

Claude Code 高效使用操作手册

Sep 29, 2026 · @Harvey

概述

Claude Code 是 Anthropic 推出的智能编程代理：它能读懂整个代码库、编辑文件、运行命令、执行测试并提交代码，你只需用自然语言描述目标。用好它的关键不是“让它写代码”，而是给足上下文、先计划后动手、用测试验证结果。

适合交给它的任务：

- 熟悉陌生代码库、回答“这段逻辑在哪里/为什么这样写”
- 按明确需求实现功能、写测试、修 Bug
- 批量重构、依赖升级、迁移（如 JS → TS）
- 处理 Git：写提交信息、解决冲突、创建 PR
- 写脚本、排查报错日志、调试 CI

使用入口：

入口	适合场景
终端（CLI）	最完整的功能，日常主力
VS Code / JetBrains 插件	在编辑器里看 diff、边写边用
桌面应用 / 网页 / 手机端	远程派发任务、并行跑多个会话
无头模式（ <code>claude -p</code> ） / GitHub Actions	脚本化、自动化、CI 流水线

安装与首次配置

推荐用官方原生安装器，它会在后台自动更新；装好后在项目目录里运行 `claude` 即可开始（[官方安装文档](#)）。

1. 安装

```
# macOS / Linux / WSL
curl -fsSL https://claude.ai/install.sh | bash

# Windows PowerShell
irm https://claude.ai/install.ps1 | iex

# 其他方式
brew install --cask claude-code          # Homebrew (需手动升级)
winget install Anthropic.ClaudeCode      # WinGet (需手动升级)
npm install -g @anthropic-ai/claude-code # npm (需 Node.js 22+, 不要加 sudo)
```

2. 验证与登录

```
claude --version  # 打印版本号即安装成功
claude doctor     # 检查安装与配置健康状况
cd your-project && claude  # 首次运行会打开浏览器登录
```

- 需要 Pro、Max、Team、Enterprise 或 Console 账号；免费版不含 Claude Code。
- Windows 可原生运行，也可在 WSL 中运行；装了 Git for Windows 会使用 Git Bash，否则用 PowerShell。
- 想要更稳的版本：在 `/config` 中把更新通道设为 `stable`（约滞后一周，跳过有严重回归的版本）。

3. 首次进入项目建议做的三件事

1. 运行 `/init`，让它扫描项目并生成 `CLAUDE.md`，再人工精简校对。
2. 运行 `/permissions`，把常用的安全命令（如 `npm test`、`git status`）加入白名单，减少反复确认。
3. 在 IDE 中安装 Claude Code 插件，方便查看 diff 和共享当前选中的代码。

核心工作流：探索 → 计划 → 编码 → 验证 → 提交

最常见的失败是“一上来就让它写”。把任务拆成五步，每一步都让它停下来等你确认，返工会少很多。

1. **探索**：先让它读，不许改。
 - 例：“阅读 `src/auth` 目录和相关测试，解释登录流程，先不要修改任何文件。”
2. **计划**：按 `Shift+Tab` 切到计划模式，或直接说“先给出计划”。复杂问题加上“仔细想一想”（think hard）。
 - 检查计划是否漏了边界情况、是否改动过大，满意再放行。

- 3. **编码**：让它按计划实现，并在过程中自行运行测试、类型检查、lint。
- 4. **验证**：给它一个可检验的目标——测试用例、截图、期望输出。“让测试通过”比“修好这个 bug”有效得多。
 - 测试驱动写法：先让它写失败的测试并提交，再写实现直到测试通过，且不许修改测试。
- 5. **提交**：让它写提交信息、推送并开 PR（配合 `gh` 命令行效果最好）。

小任务可以跳过前两步：改个文案、修个明显的报错，直接下指令即可。判断标准：如果你能用一句话说清楚改动，就不需要计划。

CLAUDE.md 项目记忆与上下文管理

`CLAUDE.md` 会在每次会话开始时自动载入，是投入产出比最高的配置：写一次，之后每次对话都受益。

放在哪里

位置	作用范围
<code>~/.claude/CLAUDE.md</code>	你个人的所有项目
项目根目录 <code>CLAUDE.md</code>	整个项目，提交到 Git 与团队共享
子目录 <code>CLAUDE.md</code>	在该目录工作时按需载入

该写什么

- 常用命令：构建、测试、单测单文件、lint 的准确写法
- 代码规范：命名、目录约定、禁止使用的库或写法
- 项目特有的坑：环境变量、必须先启动的服务、容易踩的历史遗留
- 工作习惯：如“改完代码必须跑类型检查”“提交信息用中文”

