SUPERTOKEN 中文文档

Claude Code 中文使用手册

CLI、IDE、远程入口、团队定制、实战与排错

SuperToken 中文手册

本机官方 CLI 实际输出

Anthropic Claude Code

```
$ claude --version
2.1.211 (Claude Code)

$ claude --help
Usage: claude [options] [command] [prompt]

Claude Code - starts an interactive session by default, use -p/--print for
non-interactive output

Arguments:
  prompt                                Your prompt

Options:

Selected commands and options:
  --allowedTools, --allowed-tools <tools...>
  --bg, --background                    Start the session as a background agent
  --bare                                Minimal mode: skip hooks, LSP, plugin
  -c, --continue                        Continue the most recent conversation in
  --effort <level>                      Effort level for the current session
  --ide                                 Automatically connect to IDE on startup
  --include-hook-events                 Include all hook lifecycle events in the
```

来源：Anthropic Claude Code 2.1.211 (Claude Code)

核对日期：2026-07-16

Anthropic Claude Code

基于 Anthropic 官方 Desktop、VS Code、远程入口文档与当前官方 CLI 整理，非 Anthropic 官方译本。

版本日期

2026-07-16

目录

本 PDF 由 SuperToken 文档源自动生成。网页中的命令和配置更新后，运行 `npm run pdf:build` 即可重新导出。

第 1 部分

入口与快速开始

Claude Code 中文使用手册

Claude Code 是 Anthropic 的编程代理。它以终端会话为核心，也可以连接 IDE、启动后台代理、创建 Worktree、使用 Chrome 或 Remote Control，并通过 MCP、技能、代理、Hooks 和插件扩展工作流。

SuperToken 中文整理版 · 核对日期 2026-07-15 · 当前模型 Fable 5 / Opus 4.8 / Sonnet 5 · 实测 CLI `2.1.209` · 非 Anthropic 官方译本

> **本版证据边界**
>
> 本版已重新读取 Anthropic 当前 Model configuration、Desktop、VS Code 与 What's New 官方页面，并与本机 CLI `2.1.209` 的 `--help` 交叉核对。模型是否出现在选择器中仍受账号、组织、provider、地区与数据保留策略影响，不能把一个账号的界面外推给所有用户。

先选对入口

入口	当前已确认能力	最适合	边界
Claude Desktop Code	Local、Remote、SSH、WSL、可视化 diff、终端、文件与 Browser pane	多会话、图形审查和本地/远程项目	订阅、组织策略与 Desktop 版本会影响入口和模式
CLI	交互、计划、权限、恢复、非交互、MCP、插件	本地仓库和终端工作流	需要你自己检查 Git diff 和命令输出
IDE 连接	`claude --ide` 在只有一个有效 IDE 时自动连接	编辑器项目与终端开发循环	当前帮助未说明选区怎样注入，本版不写未复核的面板按钮
Background agents	`claude --background` 与 `claude agents`	独立、可并行的后台任务	仍共享外部资源和权限边界
Worktree + tmux	`--worktree`，可配 `--tmux`	隔离 Git 修改和并行开发	不隔离数据库、端口、缓存或云账号
Remote Control	`--remote-control [name]`	订阅账号登录后的远程续接	当前 CLI 明确要求 claude.ai 订阅登录；可用性因账号而异
Chrome	`--chrome` / `--no-chrome`	使用登录态测试网页和读取控制台	站点授权与敏感动作仍需单独确认；第三方 provider 可用性不同

入口	当前已确认能力	最适合	边界
Ultrareview	云端多代理审查当前分支、PR 或基线	独立代码审查	属于云端能力，账号和网络可用性需单独确认

第一次使用建议从 **CLI + plan 权限模式** 开始。它最容易看清目录、命令、权限请求和验证结果。

> **当前模型基线**
>
> Anthropic 官方名称是 **Fable 5**，不是“Fable 5.0”。`best` 在组织有权限时选择 Fable 5，否则回退到最新 Opus；`fable` 明确选择 Fable 5；Anthropic API 下 `opus` 当前映射到 Opus 4.8，`sonnet` 当前映射到 Sonnet 5。第三方 provider 的别名映射可能不同，详见 Desktop、IDE 与远程接续。

10 分钟完成第一次任务

1. 在可信仓库启动

```bash
cd /path/to/your-project
git status --short --branch
claude --permission-mode plan
```

第一次进入项目时，只信任来源明确、内容可审查的仓库。项目配置、Hooks、MCP 和技能都可能参与后续工作。

预期结果：Claude Code 进入 plan 模式；当前目录和原有未提交修改已经明确。

2. 只读建立事实

```text
先不要修改文件。请检查这个项目并报告：
1. 产品目标、主要入口和模块边界；
2. 安装、测试、类型检查和构建命令分别来自哪里；
3. 当前 Git 状态中哪些改动已经存在；
4. 如果只修复 README 一处错别字，最小验证是什么。

不要读取或输出 .env、密钥和账号信息。结论引用具体文件。
```

提示词里的“不要读取”是行为要求，不是强制访问控制。仓库中确实存在密钥时，应把密钥移出工作区，或同时配置精确 deny 规则与外部文件隔离。

预期结果：Claude 先读文件和 Git 状态，没有产生新 diff；命令来自项目配置而不是猜测。

3. 确认计划再实现

```text
只修复刚才确认的那一处错别字。
不要改写段落，不调整格式，不修改其他文件。
完成后显示完整 diff，运行最小验证，并报告实际执行的命令与结果。
```

切换到默认或合适的编辑权限前，先检查计划中的目标文件和命令。不要为了一个文档修改启用宽泛 Bash 或跳过所有权限。

4. 独立验收

```bash
git diff -- README.md
git status --short
```

确认：

- 只有目标行发生变化。
- 原有未提交修改仍在，没有被覆盖或还原。
- 验证命令确实运行并返回结果。
- Claude 明确区分已验证、人工检查和未验证项。

第一次不要这样做

- 不从主目录或包含多个项目的上级目录启动。
- 不把 `--dangerously-skip-permissions` 当成“减少弹窗”的普通选项。
- 不在没看 diff、没验证时让 Claude 自动提交、推送或发布。
- 不把整个 `.env`、Token、浏览器会话或客户数据交给提示词、MCP 或日志。
- 不在根因仍是猜测时连续修改生产代码。
- 不因为 Worktree 或后台代理存在，就假设数据库和外部服务已经隔离。

手册怎么读

你的目标	阅读
掌握安装、信任、权限、上下文、恢复和非交互	CLI 使用
使用 Desktop、IDE、Background、Worktree、Remote Control、Chrome 或 Ultrareview	Desktop、IDE 与远程接续
配置 CLAUDE.md、settings、MCP、Skills、Agents、Hooks 和 Plugins	项目与团队定制

你的目标	阅读
跟着完整案例练习 Bug、功能、重构、审查和自动化	项目实战
解决安装、认证、权限、上下文、MCP、IDE 或网关错误	故障排除与速查

当前 `2.1.209` 能力地图

类别	已由本机 CLI 确认
会话	`--continue`、`--resume`、`--fork-session`、`--from-pr`、`--name`
认证	`auth login`、`auth logout`、`auth status`、`setup-token`
入口	`--ide`、`--chrome`、`--remote-control`、`--background`
隔离	`--worktree`、`--tmux`、`--add-dir`
权限	`manual`、`acceptEdits`、`plan`、`dontAsk`、`auto`、`bypassPermissions`
自动化	text / json / stream-json、JSON Schema、工具集合限制、allow/deny、预算上限、effort、Hook events
定制	CLAUDE.md、settings、MCP、skills、agents、plugins、Hooks
诊断	`doctor`、`auto-mode defaults/config`、`--safe-mode`、`--bare`、debug 日志
审查	本地提示词审查、`ultrareview` 云端多代理审查

一条成熟的日常工作流

1. 确认仓库、分支、已有修改和权限模式。
2. 在 plan 模式建立事实和复现路径。
3. 明确目标、边界、修改文件、失败状态和验证命令。
4. 小步实现，及时纠正越界修改。
5. 检查完整 Git diff，不只看最终回答。
6. 运行真实测试、类型检查、lint、构建或界面验证。
7. 列出证据、失败、未验证项和剩余风险。
8. 把稳定项目事实写进 `CLAUDE.md`，把重复流程放进技能或插件。

第 2 部分

CLI 使用

Claude Code CLI 使用

Claude Code CLI 把项目文件、终端工具、权限规则和会话历史组合成一个可持续的开发循环。稳定使用的关键是先定工作区与权限，再让输出接受 Git 和测试验证。

当前实测版本：`2.1.209 (Claude Code)` · 当前模型 Fable 5 / Opus 4.8 / Sonnet 5 · 核对日期 2026-07-15

安装与更新

macOS / Linux 原生安装：

```bash
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell：

```powershell
irm https://claude.ai/install.ps1 | iex
```

验证和更新：

```bash
claude --version
claude update
claude doctor
```

如果已有 npm 版本，可以用当前 CLI 的原生安装命令迁移：

```bash
claude install stable
```

企业电脑执行在线安装脚本前，应使用批准的软件分发方式或按组织要求审查来源。使用 SuperToken 时继续看 Claude Code 安装与接入。

项目信任与第一次启动

```bash
cd /path/to/project
git status --short --branch
claude --permission-mode plan
```

第一次进入项目时，Claude Code 会要求确认是否信任当前目录。只信任你了解来源的仓库，因为以下内容可能影响会话：

- 项目 `CLAUDE.md` 和自动记忆。
- `.claude/settings.json`、Hooks、skills 和 plugins。
- 项目 `.mcp.json` 中的服务启动命令。
- 仓库脚本、包管理器生命周期脚本和测试命令。

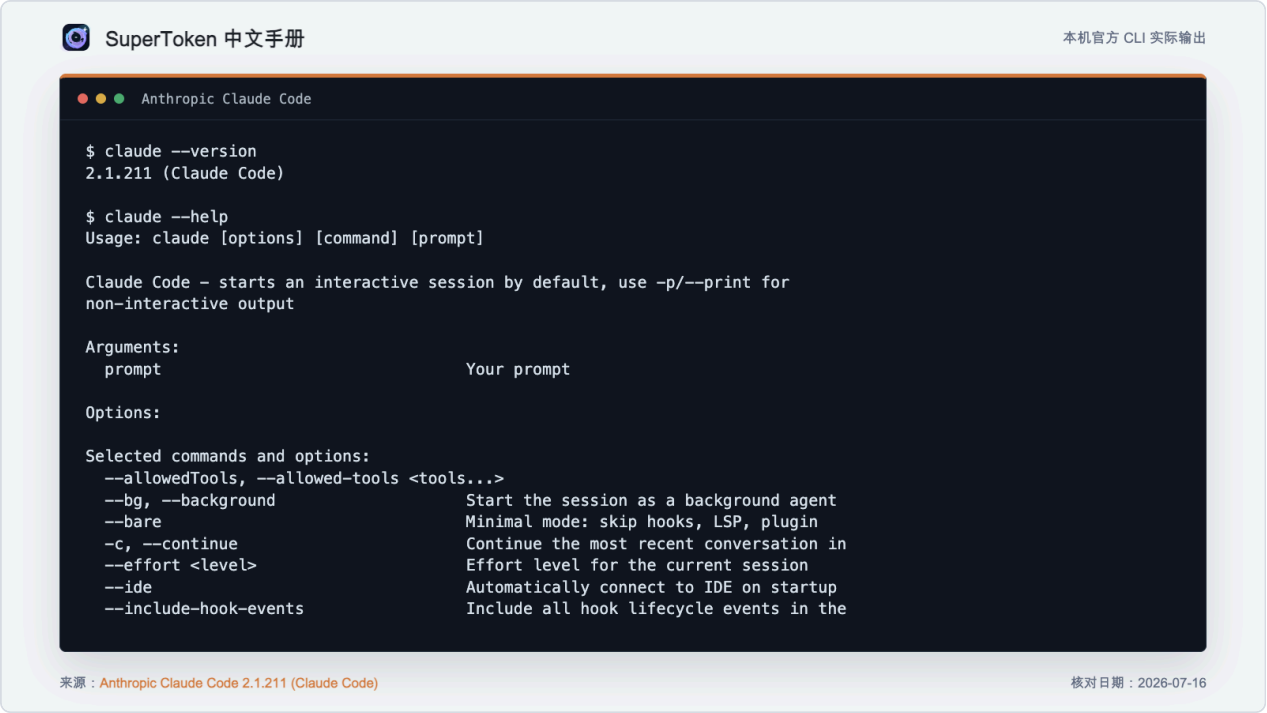
显式 `-p`，以及 stdout 不是 TTY 的管道或重定向，都会进入非交互模式并跳过 workspace trust 对话，因此只在预先信任的固定目录中使用。Print 模式遇到无法通过校验的 settings 文件时会静默忽略它；自动化必须单独校验设置和实际工具范围。

数据与模型 Endpoint 边界

工具可以在本机读文件和执行命令，但提示词、被选入上下文的代码、图片和工具输出仍会发送到当前配置的模型 endpoint。“本地执行”不等于“内容不出机”。

