

# Claude Code 社区指南

For updates and contributions, visit the [official Claude Code documentation](#)

AI Claude Code v1.0.98

Status Active

License Anthropic

板块	状态
Windows、Linux、MacOS 安装指南	✓
技巧和窍门	✓
MCP 概述及使用方法	✓
社区指南	✓
故障排除	✓
Claude Code 最优使用方法	✓

如何通过 Discord 使用 Claude Code [点击这里!](#)

[点击查看每日更新的 Claude 更新日志和新闻](#)

## 目录

快速链接: [安装](#) · [命令](#) · [快捷键](#) · [MCP](#) · [故障排除](#)

- [入门指南](#)
  - [快速开始](#)
  - [初始设置](#)
- [配置与环境](#)
  - [环境变量](#)
  - [配置文件](#)
- [命令与使用](#)
  - [Claude 命令](#)
  - [备忘单](#)
- [界面与输入](#)
  - [键盘快捷键](#)
  - [Vim 模式](#)

- [高级功能](#)
  - [子代理](#)
  - [MCP 集成](#)
  - [钩子系统](#)
- [安全与权限](#)
  - [危险模式](#)
  - [安全最佳实践](#)
- [自动化与集成](#)
  - [自动化与脚本](#)
  - [自动 PR 审查](#)
  - [问题分类](#)
- [帮助与故障排除](#)
  - [安装问题](#)
  - [MCP 问题](#)
  - [最佳实践](#)
- [第三方集成](#)
  - [DeepSeek 集成](#)

---

# 入门指南

---

Enable sound alerts when claude completes the task:

```
claude config set --global preferredNotifChannel terminal_bell
```

## 快速开始

---

[!TIP]

Send **claude** or **npx claude** in terminal to start the interface

Go to [Help & Troubleshooting](#) to fix issues...

```
# Node.js 18+★
/*通用方法      */ npm install -g @anthropic-ai/claude-code
-----
-----
-----

# Windows
/* 通过 CMD      */ npm install -g @anthropic-ai/claude-code
/* 通过 Powershell */ irm https://claude.ai/install.ps1 | iex
-----
-----
-----
```

```

# WSL/GIT
/* 通过终端          */ npm install -g @anthropic-ai/claude-code
/* 通过终端          */ curl -fsSL https://claude.ai/install.sh | bash

-----

# MacOS
/* brew install node && npm install -g @anthropic-ai/claude-code

-----

# Linux
/* 通过终端          */ sudo apt update && sudo apt install -y nodejs npm
/* 通过终端          */ npm install -g @anthropic-ai/claude-code
/* 通过终端          */ curl -fsSL https://claude.ai/install.sh | bash

-----

# Arch
/* 通过终端          */ yay -S claude-code*/

-----

# Docker
/* Windows (CMD)          */ docker run -it --rm -v "%cd%:/workspace" -e
ANTHROPIC_API_KEY="sk-your-key" node:20-slim bash -lc "npm i -g @anthropic-ai/claude-code && cd /workspace && claude"
/* macOS/Linux (bash/zsh)*/ docker run -it --rm -v "$PWD:/workspace" -e
ANTHROPIC_API_KEY="sk-your-key" node:20-slim bash -lc 'npm i -g @anthropic-ai/claude-code && cd /workspace && claude'
/* 无 bash 备用方案      */ docker run -it --rm -v "$PWD:/workspace" -e
ANTHROPIC_API_KEY="sk-your-key" node:20-slim sh -lc 'npm i -g @anthropic-ai/claude-code && cd /workspace && claude'

-----

# 检查 claude 是否正确安装
/* Linux          */ which claude
/* Windows        */ where claude
/* 通用           */ claude --version

-----

# 常用管理
/*claude config          */ 配置设置
/*claude mcp list        */ 设置 MCP 服务器, 你也可以用 add/remove 替换 "list"
/*claude /agents         */ 为不同任务配置/设置子代理
/*claude update          */ 更新到最新版本

```

[!Tip]

Open Project Via Terminal Into VS Code / Cursor

```
$ - cd /path/to/project
```

```
$ - code .
```

Make sure you have the **(Claude Code extension)** installed in your VS Code / Cursor

## 系统要求

- OS: macOS 10.15+, Ubuntu 20.04+/Debian 10+, or Windows 10/11 or WSL
- Hardware: 4GB RAM minimum 8GB+ recommended
- Software: Node.js 18+ or git 2.23+ (optional) & GitHub or GitLab CLI for PR workflows (optional)
- Internet: Connection for API calls
- Node.js 18+

## 初始设置

[!Tip]

Find API key from [Anthropic Console](#)

**Do NOT commit real keys use git-ignored files, OS key stores, or secret managers**

```
# Universal
/* start login process                */ claude /login
/* Setup long-lived authentication token */ claude setup-token
-----

# Windows
/* Set-api-key          */ set ANTHROPIC_API_KEY=sk-your-key-here-here
/* cmd-masked-check     */ echo OK: %ANTHROPIC_API_KEY:~0,8%***
/* Set-persistent-key   */ setx ANTHROPIC_API_KEY "sk-your-key-here-here"
/* cmd-unset-key        */ set ANTHROPIC_API_KEY=
-----

# Linux
/* Set-api-key          */ export ANTHROPIC_API_KEY="sk-your-key-here-here"
/* masked-check         */ echo "OK: ${ANTHROPIC_API_KEY:0:8}***"
/* remove-session       */ unset ANTHROPIC_API_KEY
-----

# Powershell
/* ps-会话              */ $env:ANTHROPIC_API_KEY = "sk-your-key-here-here"
/* ps-掩码检查          */ "OK: $($env:ANTHROPIC_API_KEY.Substring(0,8))***"
```

```
/* ps-持久化          */ [Environment]::SetEnvironmentVariable("ANTHROPIC_API_KEY","sk-  
your-key-here-here","User")  
/* ps-轮换            */ $env:ANTHROPIC_API_KEY = "sk-new-key"  
/* ps-移除            */ Remove-Item Env:\ANTHROPIC_API_KEY
```

```
-----  
-----  
# 其他...  
# 持久化-bash/*        */ echo 'export ANTHROPIC_API_KEY="sk-your-key-here-here"' >>  
~/.bashrc && source ~/.bashrc  
# 持久化-zsh /*        */ echo 'export ANTHROPIC_API_KEY="sk-your-key-here-here"' >>  
~/.zshrc && source ~/.zshrc  
# 持久化-fish/*        */ fish -lc 'set -Ux ANTHROPIC_API_KEY sk-your-key-here-here'  
-----  
-----
```

## 配置与环境

### 环境变量

You can also set any of these in settings.json under the "env" key for automatic application.

[!Important]

Windows Users replace `export` with `set` or `setx` for perm

```
# 环境切换（放在 ~/.bashrc 或 ~/.zshrc 中）  
export ANTHROPIC_API_KEY="sk-your-key-here-here"      # 作为 X-API-Key 头发送的 API 密钥  
（交互使用：/login）  
export ANTHROPIC_AUTH_TOKEN="my-auth-token"          # 自定义授权头；Claude 自动添加  
"Bearer " 前缀  
export ANTHROPIC_CUSTOM_HEADERS="X-Trace-Id: 12345"  # 额外的请求头（格式："Name: Value"）  
  
export ANTHROPIC_MODEL="claude-sonnet-4-20250514"    # 要使用的自定义模型名称  
export ANTHROPIC_DEFAULT_SONNET_MODEL="claude-sonnet-4-20250514" # 默认 Sonnet 模型别名  
export ANTHROPIC_DEFAULT_OPUS_MODEL="claude-opus-4-20250514"    # 默认 Opus 模型别名  
export ANTHROPIC_SMALL_FAST_MODEL="haiku-model"        # 用于后台任务的 Haiku 类  
模型（占位符）  
export ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION="REGION"    # 覆盖 Bedrock 上小型/快  
速模型的 AWS 区域（占位符）  
  
export AWS_BEARER_TOKEN_BEDROCK="bedrock_..."        # 用于身份验证的 Amazon Bedrock API  
密钥/令牌  
  
export BASH_DEFAULT_TIMEOUT_MS=60000                  # 长时间运行 bash 命令的默认超时（毫  
秒）  
export BASH_MAX_TIMEOUT_MS=300000                    # 长时间运行 bash 命令允许的最大超时  
（毫秒）  
export BASH_MAX_OUTPUT_LENGTH=20000                  # bash 输出中间截断前的最大字符数
```

