道雾轩 · daiw.org

Claude Code 中文手册

基于 Claude Code v2.1.212 官方文档整理的 Claude Code 中文手册。

全 17 篇 · 作者 David

概览

Claude Code 是 Anthropic 官方提供的命令行 AI 编程助手，以 `claude` 命令启动，在终端里读写文件、执行命令、调用 MCP 工具，完成从代码理解到提交 PR 的完整开发流程。

本手册基于 Claude Code v2.1.212 的官方文档整理，内容随官方文档更新而更新，但不追踪每一个补丁版本——具体行为请以你本机 `claude doctor` 与 `claude --version` 的实际结果为准。

```
claude --version
```

手册结构

- 安装与升级——原生安装脚本、Homebrew/WinGet/包管理器、npm、二进制完整性校验、自动更新与卸载。
- 启动命令行选项——`claude` 子命令与 `--flag` 全量参考。
- 环境变量——按用途分类的环境变量速查。
- 配置文件——`settings.json`、`CLAUDE.md`、`.mcp.json` 等配置文件的位置、格式与优先级。
- 上手使用——从熟悉新代码库到修 bug、重构、写测试、开 PR、并行开发的实操套路。
- 命令参考——按使用场景拆成会话管理、模型与权限、并行协作、代码评审、自定义扩展、其它实用命令六个子页面，涵盖全部内置斜杠命令与随附技能（skill）。
- 进阶——Commands、Skills、Agents、Plugins。
- 版本 Changelog——从 1.0 GA 到 2.1.212 的关键里程碑（见「其它」分组）。

最快上手路径

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

# 2. 在项目根目录启动并登录
cd /path/to/project
claude

# 3. 首次进入项目，先生成 CLAUDE.md
/init
```

之后的操作大多是「用自然语言描述任务 + 在需要时用斜杠命令控制会话」的组合，具体可参考上手使用。

安装与升级

系统要求

- 操作系统：macOS 13.0+、Windows 10 1809+ / Windows Server 2019+、Ubuntu 20.04+、Debian 10+、Alpine Linux 3.19+。
- 硬件：4 GB+ 内存，x64 或 ARM64 处理器。
- 网络：需要联网（见下方代理相关环境变量）。
- Shell:Bash、Zsh、PowerShell 或 CMD。
- 依赖：`ripgrep` 通常随 Claude Code 一起打包，无需单独安装（Alpine 等 musl 发行版例外，见下文）。

安装方式

原生安装脚本（推荐）

macOS / Linux / WSL:

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

Windows PowerShell:

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

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del inst
```

原生安装会在后台自动更新。Windows 原生环境建议安装 Git for Windows，以便 Claude Code 使用 Bash 工具；不安装则回退到 PowerShell 工具。WSL 环境不需要 Git for Windows。

Homebrew

```
brew install --cask claude-code
```

Homebrew 提供两个 cask：`claude-code` 跟踪 `stable` 渠道（通常滞后约一周，会跳过有重大问题的版本），`claude-code@latest` 跟踪 `latest` 渠道。Homebrew 安装不会自动更新，需要手动运行 `brew upgrade claude-code`（或 `@latest`）。

WinGet

```
winget install Anthropic.ClaudeCode
```

同样不会自动更新，需定期运行 `winget upgrade Anthropic.ClaudeCode`。

Linux 包管理器（apt / dnf / apk）

官方为 Debian/Ubuntu（apt）、Fedora/RHEL（dnf）、Alpine（apk）提供了签名仓库，每个仓库都有 `stable` 与 `latest` 两个渠道。以 apt 的 `stable` 渠道为例：

```
sudo install -d -m 0755 /etc/apt/keyrings
sudo curl -fsSL https://downloads.claude.ai/keys/claude-code.asc \
  -o /etc/apt/keyrings/claude-code.asc
echo "deb [signed-by=/etc/apt/keyrings/claude-code.asc] https://downloads.claude.a
  | sudo tee /etc/apt/sources.list.d/claude-code.list
sudo apt update
sudo apt install claude-code
```