```bash
claude auth status --text
```

开始敏感项目之前，核对当前认证方式、provider、`ANTHROPIC_BASE_URL` 或 SuperToken 接入设置。只提供任务必需的文件与日志，不根据未复核资料承诺 provider 的保留策略；客户代码和受监管数据应遵守团队的数据处理政策。



图：Claude Code CLI 版本与高频参数实测。 Anthropic Claude Code 2.1.209；来源：SuperToken 本机官方 CLI 实际输出，非 Anthropic 宣传图；核对日期：2026-07-14；脱敏状态：不含账号与密钥

启动方式

命令	作用
`claude`	启动交互会话
`claude "先解释项目，不要修改"`	带第一条任务启动
`claude -c`	继续当前目录最近一次会话
`claude -r`	选择历史会话；也可提供会话 ID 或搜索词
`claude --fork-session -r SESSION_ID`	从历史会话分叉新的会话 ID
`claude --from-pr PR_NUMBER_OR_URL`	恢复与 Pull Request 关联的会话
`claude --name fix-login`	为会话设置可搜索名称
`claude --add-dir ../shared`	额外开放目录访问
`claude --no-session-persistence -p "..."`	非交互且不保存会话

`--add-dir` 会扩大工具范围。只添加任务必须的具体目录，不开放整个用户目录或多个客户项目的共同上级目录。

Agent View：管理并行会话

当你同时运行多个长任务时，不必靠一排终端标签记住谁在等待、谁仍在工作。Anthropic 在 2026 年 5 月发布了 Agent View；当前本机 `2.1.209` 也已实证提供 `claude agents` 和 `--bg`。

> **可用性边界**
>
> 截至本次核对，Anthropic 将 Agent View 标为 Research Preview，面向 Pro、Max、Team、Enterprise 和 Claude API plans，并受正常 rate limits 约束。账号、组织策略或版本不符合条件时，即使命令存在也可能无法完整使用。

::: note 截图更新说明 Agent View 发布时的官方图显示 Claude Code `2.1.129` 和 Opus 4.7。功能步骤已用当前 `2.1.209` 与最新官方文档复核，但旧模型图不再收入本版手册。 :::

先启动一个边界清楚的后台任务，再打开总览：

```bash
claude --bg --permission-mode plan \
  "只读定位支付回调测试失败的根因，列出证据和最小修复建议，不修改文件"

claude agents
```

也可以在任意前台会话中按左方向键进入 Agent View，或输入 `/bg` 把当前会话转到后台。总览中的每一行会显示会话名称、最新回复、最后交互时间和当前状态：

- **Needs input**：Claude 在等待你的决定，应先处理会改变方向或权限边界的问题；
- **Working**：任务仍在运行，可以继续处理其他会话；
- **Completed**：任务已经结束，但仍需打开结果、检查 diff 和运行验证。

查看、回复和接管会话

1. 用方向键选中一行，先查看该会话最后一轮，不必离开总览。
2. 会话正在等待决定时，直接在底部输入回复；发送后它会继续运行。
3. 需要查看完整 transcript 或连续操作时，按 Enter 附着到该会话。
4. 在会话中按左方向键回到 Agent View；从总览按右方向键返回原会话。
5. 继续处理下一条，最后逐个检查产物、Git diff 和验证结果。

脚本查询状态

```bash
claude agents --json
claude agents --json --all
claude agents --json --cwd /path/to/project
```

`--all` 只和 `--json` 配合，用于包含已完成会话。JSON 可能包含绝对目录、会话名称和 session ID，写入日志或分享前先脱敏。

Agent View 解决的是“看见和接续多个会话”，不会自动隔离它们的文件、端口、数据库或云端账号。两个会修改代码的任务应使用不同 worktree；共享同一目录的并行任务至少要做到文件范围不重叠，并禁止同时执行迁移、发布或写入同一个外部环境。

怎样描述复杂任务

使用目标、事实、边界和完成标准：

```text
目标：
修复上传头像后页面仍显示旧图片的问题。

已知事实：
- 上传请求成功，刷新页面后能看到新头像；
- 组件在 src/features/profile/AvatarEditor.tsx；
- 数据请求使用现有 query client。

边界：
- 不修改后端缓存策略；
- 不引入新的状态管理依赖；
- 保持失败重试和错误提示不变。

完成标准：
- 先复现并解释根因；
- 添加回归测试；
- 做最小修改；
- 运行相关测试、类型检查和 lint；
- 最后汇报验证结果与剩余风险。
```

仍有关键歧义时：

```text
先不要写代码。检查仓库后，只问最多五个会改变实现方向的问题。
同时列出已经能从代码确认的事实，避免询问仓库中已有答案的问题。
```

权限模式

当前 `2.1.209` 的 `--permission-mode` 明确提供：

模式	适合	注意
`plan`	陌生仓库、架构、复杂任务开局	先调查和规划，不应直接实现
`manual`	日常稳妥开发	工具动作按当前规则和风险由你确认
`acceptEdits`	熟悉仓库的连续文件编辑	接受编辑不等于允许所有 Bash、网络和外部动作

模式	适合	注意
`dontAsk`	只运行预先允许的工具	未授权动作直接失败，不通过弹窗扩大范围
`auto`	当前版本和组织策略允许的自动判断	仍需保持敏感目录、命令和外部资源边界
`bypassPermissions`	外部已经严格隔离的 runner	普通本机不要使用

启动示例：

```bash
claude --permission-mode plan
claude --permission-mode acceptEdits
```

> **DANGER**
>
> `--dangerously-skip-permissions` / `bypassPermissions` 会绕过正常权限检查。只有外部 sandbox 同时做到无互联网、无敏感数据与凭据、文件系统可丢弃且任务范围被独立限制时才考虑使用。

检查 Auto mode

不要只看模式名称，先检查当前规则：

```bash
claude auto-mode defaults
claude auto-mode config
claude auto-mode critique
```

`defaults` 输出默认 environment、allow、soft_deny 和 hard_deny；`config` 输出合并后的有效配置。当前 `2.1.209` 默认 allow 中可能包含向工作分支执行 Git push，因此 Auto mode 也可能允许外部写入。使用前审查实际规则、仓库远端和凭据，生产发布仍保留独立人工审批。

`critique` 会调用模型评价自定义规则，不是离线语法检查。只想查看本地有效配置时使用 `defaults` 和 `config` 即可。

限制可用工具与权限规则

一次性限制：

```bash
claude \
  --tools "Read,Edit,Bash" \
  --allowedTools "Read,Edit,Bash(npm run lint),Bash(npm test *)" \
  --disallowedTools "Bash(git push *),Bash(npm publish *)"
```

`--tools` 限制本次会话有哪些内置工具可用。`--allowedTools` 让匹配项无需再次询问即可执行，并不是“除此之外的工具都不存在”；`--disallowedTools` 拒绝匹配项。工具名和匹配语法以当前 CLI 与设置文档为准。每一项都应能解释为什么需要；过宽 `Bash(*)` 会失去审批意义。

这些工具选项接收一个或多个值。后面还要提供位置参数 prompt 时，用独立的 `--` 结束选项解析，避免 prompt 被当成额外工具名。

会话内常用入口

输入 `/` 查看当前安装的完整菜单。常见入口：

命令	用途
`/help`	查看当前版本帮助
`/init`	生成项目 `CLAUDE.md` 初稿
`/plan`	进入计划工作流
`/permissions`	查看或调整权限
`/model`	选择当前可用模型
`/context`	查看上下文占用
`/compact`	压缩长会话并保留关键状态
`/clear`	开始新的对话上下文
`/resume`	恢复历史会话
`/memory`	查看或管理记忆
`/mcp`	查看 MCP 状态
`/doctor`	完整检查并在允许时修复问题

插件和 skills 会增加斜杠入口，因此手册不把这张表当成完整命令清单。

管理上下文

- 用 `@path/to/file` 指向关键文件，不一次塞入整个仓库。
- 先让 Claude 搜索定义、调用方和测试，再决定加载哪些内容。
- 用 `/context` 检查占用；阶段结束后用 `/compact`。
- 目标、仓库或信任边界改变时开新会话。
- 发现跑偏立即纠正，不等整轮结束。

推荐压缩指令：

```text
/compact 保留：原始需求、已确认根因、修改文件、验证命令、失败结果、未完成步骤和剩余风险。
删除已被实验否定的假设和无关探索过程。
```

恢复会话后先复核目录、分支、Git diff 和未完成项。历史内容可能正确，但工作区已经变化。

非交互执行

非交互运行应同时固定可信工作目录、permission mode 和 `--tools`。不需要项目 CLAUDE.md、Hooks、MCP 或插件的纯 stdin 任务，使用 `--safe-mode --tools ""`；内容不应写入本地会话历史时再加 `--no-session-persistence`。

只读摘要：

```bash
claude -p \
  --permission-mode plan \
  --tools "Read,Glob,Grep" \
  -- "概括仓库架构、关键命令和三个主要风险；引用文件，不修改"
```

JSON 输出：

```bash
claude -p --output-format json --permission-mode plan \
  --tools "Read,Glob,Grep" \
  -- "分析仓库并返回摘要与风险，不修改文件" > report.json
```

结构化结果：

```bash
claude -p --output-format json \
  --permission-mode plan \
  --tools "Read,Glob,Grep" \
  --json-schema '{"type":"object","properties":{"summary":{"type":"string"},"risks":
{"type":"array","items":{"type":"string"}}},"required":["summary","risks"]}' \
  -- "分析当前仓库，不修改文件" > report.json
```

流式自动化使用 `--input-format stream-json` 和 `--output-format stream-json`；只有 `--print` 模式支持，当前 `2.1.209` 的 stream-json 输出还要求 `--verbose`。`--include-partial-messages` 输出增量消息，`--include-hook-events` 把 Hook 生命周期事件加入 stream-json，`--replay-user-messages` 只在输入和输出都为 stream-json 时回显输入确认。调用方必须解析事件、处理部分消息、检查最终状态和退出码。

成本、模型与回退

当前 CLI 在非交互模式支持预算上限：

```bash
claude -p --max-budget-usd 2 --permission-mode plan \
  --tools "Read,Glob,Grep" \
  -- "只读分析这个仓库，不修改"
```

`--model` 可以使用当前账号支持的别名或完整模型名；`--fallback-model` 仅用于 `--print`，主模型不可用时按配置尝试回退。

别名	Anthropic API 当前含义
`best`	组织有权限时使用 Fable 5，否则使用最新 Opus
`fable`	Claude Fable 5，适合最长、最困难的自主任务；不是默认模型
`opus`	Claude Opus 4.8
`sonnet`	Claude Sonnet 5
`haiku`	当前快速、轻量的 Haiku

例如 `claude --model fable` 或 `claude --model claude-opus-4-8`。Fable 5 需要 Claude Code `2.1.170+`，Opus 4.8 需要 `2.1.154+`，Sonnet 5 需要 `2.1.197+`；本机 `2.1.209` 已满足。第三方 provider、自定义 `ANTHROPIC_BASE_URL` 和组织 allowlist 可能改变可用模型或别名映射，不要在团队脚本中假设所有账号完全相同。

`--effort` 可以选择 `low`、`medium`、`high`、`xhigh` 或 `max`。当前 `--help` 只确认这些取值，没有给出每个模型下的质量、时延或用量换算；应在实际账号与模型上验证，不把 `max` 假定为所有任务的最佳默认值。

Safe mode 与 Bare mode

排查项目定制：