<code>export CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR=1</code>	# (0 或 1) 每次 Bash 命令后返回原始项目目录
<code>export CLAUDE_CODE_API_KEY_HELPER_TTL_MS=600000</code> (毫秒)	# 使用 apiKeyHelper 时刷新凭据的间隔
<code>export CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL=1</code>	# (0 或 1) 跳过 IDE 扩展的自动安装
<code>export CLAUDE_CODE_MAX_OUTPUT_TOKENS=4096</code>	# 大多数请求的最大输出令牌数
 <code>export CLAUDE_CODE_USE_BEDROCK=1</code>	 # (0 或 1) 使用 Amazon Bedrock
<code>export CLAUDE_CODE_USE_VERTEX=0</code>	# (0 或 1) 使用 Google Vertex AI
<code>export CLAUDE_CODE_SKIP_BEDROCK_AUTH=0</code> 验证 (例如, 通过 LLM 网关)	# (0 或 1) 跳过 Bedrock 的 AWS 身份验证
<code>export CLAUDE_CODE_SKIP_VERTEX_AUTH=0</code> 身份验证 (例如, 通过 LLM 网关)	# (0 或 1) 跳过 Vertex 的 Google 身份验证
 <code>export CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC=0</code> 面的 <code>DISABLE_*</code> )	 # (0 或 1) 禁用非必要的流量 (等同于下面的 <code>DISABLE_*</code> )
<code>export CLAUDE_CODE_DISABLE_TERMINAL_TITLE=0</code>	# (0 或 1) 禁用自动终端标题更新
 <code>export DISABLE_AUTOUPDATER=0</code> <code>autoUpdates</code> 设置)	 # (0 或 1) 禁用自动更新 (覆盖 <code>autoUpdates</code> 设置)
<code>export DISABLE_BUG_COMMAND=0</code>	# (0 或 1) 禁用 <code>/bug</code> 命令
<code>export DISABLE_COST_WARNINGS=0</code>	# (0 或 1) 禁用成本警告消息
<code>export DISABLE_ERROR_REPORTING=0</code>	# (0 或 1) 退出 Sentry 错误报告
<code>export DISABLE_NON_ESSENTIAL_MODEL_CALLS=0</code>	# (0 或 1) 禁用非关键路径的模型调用
<code>export DISABLE_TELEMETRY=0</code>	# (0 或 1) 退出 Statsig 遥测
 <code>export HTTP_PROXY="http://proxy:8080"</code>	 # HTTP 代理服务器 URL
<code>export HTTPS_PROXY="https://proxy:8443"</code>	# HTTPS 代理服务器 URL
 <code>export MAX_THINKING_TOKENS=0</code> 考预算	 # (0 或 1 关闭/开启) 强制为模型设置思考预算
<code>export MCP_TIMEOUT=120000</code>	# MCP 服务器启动超时 (毫秒)
<code>export MCP_TOOL_TIMEOUT=60000</code>	# MCP 工具执行超时 (毫秒)
<code>export MAX_MCP_OUTPUT_TOKENS=25000</code> 25000)	# MCP 工具响应中允许的最大令牌数 (默认 25000)
 <code>export USE_BUILTIN_RIPGREP=0</code> 而不是打包的	 # (0 或 1) 设置为 0 使用系统安装的 <code>rg</code> 而不是打包的
 <code>export VERTEX_REGION_CLAUDE_3_5_HAIKU="REGION"</code> 区域覆盖	 # Vertex AI 上 Claude 3.5 Haiku 的区域覆盖
<code>export VERTEX_REGION_CLAUDE_3_5_SONNET="REGION"</code> 的区域覆盖	# Vertex AI 上 Claude 3.5 Sonnet 的区域覆盖
<code>export VERTEX_REGION_CLAUDE_3_7_SONNET="REGION"</code> 的区域覆盖	# Vertex AI 上 Claude 3.7 Sonnet 的区域覆盖
<code>export VERTEX_REGION_CLAUDE_4_0_OPUS="REGION"</code> 区域覆盖	# Vertex AI 上 Claude 4.0 Opus 的区域覆盖

```
export VERTEX_REGION_CLAUDE_4_0_SONNET="REGION" # Vertex AI 上 Claude 4.0 Sonnet 的区域覆盖
export VERTEX_REGION_CLAUDE_4_1_OPUS="REGION" # Vertex AI 上 Claude 4.1 Opus 的区域覆盖
```

## 全局配置选项

```
claude config set -g theme dark # 主题: dark | light | light-daltonized | dark-daltonized
claude config set -g preferredNotifChannel iterm2_with_bell # 通知渠道: iterm2 | iterm2_with_bell | terminal_bell | notifications_disabled
claude config set -g autoUpdates true # 自动下载和安装更新 (重启时使用)
claude config set -g verbose true # 显示完整的 bash/命令输出

claude config set -g includeCoAuthoredBy false # 在 git 提交/PR 中省略 "co-authored-by Claude"
claude config set -g forceLoginMethod console # 限制登录到 Anthropic Console (API 计费)
claude config set -g model "claude-3-5-sonnet-20241022" # 默认模型覆盖
claude config set -g statusLine '{"type":"command","command":"~/\.claude/statusline.sh"}' # 自定义状态栏

claude config set -g enableAllProjectMcpServers true # 自动批准来自 .mcp.json 的所有 MCP 服务器
claude config set -g enabledMcpjsonServers '["memory","github"]' # 批准特定的 MCP 服务器
claude config set -g disabledMcpjsonServers '["filesystem"]' # 拒绝特定的 MCP 服务器
```

[!Important]

Windows Users replace `export` with `set`

```
export DISABLE_AUTOUPDATER=1 # 全局关闭自动更新 (覆盖 autoUpdates)
export CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC=1 # 禁用非必要的流量 (等同于下面的 DISABLE_* 切换)
export DISABLE_TELEMETRY=1 # 退出 Statsig 遥测
export DISABLE_ERROR_REPORTING=1 # 退出 Sentry 错误报告
export DISABLE_BUG_COMMAND=1 # 禁用 /bug 命令
export DISABLE_COST_WARNINGS=0 # 保持成本警告 (设置为 1 隐藏)
export DISABLE_NON_ESSENTIAL_MODEL_CALLS=1 # 跳过非关键模型调用 (装饰性文本等)

export CLAUDE_CODE_DISABLE_TERMINAL_TITLE=1 # 停止自动更新终端标题
export CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR=1 # 每次 Bash 命令后返回原始项目目录
export CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL=1 # 跳过 IDE 扩展的自动安装
export USE_BUILTIN_RIPGREP=0 # 使用系统的 'rg' (0) 而不是打包的 'rg'

export MAX_THINKING_TOKENS=0 # (0 或 1 关闭/开启) 强制为模型设置思考预算
export CLAUDE_CODE_MAX_OUTPUT_TOKENS=4096 # 限制典型响应大小 (示例值)
```

```
export HTTP_PROXY="http://proxy.company:8080"      # HTTP 代理（如果需要）
export HTTPS_PROXY="https://proxy.company:8443"    # HTTPS 代理（如果需要）
```

## 配置文件

（内存类型） Claude Code 提供四个分层结构的内存位置，每个都有不同的用途：

内存类型	位置	用途	使用案例示例	共享对象
企业策略	macOS: <code>/Library/Application Support/ClaudeCode/CLAUDE.md</code> Linux: <code>/etc/claude-code/CLAUDE.md</code> Windows: <code>C:\ProgramData\ClaudeCode\CLAUDE.md</code>	由 IT/DevOps 管理的组织范围指令	公司编码标准、安全策略、合规要求	组织中的所有用户
项目内存	<code>./CLAUDE.md</code>	项目的团队共享指令	项目架构、编码标准、通用工作流程	通过源代码控制的团队成员
用户内存	<code>~/.claude/CLAUDE.md</code>	所有项目的个人偏好设置	代码样式偏好、个人工具快捷方式	仅限你（所有项目）
项目内存（本地）	<code>./CLAUDE.local.md</code>	项目特定的个人偏好设置	<i>（已弃用，见下文）</i> 你的沙盒 URL、首选测试数据	仅限你（当前项目）

所有内存文件在 Claude Code 启动时会自动加载到上下文中。层次结构中较高的文件优先加载，为更具体的内存提供基础。

## 命令与使用

### Claude 命令



命令	用途
<code>/add-dir</code>	添加附加的工作目录
<code>/agents</code>	管理用于专门任务的自定义 AI 子代理
<code>/bug</code>	报告错误（将对话发送给 Anthropic）
<code>/clear</code>	清除对话历史
<code>/compact [instructions]</code>	压缩对话，可选聚焦指令
<code>/config</code>	查看/修改配置
<code>/cost</code>	显示令牌使用统计和计费信息
<code>/doctor</code>	检查你的 Claude Code 安装的健康状态
<code>/help</code>	获取使用帮助
<code>/init</code>	使用 CLAUDE.md 指南初始化项目
<code>/login</code>	切换 Anthropic 账户
<code>/logout</code>	从你的 Anthropic 账户登出
<code>/mcp</code>	管理 MCP 服务器连接和 OAuth 身份验证
<code>/memory</code>	编辑 CLAUDE.md 内存文件
<code>/model</code>	选择或更改 AI 模型
<code>/permissions</code>	查看或更新工具权限
<code>/pr_comments</code>	查看拉取请求评论
<code>/review</code>	请求代码审查
<code>/status</code>	查看账户和系统状态
<code>/terminal-setup</code>	为换行安装 Shift+Enter 键绑定（仅限 iTerm2 和 VSCode）
<code>/vim</code>	进入 vim 模式，交替插入和命令模式