导入密钥后务必核对指纹：`31DD DE24 DDFA B679 F42D 7BD2 BAA9 29FF 1A7E CACE`。dnf、apk 的用法类似，分别用 `.repo` 文件和 `/etc/apk/repositories` 追加一行，详见官方文档。这三种包管理器安装均不会自动更新，升级走各自系统的正常升级命令（`apt upgrade`、`dnf upgrade`、`apk upgrade`）。

npm 全局安装

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

自 v2.1.198 起，npm 包要求 Node.js 22+；更早的 Node 版本只会打印 `EBADENGINE` 警告，不影响安装和运行——因为安装的其实是各平台的原生二进制（如 `@anthropic-ai/claude-code-darwin-arm64`），`claude` 命令本身不依赖 Node.js 运行时。

支持的平台：`darwin-arm64`、`darwin-x64`、`linux-x64`、`linux-arm64`、`linux-x64-musl`、`linux-arm64-musl`、`win32-x64`、`win32-arm64`。

升级请用：

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

> ⚠ 不要使用 `npm update -g`，它遵循安装时的 semver 范围，可能不会真正升到最新版。也不要用 `sudo npm install -g`，否则容易引发权限问题，遇到权限报错应参考官方 troubleshooting 文档而不是加 `sudo`。

Alpine / musl 发行版的额外依赖

Alpine 等基于 musl/uClibc 的发行版，原生安装器需要额外的 `libgcc`、`libstdc++`、`ripgrep`：

```
apk add libgcc libstdc++ ripgrep
```

并在 `settings.json` 里关闭内置 ripgrep，改用系统版本：

```
{
  "env": {
    "USE_BUILTIN_RIPGREP": "0"
  }
}
```

验证安装

```
claude --version
claude doctor
```

`claude doctor` 会给出更详细的安装与配置自检结果（包括最近一次自动更新是否成功）。

认证

Claude Code 需要 Pro / Max / Team / Enterprise / Console 账号（免费版 claude.ai 不包含 Claude Code 访问权限），也可以通过 Amazon Bedrock、Google Cloud's Agent Platform、Microsoft Foundry 等第三方 API 提供方接入。安装完成后直接运行 `claude`，按浏览器提示完成登录。

安装后推荐配置

用 shell 函数默认跳过权限确认

如果你每次都在可信、可回滚的环境里用 Claude Code（比如自己的项目仓库、容器、或有 Git 兜底的目录），反复点“允许”会很累。可以在 shell 配置里包一层函数，让 `claude` 每次都以 `bypassPermissions` 模式启动：

```
# 加到 ~/.bashrc 或 ~/.zshrc，然后 source ~/.zshrc（或重开终端）生效
claude() {
  command claude --permission-mode bypassPermissions "$@"
}
```

- `command claude` 是关键：函数名也叫 `claude`，必须用 `command` 显式调用真正的可执行文件，否则会无限递归调用函数自身。
- `"$@"` 把你敲的所有参数原样透传，所以 `claude -p "..."`、`claude update`、`claude doctor` 等用法都不受影响。

> ⚠ 先想清楚这到底跳过了什么。`bypassPermissions`（等价于 `--dangerously-skip-permissions`）会跳过几乎所有确认——Claude 可以不经询问就执行任意 shell 命令、写删文件、发网络请求。它适合隔离/可回滚的场景，不要在生产代码库、有写权限的服务器、有敏感数据的机器、或克隆下来的不可信仓库里默认开启。各权限模式的准确边界见权限模式说明。

想要折中的话，把上面的 `bypassPermissions` 换成 `acceptEdits`——它只自动接受文件编辑，shell 命令等仍会逐个确认，日常写代码摩擦小很多，又保留了对“执行命令”这类高风险操作的把关：

```
claude() {
  command claude --permission-mode acceptEdits "$@"
}
```