```bash
claude --safe-mode
```

Safe mode 暂时禁用 CLAUDE.md、skills、plugins、Hooks、MCP、自动记忆和其他自定义，但保留认证、模型、内置工具与管理员策略。它适合判断异常是否来自定制层。

最小运行模式：

```bash
claude --bare
```

Bare mode 跳过 Hooks、LSP、插件同步、自动记忆、后台预取、keychain 读取和 CLAUDE.md 自动发现。它不会读取 OAuth 或 keychain 凭据，只接受 `ANTHROPIC_API_KEY`，或通过 `--settings` 提供的 `apiKeyHelper`；Bedrock、Vertex 和 Foundry 等第三方 provider 使用各自凭据。它对自动化缓存和最小环境有用，但你必须显式提供需要的认证、设置、目录和上下文。

无障碍终端输出

屏幕阅读器模式：

```bash
claude --ax-screen-reader
```

它使用扁平文本，减少装饰边框和动画。团队文档和 CI 日志也应避免只靠颜色表达状态。

CLI 交付检查清单

- 工作目录、分支和已有修改清楚。
- 权限模式与任务风险匹配，没有危险绕过。
- 结论来自代码、命令和测试证据。
- 完整 diff 已审查，无密钥、无关文件和调试产物。
- 非交互调用校验退出码、JSON、预算和必需字段。
- 最终结果区分通过、失败、未运行和人工验证。

第 3 部分

IDE 与远程入口

Claude Code Desktop、IDE 与远程接续

Desktop 适合把并行会话、diff、终端、文件编辑器和预览放进一个可审查工作区；VS Code 适合围绕当前选区和编辑器 diff 工作；Remote Control 用来从网页或手机继续操控仍在本机运行的 CLI 或 VS Code 会话。

官方资料核对日期 2026-07-15 · 当前模型 Fable 5 / Opus 4.8 / Sonnet 5 · 图形工作区要求 Claude Desktop 1.2581.0 或更高版本

> **图片证据边界**
>
> Anthropic 的 Desktop reference 仍没有静态界面图，但 2026 年第 28 周官方更新页发布了 Desktop 内置浏览器视频。本章使用该官方视频的原始帧，并保留 Web、VS Code 与模型配置页的官方素材；不使用第三方旧截图或生成界面冒充产品画面。

先选对入口

入口	代码实际在哪里运行	适合的工作
Desktop **Code > Local**	你的电脑	可视化 diff、本地预览、多个隔离会话
Desktop **Code > Remote**	Anthropic 云端	关掉 App 或电脑后仍继续的长任务
Desktop **Code > SSH / WSL**	SSH 远端或本机 WSL 2	依赖在服务器、容器或 Linux 工具链中的项目
VS Code 扩展	当前编辑器打开的项目	选区提问、行号引用、编辑器内 proposed diff
Claude Code on the web	Anthropic 云端	没有本地 checkout 时启动云端任务
Remote Control	仍在运行的本机 CLI 或 VS Code 进程	从 `claude.ai/code`、手机或平板接续本地工作

> **Remote 不是 Remote Control**
>
> Desktop 的 **Remote** 会话在 Anthropic 云端执行，电脑离线后仍可继续。**Remote Control** 只把本机正在运行的 CLI 或 VS Code 会话接到网页或 Claude 移动 App；本机进程、项目和工具仍是执行端，进程退出后远程会话也会结束。

安装 Desktop 并打开 Code

系统与账号要求

平台	官方安装入口	安装说明
macOS	Universal DMG	同时支持 Intel 与 Apple Silicon
Windows x64	x64 安装器	本地会话还需要 Git for Windows
Windows ARM64	ARM64 安装器	安装 Git 后应重启 App
Linux beta	Ubuntu / Debian 安装说明	通过 apt 或下载 `.deb` 安装

Code 标签需要 Pro、Max、Team 或 Enterprise 订阅。Desktop 已经包含运行 Code 标签所需的 Claude Code，不需要另装 Node.js 或 CLI；只有要在终端输入 `claude` 时才需单独安装 CLI。

首次启动

1. 安装后从 macOS Applications、Windows 开始菜单或 Linux 应用启动器打开 Claude。

2. 使用 Anthropic 账号登录。
3. 点击窗口顶部中央的 **Code**。
4. 若 Code 要求在线登录，完成浏览器授权后重启 App；若要求升级，先确认当前账号具备付费订阅。
5. Windows 首次打开 Code 前确认 Git 已安装；安装后重启 App。

看到 `403` 时，不要反复重装。先核对当前登录账号、订阅和组织，再按官方 Desktop 页的 **403 or authentication errors in the Code tab** 分支处理。

当前模型选择

当前 Claude Code 可以用别名选择模型，不必在日常操作中记住完整 ID。本机 `2.1.209` 已满足 Fable 5、Opus 4.8 和 Sonnet 5 的最低 CLI 版本要求。

选择	当前含义	适合
`default`	清除个人覆盖，回到账号或组织推荐默认值	不想固定模型时
`best`	有权限时使用 Fable 5，否则使用最新 Opus	希望自动选择当前最强可用模型
`fable`	Claude Fable 5；不是默认模型	跨较长时间、开放式、需要持续调查与验证的困难任务
`opus`	Anthropic API 当前为 Opus 4.8	复杂推理、架构判断和高价值代码工作
`sonnet`	Anthropic API 当前为 Sonnet 5	日常编码、工具调用和较高吞吐
`haiku`	当前快速、轻量的 Haiku	边界清楚的简单任务

Desktop 中从发送按钮旁的模型菜单选择；CLI 中使用 `/model fable`、`/model opus`，或启动时运行 `claude --model fable`。Fable 5 受组织权限和数据保留策略限制；provider 不同，`opus` 与 `sonnet` 的实际映射也会不同。

Provider	`opus`	`sonnet`
Anthropic API	Opus 4.8	Sonnet 5
Claude Platform on AWS	Opus 4.8	Sonnet 4.6
Amazon Bedrock, Google Cloud's Agent Platform	Opus 4.8	Sonnet 4.5
Microsoft Foundry	Opus 4.6	Sonnet 4.5

图：别名要按 provider 解释：Anthropic API 当前为 Opus 4.8 与 Sonnet 5。 Anthropic Claude Code Model configuration 核对于 2026-07-15；来源：Anthropic 官方 Model configuration 页面实时渲染截图，界面英文保持原样；核对日期：2026-07-15；官方公开文档表格；不含账号、API Key、私人项目或会话数据

完成第一次本地修改

1. 选择运行环境和项目

在 Code 标签的新会话输入区完成四项设置：

1. 环境选择 **Local**。
2. 点击 **Select folder**，选择仓库根目录，而不是仓库的上级目录。
3. 从发送按钮旁的菜单选择模型。
4. 首次使用保留 **Manual**；复杂任务可先切到 **Plan**。

Local 会直接访问所选目录。Windows 本地会话必须有 Git；非 Git 目录仍可工作，但不会获得 Git worktree 隔离和完整的版本差异能力。

2. 先让 Claude 建立基线

第一次任务应小而可验证，例如：

```text
先不要修改。确认项目根目录、当前分支、未提交文件，
定位登录按钮重复提交的原因，并列出最小修改方案与验证命令。
得到我确认后再改。
```

可用 `@文件名` 把本地文件加入上下文，也可通过附件按钮或拖放加入图片、PDF 等材料。`@mention` 不适用于云端或 WSL 会话。

3. 选择合适的权限模式

模式	Desktop 行为	适用时机
Manual	编辑文件和运行命令前询问；文件修改先显示 diff	第一次使用、高风险仓库
Accept edits	自动接受文件编辑及常见文件系统命令，其他终端命令仍询问	修改范围已明确的快速迭代
Plan	读取和探索后给出方案，不修改源代码	大改、迁移、先审方案
Auto	以后台安全检查代替大部分逐项确认；是否显示受账号、模型与组织策略影响	边界清晰且可回滚的任务
Bypass permissions	跳过大多数确认，等同 CLI 的 `--dangerously-skip-permissions`	仅限隔离容器或虚拟机

Desktop 不提供 CLI 的 `dontAsk` 模式。模式可以在会话中切换；建议先用 Plan 明确范围，再切回 Manual 或 Accept edits 实施。

4. 接受或拒绝 proposed diff

Manual 模式下，Claude 提议编辑时会显示逐文件 diff 和 **Accept / Reject**。接受前检查：

- 文件是否都属于目标范围；
- 是否混入格式化、依赖锁文件、调试日志或密钥；
- 删除和行为变化是否有依据；
- 原本存在的未提交修改是否被误算成 Claude 的工作。

拒绝后直接说明应保留什么、换哪种实现。官方 quickstart 明确说明：Manual 流程中，接受之前文件不会被这次提议修改。

会话、工作区与 Git 隔离

并行会话

点击侧栏的 **+ New session**，或按 `Cmd+N` / `Ctrl+N` 新建任务。每个会话有独立的对话上下文、项目目录和代码改动；用 `Ctrl+Tab` / `Ctrl+Shift+Tab` 在会话间切换。

要并排查看两个会话，在 macOS 按住 `Cmd`、Windows 按住 `Ctrl`，再点击侧栏中的另一个会话。当前布局会显示两个 session pane；关闭聚焦 pane 后回到单会话。

自动 worktree

对 Git 仓库，Desktop 为并行会话自动创建隔离 worktree，默认位于：

```text
<project-root>/.claude/worktrees/
```

可以在 **Settings > Claude Code > Worktree location** 改位置，也可设置统一的分支名前缀。完成后在侧栏归档会话会移除对应 worktree；也可以开启 PR 合并或关闭后的自动归档。

需要把某些 gitignored 文件复制进新 worktree 时，可在项目根目录配置 `.worktreeinclude`。不要把生产密钥、个人 token 或不必要的 `.env` 当成普通依赖复制；worktree 只隔离 Git 文件，不会自动隔离数据库、端口、缓存和外部账号。

每个并行任务开始时都应记录：worktree 路径、分支、验证命令、端口和测试数据。多个会话不要同时写同一外部资源。

终端、文件编辑器与布局

> **版本门槛**
>
> 官方 Desktop 文档明确说明：可拖拽 pane 布局、集成终端、文件编辑器和 Transcript 视图要求 Desktop 1.2581.0+。macOS 用 **Claude > Check for Updates**，Windows 用 **Help > Check for Updates**。

集成终端

从会话工具栏的 **Views** 菜单打开 Terminal，或按 `Ctrl` + 反引号键。终端会在当前会话的工作目录启动，并与 Claude 共享环境，因此 `git status` 和测试命令看到的是 Claude 正在编辑的同一份文件。