## 命令行标志

标志 / 命令	描述	示例
<code>-d, --debug</code>	启用调试模式（显示详细的调试输出）。	<code>claude -d -p "query"</code>
<code>--mcp-debug</code>	[已弃用] MCP 调试模式（显示 MCP 服务器错误）。请使用 <code>--debug</code> 代替。	<code>claude --mcp-debug</code>
<code>--verbose</code>	覆盖配置中的详细模式设置（显示扩展日志 / 逐步输出）。	<code>claude --verbose</code>
<code>-p, --print</code>	打印响应并退出（用于管道输出）。	<code>claude -p "query"</code>
<code>--output-format &lt;format&gt;</code>	输出格式（仅适用于 <code>--print</code> ）： <code>text</code> （默认）、 <code>json</code> （单个结果）或 <code>stream-json</code> （实时流）。	<code>claude -p "query" --output-format json</code>
<code>--input-format &lt;format&gt;</code>	输入格式（仅适用于 <code>--print</code> ）： <code>text</code> （默认）或 <code>stream-json</code> （实时流输入）。	<code>claude -p --output-format stream-json --input-format stream-json</code>
<code>--replay-user-messages</code>	将用户消息从 stdin 重新发送到 stdout 以进行确认 — 仅适用于 <code>--input-format=stream-json</code> 和 <code>--output-format=stream-json</code> 。	<code>claude --input-format stream-json --output-format stream-json --replay-user-messages</code>
<code>--allowedTools, --allowed-tools &lt;tools...&gt;</code>	允许的工具名称的逗号/空格分隔列表（例如 <code>"Bash(git:*)" Edit"</code> ）。	<code>--allowed-tools "Bash(git:*)" Edit"</code>
<code>--disallowedTools, --disallowed-tools &lt;tools...&gt;</code>	拒绝的工具名称的逗号/空格分隔列表（例如 <code>"Bash(git:*)" Edit"</code> ）。	<code>--disallowed-tools "Edit"</code>
<code>--mcp-config &lt;configs...&gt;</code>	从 JSON 文件或字符串加载 MCP 服务器（空格分隔）。	<code>claude --mcp-config ./mcp-servers.json</code>
<code>--strict-mcp-config</code>	仅使用来自 <code>--mcp-config</code> 的 MCP 服务器，忽略其他 MCP 配置。	<code>claude --mcp-config ./a.json --strict-mcp-config</code>
<code>--append-system-prompt &lt;prompt&gt;</code>	在默认系统提示后附加系统提示（在打印模式中有用）。	<code>claude -p --append-system-prompt "Do X then Y"</code>
<code>--permission-mode &lt;mode&gt;</code>	会话的权限模式（选项包括 <code>acceptEdits</code> 、 <code>bypassPermissions</code> 、 <code>default</code> 、 <code>plan</code> ）。	<code>claude --permission-mode plan</code>
<code>--permission-prompt-tool &lt;tool&gt;</code>	指定一个 MCP 工具来在非交互模式中处理权限提示。	<code>claude -p --permission-prompt-tool mcp_auth_tool "query"</code>
<code>--fallback-model &lt;model&gt;</code>	当默认模型超载时，启用自动备用到指定模型（注：仅适用于 <code>--print</code> ）。	<code>claude -p --fallback-model claude-haiku-20240307 "query"</code>
<code>--model &lt;model&gt;</code>	当前会话的模型。接受别名如 <code>sonnet / opus</code> 或完整模型名称（例如 <code>claude-sonnet-4-20250514</code> ）。	<code>claude --model sonnet</code>
<code>--settings &lt;file-or-json&gt;</code>	从 JSON 文件或 JSON 字符串加载附加设置。	<code>claude --settings ./settings.json</code>
<code>--add-dir &lt;directories...&gt;</code>	允许工具访问的附加目录。	<code>claude --add-dir ../apps ../lib</code>
<code>--ide</code>	如果有且仅有一个有效的 IDE 可用，则在启动时自动连接到 IDE。	<code>claude --ide</code>
<code>-c, --continue</code>	继续当前目录中最近的对话。	<code>claude --continue</code>
<code>-r, --resume [sessionId]</code>	恢复对话；提供会话 ID 或交互式选择一个。	<code>claude -r "abc123"</code>
<code>--session-id &lt;uuid&gt;</code>	为对话使用特定的会话 ID（必须是有效的 UUID）。	<code>claude --session-id 123e4567-e89b-12d3-a456-426614174000</code>
<code>--dangerously-skip-permissions</code>	绕过所有权限检查（仅限可信的沙盒）。	<code>claude --dangerously-skip-permissions</code>
<code>-v, --version</code>	显示已安装的 <code>claude</code> CLI 版本。	<code>claude --version</code>
<code>-h, --help</code>	显示帮助 / 用法。	<code>claude --help</code>

The `--output-format json` flag is particularly useful for scripting and automation, allowing you to parse Claude's responses programmatically.

# 备忘单

```
## Claude 备忘单
# 基本 / 交互式

claude                                # 启动交互式 REPL
claude "explain this project"         # 使用提示启动 REPL
claude -p "summarize README.md"       # 非交互式打印模式 (SDK 支持)
cat logs.txt | claude -p "explain"    # 将输入传递给 Claude 并退出
claude -c                             # 继续最近的对话 (--continue 的别名)
claude -r "<session-id>" "finish this" # 通过 ID 恢复特定会话 (--resume 的别名)
claude --model claude-sonnet-4-20250514 # 为此次运行选择模型
claude --max-turns 3 -p "lint this"   # 在打印模式中限制代理轮次
claude --replay-user-messages         # 将用户消息重放到 stdout 用于调试 / SDK 工作流

# 更新与安装

claude update                         # 手动更新 Claude Code
claude doctor                         # 诊断安装/版本和设置
claude install                       # 启动原生二进制安装程序 (测试版)
claude migrate-installer             # 从全局 npm 迁移到本地安装程序

# 配置: 交互式向导 + 直接操作

claude config                         # 交互式配置向导
claude config get <key>               # 获取值 (例如: claude config get theme)
claude config set <key> <val>         # 设置值 (例如: claude config set theme dark)
claude config add <key> <vals...>    # 追加到数组类型键 (例如: claude config add env
DEV=1)
claude config remove <key> <vals...> # 从列表类型键中删除项目
claude config list                   # 显示项目的所有当前设置 (默认为项目范围)

# 项目范围设置示例

claude config set model "claude-3-5-sonnet-20241022" # 为此项目覆盖默认模型
claude config set includeCoAuthoredBy false          # 禁用 git/PR 中的 "co-authored-
by Claude" 署名行
claude config set forceLoginMethod claudeai          # 限制登录流程: claudeai | console
claude config set enableAllProjectMcpServers true    # 自动批准来自 .mcp.json 的所有 MCP
服务器
claude config set defaultMode "acceptEdits"         # 设置默认权限模式
claude config set disableBypassPermissionsMode disable # 阻止绕过权限模式 (示例键)

# 管理列表设置 (项目范围)

claude config add enabledMcpjsonServers github      # 批准来自 .mcp.json 的特定 MCP 服
务器
claude config add enabledMcpjsonServers memory      # 添加另一个
claude config remove enabledMcpjsonServers memory   # 删除一个条目
claude config add disabledMcpjsonServers filesystem # 明确拒绝特定的 MCP 服务器

# 全局范围 (使用 -g 或 --global)

claude config set -g autoUpdates false              # 全局关闭自动更新
```

```

claude config set --global preferredNotifChannel iterm2_with_bell
claude config set -g theme dark # 主题: dark | light | light-
daltonized | dark-daltonized
claude config set -g verbose true # 在所有地方显示完整的 bash/命令输出
claude config get -g theme # 确认全局值

# MCP (模型上下文协议) 管理
claude mcp # 启动 MCP 向导/配置 MCP 服务器
claude mcp list # 列出配置的 MCP 服务器
claude mcp get <name> # 显示服务器详细信息
claude mcp remove <name> # 删除服务器
claude mcp add <name> <command> [args...] # 添加本地 stdio 服务器
claude mcp add --transport sse <name> <url> # 添加远程 SSE 服务器
claude mcp add --transport http <name> <url> # 添加远程 HTTP 服务器
claude mcp add <name> --env KEY=VALUE -- <cmd> [args...] # 向服务器命令传递环境变量
claude mcp add --transport sse private-api https://api.example/mcp \
  --header "Authorization: Bearer TOKEN" # 添加带认证头的服务器
claude mcp add-json <name> '<json>' # 通过 JSON 数据添加服务器
claude mcp add-from-claude-desktop # 从 Claude Desktop 导入服务器
claude mcp reset-project-choices # 重置项目 .mcp.json 服务器的批准状态
claude mcp serve # 将 Claude Code 本身作为 MCP stdio 服务器运行

# 其他有用的标志 (打印/SDK 模式)
claude --add-dir ../apps ../lib # 添加额外的工作目录
claude --allowedTools "Bash(git log:*)" "Read" # 允许列出的工具无需权限提示
claude --disallowedTools "Edit" # 禁止列出的工具无需权限提示
claude --append-system-prompt "Custom instruction" # 追加到系统提示 (仅适用于 -p)
claude -p "query" --output-format json --input-format stream-json # 控制脚本的 IO 格式
claude --verbose # 详细日志记录 (逐轮)
claude --dangerously-skip-permissions # 跳过权限提示 (谨慎使用)

# 快速验证/注意事项
# - 'claude config' 默认为项目范围; 使用 -g/--global 影响所有项目。
# - 设置优先级: 企业 > CLI 参数 > 本地项目 > 共享项目 > 用户 (~/.claude) 。
# - 仅对列表类型键使用 'add'/'remove' (例如: enabledMcpjsonServers) 。
# - CLI 参考和发布说明是标志和最新添加功能的权威来源。

```

## 界面与输入

### 键盘快捷键

快捷键	描述	上下文
<code>Ctrl+C</code>	取消当前输入或生成	标准中断
<code>Ctrl+D</code>	退出 Claude Code 会话	EOF 信号
<code>Ctrl+L</code>	清除终端屏幕	保持对话历史
<code>Up/Down arrows</code>	导航命令历史	回顾之前的输入
<code>Esc + Esc</code>	编辑上一条消息	双击 Escape 进行修改

## Multiline Input

方法	快捷键	上下文
快速转义	<code>\ + Enter</code>	在所有终端中工作
macOS 默认	<code>Option+Enter</code>	在 macOS 上的默认设置
终端设置	<code>Shift+Enter</code>	在 <code>/terminal-setup</code> 之后
控制序列	<code>Ctrl+J</code>	用于多行的换行字符
粘贴模式	直接粘贴	用于代码块、日志

## Quick Commands

快捷键	描述	备注
开头的 <code>#</code>	内存快捷键添加到 CLAUDE.md	提示选择文件
开头的 <code>/</code>	斜杠命令	

## Vim 模式

[!Note]

Enable vim-style editing with `/vim` command or configure permanently via `/config`.

