 读取 文件, 搜索 代码, and检查 the 代码基础 to 验证 条件. 代理 钩子 支持 the 相同 事件s as 及时-基础d 钩子.
```

How 代理 钩子 工作

When an 代理 hook f怒s:

1. Claude 代码 spawns a sub代理 with your 及时 and the hook's JSON 输入
2. The sub代理 can 使用 工具 像 读取, Grep, and Glob to调查
3. 之后 上 to 50 turns, the sub代理 回报s a 结构化 `{ "ok": 真实/虚假 }` 决定
4. Claude 代码 流程es the 决定 the 相同 方式 as a 及时 hook

代理 钩子 are 有用 when 验证 requ怒s审查 实际 文件 or 测试 输出, not 公正评估 the hook 输入 数据 alone.

代理 hook 配置

设置 `类型` to `"代理"` and provide a `及时` 字符串. The 配置 字段s are the 相同 as 及时钩子, with a 长er 默认 超时:

字段	必需	描述
`类型`	yes	Must be `"代理"`
`及时`	yes	及时描述 what to 验证. 使用 `$参数` as a placeh更旧 for the hook 输入 JSON
`模型`	no	模型 to 使用. 默认s to a 快 模型
`超时`	no	超时 in seconds. 默认: 60

The 响应 模式 is the 相同 as 及时 钩子: `{ "ok": 真实 }` to 允许 or `{ "ok": 虚假, "原因": "..." }` to 块.

This `停止` hook verifies that 所有 单位 测试s pass 之前允许 Claude to finish:

```json 主题={空}
{
"钩子": {
"停止": [
{
"钩子": [
```

{
"类型": "代理",
"及时": "验证 that 所有 单位 测试s pass. 运行 the 测试 suite and 检查 the 结果s. $参数",
"超时": 120
}
]
}
]
}
}

```
## 运行 钩子 in the 背景

By 默认, 钩子 块 Claude's 执行 until they 完成. For 长-运行中 任务s 像 部署ment

### Con图 an 异步 hook

添加 `"异步": 真实` to a 命令 hook's 配置 to 运行 it in the 背景 without 阻塞 C

This hook 运行s a 测试 脚本 之后 每个 `写入` 工具 c所有. Claude 继续s工作 立即ly

```json  主题={空}
{
  "钩子": {
    "工具使用后": [
      {
        "匹配er": "写入",
        "钩子": [
          {
            "类型": "命令",
            "命令": "/路径/to/运行-测试s.sh",
            "异步": 真实,
            "超时": 120
          }
        ]
      }
    ]
  }
}
```

The `超时` 字段 设置s the 最大 时间 in seconds for the 背景 流程. If not 指定, 异步 钩子 使用 the 相同 10-微小 默认 as 同步 钩子.

How 异步 钩子 执行

When an 异步 hook f怒s, Claude 代码 启动s the hook 流程 and 立即ly 继续s without 等待ing for it to finish. The hook 接收s the 相同 JSON 输入 via stdin as a 同步 hook.
```

之后 the 背景 流程 退出s, if the hook produced a JSON 响应 with a `系统消息` or `添加itional上下文` 字段, that 满意 is 交付 to Claude as 上下文 on the 下一个 对话 turn.

异步 hook completion 通知s are s上pressed by 默认. To看见 them, 启用 详细 模式 with `Ctrl+O` or 启动 Claude 代码 with `--详细`.

示例: 运行 测试s 之后 文件 更改s

This hook 启动s a 测试 suite in the 背景 whenever Claude 写入s a 文件, then 报告s the 结果s 返回 to Claude when the 测试s finish. 保存 this 脚本 to `.claude/钩子/运行-测试s-异步.sh` in your 项目 and make it 可执行文件 with `chmod +x`:

```bash 主题={空}
#!/bin/bash

# 运行-测试s-异步.sh

# 读取 hook 输入 from stdin
输入=$(cat)
文件_路径=$(echo "$输入" | jq -r '.工具_输入.文件_路径 // 空')

# Only 运行 测试s for 来源 文件
if [[ "$文件_路径" != .ts && "$文件_路径" != .js ]]; then
退出 0
fi
```

运行 测试s and 报告 结果s via 系统消息

结果=$(npm 测试 2>&1)
退出_代码=$?

if [$退出_代码 -eq 0]; then
echo "{\"系统消息\": \"测试s passed 之后 编辑ing $文件_路径\"}"
else
echo "{\"系统消息\": \"测试s 失败 之后 编辑ing $文件_路径: $结果\"}"
fi

````
Then 添加 this 配置 to `.claude/设置.json` in your 项目 根. The `异步: 真实`

```json  主题={空}
{
  "钩子": {
    "工具使用后": [
      {
        "匹配er": "写入|编辑",
        "钩子": [
          {
            "类型": "命令",
            "命令": "\"$CLAUDE_项目_DIR\"/.claude/钩子/运行-测试s-异步.sh",
            "异步": 真实,
            "超时": 300
          }
        ]
      }
    ]
  }
}
````
````

限制

异步 钩子 have 几个 常量雨ts 比较d to 同步 钩子:

- Only `类型："命令"` 钩子 支持 `异步`. 及时-基础d 钩子 cannot 运行 异步ly.
- 异步 钩子 cannot 块 工具 c所有s or 回报 决定s. By the 时间 the hook 完成s, the triggering 行动 has al就绪 proceeded.
- Hook 输出 is 交付 on the 下一个 对话 turn. If the 会话 is 空闲, the 响应 等待s until the 下一个 用户 交互.
- 每个 执行 创建s a 单独 背景 流程. There is no ded上lication across mul提示le firings of the 相同 异步 hook.

安全 conside比率ns

Discl瞄准er

命令 钩子 运行 with your 系统 用户's 满 权限.

命令 钩子 执行 命令行 命令 with your 满 用户 权限. They can 修改, 删除, or access 任何 文件 your 用户 说明 can access. Re视图 and 测试 所有 hook 命令 之前添加 them to your 配置.

安全 最佳实践

Keep these 实践 in mind when写作 钩子:

- **验证 and sanitize 输入s**: never 信任 输入 数据 blindly
- **Al方式s quote 命令行 可变s**: 使用 `"$VAR"` not `$VAR`
- **块 路径 traversal**: 检查 for `..` in 文件 路径s
- **使用 absolute 路径s**: 规格ify 满 路径s for 脚本s, using `"$CLAUDE_项目_DIR"` for the 项目 根
- **跳 敏感 文件**: a空白 `.env`, `.Git/`, 键s, etc.

风ows 权力命令行 工具

On 风ows, you can 运行 个人 钩子 in 权力命令行 by设置 `"命令行"："权力命令行"` on a 命令 hook. 钩子 spawn 权力命令行 直接ly, so this 工作s 注意较少 of whether `CLAUDE_代码_使用_权力命令行_工具` is 设置. Claude 代码 auto-detects `pwsh.exe` (权力命令行 7+) with a 下降返回 to `权力命令行.exe` (5.1).

```json 主题={空}
{
"钩子": {
"工具使用后": [
{
"匹配er": "写入",
"钩子": [
{
"类型": "命令",
"命令行": "权力命令行",
"命令": "写入-Host '文件 written'"
}
]
}
]
}
}
```

```
## 调试 钩子

运行 `claude --调试` to看见 hook 执行 详情s,包括 which 钩子 匹配ed, their 退出
```

```文本  主题={空}
[调试]执行 钩子 for 工具使用后:写入
[调试] Found 1 hook 命令 to 执行
[调试]执行 hook 命令:  with 超时 600000ms
[调试] Hook 命令 完成 with 状态 0:
```

For 更多 颗粒 hook 匹配 详情s, 设置 `CLAUDE_代码_调试_日志_级别=详细` to看见 添加itional 日志 行s such as hook 匹配er counts and 查询 匹配.

For 故障排除 常见问题 像 钩子 not firing, infinite 停止 hook loops, or 配置 错误,看见 限制 and 故障排除 in the 指南.