```bash
git status --short --branch
# 再运行仓库真实存在的 test、typecheck、lint 或 build 命令
```

点击 terminal pane 标题栏的 **+** 可开第二个终端；也可右键聊天中的目录并选择 **Open in terminal**。集成终端只适用于 Local 会话。

文件编辑器

点击聊天或 diff 中的文件路径会在 file pane 打开文件。可做小范围编辑并点击 **Save**；如果磁盘上的文件已在打开后发生变化，Desktop 会警告并让你覆盖或放弃。**Discard** 会丢弃 file pane 内尚未保存的编辑。

file pane 适用于 Local 和 SSH 会话。右键文件路径还可以：

- **Attach as context**：加入下一条提示；
- **Open in**：交给已安装的 VS Code、Cursor 或 Zed；
- **Show in Finder / Explorer**：定位文件；
- **Copy path**：复制绝对路径。

聊天、diff、Browser、terminal、file、plan、tasks 和 subagent 都是可拖拽 pane。从 **Views** 打开所需视图，拖动标题调整位置，拖动边缘调整大小。

Diff、代码审查与 PR

审完整 diff

Claude 修改文件后，会出现类似 `+12 −1` 的统计。点击它打开 diff viewer：左侧是文件列表，右侧是逐行差异。

1. 逐个文件检查完整 diff。
2. 点击具体行打开评论框，输入反馈并按 Enter 添加。
3. 多处评论完成后，macOS 按 `Cmd+Enter`、Windows 按 `Ctrl+Enter` 一次提交。
4. Claude 根据评论修改后，再审新 diff，不要把“已响应评论”当成验证通过。

点击右上角 **Review code**，Claude 会针对当前 diff 留下行内审查意见。官方说明该审查聚焦编译错误、确定的逻辑错误、安全漏洞和明显 bug，不负责替代 linter，也不会主动报告纯风格或既有问题。

监控 PR

创建 PR 后，会话中会出现 CI 状态栏。该能力依赖本机已安装并认证的 GitHub CLI `gh`：

- **Auto-fix**：读取失败检查并尝试修复；
- **Auto-merge**：检查全部通过后 squash merge，且仓库必须先允许 GitHub auto-merge。

这两个开关都应按 PR 单独决定。涉及受保护分支、发布或数据库迁移时，不应把 CI 通过直接等同于允许自动合并。

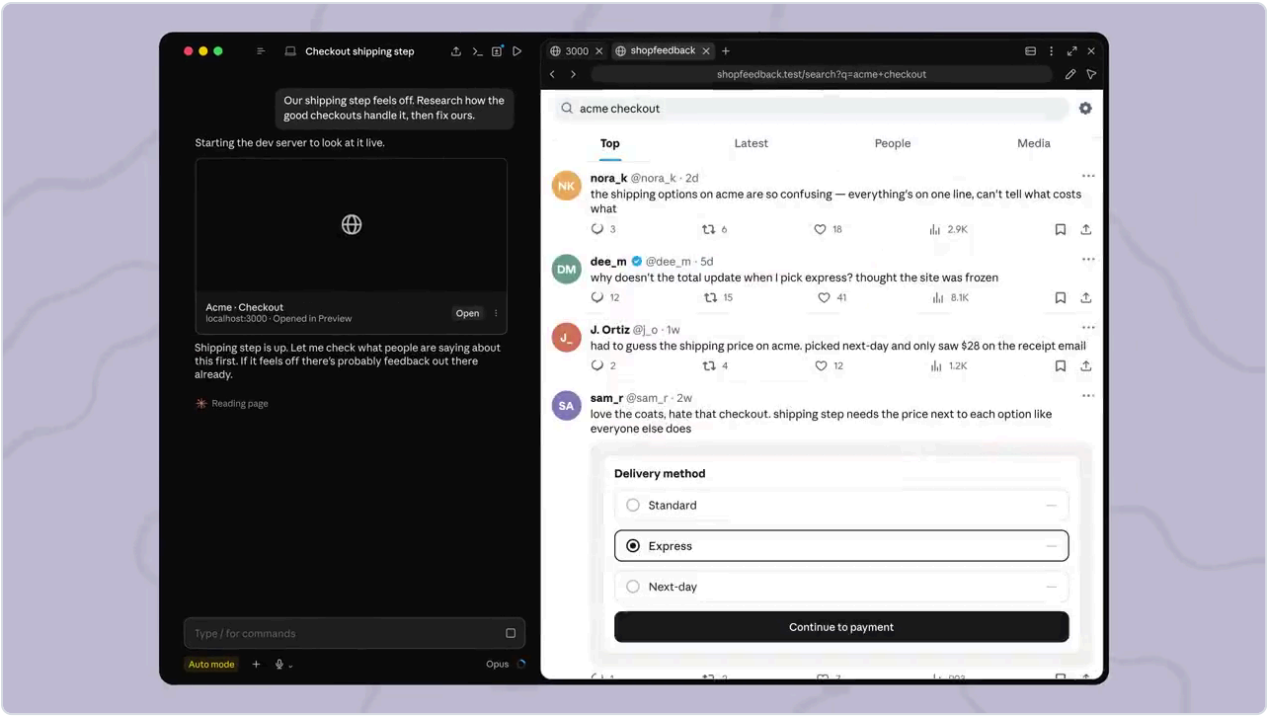
Preview 与 Browser pane

让 Claude 验证正在运行的 App

完成改动后可直接要求：

```text
启动本项目的开发服务器，在 Browser pane 验证登录流程；
检查控制台与服务日志，报告实际访问地址、验证步骤和仍未覆盖的状态。
```

Desktop 会检测开发服务器并在 Browser pane 打开。Claude 可以截图、检查 DOM、点击和填写表单，并根据发现继续修改。`autoVerify` 默认开启，会在编辑后自动验证；也可以从 server 菜单关闭。



图：在 Desktop 内把调查、代码修改和浏览器验证放在同一工作区。 Anthropic Claude Code Desktop 官方更新覆盖 Claude Code 2.1.202 至 2.1.206；截图获取于 2026-07-15；来源：Anthropic 官方 2026 年第 28 周 Desktop 内置浏览器更新视频原始帧，界面英文保持原样；核对日期：2026-07-15；Anthropic 官方演示项目与测试页面；不含账号、凭据或私人数据

服务器配置位于所选项目根目录的 `.claude/launch.json`。若自动识别错了命令或端口，从 server 下拉菜单点 **Edit configuration**，例如：