## Vim Mode Switching

命令	操作	来源模式
<code>Esc</code>	进入 NORMAL 模式	INSERT
<code>i</code>	在光标前插入	NORMAL
<code>I</code>	在行首插入	NORMAL
<code>a</code>	在光标后插入	NORMAL
<code>A</code>	在行尾插入	NORMAL
<code>o</code>	在下方开新行	NORMAL
<code>O</code>	在上方开新行	NORMAL

## Vim Navigation

命令	操作
<code>h / j / k / l</code>	左/下/上/右移动
<code>w</code>	下一个单词
<code>e</code>	单词结尾
<code>b</code>	上一个单词
<code>0</code>	行首
<code>\$</code>	行尾
<code>^</code>	首个非空字符
<code>gg</code>	输入开始
<code>G</code>	输入结束

## Vim Editing

命令	操作
x	删除字符
dd	删除行
D	删除到行尾
dw / de / db	删除单词/到尾/向后
cc	修改行
C	修改到行尾
cw / ce / cb	修改单词/到尾/向后
.	重复上次修改

[!Tip]  
Configure your preferred line break behavior in terminal settings. Run `/terminal-setup` to install Shift+Enter binding for iTerm2 and VS Code terminals.

## 命令历史

Claude Code maintains command history for the current session:

```
* History is stored per working directory
* Cleared with `/clear` command
* Use Up/Down arrows to navigate (see keyboard shortcuts above)
* **Ctrl+R**: Reverse search through history (if supported by terminal)
* **Note**: History expansion (`!`) is disabled by default
```

## 高级功能

### 思考关键词

[!Note]  
通过在提示中添加以下关键词之一，为 Claude 提供额外的预答案规划时间。  
顺序（从低到高） token 消耗量

think

-----> Lowest

think hard

think harder

ultrathink

-----> Highest

## 这会让 Claude 花费更多时间：

1. 规划解决方案
2. 分解步骤
3. 权衡备选方案/权衡利弊
4. 检查约束条件和边界情况

更高级别通常会增加延迟和token 使用量，请选择有效的最小级别。

示例

```
# 小幅提升
claude -p "Think. 概述重构认证模块的计划。"

# 中等提升
claude -p "Think harder. 起草从 REST 到 gRPC 的迁移计划。"

# 最大提升
claude -p "Ultrathink. 提出修复不稳定支付测试并添加防护措施的分步策略。"
```

## 子代理

子代理是专门构建的助手，具有它们自己的提示、工具和隔离的上下文窗口。将其视为您每个仓库组合的“混合专家”。

### 何时使用

- 您需要高信号响应（计划、审查、差异）而无支线任务。
- 您希望与代码库一起进行版本控制的提示和工具策略。
- 您在 PR 驱动的团队中工作，希望按角色进行限定范围的编辑。

## 每个子代理都有自己的上下文

### 阵容设计规则

- 为每个代理定义一个明确的职责。
- 保持该角色所需的最小工具集。
- 对于分析/审查任务，优先使用只读代理。
- 给尽可能少的代理编辑权限。



说明：终端中的代理选择 UI。

## 配置代理

将代理保持在项目中，这样它们会与仓库一起进行版本控制，并通过 PR 演进。

## 快速开始

更新 CLI 并打开代理面板

```
claude update  
/agents
```

## 创建核心代理

- **planner**（只读）：将功能/问题转化为小的、可测试的任务；输出任务列表或 plan.md。
- **codegen**（可编辑）：实现任务；限制于 `src/` + `tests/`。
- **tester**（只读或仅补丁）：编写一个失败测试或最小复现。
- **reviewer**（只读）：留下结构化的审查意见；从不编辑。
- **docs**（可编辑）：仅更新 `README.md` / `docs/`。

**策略提示：**对于可编辑的代理，优先使用**补丁输出**，这样更改会通过您的正常 Git 工作流落地。

说明：仅选择代理真正需要的工具（例如，咨询访问 vs 编辑访问）。

# 示例提示

保持提示简短、可测试且特定于仓库。将它们检入 `agents/`：

说明：测试覆盖率分析器代理的示例提示。

## tester.prompt.md（示例）

角色：为我描述的特定场景编写一个单一、专注的失败测试。  
范围：仅在 `tests/` 下创建/修改测试。不要更改 `src/`。  
输出：简要的理由 + 统一差异或补丁。  
如果场景不清楚，请只问一个澄清问题。

# 预期输出

您的测试器代理应该产生一个小的差异或补丁加上简短的理由：

说明：测试覆盖率分析器代理的示例响应。

# 为什么这种转变很重要

## 运营益处

- 更少的上下文切换：您保持在一种思维模式中；代理做其余工作。
- 更清洁的 PR：狭窄的提示 + 有限的工具 → 更小、可审查的差异。
- 更少的退化：测试器/审查器代理在合并前捕获缺口。
- 可重复性：提示 + 策略住在仓库中，并与分支一起移动。

## 安全与治理

- 按路径限制写入访问（例如，`src/`、`tests/`、`docs/`）。
- 对于高风险区域，优先使用只读分析。
- 将助手输出作为补丁记录/提交，以便审计。

# 思维转变

## 做

- 将代理视为具有工作描述的队友。
- 从只读开始；最后授予写入访问权限。
- 将提示保持在版本控制中，并通过 PR 迭代。

## 不要

- 要求一个代理在单次轮次中计划、编码和测试。
- 给予全面的写入权限。
- 当您要求一个测试时，接受多文件差异。

# MCP 集成

## 理解 MCP（模型上下文协议）

### MCP 是什么？

MCP 通过连接到外部服务、数据库、API 和工具（文件系统、Puppeteer、GitHub、Context7 等）来扩展 Claude 的能力。

### MCP 架构：



# MCP 设置与配置

## 基本 MCP 命令

```
claude mcp                # 交互式 MCP 配置
claude mcp list            # 列出配置的服务器
claude mcp add <name> <cmd> # 添加新服务器
claude mcp remove <name>  # 删除服务器
```

## MCP 配置文件位置

```
~/.claude.json    # 全局文件
`.mcp.json`       # 项目范围的服务器存储在项目根目录的文件中
```

# 流行的 MCP 服务器

## 开发工具

```
# npm install -g git-mcp-server

# claude mcp add git "git-mcp-server"
# claude mcp add github "github-mcp-server --token $GITHUB_TOKEN"
```

## 数据库集成

```
npm install -g postgres-mcp-server
npm install -g mysql-mcp-server
npm install -g sqlite-mcp-server

# 设置示例可能如下所示：
# export POSTGRES_URL="postgresql://user:password@localhost:5432/mydb"
# claude mcp add postgres "postgres-mcp-server --url $POSTGRES_URL"
```

## MCP 工具权限

```
# 允许特定的 MCP 工具
claude --allowedTools "mcp__git__commit,mcp__git__push"

# 允许来自特定服务器的所有工具
claude --allowedTools "mcp__postgres__*"

# 与内置工具结合
claude --allowedTools "Edit,View,mcp__git__*"
```

# 钩子系统

本页提供在 Claude Code 中实现钩子的参考文档。

要获取带有示例的快速入门指南，请参阅 [Claude Code 钩子入门](#)。

## 配置

Claude Code 钩子在您的 [设置文件](#) 中配置：

- `~/.claude/settings.json` - 用户设置
- `.claude/settings.json` - 项目设置
- `.claude/settings.local.json` - 本地项目设置（不提交）
- 企业管理的策略设置

## 结构

钩子按匹配器组织，每个匹配器可以有多个钩子：

```
{
  "hooks": {
    "EventName": [
      {
        "matcher": "ToolPattern",
        "hooks": [
          {
            "type": "command",
            "command": "your-command-here"
          }
        ]
      }
    ]
  }
}
```

- **matcher:** 匹配工具名称的模式，区分大小写（仅适用于 `PreToolUse` 和 `PostToolUse`）
  - 简单字符串精确匹配： `Write` 仅匹配 `Write` 工具
  - 支持正则表达式： `Edit|Write` 或 `Notebook.*`
  - 使用 `*` 匹配所有工具。您也可以使用空字符串（`""`）或留空 `matcher`。
- **hooks:** 当模式匹配时执行的命令数组
  - `type`: 目前仅支持 `"command"`
  - `command`: 要执行的 bash 命令（可以使用 `$CLAUDE_PROJECT_DIR` 环境变量）
  - `timeout`: （可选）在取消该特定命令之前，命令应该运行多久（以秒为单位）。

对于不使用匹配器的事件（如 `UserPromptSubmit`、`Notification`、`Stop` 和 `SubagentStop`），您可以省略 `matcher` 字段：

```
{
  "hooks": {
    "UserPromptSubmit": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/prompt-validator.py"
          }
        ]
      }
    ]
  }
}
```

## 项目特定的钩子脚本

您可以使用环境变量 `CLAUDE_PROJECT_DIR`（仅在 Claude Code 生成钩子命令时可用）来引用存储在项目中的脚本，确保它们无论 Claude 的当前目录如何都能正常工作：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "$CLAUDE_PROJECT_DIR/.claude/hooks/check-style.sh"
          }
        ]
      }
    ]
  }
}
```

## 钩子事件

### 工具使用前

在 Claude 创建工具参数之后、处理工具调用之前运行。

常用匹配器：

- `Task` - 子代理任务（参见 [子代理文档](#)）

- `Bash` - Shell 命令
- `Glob` - 文件模式匹配
- `Grep` - 内容搜索
- `Read` - 文件读取
- `Edit`, `MultiEdit` - 文件编辑
- `Write` - 文件写入
- `WebFetch`, `WebSearch` - Web 操作

## 工具使用后

在工具成功完成后立即运行。

识别与 `PreToolUse` 相同的匹配器值。

## 通知

在 Claude Code 发送通知时运行。在以下情况下会发送通知：

1. Claude 需要您的权限来使用工具。示例：“Claude 需要您的权限来使用 `Bash`”
2. 提示输入已空闲至少 60 秒。“Claude 正在等待您的输入”

## 用户提示提交

在用户提交提示时、Claude 处理之前运行。这允许您根据提示/对话添加额外上下文、验证提示或阻止某些类型的提示。

## 停止

在主 Claude Code 代理完成响应时运行。如果停止是由于用户中断导致的，则不会运行。

## 子代理停止

在 Claude Code 子代理（Task 工具调用）完成响应时运行。

## 压缩前

在 Claude Code 即将运行压缩操作之前运行。

匹配器：

- `manual` - 从 `/compact` 调用
- `auto` - 从自动压缩调用（由于上下文窗口已满）

## 会话开始

在 Claude Code 启动新会话或恢复现有会话时运行（目前在底层会启动新会话）。对于加载开发上下文（如现有问题或代码库的最近更改）很有用。

匹配器：

- `startup` - 从启动调用
- `resume` - 从 `--resume`、`--continue` 或 `/resume` 调用
- `clear` - 从 `/clear` 调用

## 钩子输入

钩子通过 stdin 接收包含会话信息和事件特定数据的 JSON 数据：