需要临时用回完整确认（default 模式）时，直接调 `command claude` 绕开这个函数即可，无需改配置。

在服务器上长期运行：建一个专用非 root 用户

云服务器默认常常直接以 `root` 登录。但 Claude Code 拒绝在 root / sudo 身份下使用 `--dangerously-skip-permissions`（也就是上面 shell 函数里用的 `bypassPermissions`），会直接报错退出：

```
--dangerously-skip-permissions cannot be used with root/sudo privileges for securi
```

这是有意的安全护栏——一个会跳过所有确认的 agent 若再叠加 root 权限，破坏面就是整台机器。正确做法是建一个专用的非 root 用户来跑 Claude Code：

```
# 1. 新建专用用户(会提示设置密码,其余信息一路回车即可)
sudo adduser claude

# 2. 加入 sudo 组,方便这个用户自己做运维(装依赖、重启服务等)
sudo usermod -aG sudo claude

# 3. 给它配好 SSH 公钥登录:把你的公钥写进 /home/claude/.ssh/authorized_keys
#    再从本地以该用户登录服务器:
#    ssh claude@你的服务器地址

# 4. 以 claude 用户身份运行,这次不会再报 root 错误
claude --dangerously-skip-permissions
```

> 加进 `sudo` 组不会让第 4 步重新报错——那条校验只看「当前进程是不是 root、是不是经 `sudo` 启动」，与组成员身份无关。你以 `claude` 用户直接 SSH 登录后跑 `claude`，uid 不为 0，校验通过；而无人值守时 Claude 也不会替你敲 `sudo`（那要密码）。真正的收益是：万一这个 agent 被诱导做危险操作，权限也被圈在这个普通账户里，而不是 root。

只是想少点确认、又不想彻底放开的话，更稳的选择是 `--permission-mode auto`（带分类器逐条预审），或上面的 `acceptEdits`；若要从管理侧彻底禁用 bypass，用 managed settings 的 `permissions.disableBypassPermissionsMode: "disable"`。

更新策略

原生安装在启动时以及运行过程中会定期检查更新，并在后台静默下载，下次启动时生效——不影响当前会话。

配置发布渠道

用 `autoUpdatesChannel` 控制走哪个渠道：

```
{
  "autoUpdatesChannel": "stable"
}
```

- `"latest"`（默认）：第一时间拿到新特性。
- `"stable"`：滞后约一周，跳过有重大回归的版本。

也可以在会话内用 `/config` → Auto-update channel 切换。企业部署可以通过 managed settings 强制统一渠道。

钉住最低版本

`minimumVersion` 设定一个更新下限，自动更新和 `claude update` 都不会把版本降到这个值以下：

```
{
  "autoUpdatesChannel": "stable",
  "minimumVersion": "2.1.100"
}
```

如果需要“版本不在范围内就直接拒绝启动”这种更强的约束，要用 managed settings 里的 `requiredMinimumVersion` / `requiredMaximumVersion`，而不是 `minimumVersion`。

禁用自动更新

```
{
  "env": {
    "DISABLE_AUTOUPDATER": "1"
  }
}
```

`DISABLE_AUTOUPDATER` 只关闭后台检查，`claude update` 手动更新依旧可用。如果连手动更新也要禁止（比如你通过自己的渠道分发 Claude Code，不希望用户自行升级），要设置更严格的 `DISABLE_UPDATES`。

手动更新 / 安装指定版本

```
claude update
```

原生安装器也支持直接装某个具体版本或渠道：

```
curl -fsSL https://claude.ai/install.sh | bash -s 2.1.89
curl -fsSL https://claude.ai/install.sh | bash -s stable
```