```json
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "web",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000,
      "autoPort": true
    }
  ]
}
```

`autoPort: true` 会在端口冲突时寻找空闲端口；OAuth callback 或 CORS 必须固定端口时用 `false`。不要把 secret 写进可提交的 `launch.json`，本地变量应放入 Local 环境编辑器。

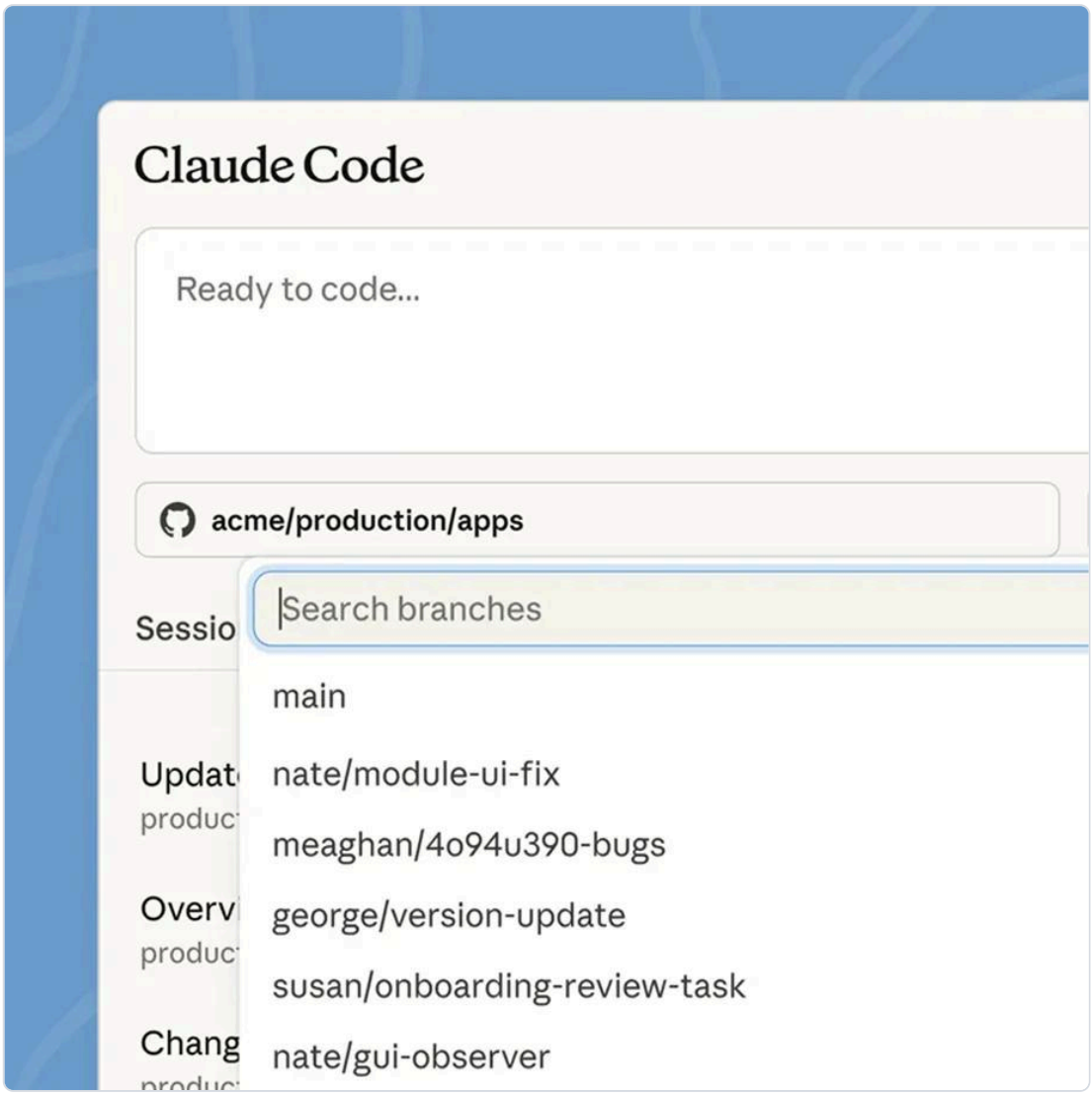
外部站点边界

Browser pane 使用与个人浏览器分离的干净 profile，不继承个人历史或登录。Claude 第一次在外部站点执行点击、输入等动作时，会显示 **Allow once / Always allow / Deny**；不同站点和子域分别授权。本地开发服务器和项目文件不需要这项外站授权。

测试登录、支付、管理后台等流程时使用测试账号。即使站点已允许，购买、创建账号和 CAPTCHA 仍需要人工介入。

Remote、SSH 与跨界面接续

Desktop 的 **Remote** 与 Claude Code on the web 都把任务放在 Anthropic 托管环境中，适合不依赖本机未提交文件、且需要在电脑离线后继续的工作。启动前应确认云端选择的是正确仓库和分支。



图：在云端入口选择仓库、分支并查看会话。 Anthropic Claude Code on the web 产品页核对于 2026-07-14；官方未注明界面版本；来源：Anthropic 官方 Claude Code 产品页，原图未改；核对日期：2026-07-14；Anthropic 官方示例仓库与分支名；不含账号、凭据或私人数据

Desktop 的三种非本地环境

环境	行为	关键限制
Remote	在 Anthropic 云端运行，关闭 App 或电脑后继续	本地未提交文件不会凭空出现在云端

环境	行为	关键限制
SSH	Desktop 自动在 Linux 或 macOS 远端安装并运行 Claude Code	需要可用 SSH 账号、主机和密钥；file pane 可用，集成终端的本地限定仍需注意
WSL	Windows 上在 WSL 2 内用 Linux 路径、Git 和工具链执行	先准备可用 WSL 2 distribution

从新会话的 Environment 菜单选择环境。SSH 首次添加时填写名称、`user@host`、端口和可选 identity file；Remote 会话还可以通过仓库 pill 旁的 + 加多个仓库。

Continue in

会话工具栏右下角 VS Code 图标提供 **Continue in**：

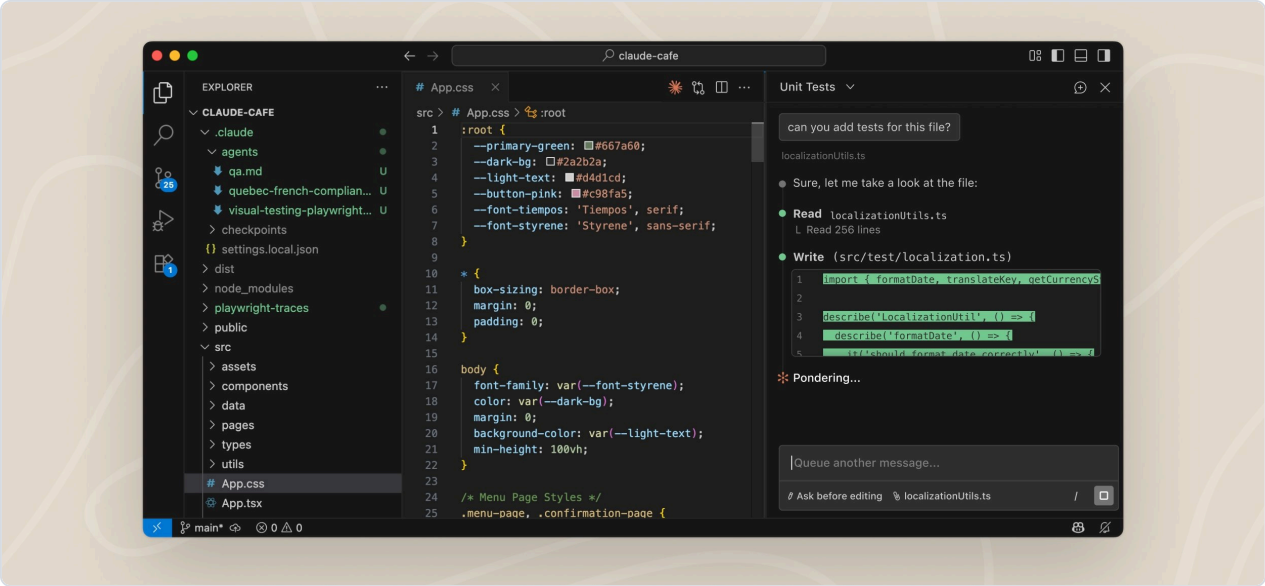
- **Claude Code on the Web**：Desktop 推送分支、生成对话摘要并创建云端会话。要求 working tree 干净，不适用于 SSH 会话。
- **Your IDE**：在支持的 IDE 中，以当前会话 working directory 打开项目。

CLI 会话要转入 Desktop，可在 macOS 或 Windows 的订阅登录会话中运行 `/desktop`。CLI 保存会话、打开 Desktop 后退出；API key 登录及 Bedrock、Google Cloud Agent Platform、Microsoft Foundry 不支持此交接。

在 VS Code 中工作

VS Code 扩展是 Anthropic 官方推荐的 VS Code 图形入口，要求 VS Code 1.98.0 或更高版本。付费 Claude 订阅或 Claude Console 账号均可登录，扩展面板本身不要求手工填写 API Key。

扩展为图形面板自带一份私有 CLI，但不会把 `claude` 放入 PATH。要在 VS Code terminal 中使用 CLI，仍需单独安装 Claude Code CLI。



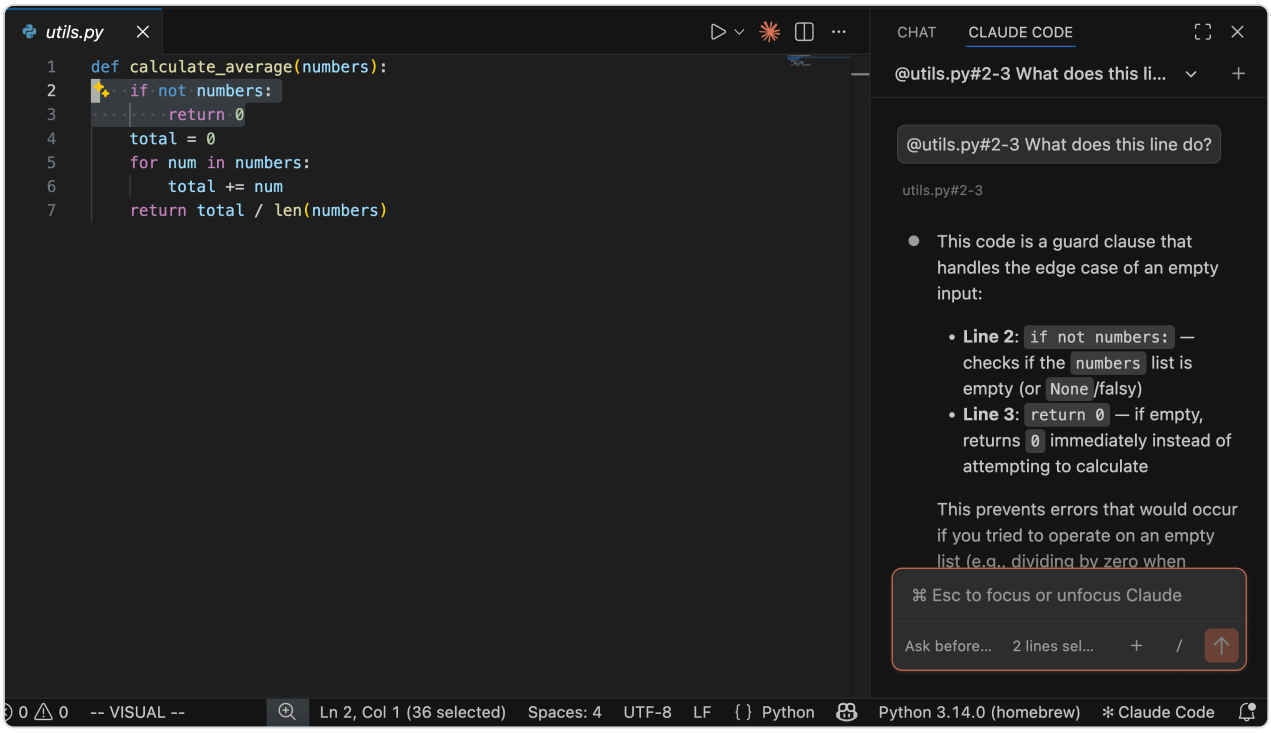
图：Claude Code 官方 VS Code 扩展整体界面。 Anthropic Claude Code for VS Code VS Code 1.98+；官方未注明截图中的扩展版本；来源：Anthropic 官方 Claude Code 文档：Use Claude Code in VS Code，原图未改；核对日期：2026-07-14；Anthropic 官方示例；未发现真实账号、密钥或私人数据

安装、登录和打开面板

1. 在 VS Code 按 `Cmd+Shift+X` / `Ctrl+Shift+X` 打开 Extensions。
2. 搜索 **Claude Code**，确认发布者为 Anthropic 后点击 **Install**。
3. 打开一个文件，点击编辑器右上角的 Spark 图标；也可用左侧 Activity Bar 的 Spark、Command Palette 中的 **Claude Code: Open in New Tab**，或右下角 **Claude Code** 状态项。
4. 首次打开点击 **Sign in**，在浏览器完成授权。
5. 若安装后入口未出现，执行 **Developer: Reload Window** 或重启 VS Code。

把选区和文件交给 Claude

在编辑器选择代码后，Claude 能自动看到该选区。按 `Option+K` / `Alt+K` 会在提示中插入带文件和行号的引用，例如 `@app.ts#5-10`；手工输入 `@` 可模糊匹配文件或目录。

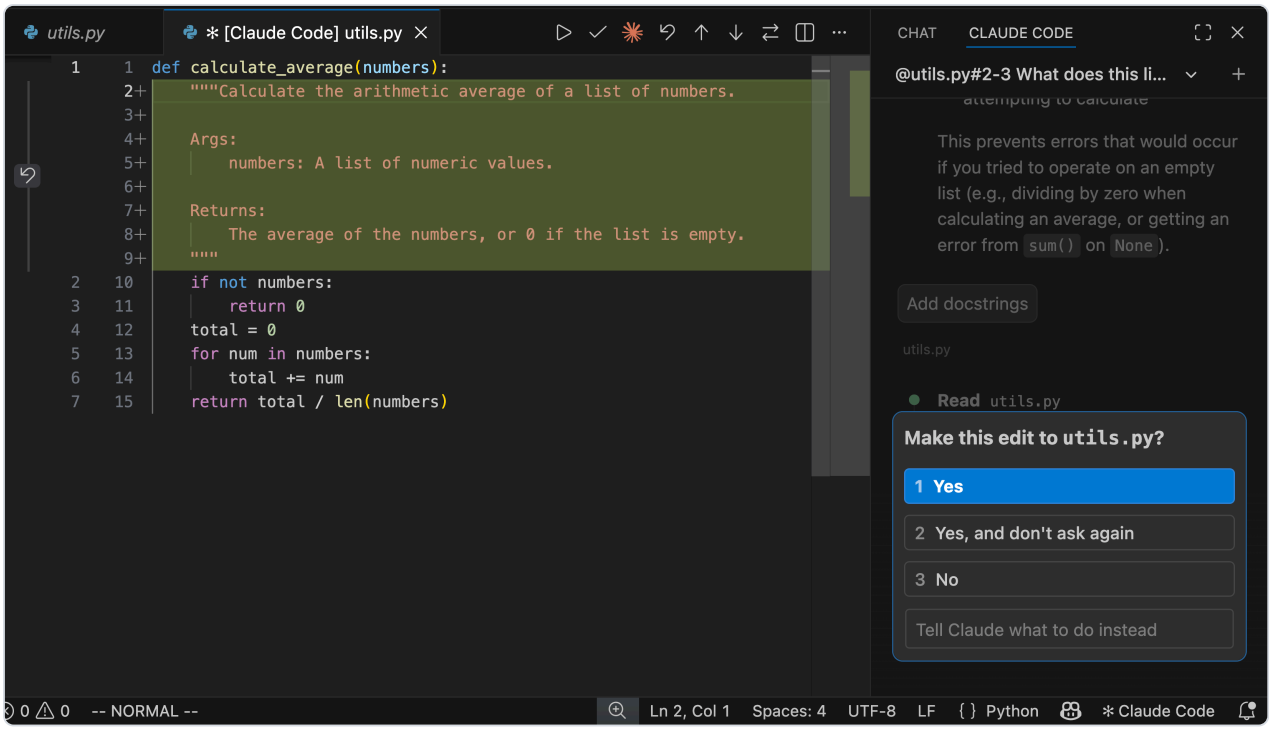


图：把当前选区作为带行号上下文发送。 Anthropic Claude Code for VS Code VS Code 1.98+；官方未注明截图中的扩展版本；来源：Anthropic 官方 Claude Code 文档：Use Claude Code in VS Code，原图未改；核对日期：2026-07-14；Anthropic 官方示例代码；未发现真实账号、密钥或私人数据

Plan 模式会把方案作为完整 Markdown 文档打开，可直接留下行内评论后再允许实施。需要并行会话时，从 Command Palette 使用 **Open in New Tab** 或 **Open in New Window**；每个会话保留独立历史和上下文。

审阅 proposed edit

Claude 请求编辑文件时，VS Code 显示原始内容与 proposed change 的并排 diff。你可以接受、拒绝，或直接修改 proposed 内容；若你在接受前手工改过 diff，扩展会把这一事实告诉 Claude，不会让它误以为原提议被原样应用。



图：在写入前审阅并接受或拒绝 proposed diff。 Anthropic Claude Code for VS Code VS Code 1.98+；官方未注明截图中的扩展版本；来源：Anthropic 官方 Claude Code 文档：Use Claude Code in VS Code，原图未改；核对日期：2026-07-14；Anthropic 官方示例代码；未发现真实账号、密钥或私人数据

要在 VS Code 内运行完整 CLI，在 integrated terminal 输入 `claude`；它会自动集成 IDE diff 和诊断。若 CLI 在外部 terminal 运行，在会话内输入 `/ide` 连接 VS Code。扩展与 CLI 共享会话历史，可在 terminal 用 `claude --resume` 选择并继续扩展会话。

Remote Control：从手机或浏览器继续本机会话

Remote Control 当前属于 research preview。它适合接续正在运行的本机会话，不应当作无人值守的云端执行环境。

启用前检查

Remote Control 适用于 Pro、Max、Team 和 Enterprise，API Key 不支持。Team 与 Enterprise 需要 Owner 先在 Claude Code 管理设置中打开 Remote Control。

同时确认：

- 已通过 `claude` 中的 `/login` 或 `claude auth login` 使用 claude.ai 账号登录；
- 已在目标项目运行过 `claude` 并接受 workspace trust；
- 会话直接连接 `api.anthropic.com`；Bedrock、Google Cloud Agent Platform、Microsoft Foundry 和自定义 `ANTHROPIC_BASE_URL` 不支持；
- 本机进程会保持运行，网络不会长期断开。

启动方式

```bash
# 一个可在本机输入、也可远程接续的交互会话
cd /path/to/project
claude --remote-control "修复登录重复提交"