```
{
  // 公共字段
  session_id: string
  transcript_path: string // 对话 JSON 的路径
  cwd: string             // 调用钩子时的当前工作目录

  // 事件特定字段
  hook_event_name: string
  ...
}
```

## 工具使用前输入

`tool_input` 的确切模式取决于工具。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "PreToolUse",
  "tool_name": "Write",
  "tool_input": {
    "file_path": "/path/to/file.txt",
    "content": "file content"
  }
}
```



## 工具使用后输入

`tool_input` 和 `tool_response` 的确切模式取决于工具。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "PostToolUse",
  "tool_name": "Write",
  "tool_input": {
    "file_path": "/path/to/file.txt",
    "content": "file content"
  },
  "tool_response": {
    "filePath": "/path/to/file.txt",
    "success": true
  }
}
```

## 通知输入

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "Notification",
  "message": "Task completed successfully"
}
```

## 用户提示提交输入

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "UserPromptSubmit",
  "prompt": "Write a function to calculate the factorial of a number"
}
```

## 停止和子代理停止输入

当 Claude Code 已经由于停止钩子而继续运行时，`stop_hook_active` 为 `true`。检查此值或处理记录以防止 Claude Code 无限期运行。

```
{
  "session_id": "abc123",
  "transcript_path": "~/.claude/projects/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "hook_event_name": "Stop",
  "stop_hook_active": true
}
```

## 压缩前输入

对于 `manual`，`custom_instructions` 来自用户传递给 `/compact` 的内容。对于 `auto`，`custom_instructions` 为空。

```
{
  "session_id": "abc123",
  "transcript_path": "~/.claude/projects/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "hook_event_name": "PreCompact",
  "trigger": "manual",
  "custom_instructions": ""
}
```

## 会话开始输入

```
{
  "session_id": "abc123",
  "transcript_path": "~/.claude/projects/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "hook_event_name": "SessionStart",
  "source": "startup"
}
```

## 钩子输出

钩子有两种方式将输出返回给 Claude Code。输出传达是否阻止以及应该向 Claude 和用户显示的任何反馈。

### 简单：退出代码

钩子通过退出代码、`stdout` 和 `stderr` 来传达状态：

- **退出代码 0**：成功。`stdout` 在记录模式中显示给用户（CTRL-R），除了 `UserPromptSubmit` 和 `SessionStart`，其 `stdout` 被添加到上下文中。

- **退出代码 2：**阻止错误。`stderr` 被反馈给 Claude 自动处理。  
请参阅下面的每个钩子事件行为。
- **其他退出代码：**非阻止错误。`stderr` 显示给用户，执行继续。

提醒：如果退出代码为 0，Claude Code 不会看到 `stdout`，除了 `UserPromptSubmit` 钩子，其中 `stdout` 被注入为上下文。

### 退出代码 2 行为

钩子事件	行为
<code>PreToolUse</code>	阻止工具调用，将 <code>stderr</code> 显示给 Claude
<code>PostToolUse</code>	将 <code>stderr</code> 显示给 Claude（工具已运行）
<code>Notification</code>	不适用，仅将 <code>stderr</code> 显示给用户
<code>UserPromptSubmit</code>	阻止提示处理，清除提示，仅将 <code>stderr</code> 显示给用户
<code>Stop</code>	阻止停止，将 <code>stderr</code> 显示给 Claude
<code>SubagentStop</code>	阻止停止，将 <code>stderr</code> 显示给 Claude 子代理
<code>PreCompact</code>	不适用，仅将 <code>stderr</code> 显示给用户
<code>SessionStart</code>	不适用，仅将 <code>stderr</code> 显示给用户

## 高级：JSON 输出

钩子可以在 `stdout` 中返回结构化的 JSON 以进行更复杂的控制：

### 公共 JSON 字段

所有钩子类型都可以包含这些可选字段：

```
{
  "continue": true, // Claude 是否应在钩子执行后继续（默认：true）
  "stopReason": "string" // 当 continue 为 false 时显示的消息
  "suppressOutput": true, // 在记录模式中隐藏 stdout（默认：false）
}
```

如果 `continue` 为 `false`，Claude 在钩子运行后停止处理。

- 对于 `PreToolUse`，这与 `"permissionDecision": "deny"` 不同，后者仅阻止特定的工具调用并向 Claude 提供自动反馈。
- 对于 `PostToolUse`，这与 `"decision": "block"` 不同，后者向 Claude 提供自动化反馈。
- 对于 `UserPromptSubmit`，这会阻止提示被处理。

- 对于 `stop` 和 `SubagentStop`，这优先于任何 `"decision": "block"` 输出。
- 在所有情况下，`"continue" = false` 优先于任何 `"decision": "block"` 输出。

`stopReason` 伴随 `continue` 提供显示给用户的原因，不显示给 Claude。

### **PreToolUse** 决策控制

`PreToolUse` 钩子可以控制工具调用是否继续。

- `"allow"` 绕过权限系统。 `permissionDecisionReason` 显示给用户但不显示给 Claude。（已弃用的 `"approve"` 值 + `reason` 具有相同行为。）
- `"deny"` 阻止工具调用执行。 `permissionDecisionReason` 显示给 Claude。（`"block"` 值 + `reason` 具有相同行为。）
- `"ask"` 要求用户在 UI 中确认工具调用。  
`permissionDecisionReason` 显示给用户但不显示给 Claude。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow" | "deny" | "ask",
    "permissionDecisionReason": "My reason here (shown to user)"
  },
  "decision": "approve" | "block" | undefined, // Deprecated for PreToolUse but still supported
  "reason": "Explanation for decision" // Deprecated for PreToolUse but still supported
}
```

### **PostToolUse** 决策控制

`PostToolUse` 钩子可以控制工具调用是否继续。

- `"block"` 自动使用 `reason` 提示 Claude。
- `undefined` 不做任何事。忽略 `reason`。

```
{
  "decision": "block" | undefined,
  "reason": "Explanation for decision"
}
```

## UserPromptSubmit 决策控制

UserPromptSubmit 钩子可以控制用户提示是否被处理。

- "block" 阻止提示被处理。提交的提示从上下文中删除。"reason" 显示给用户但不添加到上下文。
- undefined 允许提示正常进行。忽略 "reason"。
- "hookSpecificOutput.additionalContext" 如果未被阻止，将字符串添加到上下文。

```
{
  "decision": "block" | undefined,
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "My additional context here"
  }
}
```

## Stop / SubagentStop 决策控制

Stop 和 SubagentStop 钩子可以控制 Claude 是否必须继续。

- "block" 阻止 Claude 停止。您必须填写 reason 以便 Claude 知道如何继续。
- undefined 允许 Claude 停止。忽略 reason。

```
{
  "decision": "block" | undefined,
  "reason": "Must be provided when Claude is blocked from stopping"
}
```

## SessionStart 决策控制

SessionStart 钩子允许您在会话开始时加载上下文。

- "hookSpecificOutput.additionalContext" 将字符串添加到上下文。

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "My additional context here"
  }
}
```

## 退出代码示例：Bash 命令验证

```
#!/usr/bin/env python3
import json
import re
import sys

# 将验证规则定义为（正则模式，消息）元组的列表
VALIDATION_RULES = [
    (
        r"\bgrep\b(?!.*\|)",
        "使用 'rg' (ripgrep) 而不是 'grep' 以获得更好的性能和功能",
    ),
    (
        r"\bfind\s+\S+\s+name\b",
        "使用 'rg --files | rg pattern' 或 'rg --files -g pattern' 而不是 'find -name' 以获得更好的性能",
    ),
]

def validate_command(command: str) -> list[str]:
    issues = []
    for pattern, message in VALIDATION_RULES:
        if re.search(pattern, command):
            issues.append(message)
    return issues

try:
    input_data = json.load(sys.stdin)
except json.JSONDecodeError as e:
    print(f"Error: Invalid JSON input: {e}", file=sys.stderr)
    sys.exit(1)

tool_name = input_data.get("tool_name", "")
tool_input = input_data.get("tool_input", {})
command = tool_input.get("command", "")

if tool_name != "Bash" or not command:
    sys.exit(1)

# 验证命令
issues = validate_command(command)

if issues:
    for message in issues:
        print(f"• {message}", file=sys.stderr)
    # 退出代码 2 阻止工具调用并向 Claude 显示 stderr
```

```
sys.exit(2)
```

## JSON 输出示例：UserPromptSubmit 添加上下文和验证

对于 `UserPromptSubmit` 钩子，您可以使用任一方法注入上下文：

- 退出代码 0 加 stdout：Claude 看到上下文（`UserPromptSubmit` 的特殊情况）
- JSON 输出：提供对行为的更多控制