二进制完整性校验

每个发布版本都会附带一份 `manifest.json`，列出各平台二进制的 SHA256 校验和，并用 Anthropic 的 GPG 密钥签名（`31DD DE24 DDFA B679 F42D 7BD2 BAA9 29FF 1A7E CACE`）。macOS 二进制额外经过 Apple 公证（`codesign --verify --verbose ./claude`），Windows 二进制由 “Anthropic, PBC” 签名（`Get-AuthenticodeSignature .\claude.exe`）。通过 apt / dnf / apk 安装时，包管理器会自动用仓库签名密钥校验，无需手动操作。

卸载

不同安装方式对应不同的卸载命令，例如原生安装：

```
rm -f ~/.local/bin/claude
rm -rf ~/.local/share/claude
```

若要彻底清除配置、已登录凭证、MCP 配置与会话历史（不可恢复）：

```
# 用户级配置
rm -rf ~/.claude
rm ~/.claude.json

# 项目级配置（在项目目录下执行）
rm -rf .claude
rm -f .mcp.json
```

> ⚠ VS Code 插件、JetBrains 插件、Desktop 客户端也会写入 `~/.claude/`。只要其中任意一个还装着，这个目录就会在它下次运行时被重新创建。要彻底卸载，需要先卸载这些客户端再删除配置目录。

启动命令行选项

`claude --help` 不会列出全部参数——某个参数没出现在 `--help` 里，不代表它不存在。本页尽量按官方文档收全，但请以你本机版本的实际行为为准。

子命令

除了直接 `claude` 进入交互式会话，`claude` 后面还可以跟一个子命令：

子命令	作用
`claude`	启动交互式会话
`claude "query"`	携带初始 prompt 启动交互式会话
`claude -p "query"`	非交互模式：执行一次查询后退出（脚本 / CI 场景）
`claude -c` / `claude --continue`	继续当前目录下最近一次会话
`claude -r "<id|name>" "query"`	按 ID 或名称恢复指定会话
`claude update`	更新到最新版本
`claude install [version]`	安装/重装原生二进制，可指定版本号或 `stable` / `latest`
`claude auth login` / `logout` / `status`	登录、登出、查看认证状态
`claude mcp`	管理 MCP 服务器（详见 MCP 文档）
`claude mcp login/logout <server>`	对指定 MCP 服务器单独走 OAuth 登录/登出（v2.1.186+）
`claude plugin`	管理插件（别名 `claude plugins`）
`claude agents`	打开 agent 视图，监控与派发并行的后台会话
`claude attach <id>`	在当前终端里接管一个后台会话
`claude respawn <id>`	重启一个后台会话（会话内容保留），`--all` 重启全部
`claude stop <id>` / `claude kill <id>`	停止一个后台会话
`claude logs <id>`	打印某个后台会话最近的输出
`claude rm <id>`	从列表移除一个后台会话（本地文件仍保留，仍可 `--resume`）
`claude daemon status` / `claude daemon stop --any`	查看/停止后台会话的 supervisor 进程，用于故障恢复
`claude project purge [path]`	删除某个项目在本地的全部状态（会话记录、任务列表、编辑历史等）

子命令	作用
`claude setup-token`	为 CI/脚本生成长期有效的 OAuth token
`claude remote-control`	以服务模式启动，供 claude.ai / Claude App 远程控制
`claude gateway --config gateway.yaml`	启动自托管的 Claude apps 网关（面向企业管理员，v2.1.195+）
`claude ultrareview [target]`	非交互式跑一次 ultrareview 深度评审