写作原则：短、具体、可执行。一行一条规则，避免大段背景介绍；发现它反复犯同一个错时，就补一条。可以用 `@路径` 引用其他文档，而不是把内容全部复制进来。

管理上下文窗口

- 一个任务一个会话：切换到无关任务时用 `/clear` 清空，避免旧信息干扰。
- 长会话变慢或开始“忘事”时，用 `/compact` 压缩历史，可附带说明保留哪些要点。
- 用 `/context` 查看上下文占用情况。
- 大量搜索、阅读日志等“脏活”交给子代理做，只把结论带回主会话。

- 需要找回之前的会话：`claude --continue` 接着上次，`claude --resume` 选择历史会话。

常用斜杠命令与快捷键速查

输入 `/` 即可看到当前可用的全部命令并按字母筛选；下表是日常最常用的一组（[完整命令列表](#)）。

斜杠命令

命令	用途	何时用
<code>/init</code>	生成初始 <code>CLAUDE.md</code>	第一次进入项目
<code>/memory</code>	编辑 <code>CLAUDE.md</code> 与自动记忆	调整项目规则
<code>/plan</code>	进入计划模式	大改动之前
<code>/clear</code>	清空上下文开新对话	切换到无关任务
<code>/compact</code>	压缩对话历史	会话过长、变慢
<code>/context</code>	查看上下文占用	排查“忘事”
<code>/rewind</code>	回退代码和对话到检查点	改坏了想撤回
<code>/resume</code>	恢复历史会话	接着之前的工作
<code>/diff</code>	查看工作区改动	提交前检查
<code>/code-review</code>	审查当前改动的正确性问题	提交或开 PR 前
<code>/security-review</code>	检查当前分支的安全漏洞	涉及鉴权、输入处理时
<code>/model</code> 、 <code>/effort</code>	切换模型、调整推理强度	难题调高，简单任务调低
<code>/permissions</code>	管理允许/询问/拒绝规则	减少重复确认
<code>/mcp</code>	管理 MCP 服务器连接	接入外部工具
<code>/btw</code>	问一个不进入对话历史的旁支问题	临时插问
<code>/usage</code>	查看用量与额度	控制成本
<code>/doctor</code>	诊断并修复安装与配置问题	出现异常时

快捷操作

操作	作用
<code>Esc</code>	中断当前操作，保留已完成的工作，可立即纠正方向
<code>Esc</code> 连按两次	回到之前的消息重新编辑，或打开回退菜单
<code>Shift+Tab</code>	循环切换权限模式（普通 / 自动接受编辑 / 计划模式）
<code>@文件路径</code>	把指定文件或目录直接加入上下文
<code>!命令</code>	直接执行 shell 命令，输出进入上下文
粘贴或拖入图片	让它看设计稿、报错截图
<code>/terminal-setup</code>	配置 <code>Shift+Enter</code> 换行输入多行提示

命令行启动参数

```
claude "解释这个项目的架构"    # 带着首条提示启动
claude -c                        # 继续最近一次会话
claude -r                        # 选择历史会话恢复
claude -p "查询"                 # 无头模式，输出后退出
cat error.log | claude -p "找出根因"  # 管道输入
```

写好提示词的技巧

好提示词 = 目标 + 上下文 + 约束 + 验收标准。它越具体，第一次就做对的概率越高。

模糊写法	高效写法
给 foo.py 加测试	为 foo.py 写测试，覆盖用户未登录的边界情况，不要用 mock
为什么这个接口这么奇怪？	翻一下 git 历史，总结这个接口为什么设计成现在这样
加一个日历组件	参考首页现有组件的写法，实现一个支持选月份和年份的日历组件，不引入新依赖
修复这个 bug	用户反馈会话超时后登录失败。检查 src/auth 的令牌刷新逻辑，先写能复现的测试，再修复

实用技巧

- **指明文件**：用 @ 引用关键文件，省去它自己搜索的时间。
- **给参照物**：“按 XX 模块的风格来写”比描述风格更准确。
- **给图**：UI 任务直接贴设计稿或截图，并要求它截图对比结果。
- **说清楚不要什么**：不许改的文件、不许引入的依赖、不许删的测试。
- **要求先提问**：需求不明确时说“有不清楚的地方先问我”。
- **及早纠偏**：看到方向不对立即按 Esc 打断，别等它做完再返工。
- **长任务写清单**：让它把任务拆成 Markdown 检查清单写入文件，逐项完成并打勾。