```
#!/usr/bin/env python3
import json
import sys
import re
import datetime

# Load input from stdin
try:
    input_data = json.load(sys.stdin)
except json.JSONDecodeError as e:
    print(f"Error: Invalid JSON input: {e}", file=sys.stderr)
    sys.exit(1)

prompt = input_data.get("prompt", "")

# 检查敏感模式
sensitive_patterns = [
    (r"(?i)\b(password|secret|key|token)\s*[:=]", "提示包含潜在机密"),
]

for pattern, message in sensitive_patterns:
    if re.search(pattern, prompt):
        # 使用 JSON 输出以特定原因阻止
        output = {
            "decision": "block",
            "reason": f"安全策略违反: {message}。请重新表述您的请求，不要包含敏感信息。"
        }
        print(json.dumps(output))
        sys.exit(0)

# 将当前时间添加到上下文
context = f"当前时间: {datetime.datetime.now()}"
print(context)

"""
以下也是等效的：
print(json.dumps({
    "hookSpecificOutput": {
        "hookEventName": "UserPromptSubmit",
```

```

        "additionalContext": context,
    },
    )))
"""

# 允许提示使用额外上下文继续
sys.exit(0)

```

## JSON 输出示例: PreToolUse 批准

```

#!/usr/bin/env python3
import json
import sys

# Load input from stdin
try:
    input_data = json.load(sys.stdin)
except json.JSONDecodeError as e:
    print(f"Error: Invalid JSON input: {e}", file=sys.stderr)
    sys.exit(1)

tool_name = input_data.get("tool_name", "")
tool_input = input_data.get("tool_input", {})

# 示例: 自动批准文档文件的文件读取
if tool_name == "Read":
    file_path = tool_input.get("file_path", "")
    if file_path.endswith((".md", ".mdx", ".txt", ".json")):
        # 使用 JSON 输出自动批准工具调用
        output = {
            "decision": "approve",
            "reason": "文档文件自动批准",
            "suppressOutput": True # 不在记录模式中显示
        }
        print(json.dumps(output))
        sys.exit(0)

# 对于其他情况, 让正常的权限流程继续
sys.exit(0)

```

## 使用 MCP 工具

Claude Code 钩子与 [模型上下文协议 \(MCP\) 工具](#) 无缝协作。当 MCP 服务器提供工具时, 它们以您可以在钩子中匹配的特殊命名模式出现。



## MCP 工具命名

MCP 工具遵循 `mcp__<server>__<tool>` 模式，例如：

- `mcp__memory__create_entities` - 内存服务器的创建实体工具
- `mcp__filesystem__read_file` - 文件系统服务器的读取文件工具
- `mcp__github__search_repositories` - GitHub 服务器的搜索工具

## 为 MCP 工具配置钩子

您可以针对特定的 MCP 工具或整个 MCP 服务器：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__memory__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo '内存操作已启动' >> ~/mcp-operations.log"
          }
        ]
      },
      {
        "matcher": "mcp__.*__write.*",
        "hooks": [
          {
            "type": "command",
            "command": "/home/user/scripts/validate-mcp-write.py"
          }
        ]
      }
    ]
  }
}
```

## 示例

要获取包括代码格式化、通知和文件保护在内的实用示例，请参阅入门指南中的 [更多示例](#)。

## 安全考虑

## 免责声明

**风险自担：** Claude Code 钩子会在您的系统上自动执行任意 shell 命令。通过使用钩子，您承认：

- 您对配置的命令承担全部责任
- 钩子可以修改、删除或访问您的用户账户可以访问的任何文件
- 恶意或编写不当的钩子可能导致数据丢失或系统损坏
- Anthropic 不提供任何保证，并不对使用钩子造成的任何损害承担责任
- 您应该在生产使用前在安全环境中充分测试钩子

在将任何钩子命令添加到配置中之前，始终要审查并理解它们。

##

## 安全最佳实践

---

以下是编写更安全钩子的一些关键实践：

1. **验证和清理输入** - 永远不要盲目信任输入数据
2. **始终引用 shell 变量** - 使用 `"$VAR"` 而不是 `$VAR`
3. **阻止路径遍历** - 检查文件路径中的 `..`
4. **使用绝对路径** - 为脚本指定完整路径（使用 `$CLAUDE_PROJECT_DIR` 作为项目路径）
5. **跳过敏感文件** - 避免 `.env`、`.git/`、密钥等

## 配置安全

对设置文件中钩子的直接编辑不会立即生效。Claude Code：

1. 在启动时捕获钩子的快照
2. 在整个会话中使用此快照
3. 如果钩子被外部修改，则警告
4. 需要在 `/hooks` 菜单中审查才能应用更改

这可以防止恶意的钩子修改影响您的当前会话。

## 钩子执行详细信息

- **超时：** 默认 60 秒执行限制，每个命令可配置。
  - 单个命令的超时不会影响其他命令。
- **并行化：** 所有匹配的钩子并行运行
- **环境：** 在当前目录中使用 Claude Code 的环境运行
  - `CLAUDE_PROJECT_DIR` 环境变量可用，包含项目根目录的绝对路径
- **输入：** 通过 stdin 的 JSON

- 输出：
  - PreToolUse/PostToolUse/Stop: 在记录中显示进度 (Ctrl-R)
  - Notification: 仅记录到调试 (`--debug`)

## 调试

### 基本故障排除

如果您的钩子不起作用：

1. **检查配置** - 运行 `/hooks` 查看您的钩子是否已注册
2. **验证语法** - 确保您的 JSON 设置有效
3. **测试命令** - 先手动运行钩子命令
4. **检查权限** - 确保脚本可执行
5. **审查日志** - 使用 `claude --debug` 查看钩子执行详细信息

常见问题：

- 引号未转义 - 在 JSON 字符串内使用 `\`
- 匹配器错误 - 检查工具名称是否完全匹配（区分大小写）
- 找不到命令 - 为脚本使用完整路径

### 高级调试

对于复杂的钩子问题：

1. **检查钩子执行** - 使用 `claude --debug` 查看详细的钩子执行情况
2. **验证 JSON 模式** - 使用外部工具测试钩子输入/输出
3. **检查环境变量** - 验证 Claude Code 的环境是否正确
4. **测试边界情况** - 使用异常文件路径或输入测试钩子
5. **监控系统资源** - 检查钩子执行期间的资源耗尽情况
6. **使用结构化日志** - 在您的钩子脚本中实现日志记录

### 调试输出示例

使用 `claude --debug` 查看钩子执行详细信息：

```
[DEBUG] Executing hooks for PostToolUse:Write
[DEBUG] Getting matching hook commands for PostToolUse with query: Write
[DEBUG] Found 1 hook matchers in settings
[DEBUG] Matched 1 hooks for query "Write"
[DEBUG] Found 1 hook commands to execute
[DEBUG] Executing hook command: <Your command> with timeout 60000ms
[DEBUG] Hook command completed with status 0: <Your stdout>
```

进度消息在记录模式 (Ctrl-R) 中显示：

- 正在运行的钩子
- 正在执行的命令
- 成功/失败状态
- 输出或错误消息

---

## 安全与权限

### 工具权限模式

```
# Allow specific tools (read/edit files)
claude --allowedTools "Edit,Read"

# Allow tool categories incl. Bash (but still scoped below)
claude --allowedTools "Edit,Read,Bash"

# Scoped permissions (all git commands)
claude --allowedTools "Bash(git:*)"

# Multiple scopes (git + npm)
claude --allowedTools "Bash(git:*),Bash(npm:*)"
```

## Dangerous Mode

[!Warning]

NEVER use in Production systems, shared machines, or any systems with important data  
Only use with isolated environments like a **Docker container**, using this mode can cause data loss and compromise your system!

```
claude --dangerously-skip-permissions
```

## 安全最佳实践

### Start Restrictive

### Protect Sensitive Data

- Keep `~/.claude.json` private (`chmod 600`).
- Prefer environment variables for API keys over plain-text.
- Use `--strict-mcp-config` to only load MCP servers from specified config files

## 自动化与集成

---

# 使用 Claude Code 的自动化与脚本

GitHub Actions you can copy/paste :p

1. **Install the Claude GitHub App** on your org/repo (required for Actions to comment on PRs/issues).
2. In your repo, add a secret `ANTHROPIC_API_KEY` Settings → Secrets and variables → Actions → New repository secret
3. Copy the workflows below into `.github/workflows/`.
4. Open a **test PR** (or a new issue) to see them run.

[!TIP]

Pin Actions to a release tag (e.g. `@v1`) when you adopt them long-term. The snippets below use branch tags for readability.

## 自动 PR 审查（内联评论）

Creates a structured review (with inline comments) as soon as a PR opens or updates.

File: `.github/workflows/claude-pr-auto-review.yml`

```
name: Auto review PRs
on:
  pull_request:
    types: [opened, synchronize, reopened, ready_for_review]

permissions:
  contents: read
  pull-requests: write

jobs:
  auto-review:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 1

      - name: Claude PR review
        uses: anthropics/claude-code-action@main
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          # Claude will fetch the diff and leave inline comments
          direct_prompt: |
            Review this pull request's diff for correctness, readability, testing,
            performance, and DX.
            Prefer specific, actionable suggestions. Use inline comments where
            relevant.
          # GitHub tools permitted during the run:
```

```
allowed_tools: >-
  mcp_github__get_pull_request_diff,
  mcp_github__create_pending_pull_request_review,
  mcp_github__add_comment_to_pending_review,
  mcp_github__submit_pending_pull_request_review
```

## 每个 PR 的安全审查

Runs a focused security scan and comments findings directly on the PR.