# 同一进程按需提供多个会话，并让新会话使用隔离 worktree
claude remote-control --name "Web 项目" --spawn worktree --capacity 4
```

已有 CLI 会话中输入：

```text
/remote-control 修复登录重复提交
```

VS Code 扩展的提示框中输入 `/remote-control` 或 `/rc`；连接后 banner 会显示状态，可点 **Open in browser**。VS Code 入口不接受名称参数，也不显示 QR code。

从另一台设备连接

1. 打开终端给出的 session URL，进入 `claude.ai/code`；或扫描 QR code。
2. 在 Claude 移动 App 中也可进入 **Code**，按名称找到带绿色在线状态的会话。
3. 远程发送指令、查看工具动作并处理权限请求。
4. 回到电脑后继续使用同一会话；不要另开一个未隔离进程同时修改相同文件。

Remote Control 只建立本机到 Anthropic API 的出站 HTTPS 连接，不会在电脑上开放入站端口。流量经 TLS 传输，并使用用途独立、短期有效的凭据。

结束条件与故障边界

- 普通交互进程一次只有一个 Remote Control session；多会话使用 server mode。
- 关闭 terminal、退出 VS Code 或停止 `claude` 进程会结束远程会话。
- 本机在线但无法联网约 10 分钟后，会话会超时并退出，需要重新启动。
- `Remote Control requires a claude.ai subscription` 通常表示当前不是 claude.ai OAuth 登录；先清除冲突的 API Key 环境，再重新登录。
- Team / Enterprise 报组织策略禁用时，由 Owner 检查管理端 Remote Control 开关；本机无法绕过。

本地定时任务

Desktop 的 **Routines** 可创建访问本地文件的周期任务：

1. 打开侧栏 **Routines**，点击 **New routine > Local**。
2. 填写 Name、Description、Instructions、模型、权限模式和工作目录。
3. Git 项目建议打开 worktree 隔离；否则任务会直接基于当前 working directory，包括未提交修改。

4. 选择 Manual、Hourly、Daily、Weekdays 或 Weekly；更复杂频率可在会话中用自然语言要求 Claude 设置。
5. 保存后先点 **Run now**，审一次完整权限、diff 和结果，再启用周期执行。

本地 routine 只在 Desktop 开着且电脑醒着时运行。唤醒后会检查最近 7 天的遗漏，但只补跑最近一次；需要电脑关机后仍可靠运行的任务，应选择云端 routine，而不是把本地计划任务当成后台服务器。

交付检查

- 会话环境确实是预期的 Local、Remote、SSH 或 WSL。
- 每个并行 session 的 worktree、分支、端口和外部资源互不冲突。
- Manual proposed diff 与最终完整 Git diff 都已检查。
- 测试、类型检查、构建或预览在正确 working directory 实际运行。
- Preview 使用测试账号，外部站点只授权必要域名和动作。
- Remote Control 与云端 Remote 没有混淆；本机进程退出的影响已经明确。
- PR 的 Auto-fix、Auto-merge 和定时任务都经过单独授权与首次人工验收。

第 4 部分

项目与团队定制

Claude Code 项目与团队定制

先把项目事实和权限边界写清，再增加技能、代理、Hooks 和插件。扩展越多，启动上下文、权限面和故障组合也越大。

CLI 行为基于本机 `2.1.209` · 文件约定等待 Anthropic 官方站恢复后再次复核

选择最小的定制层

需求	放在哪里
当前任务的一次性目标和边界	提示词或当前会话
项目结构、命令、约定和完成标准	`CLAUDE.md`
工具权限、Hooks、环境和团队设置	`.claude/settings.json`
本机对项目的私有覆盖	`.claude/settings.local.json`
外部实时数据和动作	MCP

需求	放在哪里
可复用专业流程	Skill
独立角色和隔离上下文	Agent / subagent
工具事件前后的确定性检查	Hook
一组可安装、可更新的能力	Plugin

一次提示词能解决的问题不要急着做 Skill；一条项目规则能解决的问题不要急着做 Hook。

CLAUDE.md：保存稳定项目事实

常见位置：

- `~/.claude/CLAUDE.md`：个人跨项目说明。
- 仓库根目录 `CLAUDE.md` 或 `.claude/CLAUDE.md`：团队共享规则。
- 子目录中的 `CLAUDE.md`：模块级更具体说明。

用 `/init` 生成草案后人工整理：

```md
# CLAUDE.md

## 项目结构
- apps/web：前端入口和页面
- packages/core：共享领域逻辑
- tests：集成测试

## 常用命令
- 安装：pnpm install --frozen-lockfile
- 开发：pnpm dev
- 检查：pnpm lint && pnpm typecheck
- 测试：pnpm test
- 构建：pnpm build

## 工作约定
- 修改前阅读同目录实现、测试和最近调用方。
- 不提交 .env、密钥、日志、覆盖率或生成目录。
- Bug 修复不升级依赖，不顺手重构无关模块。
- 新功能覆盖成功、失败、空数据和权限状态。

## 完成标准
- 运行与改动直接相关的检查。
- 审查完整 diff，保留用户原有修改。
- 最终列出改动、证据、未验证项和剩余风险。
```

可以用 `@README.md` 或 `@docs/architecture.md` 导入已有说明。只导入每次任务都值得读取的稳定内容；大文档和临时日志会持续占用上下文。

`CLAUDE.md` 适合团队审查的显式规则，自动记忆适合本机积累经验。关键命令和安全边界仍应进入版本控制，而不是只依赖自动记忆。

设置文件与加载来源

文件	作用
`~/.claude/settings.json`	个人跨项目设置
`.claude/settings.json`	团队共享设置，可提交 Git
`.claude/settings.local.json`	当前电脑或当前项目私有覆盖，通常不提交

CLI 可用 `--setting-sources user,project,local` 限定加载来源，也可用 `--settings SETTINGS_JSON_OR_FILE` 追加一次性设置。排错时一次只改变一个来源。

提交团队设置前检查：

- 没有绝对个人路径、Key、代理凭据和账号 ID。
- Hooks 与 MCP 启动命令来源可信、可在目标平台运行。
- allow 规则足够具体，deny 规则覆盖敏感文件和危险动作。
- Windows、macOS 和 Linux 的路径与 shell 差异已处理。

权限规则

一个保守起点：

```JSON
{
  "permissions": {
    "allow": [
      "Bash(pnpm lint)",
      "Bash(pnpm test *)"
    ],
    "deny": [
      "Read(./.env)",
      "Read(./secrets/**)",
      "Bash(git push *)",
      "Bash(npm publish *)"
    ]
  }
}
```

规则设计：

1. 从真实审批记录找稳定、可重复的命令。
2. 允许项目检查，不允许任意 shell。
3. 拒绝密钥、发布、远程写入和不可逆删除。

4. 定期删除过时规则，避免“临时放宽”变成永久默认。

权限模式和 allow/deny 共同决定行为；`acceptEdits` 不自动允许所有 Bash，`dontAsk` 不自动授予未允许工具。

MCP

添加 HTTP MCP：

```bash
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

添加 stdio MCP：

```bash
claude mcp add my-server -- command-that-starts-the-server
```

作用域：

```bash
claude mcp add --scope local my-server -- command-that-starts-the-server
claude mcp add --scope project my-server -- command-that-starts-the-server
claude mcp add --scope user my-server -- command-that-starts-the-server
```

- `local`：当前项目、当前用户，默认。
- `project`：写入项目 `.mcp.json`，适合团队共享，首次加载需要审批。
- `user`：当前用户所有项目。

管理和诊断：

```bash
claude mcp list
claude mcp get my-server
claude mcp login my-server
claude mcp login --no-browser my-server
claude mcp logout my-server
claude mcp reset-project-choices
```

`--no-browser` 适合 SSH 或无图形界面的 OAuth 登录，它会打印授权 URL 并等待回填 redirect URL。当前 CLI 还提供 `claude mcp add-from-claude-desktop`（仅 Mac 和 WSL）导入 Claude Desktop 配置，以及 `claude mcp serve` 启动 Claude Code MCP server；使用前分别通过对应 `--help` 确认导入来源和调用方。

共享 MCP 前检查服务发布者、包名、启动命令、传输、认证、工具读写能力和日志脱敏。先做无副作用查询，再授权写操作。

Skills

项目技能通常放在 `.claude/skills/NAME/SKILL.md`，通过对应斜杠名称调用。适合：

- 已经重复并稳定的发布检查、代码审查或数据迁移流程。
- 需要参考文件、脚本、模板和明确质量门的任务。
- 希望把复杂流程从 `CLAUDE.md` 中拆出，按需加载。

一个 Skill 至少写清：触发场景、输入、前提、步骤顺序、产物、验证和停止条件。不要把整本团队手册都放进每次启动的上下文。

`--disable-slash-commands` 会禁用 skills。`--bare` 仍可以通过明确的 `/skill-name` 解析技能，但不会自动发现其他项目上下文；以当前 CLI 帮助为准。

Agents 与后台任务

一次性代理定义只对同一次 CLI invocation 生效。可以先把 JSON 放进 shell 变量：

```bash
REVIEWER_AGENT='{
  "reviewer": {
    "description": "审查当前改动中的可验证 Bug 和缺失测试",
    "prompt": "只报告有文件证据的问题，按严重程度排序"
  }
'}
```

在同一次启动中定义并选择：

```bash
claude --agents "$REVIEWER_AGENT" --agent reviewer \
  --permission-mode plan --tools "Read,Glob,Grep,Bash" \
  --allowedTools "Read,Glob,Grep,Bash(git status *),Bash(git diff *)" \
  -- "审查当前分支，不修改文件"
```

后台代理：

```bash
claude --agents "$REVIEWER_AGENT" \
  --background --agent reviewer \
  --permission-mode plan --tools "Read,Glob,Grep,Bash" \
  --allowedTools "Read,Glob,Grep,Bash(git status *),Bash(git diff *)" \
  -- "审查当前分支，不修改文件"
claude agents --json
```

如果只运行 `claude --agent reviewer`，该代理必须已经通过持久化配置存在；前一个进程中的一次性 `--agents` 不会自动传给后一个进程。

不要混淆三层概念：`--agents` 为当前 invocation 提供 custom agent 定义，`--agent` 选择当前会话使用的 agent；`--background` 把一个顶层会话放到后台，`claude agents` 管理这类后台会话；subagent 则是在会话内部被委派的隔离上下文。当前 `claude agents --help` 只承诺管理 background agents，不能把它当成所有会话内 subagent 的通用列表。

代理适合隔离搜索、审查、测试分析等独立上下文。不要把多个强依赖步骤同时并行，也不要让两个代理写同一文件或分支。

Hooks

当前顶层 `--help` 能确认 Hooks 会被 Safe/Bare mode 影响，并能在 stream-json 中输出 Hook events，但没有列出 settings JSON 的完整事件名、输入字段和退出码协议。本版不据此编造可复制的 Hook 配置模板；精确 schema 等官方资料恢复后再补。

Hook 适合在工具或会话事件前后执行确定性动作，例如：

- 编辑后运行格式化或静态检查。
- 只有当前官方 schema 明确支持阻断时，才把敏感工具调用检查做成阻断；否则由 permissions 与外部 sandbox 承担强制边界。
- 对发布、远程写入和删除动作增加审计。
- 把验证结果写入明确的交付记录。

每个 Hook 应有超时、错误输出、失败策略和平台说明。Hook 代码与仓库脚本一样需要审查；不要在 Hook 中隐藏大范围修改或把网络失败静默吞掉。

需要观察自动化中的 Hook 生命周期时，当前 CLI 可以把事件加入 stream-json：

```bash
claude -p --no-session-persistence --verbose --output-format stream-json \
  --include-hook-events \
  --permission-mode plan --tools "Read,Glob,Grep" \
  -- "只读检查项目入口，不修改文件"
```

该命令会实际加载并触发已配置 Hooks，只能在预先信任的项目中运行；不要用它“试跑”来源不明的 Hook。调用方要解析事件并对 Hook 输出脱敏。Hook 造成启动失败时先用 `claude --safe-mode` 建立无定制基线；`--bare` 会跳过 Hooks，但同时改变认证和其他加载行为，不能把两种模式的结果混为一谈。

Plugins

当前 `2.1.209` 的 `claude plugin` 支持：

- 安装、卸载、启用、禁用和更新插件。
- 管理 marketplace。
- 查看插件组件清单和预计 Token 成本。
- 校验插件或 marketplace manifest。
- 对插件运行 eval。

```bash
claude plugin list
claude plugin details PLUGIN_NAME
claude plugin validate --strict PLUGIN_PATH
```