权限、安全与模式

默认情况下，修改文件和执行命令前都会征求你的同意。效率来自“把安全的操作放行、把危险的操作拦住”，而不是一律全放。

模式	行为	适合场景
普通模式（默认）	编辑和命令都需确认	刚上手、敏感项目
自动接受编辑	文件修改自动通过，命令仍需确认	信任它的改动方向时
计划模式	只读分析、输出方案，不改任何文件	大改动、陌生代码
自动模式	由分类器自动判断操作是否安全	长任务、减少打扰
跳过全部权限（ <code>--dangerously-skip-permissions</code> ）	不做任何确认	仅限隔离的容器或沙箱

推荐做法

- 用 `/permissions` 或 `/fewer-permission-prompts` 把只读、低风险命令加入允许列表；把 `rm -rf`、`git push --force` 等加入拒绝列表。
- 团队共享的规则写进项目的 `.claude/settings.json` 并提交；个人偏好放 `.claude/settings.local.json`。
- 支持的平台上可用 `/sandbox` 开启沙箱，隔离命令的文件和网络访问。
- 始终在 Git 仓库里工作，勤提交；出问题可用 `/rewind` 或 Git 回退。
- 不要把密钥写进 `CLAUDE.md` 或提示词；让它读取 `.env` 前想清楚是否必要。

进阶：MCP、技能、子代理、Hooks、并行与自动化

基础用熟之后，以下能力能把重复劳动变成一条命令。

能力	做什么	怎么开始
MCP 服务器	接入外部工具：数据库、浏览器、Jira、Sentry、设计稿等	<code>claude mcp add ...</code> 或 <code>/mcp</code> ；团队共享写入项目的 <code>.mcp.json</code>
技能（自定义命令）	把常用流程写成可复用的 <code>/命令</code> ，可带参数	在 <code>.claude/skills/</code> 下写说明文件， <code>/skills</code> 查看
子代理	独立上下文处理搜索、审查、测试等子任务，只回传结论	让它创建子代理，或编辑 <code>.claude/agents/</code>
Hooks	在特定事件自动执行脚本，如每次改完文件自动格式化、拦截危险命令	<code>/hooks</code> 查看， <code>/update-config</code> 描述需求让它配置
插件	一次安装一整套技能、代理、MCP 和 Hooks	<code>/plugin</code>
并行工作	多个会话同时干活，互不干扰	Git worktree 开多个终端； <code>/batch</code> 拆分大改动； <code>/background</code> 转后台
无头模式	在脚本中调用，输出后退出	<code>claude -p "..."</code> ，可加 <code>--output-format json</code>
GitHub 集成	在 PR 或 issue 里 @claude 让它改代码、审查	<code>/install-github-app</code>

两个立刻能用的例子

- 1. 技能：在 `.claude/skills/fix-issue/` 写一个流程——读取 issue、定位代码、写测试、修复、提 PR。之后只需 `/fix-issue 1234`。
- 2. Hook：配置“文件编辑后自动运行 prettier”，从此不再需要提醒它格式化。

常见问题与避坑清单

现象	原因	对策
改动范围远超预期	需求太宽泛，没有先计划	用计划模式，明确“只改哪些文件”

现象	原因	对策
反复犯同一个错	规则没沉淀	写进 <code>CLAUDE.md</code>
越聊越糊涂、忘记前文	上下文被无关内容塞满	<code>/clear</code> 开新会话，或 <code>/compact</code>
为了让测试通过而改测试	验收标准不清	明确“不许修改测试文件”
声称完成但实际没跑通	没有要求验证	要求它运行测试或 <code>/verify</code> 并展示结果
频繁弹出权限确认	未配置允许列表	<code>/permissions</code> 或 <code>/fewer-permission-prompts</code>
用量消耗快	大文件、长会话、模型过重	精准 @ 引用、及时清空、简单任务用轻量模型
安装后找不到 <code>claude</code> 命令	安装目录不在 PATH	重开终端，运行 <code>claude doctor</code>

每日自检清单

- ☐ 新任务开新会话
- ☐ 大改动先计划，确认后再动手
- ☐ 每个任务都有可验证的完成标准
- ☐ 提交前 `/diff` 或 `/code-review` 过一遍
- ☐ 踩到的新坑写进 `CLAUDE.md`

参考资料：[官方安装文档](#) · [命令参考](#)