File: `.github/workflows/claude-security-review.yml`

```
name: Security Review
on:
  pull_request:

permissions:
  contents: read
  pull-requests: write

jobs:
  security:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          ref: ${ github.event.pull_request.head.sha || github.sha }
          fetch-depth: 2

      - name: Claude Code Security Review
        uses: anthropics/claude-code-security-review@main
        with:
          claude-api-key: ${ secrets.ANTHROPIC_API_KEY }
          comment-pr: true
          # Optional:
          # exclude-directories: "docs,examples"
          # claudecode-timeout: "20"
          # claude-model: "claude-3-5-sonnet-20240620"
```

## 问题分类（建议标签和严重程度）

When a new issue opens, Claude proposes labels/severity and posts a tidy triage comment. You can enable auto-apply labels\*\* by flipping a single flag\*\*

File: `.github/workflows/claude-issue-triage.yml`

```
name: Claude Issue Triage
on:
```

```

issues:
  types: [opened, edited, reopened]

permissions:
  contents: read
  issues: write

jobs:
  triage:
    runs-on: ubuntu-latest
    env:
      CLAUDE_MODEL: claude-3-5-sonnet-20240620
    steps:
      - name: Collect context & similar issues
        id: gather
        env:
          GH_TOKEN: ${ secrets.GITHUB_TOKEN }
        run: |
          TITLE="${ github.event.issue.title }"
          BODY="${ github.event.issue.body }"
          # naive similar search by title words
          Q=$(echo "$TITLE" | tr -dc '[:alnum:] ' | awk '{print $1" "$2" "$3" "$4}')
          gh api -X GET search/issues -f q="repo:${ github.repository } is:issue $Q"
      -f per_page=5 > similars.json
        echo "$TITLE" > title.txt
        echo "$BODY" > body.txt

      - name: Ask Claude for triage JSON
        env:
          ANTHROPIC_API_KEY: ${ secrets.ANTHROPIC_API_KEY }
        run: |
          cat > payload.json << 'JSON'
          {
            "model": "${ env.CLAUDE_MODEL }",
            "max_tokens": 1500,
            "system": "You are a pragmatic triage engineer. Be specific, cautious with
duplicates.",
            "messages": [{
              "role": "user",
              "content": [{
                "type": "text",
                "text": "Given the issue and similar candidates, produce STRICT JSON
with keys: labels (array of strings), severity (one of: low, medium, high, critical),
duplicate_url (string or empty), comment_markdown (string brief). Do not include any
extra keys."
              }],
              "type": "text", "text": "Issue title:\n",
              "type": "text", "text": (include from file) },
              "type": "text", "text": "\n\nIssue body:\n",

```

```

        {"type": "text", "text": (include from file) },
        {"type": "text", "text": "\n\nSimilar issues (JSON):\n"},
        {"type": "text", "text": (include from file) }]
    ]]
}
JSON
# Inject files safely
jq --arg title "$(cat title.txt)" '.messages[0].content[2].text = $title'
payload.json \
| jq --arg body "$(cat body.txt)" '.messages[0].content[4].text = $body' \
| jq --arg sims "$(cat similars.json)" '.messages[0].content[6].text = $sims'
> payload.final.json

curl -s https://api.anthropic.com/v1/messages \
-H "x-api-key: $ANTHROPIC_API_KEY" \
-H "anthropic-version: 2023-06-01" \
-H "content-type: application/json" \
-d @payload.final.json > out.json
jq -r '.content[0].text' out.json > triage.json || echo '{}' > triage.json
# Validate JSON to avoid posting garbage
jq -e . triage.json >/dev/null 2>&1 || echo '{"labels":
[], "severity": "low", "duplicate_url": "", "comment_markdown": "(triage failed to parse)"}'
> triage.json

- name: Apply labels (optional)
  if: ${{ false }} # flip to `true` to auto-apply labels
  uses: actions/github-script@v7
  with:
    script: |
      const triage = JSON.parse(require('fs').readFileSync('triage.json', 'utf8'))
      if (triage.labels?.length) {
        await github.rest.issues.addLabels({
          owner: context.repo.owner,
          repo: context.repo.repo,
          issue_number: context.issue.number,
          labels: triage.labels
        })
      }

- name: Post triage comment
  uses: actions/github-script@v7
  with:
    script: |
      const fs = require('fs')
      const triage = JSON.parse(fs.readFileSync('triage.json', 'utf8'))
      const md = `### 🧐 Triage
      - **Suggested labels:** ${triage.labels?.join(', ') || '-'}
      - **Severity:** ${triage.severity || '-'}

```



```
    ${triage.duplicate_url ? `- **Possible duplicate:**`
${triage.duplicate_url}\n` : ''}

    ---

    ${triage.comment_markdown || ''}`
    await github.rest.issues.createComment({
      owner: context.repo.owner,
      repo: context.repo.repo,
      issue_number: context.issue.number,
      body: md
    })
  })
}
```

[!NOTE]

The triage workflow posts a **suggestion comment** by default. Flip the `Apply labels` step to `true` if you want labels applied automatically.

## Configuration & Customization

- **Model selection:** set `CLAUDE_MODEL` (e.g., `claude-3-5-sonnet-20240620`) where shown.
- **Secrets:** `ANTHROPIC_API_KEY` is required. The built-in `GITHUB_TOKEN` is sufficient for posting comments and applying labels.
- **Permissions:** each workflow declares the least privileges it needs (`pull-requests: write` and/or `issues: write`). Adjust only if your org requires stricter policies.
- **Scope:** use `paths:` filters on triggers to limit when workflows run (e.g., only for `/src` or exclude `/docs`).

## Troubleshooting

Check the **Actions logs** first—most issues are missing secrets/permissions or a mis-indented YAML block.

- **No comments appear on PRs:** Verify the Claude GitHub App is installed and the workflow has `pull-requests: write` permission.
- **403 when applying labels:** Ensure the job or step has `issues: write`. The default `GITHUB_TOKEN` must have access to this repo.
- **Anthropic API errors:** Confirm `ANTHROPIC_API_KEY` is set at repository (or org) level and not expired.
- **“YAML not well-formed”:** Validate spacing—two spaces per nesting level; no tabs.

---

## 帮助与故障排除

[!TIP]

Q: `claude` not found, but `npx claude` works?

A: Your `PATH` is missing the npm global bin. See the `PATH` issue section for [Windows](#) or [Linux](#)

Q: Which Node.js version do I need?

A: Node.js 18+\*\* (ideally 20+). Check with `node --version`.\*\*

Q: Where do I see logs

A: Run `claude doctor` and `claude --verbose` the diagnostic window will point to log locations.

Q: Do I need to reboot after editing PATH?

A: No reboot required, but you **must** open a **new** terminal window.

## 调试快速命令

Check the output of `claude doctor` for log locations and environment checks.

[!Note]

```
claude                # Open Claude UI (if on PATH)
claude --version       # Show CLI version (e.g., 1.0.xx)
claude update          # Update the CLI (if supported)

claude doctor          # Open diagnostic / debug window
npx claude /doctor     # Opens diagnostic/debug window

claude --debug         # Launch claude with diagnostics
claude --verbose       # Verbose logging

where claude           # Windows (cmd)
which claude           # macOS/Linux (bash/zsh)

npm bin -g             # Linux Verify your global bin path
npm prefix -g          # Windows Verify your global bin path
```

## 路径临时修复

Your PATH\*\* likely doesn't include the global npm bin directory.\*\*

[!Note]

## Windows (CMD):

```
set PATH=%USERPROFILE%\AppData\Roaming\npm;C:\Program Files\nodejs;%PATH%
where claude
claude --debugg
```

## Windows (PowerShell):

```
$env:Path = "$env:USERPROFILE\AppData\Roaming\npm;C:\Program
Files\nodejs;$env:Path"
where claude
claude --debugg
```

## Linux/MacOS (bash/zsh)

```
export PATH="$ (npm config get prefix) /bin:$HOME/.local/bin:$PATH"
which claude
claude doctor
```

# Windows 路径权限修复

Replace `<you>` with your own Windows username (without the angle brackets)

- Start → type: `Environment Variables`
- Open `Edit the system environment variables` → `Environment Variables`
- Under `User variables for` select `Path` → `Edit` → `New` add:

```
C:\Users\<you>\AppData\Roaming\npm
C:\Program Files\nodejs</kbd>
```

### Optional locations to add:

```
C:\Users\<you>\.claude\local\bin
C:\Users\<you>\.local\bin
```

- Remove duplicates, any entry containing `%PATH%`, and stray quotes ( `"` ). Click `OK`.
- Open a `new` Command Prompt/PowerShell and verify:

```
where claude
claude doctor
```

[!Tip]

## Optional Run directly (when PATH is broken)

### Windows (PowerShell/cmd)

```
"%USERPROFILE%\AppData\Roaming\npm\claude.cmd" --version
"%USERPROFILE%\AppData\Roaming\npm\claude.cmd" doctor
```

### Or via npx:

```
npx claude doctor
```

## Installation / Node.js Issues

Must be Node 18+ (20+ recommended)

```
node --version
```

### Clean uninstall

```
npm uninstall -g @anthropic-ai/claude-code
```

### Fresh install

```
npm install -g @anthropic-ai/claude-code
```

## Authentication Issues

*Verify your Anthropic API key is available to the CLI.*

### PowerShell (Windows):

```
echo $env:ANTHROPIC_API_KEY
claude -p "test" --verbose
```

bash/zsh (macOS/Linux):

```
echo $ANTHROPIC_API_KEY  
claude -p "test" --verbose
```

If the variable is empty set it for your shell/profile or use your OS keychain/secrets manager.