顶层 CLI 还支持用 `--plugin-dir` 加载本地目录或 zip，以及用 `--plugin-url` 为当前会话下载远程 zip。远程插件可能同时带入 Skills、Hooks、MCP 或 agents；只使用来源、版本和内容都已审查的固定制品，不把临时 URL 当作可信安装源。

采用前检查来源、许可证、组件清单、Token 成本、MCP/Hook 权限、更新和回滚。只有一个流程时先用 Skill；需要一组可分发能力时再用 Plugin。

团队落地顺序

1. 写准确的 `CLAUDE.md` 和真实验证命令。
2. 用最小 `settings.json` 固定权限和平台约定。
3. 只连接必要的 MCP，并做只读验证。
4. 重复流程成熟后再做 Skill。
5. 需要独立角色时定义 Agent，避免无意义并行。
6. 确定性质量门才使用 Hook。
7. 多能力分发才使用 Plugin，并记录版本与回滚。
8. 使用 `--safe-mode` 做无定制基线回归。

定制变更验收

- 新会话能正确读到项目命令和约束。
- allow / deny 的正反例符合预期。
- MCP 工具只显示必要账号和数据范围。
- Skill 不把无关大文档加载到所有任务。
- Agent 之间没有同文件、同分支或同外部资源写冲突。
- Hook 失败可见，不会静默放过或破坏工作区。
- Plugin 的来源、组件、成本和更新策略可审查。

第 5 部分

项目实战

Claude Code 项目实战

这套流程把 Claude Code 当作能操作工具的工程协作者：先建立事实，再计划和实现，最后用测试与审查证明结果。

配套阅读：入口与快速开始 · CLI · IDE 与远程入口

> **入口说明**
>
> 下面的核心提示词以 CLI 为基线，也可以在已连接的 IDE 中使用。独立并行任务可使用 Background agent 或 Worktree；Remote Control、Chrome 和 Ultrareview 属于账号、网络或组织策略可能限制的入口，不作为新手默认路径。

实战总流程

1. **探索：** 找入口、读现有模式、确认命令和 Git 状态。
2. **计划：** 识别歧义、依赖、风险与测试方案。
3. **实现：** 小步修改，控制文件和依赖范围。
4. **验证：** 运行真实命令，区分新失败与已有失败。
5. **审查：** 对照原始需求检查 diff、回归和安全问题。
6. **交付：** 给出可复核摘要，不隐瞒未验证项。

复杂任务可以从 `claude --permission-mode plan` 开始，或在会话中切换到 `/plan`。

实战一：接手一个陌生仓库

启动：

```bash
cd /path/to/repo
claude --permission-mode plan
```

输入：

```text
先不要修改文件。请调查这个仓库并给我一份接手说明：
1. 产品目标、技术栈和运行入口；
2. 主要模块、依赖方向和关键数据流；
3. 安装、开发、测试、构建命令，以及它们来自哪个文件；
4. 配置和环境变量从哪里加载，但不要读取或输出密钥值；
5. 当前 Git 状态、测试状况和最值得先确认的风险；
6. 仍无法从仓库判断的问题。

不要只总结 README。结论必须尽量引用具体文件。
```

然后执行 `/init` 生成 `CLAUDE.md` 草案。人工删除猜测，只保留团队愿意长期维护的命令和约束。

实战二：定位并修复 Bug

例子：搜索结果偶尔显示上一次关键词的数据。

```text
修复搜索页的竞态问题：快速输入两个关键词时，较慢的旧请求可能覆盖新结果。

请严格按顺序：
1. 阅读搜索组件、请求封装、取消逻辑和现有测试；
2. 用测试或最小复现确认问题；
3. 解释事件时间线和根因；
4. 先添加能失败的回归测试；
5. 做最小修复，不更换请求库、不改服务端接口；
6. 验证快速输入、请求失败、清空关键词和组件卸载；
7. 运行相关测试、类型检查和 lint；
8. 最后审查 diff，说明修改、证据和剩余风险。
```

当根因不清楚时，及时收紧：

```text
暂停修改。把目前内容分为“已验证事实”“仍是推测”“下一步最小实验”。
只有实验确认根因后才继续改生产代码。
```

这能避免模型围绕第一个猜测不断打补丁。

实战三：用计划模式开发功能

需求：导出当前筛选条件下的订单 CSV。

计划阶段

```text
我们要给订单列表增加“导出 CSV”。

要求：
- 导出当前筛选和排序下的全部结果，而不只是当前页；
- 复用现有权限系统；
- 导出期间要有进度或禁用状态；
- 失败可重试，不能阻塞列表浏览；
- 文件名包含日期，但不包含用户隐私信息。

请先检查前后端现有能力、相似下载流程和测试方式。
列出需要确认的问题、推荐方案、修改文件、失败场景和验证计划。
现在不要实现。
```

计划阶段重点确认：

- 是前端生成还是服务端异步导出；
- 数据量、权限与审计要求；
- CSV 注入、编码和时区问题；
- 失败、重复点击和取消行为；
- 测试需要在哪一层覆盖。

实现阶段

```text
按已确认方案实现。先完成最小纵向闭环，再补边界状态。

不要修改无关列表逻辑，不新增依赖，沿用现有下载和通知组件。
完成后运行相关前后端测试、类型检查、lint 和构建。
最后对照原始需求逐项验收，明确哪些是自动化验证、哪些需要人工验证。
```

实战四：用 worktree 隔离任务

Claude Code 可以为任务创建独立 Git worktree：

```bash
claude --worktree order-export
```

适合：

- 当前工作区已有未完成改动；
- 同时处理两个互不依赖的 Issue；
- 想隔离一次高风险实验；
- 需要并行代理但不希望改动互相覆盖。

进入后先让 Claude 报告 worktree 路径、分支和基线提交。结束前要求它运行测试、总结 diff，但是否提交、合并和删除 worktree 仍由你决定。

> WARNING
>
> worktree 只隔离工作目录和分支，不隔离数据库、端口、全局缓存、云账号或外部服务。并行测试使用独立端口和测试数据。

实战五：重构但保持行为

```text
重构 packages/billing/src/calculateInvoice.ts，把折扣、税费和舍入拆成独立纯函数。

必须保持：
- 导出 API、参数和返回结构不变；
- 金额精度、舍入顺序和错误类型不变；
- 日志与指标字段不变；
- 不升级依赖，不修改调用方。

先列出现有可观察行为、边界值和测试缺口。
必要时先增加特征测试，再分小步重构；每步运行测试。
最后比较重构前后行为，并审查是否改变了运算顺序。
```

金融、权限、时间和并发代码尤其需要先固定现有行为。代码“更漂亮”不是完成标准。

实战六：让 Claude 做代码审查

交互会话中：

```text
审查当前相对 main 的所有改动。

优先找：
- 会造成错误行为、数据损坏或权限绕过的问题；
- 并发、缓存、边界值和错误处理回归；
- 与需求不一致或缺失的测试。

每个发现必须说明严重程度、具体文件位置、触发条件和影响。
不要报告纯风格偏好；如果没有可验证问题，明确说没有发现。
```

非交互审查已暂存 diff：

```bash
git diff --cached | claude -p \
  --no-session-persistence --safe-mode --tools "" \
  -- "审查这份 diff，只报告可验证的 Bug、安全问题和缺失测试，按严重程度排序"
```

输入前先确认 diff 不含密钥、PII、客户数据或内部地址。这里禁用了工具并不保存会话，只审查标准输入；只有 diff 可能缺少调用方和项目约束。高风险审查应在可信仓库内使用 plan 模式和受限 Read/Glob/Grep 工具读取上下文。

实战七：前端视觉与交互验证

```text
完成页面后启动本地服务，并验证桌面 1440×900 与移动端 390×844。

覆盖：
- 正常、加载、空数据、错误、禁用和长文本状态；
- 键盘导航、焦点、表单错误与按钮反馈；
- 控制台错误、失败请求和资源加载；
- 文字溢出、元素遮挡、横向滚动和布局跳动。

请用真实页面和截图验证，不要只根据组件代码推断。
修复发现的问题后重复检查，并在最终结果中列出视口和状态。
```

没有可用浏览器或测试账号时，让 Claude 明确列出未完成的人工验收项。

实战八：非交互和脚本

`claude -p` 执行一次任务并退出，适合管道、报告和 CI。

生成仓库摘要：

```bash
claude -p --no-session-persistence --permission-mode plan \
  --tools "Read,Glob,Grep" \
  -- "概括仓库架构、关键命令和三个主要风险；引用文件路径，不修改文件"
```

分析测试输出：

```bash
npm test 2>&1 | claude -p \
  --no-session-persistence --safe-mode --tools "" \
  -- "分析失败测试，找出最早根因，区分实现问题与环境问题，并给出最小修复建议"
```

管道前先对测试输出脱敏。即使禁用工具，日志正文仍会发送到当前模型 endpoint；`--no-session-persistence` 只是不保存本地会话，不改变 provider 的数据政策。

输出 JSON：

```bash
claude -p --output-format json --no-session-persistence \
  --permission-mode plan \
  --tools "Read,Glob,Grep" \
  -- "检查仓库并返回架构摘要和风险列表，不修改文件" \
  > report.json
```

要求结构化字段：

```bash
claude -p --output-format json \
  --no-session-persistence \
  --permission-mode plan \
  --tools "Read,Glob,Grep" \
  --json-schema '{"type":"object","properties":{"summary":{"type":"string"},"risks":
{"type":"array","items":{"type":"string"}}},"required":["summary","risks"]}' \
  -- "分析当前仓库，不修改文件" > report.json
```

自动修改时明确工具和权限边界：

```bash
claude -p \
  --no-session-persistence \
  --permission-mode acceptEdits \
  --tools "Read,Edit,Bash" \
  --allowedTools "Read,Edit,Bash(npm run lint),Bash(npm test *)" \
  --disallowedTools "Bash(git push *),Bash(npm publish *)" \
  -- "修复当前 lint 错误，只修改直接相关文件，然后重新运行 lint 和相关测试"
```

> **自动化安全**
>
> 显式 `-p` 或 stdout 非 TTY 都会跳过工作区信任对话，只在预先固定并审查过的可信目录中运行。Print 模式会静默忽略无效 settings，CI 应在运行前单独校验配置。不要把 `--dangerously-skip-permissions` 当成默认选项；更稳妥的做法是容器隔离、只读仓库权限、用 `--tools` 限制工具面、精确 allow/deny 和短期凭据。

实战九：把重复流程做成技能

当同一套步骤已经稳定重复三次以上，可以考虑项目技能，例如 `.claude/skills/release-check/SKILL.md`：

```md
---
name: release-check
description: 发布前检查版本、变更记录、测试和构建产物。
---

1. 读取当前版本和自上个标签以来的提交。
2. 检查 CHANGELOG 是否覆盖用户可见变更。
3. 运行项目规定的 lint、测试和构建。
4. 检查仓库是否包含密钥、调试日志或意外产物。
5. 输出通过项、阻塞项和对应证据，不执行发布。
```

技能负责可复用流程，`CLAUDE.md` 负责项目事实和约束。不要在两处复制一大段相同内容。

失败时怎样纠偏

直接使用这些短指令：

```text
停下。不要继续修改，先用 git diff 说明你已经改了什么以及为什么超出原范围。
```

```text
回到最近一个测试通过的状态。不要丢弃我原有的未提交改动；先区分哪些修改是你本轮产生的。
```

```text
你还没有证据支持这个根因。设计一个最小实验来证实或否定它，暂时不要改生产代码。
```

```text
这个错误来自环境。保留原始错误，说明缺少的前提，并继续完成不依赖该前提的检查。
```

纠偏要指出“哪条事实或边界错了”，比泛泛说“再仔细一点”有效。

交付前检查清单

- Claude 是否在正确仓库和分支工作。
- 当前 diff 是否只有需求相关改动。
- 原始问题是否真实复现，修复后是否再次验证。
- 测试、类型检查、lint 和构建是否确实运行。
- 是否用跳过测试、删断言或扩大超时掩盖失败。
- 是否读取、输出或提交了敏感信息。
- 是否区分自动验证、人工验证与未验证项。
- 最终摘要是否包含改动、证据和剩余风险。

第 6 部分

故障排除与速查

Claude Code 故障排除与速查

先保留完整错误，再用 doctor、safe mode 和最小设置逐层隔离。不要用跳过权限、删除项目状态或反复重装代替定位根因。

当前实测版本：`2.1.209 (Claude Code)` · 核对日期 2026-07-15

第一轮只读检查

```bash
claude --version
claude auth status --text
claude doctor
pwd
git status --short --branch
```

`claude doctor` 会读取当前目录设置但不弹 workspace trust。会话内 `/doctor` 可以进行更完整检查，并在允许时修复问题。

需要调试日志时：

```bash
claude --debug
claude --debug-file /tmp/claude-debug.log
```

分享日志前删除 Token、Authorization header、邮箱、账号、绝对私人路径和业务数据。

按症状查找

症状	先检查	不要先做
`claude` 找不到	`which claude`、安装方式、PATH、重开终端	原生版和 npm 版反复叠装
项目不可信或进错目录	`pwd`、Git 根目录、仓库来源	在主目录直接信任所有内容
认证失败	`claude auth status --text`、账号类型、系统时间、代理、网关变量	把 Key 粘到聊天、Issue 或截图
配置没有生效	setting sources、JSON 语法、local/project/user 层	同时修改三层设置
权限弹窗太多	当前 permission mode、allow/deny 和实际命令	`--dangerously-skip-permissions`
长会话遗漏事实	`/context`、`/compact`、目标是否变化	不断把整个仓库追加进上下文
MCP Pending / Failed	`claude mcp list/get`、项目审批、启动命令、认证	重复添加同名服务
IDE 没连接	只开一个有效 IDE、集成终端目录、`--ide`	假设所有 IDE 窗口自动同步
Remote Control 不可用	claude.ai 登录、订阅、组织策略、网络	用 API Key 推断一定可用
401 / 404 / 模型不存在	Key、Base URL、模型、分组、余额	随机在 URL 后增删路径

安装与版本冲突

```bash
which claude
ls -l "$(which claude)"
claude --version
```

原生安装更新：

```bash
claude update
```

从旧 npm 安装迁移：

```bash
claude install stable
```

更新后仍显示旧版本，通常是 PATH 先命中了另一个安装。先确认实际二进制位置，再移除不用的来源；不要盲目删除配置和会话。

认证方式与 Endpoint

先查看状态，不要先反复登录：

```bash
claude auth status --text
```

需要使用 Anthropic 官方登录时，当前 CLI 区分订阅账号和 Console API 计费账号：

```bash
claude auth login --claudeai
claude auth login --console
```

SuperToken 等自定义 endpoint 通常来自环境变量或外部配置，不应在没确认当前来源时用登录命令覆盖。状态输出可能包含认证方式和 Base URL 等环境元数据，分享前先脱敏。`claude setup-token` 生成长期认证 token 且要求 Claude 订阅，只用于明确需要的受控环境，不能把结果粘进聊天、日志或仓库。

项目信任与目录

```bash
pwd
git rev-parse --show-toplevel
git status --short --branch
```

检查项目中的：

- `CLAUDE.md` 与 `.claude/settings.json`。
- `.mcp.json` 及其启动命令。
- `.claude/skills`、agents、Hooks 和 plugins。
- 包管理器脚本与测试命令。

显式 `-p` 或 stdout 不是 TTY（管道、重定向）都会跳过 trust 对话，只能在预先固定和审查的目录中运行。Print 模式会静默忽略无法通过校验的 settings 文件，因此排错时要单独验证设置和实际工具范围。

用 Safe mode 隔离定制

```bash
claude --safe-mode
```

如果 Safe mode 正常，问题大概率在 CLAUDE.md、自动记忆、settings、MCP、skill、agent、Hook、plugin、LSP 或其他项目定制。逐项恢复，每次只改变一层。

需要更小基线：

```bash
claude --bare \
  --permission-mode plan \
  "只报告当前目录和可读取的项目入口，不修改"
```

Bare mode 不自动发现 CLAUDE.md，并跳过大部分定制。它不会读取 OAuth 或 keychain 凭据；Anthropic 认证只接受 `ANTHROPIC_API_KEY` 或通过 `--settings` 提供的 `apiKeyHelper`，Bedrock、Vertex 和 Foundry 使用各自凭据。不要把这种预期的认证变化误判成代码问题。

设置与权限

指定加载来源做对比：

```bash
claude --setting-sources user
claude --setting-sources project
claude --setting-sources user,project,local
```

一次性最小设置：

```bash
claude --settings '{"permissions":{"allow":["Read"],"deny":["Read(./.env)"]}}' \
  --permission-mode plan
```

若未授权动作在 `dontAsk` 下直接失败，这是预期行为；它不会弹窗请求扩大范围。`acceptEdits` 也不表示所有 Bash 或网络动作自动允许。

Auto mode 行为不符合预期时，比较默认规则与有效配置：

```bash
claude auto-mode defaults
claude auto-mode config
```

上下文和会话

会话内：

- `/context` 查看占用。
- `/compact` 保留需求、事实、diff、验证和未完成项。
- `/clear` 目标改变时清空当前对话。
- `/resume` 选择历史会话。

CLI：

```bash
claude -c
claude -r
claude --fork-session -r SESSION_ID
```

恢复后要求 Claude 重报目录、分支、diff 和未完成步骤。历史会话不会保证工作区仍处在同一提交。

MCP

```bash
claude mcp list
claude mcp get SERVER_NAME
```

项目级 `.mcp.json` 未审批时会显示 Pending，不会连接。检查：

1. 当前项目是否正确。
2. server 名称和 scope。
3. stdio 命令在同一 shell 能否独立启动。
4. HTTP/SSE endpoint、OAuth 或 header 是否有效。
5. 项目选择是否被拒绝；必要时使用 `reset-project-choices` 后重新审查。
6. `--strict-mcp-config` 是否忽略了其他来源。

SSH 或无图形界面环境中，使用 `claude mcp login --no-browser NAME` 打印授权 URL，并按提示回填 redirect URL。

不要把 Authorization header 直接写进截图或仓库。优先使用环境变量、OAuth 和可撤销的短期凭据。

IDE

```bash
pwd
git rev-parse --show-toplevel
claude --ide --permission-mode plan
```

`--ide` 只在恰好有一个有效 IDE 时自动连接。关闭无关窗口，确认 IDE workspace 与集成终端是同一个 checkout。使用 Worktree 时比较真实路径，不只比较项目名。

Remote Control、Chrome 和云端能力

Remote Control：

```bash
claude --remote-control
```

未登录时，当前 CLI 明确要求 claude.ai 订阅登录。若已登录仍不可用，检查套餐、组织策略、版本和网络；不要假设 API Key 等同于订阅权限。

Chrome：

```bash
claude --chrome
```

能启动参数不等于浏览器已正确授权。检查当前官方集成状态、profile、站点范围和重要动作确认。

Ultrareview：

```bash
claude ultrareview --timeout 30 main
```

`--timeout` 的单位是分钟。失败时区分登录、网络、代码上传策略、目标分支和服务可用性。它是云端审查，不是本地 diff 命令。

SuperToken 接入错误

先看 Claude Code 安装与接入 SuperToken。配置后开新终端，再检查环境变量是否存在，但不要打印完整值。

错误	常见原因	处理
401	Key 错误、过期、被撤销或变量没加载	重新创建最小权限 Key，只检查前后少量字符和变量来源
403	分组、账号或模型权限不足	检查控制台分组和模型可用范围

错误	常见原因	处理
404	Base URL 路径或兼容接口错误	Claude Code 的 SuperToken Base URL 按接入页填写，不随意追加 `/v1`
模型不存在	模型名与分组映射不一致	从当前模型广场复制准确模型 ID
429	频率、额度或余额限制	检查用量、余额、并发与重试策略
连接超时	DNS、代理、网络策略或 endpoint 不可达	分层检查网络，不输出代理凭据

“保存了配置”不是验证。必须在一个无敏感信息的测试项目里完成真实请求，并记录成功模型、入口和时间。

破损项目状态的最后手段

先预览将删除的内容：

```bash
claude project purge --dry-run /path/to/project
```

确认目标无误并备份后，使用逐项确认：

```bash
claude project purge --interactive /path/to/project
```

它会删除该项目的 Claude Code 状态，包括 transcripts、tasks、file history 和 config entry。只有确认问题来自不可恢复的项目状态、已经备份需要保留的信息，并理解无法恢复的后果时才使用。它不是普通缓存清理命令；不要在排错手册中默认使用 `--yes` 或 `--all`。

命令速查

```bash
# 健康与隔离
claude --version
claude auth status --text
claude doctor
claude --safe-mode
claude --bare

# 会话
claude --permission-mode plan
claude auto-mode config
claude -c
claude -r
claude --fork-session -r SESSION_ID

# IDE / 并行 / 远程
claude --ide
claude --background
claude agents --json
claude --worktree task-name --tmux
claude --remote-control

# MCP / 插件
claude mcp list
claude plugin list
claude plugin details PLUGIN_NAME

# 调试
claude --debug-file /tmp/claude-debug.log
```

请求支持前准备

提供版本、操作系统、入口、脱敏错误、最小复现、当前目录类型、权限模式、是否启用 Safe mode，以及问题在默认认证还是 SuperToken 网关下发生。

不要提供 API Key、完整环境变量、cookies、私人会话、客户代码或未脱敏日志。

附录

来源与版本

来源与版本

基于 Anthropic 官方 Desktop、VS Code、远程入口文档与当前官方 CLI 整理，非 Anthropic 官方译本。

本手册核对日期为 2026-07-16。产品界面、模型、命令和账号能力可能更新，快速变化的选项以当前官方页面和本机 `--help` 为准。

1. Claude Code overview
https://code.claude.com/docs/en/overview
2. Claude Code Desktop quickstart
https://code.claude.com/docs/en/desktop-quickstart
3. Claude Code Desktop
https://code.claude.com/docs/en/desktop
4. Claude Code in VS Code
https://code.claude.com/docs/en/vs-code
5. Claude Code Remote Control
https://code.claude.com/docs/en/remote-control
6. Claude Code Desktop scheduled tasks
https://code.claude.com/docs/en/desktop-scheduled-tasks
7. Claude Code platforms
https://code.claude.com/docs/en/platforms
8. Claude Code product page
https://claude.com/product/claude-code
9. CLI reference
https://code.claude.com/docs/en/cli-reference
10. Common workflows
https://code.claude.com/docs/en/common-workflows
11. Memory and CLAUDE.md
https://code.claude.com/docs/en/memory
12. Settings
https://code.claude.com/docs/en/settings
13. Permissions
https://code.claude.com/docs/en/permissions
14. MCP
https://code.claude.com/docs/en/mcp
15. Skills
https://code.claude.com/docs/en/skills
16. Hooks
https://code.claude.com/docs/en/hooks
17. Security
https://code.claude.com/docs/en/security
18. Model configuration
https://code.claude.com/docs/en/model-config
19. Agent view in Claude Code
https://claude.com/blog/agent-view-in-claude-code
20. Agent View documentation
https://code.claude.com/docs/en/agent-view
21. Interactive mode
https://code.claude.com/docs/en/interactive-mode
22. Claude Desktop on Linux
https://code.claude.com/docs/en/desktop-linux
23. Claude Code Desktop in WSL
https://code.claude.com/docs/en/desktop-wsl
24. Week 28: in-app browser on Desktop
https://code.claude.com/docs/en/whats-new/2026-w28
25. Anthropic 官方 Model configuration 页面实时渲染截图，界面英文保持原样
https://code.claude.com/docs/en/model-config#model-aliases
26. Anthropic 官方 2026 年第 28 周 Desktop 内置浏览器更新视频原始帧，界面英文保持原样
https://code.claude.com/docs/en/whats-new/2026-w28#in-app-browser-on-desktop
27. Subagents
https://code.claude.com/docs/en/sub-agents
28. Plugins
https://code.claude.com/docs/en/plugins
29. 沙箱
https://code.claude.com/docs/en/sandboxing
30. Troubleshooting
https://code.claude.com/docs/en/troubleshooting