## Permission / Allowed Tools Issues

### Inspect permissions

```
claude config get allowedTools
```

### Reset permissions

```
claude config set allowedTools "[]"
```

### Minimal safe set (example)

```
claude config set allowedTools '["Edit","View"]'
```

## MCP (Model Context Protocol) Issues

### Debug MCP servers

```
claude --mcp-debug
```

### List & remove MCP servers

```
claude mcp list  
claude mcp remove <server-name>
```

## 完全清洁重新安装（Windows / PowerShell）

[!Caution]

The following removes Claude Code binaries, caches, and config under your user profile

1. Uninstall the global npm package

```
npm uninstall -g @anthropic-ai/claude-code
```

2. Remove leftover shim files

```
Remove-Item -LiteralPath "$env:USERPROFILE\AppData\Roaming\npm\claude*" -Force -
ErrorAction SilentlyContinue
Remove-Item -LiteralPath "$env:USERPROFILE\AppData\Roaming\npm\node_modules\@anthropic-
ai\claude-code" -Recurse -Force -ErrorAction SilentlyContinue
```

### 3. Delete cached installer & native files

```
Remove-Item -LiteralPath "$env:USERPROFILE\.claude\downloads\*" -Recurse -Force -
ErrorAction SilentlyContinue
Remove-Item -LiteralPath "$env:USERPROFILE\.claude\local\bin\claude.exe" -Force -
ErrorAction SilentlyContinue
Remove-Item -LiteralPath "$env:USERPROFILE\.claude\local" -Recurse -Force -ErrorAction
SilentlyContinue
```

### 4. Remove config and project-local files

```
Remove-Item -LiteralPath "$env:USERPROFILE\.claude.json" -Force -ErrorAction
SilentlyContinue
Remove-Item -LiteralPath "$env:USERPROFILE\.claude" -Recurse -Force -ErrorAction
SilentlyContinue
```

### 5. Reinstall

```
npm install -g @anthropic-ai/claude-code
```

## 一键健康检查（复制/粘贴）

### Windows (PowerShell):

```
Write-Host "`n=== Node & npm ==="; node --version; npm --version
Write-Host "`n=== Where is claude? ==="; where claude
Write-Host "`n=== Try doctor ==="; try { claude doctor } catch { Write-Host "claude not
on PATH" }
Write-Host "`n=== API key set? ==="; if ($env:ANTHROPIC_API_KEY) { "Yes" } else { "No"
}
```

### macOS/Linux (bash/zsh):

```
echo "=== Node & npm ==="; node --version; npm --version
echo "=== Where is claude? ==="; which claude || echo "claude not on PATH"
echo "=== Try doctor ==="; claude doctor || true
echo "=== API key set? ==="; [ -n "$ANTHROPIC_API_KEY" ] && echo Yes || echo No
```

---

## 附录：常用路径

---

- **Windows npm global bin:** `C:\Users\<you>\AppData\Roaming\npm`
- **Windows Node.js:** `C:\Program Files\nodejs`
- **Claude local data (Win):** `C:\Users\<you>\.claude\`
- **Claude config (Win):** `C:\Users\<you>\.claude.json`
- **macOS/Linux npm global bin:** `$(npm config get prefix)/bin` (often `/usr/local/bin` or `$HOME/.npm-global/bin`)

## 最佳实践

---

Curated guidance for safe, fast, and correct use of the Claude Code CLI and interactive REPL. All commands and flags here match the current Anthropic docs as of **Aug 23, 2025**.

## 有效提示

---

```
# Good: Specific and detailed
claude "Review UserAuth.js for security vulnerabilities, focusing on JWT handling"

# Bad: Vague
claude "check my code"
```

Tip: `claude "query"` starts the interactive REPL pre-seeded with your prompt; `claude -p "query"` runs **print mode** (non-interactive) and exits.

---

## 安全最佳实践

---

### 1. Start with minimal permissions

- Prefer explicit allows and denies, either on the CLI or in settings files.

```
# Allow only what you need for this run
claude --allowedTools "Read" "Grep" "LS" "Bash(npm run test:*)"
```

Or commit a project policy at `.claude/settings.json`:

```
{
  "permissions": {
    "allow": ["Read", "Grep", "LS", "Bash(npm run test:)"],
    "deny": ["WebFetch", "Bash(curl:*)", "Read(../.env)", "Read(../secrets/**)"]
  }
}
```

## 2. Handle secrets correctly

- Use environment variables for SDK/automation flows:

```
export ANTHROPIC_API_KEY="your_key" # for SDK/print mode
```

- In the interactive REPL, prefer `/login` instead of hard-coding tokens.
- Deny access to sensitive files in settings (replaces older `ignorePatterns`):

```
{ "permissions": { "deny": ["Read(../.env)", "Read(../.env.*)", "Read(../secrets/**)"]
} }
```

## 3. Audit permissions regularly

```
# Project settings
claude config list
claude config get permissions.allow
claude config get permissions.deny

# Global settings
claude config list -g
```

## 4. Avoid bypass modes in production

- Do **not** use `--dangerously-skip-permissions` outside isolated/dev sandboxes.
- For unattended runs, combine narrow `--allowedTools` with `--disallowedTools` and project settings.

# 性能提示

## 1. Use machine-readable output in automations

```
claude -p "summarize this error log" --output-format json
# valid: text | json | stream-json
```

## 2. Bound non-interactive work



```
claude -p "run type checks and summarize failures" --max-turns 3
# optionally also bound thinking:
export MAX_THINKING_TOKENS=20000
```

### 3. Keep sessions tidy

```
# Retain recent sessions only (default is 30 days)
claude config set -g cleanupPeriodDays 20
```

### 4. Limit context scope

```
# Grant access only to relevant paths to reduce scanning/noise
claude --add-dir ./services/api ./packages/ui
```

### 5. Pick the right model

- CLI aliases: `--model sonnet` or `--model opus` (latest of that family).
- For reproducibility in settings, pin a full model ID (e.g., `"claude-3-5-sonnet-20241022"`).

---

## 监控与告警

### 1) Health checks

Use the built-in **doctor** command to verify installation and environment.

```
# Every 15 minutes
*/15 * * * * /usr/local/bin/claude doctor >/dev/null 2>&1 || \
mail -s "Claude Code doctor failed" admin@company.com </dev/null
```

### 2) Log analysis batch job

```
# Daily analysis with non-interactive JSON output (print mode)
0 6 * * * tail -1000 /var/log/app.log | \
claude -p "Analyze errors, regressions, and suspect patterns; output JSON." \
--output-format json > /tmp/daily-analysis.json
```

### 3) Telemetry (optional)

Claude Code emits OpenTelemetry metrics/events. Set exporters in settings/env (e.g., OTLP) and ship to your observability stack (Datadog, Honeycomb, Prometheus/Grafana, etc.).

---

## 协作最佳实践

# Team Workflows

## 1) Share versioned configuration

```
// .claude/settings.json (checked into the repo)
{
  "permissions": {
    "allow": ["Read", "Grep", "LS", "Bash(npm run lint)", "Bash(npm run test:)"],
    "deny": ["WebFetch", "Read(./env)", "Read(./env.*)", "Read(./secrets/**)"]
  },
  // Pin a model here for reproducibility if desired, using a full model ID:
  "model": "claude-3-5-sonnet-20241022"
}
```

## 2) Documentation automation

```
# Update docs with explicit tasks
claude "Update README.md to reflect the latest API endpoints and examples."
claude "Generate TypeScript types from schema.prisma and write to /types."
```

## 3) Code review standards

```
# Review a local diff with constrained tools
git fetch origin main
git diff origin/main...HEAD > /tmp/diff.patch
claude --allowedTools "Read" "Grep" "Bash(git:*)" \
  "Review /tmp/diff.patch using team standards:
  - Security best practices
  - Performance considerations
  - Code style compliance
  - Test coverage adequacy"
```

# Knowledge Sharing

## 1) Project runbooks

```
claude "Create a deployment runbook for this app: steps, checks, rollback."
claude "Document onboarding for new developers: setup, commands, conventions."
```

## 2) Architecture docs

```
claude "Update architecture docs to reflect new microservices."
claude "Create sequence diagrams for the authentication flow."
```

Tip: Keep durable context in **CLAUDE.md** at the project root. In the REPL, use `/memory` to manage it and `@path` to import file content into prompts.

---

# 常见陷阱及避免方法

---

## Security

### ❌ Don't

- Use `--dangerously-skip-permissions` on production systems
- Hard-code secrets in commands/config
- Grant overly broad permissions (e.g., `Bash( *)`)
- Run with elevated privileges unnecessarily

### ✅ Do

- Store secrets in env vars and credential helpers
- Start from minimal `permissions.allow` and expand gradually
- Audit with `claude config list` / `claude config get`
- Isolate risky operations in containers/VMs

## Performance

### ❌ Don't

- Load an entire monorepo when you only need a package
- Max out thinking/turn budgets for simple tasks
- Ignore session cleanup

### ✅ Do

- Use `--add-dir` for focused context
- Right-size with `--max-turns` and `MAX_THINKING_TOKENS`
- Set `cleanupPeriodDays` to prune old sessions

## Workflow

### ❌ Don't

- Skip project context (`CLAUDE.md`)
- Use vague prompts
- Ignore errors/logs
- Automate without testing

### ✅ Do

- Maintain and update `CLAUDE.md`
- Be specific and goal-oriented in prompts

- Monitor via logs/OTel as appropriate
- Test automation in safe environments first

---

## 第三方集成

---

### DeepSeek 集成

---

#### 1. Have claude Code installed

```
npm install -g @anthropic-ai/claude-code
```

#### 2. Config Environment Variables

```
export ANTHROPIC_BASE_URL=https://api.deepseek.com/anthropic
export ANTHROPIC_AUTH_TOKEN=${YOUR_API_KEY}
export ANTHROPIC_MODEL=deepseek-chat
export ANTHROPIC_SMALL_FAST_MODEL=deepseek-chat
```

#### 3. Now all you need to do is launch `claude`

Find more information from the [Official Deepseek Docs](#)