常用参数速查表

按使用场景分组，便于查找；完整字段名与说明见官方 CLI reference。

权限与安全

参数	说明
`--permission-mode <mode>`	以指定权限模式启动：`default`（即 `manual`）、`acceptEdits`、`plan`、`auto`、`dontAsk`、`bypassPermissions`
`--dangerously-skip-permissions`	跳过全部权限确认，等价于 `--permission-mode bypassPermissions`
`--allow-dangerously-skip-permissions`	把 `bypassPermissions` 加入 `Shift+Tab` 循环列表，但不强制一开始就用它
`--allowedTools` / `--allowed-tools`	无需确认即可执行的工具白名单，支持模式匹配，如 `"Bash(git log *)"`
`--disallowedTools` / `--disallowed-tools`	拒绝规则，例如 `"Bash(rm *)"` 只拒绝匹配的调用，裸工具名如 `"Edit"` 则整体移除该工具
`--tools`	限制 Claude 可用的内置工具集合，如 `"Bash,Edit,Read"`；不影响 MCP 工具
`--permission-prompt-tool`	指定一个 MCP 工具在非交互模式下代为处理权限确认
`--safe-mode`	禁用全部自定义项（CLAUDE.md、skills、插件、hooks、MCP 等）以排查配置问题，比 `--bare` 更彻底
`--bare`	极简模式：跳过 hooks/skills/插件/MCP/自动记忆/CLAUDE.md 自动发现，只保留 Bash、文件读写工具

模型与推理

参数	说明
`--model <name>`	设置本次会话模型，支持别名 `sonnet` / `opus` / `haiku` / `fable` 或完整模型名
`--effort <level>`	设置推理强度：`low` / `medium` / `high` / `xhigh` / `max`，具体可用档位取决于模型
`--fallback-model <a,b,...>`	主模型过载/不可用时按顺序自动降级到备用模型
`--advisor <model>`	为本次会话启用顾问模型（第二模型在关键节点提供建议，v2.1.98+）
`--betas`	附加 `anthropic-beta` 请求头（仅 API Key 用户）

非交互模式与输出

参数	说明
`--print`， `-p`	打印结果后退出，不进入交互模式
`--output-format <text|json|stream-json>`	非交互模式的输出格式
`--input-format <text|stream-json>`	非交互模式的输入格式
`--max-turns <n>`	限制 agentic 轮数，超出直接报错退出
`--max-budget-usd <n>`	单次调用的最高花费上限（仅非交互模式）
`--json-schema <schema>`	让最终输出按给定 JSON Schema 校验（结构化输出）
`--include-partial-messages`	输出流式的中间结果片段（需配合 `stream-json`）
`--include-hook-events`	输出流里包含全部 hook 生命周期事件
`--prompt-suggestions`	每轮结束后额外输出一条“预测下一句 prompt”
`--no-session-persistence`	不落盘会话记录，无法被 `--resume`
`--replay-user-messages`	把 stdin 收到的用户消息原样回写到 stdout，便于确认

系统提示词定制

参数	行为
`--system-prompt <text>`	完全替换默认系统提示词
`--system-prompt-file <path>`	用文件内容替换默认系统提示词
`--append-system-prompt <text>`	在默认提示词末尾追加内容
`--append-system-prompt-file <path>`	追加文件内容到默认提示词末尾
`--exclude-dynamic-system-prompt-sections`	把工作目录、环境信息等“因机器/用户而异”的片段移到第一条用户消息里，提升跨用户/跨机器的 prompt 缓存命中率

`--system-prompt` 与 `--system-prompt-file` 互斥；两个 append 参数可以和任一替换参数叠加使用。想保留默认的工具指引与安全说明、只是叠加规则时用 append；想完全接管身份定位（例如把 Claude Code 当成流水线里的非编程 agent）时才用替换。

会话恢复与并行

参数	说明
`--continue`，`-c`	恢复当前目录最近一次会话
`--resume`，`-r <id|name>`	按 ID/名称恢复会话，或不带参数打开交互式选择器
`--fork-session`	恢复会话时创建新会话 ID，而不是复用原 ID
`--from-pr <number|url>`	恢复与指定 PR 关联的会话（Claude 创建 PR 时会自动关联）
`--session-id <uuid>`	为本次会话指定一个 UUID
`--name`，`-n <name>`	设置会话显示名称，可用 `--resume <name>` 找回
`--worktree`，`-w [name]`	在独立的 git worktree 里启动，便于并行开发互不干扰
`--tmux`	为 worktree 创建 tmux 会话（需配合 `--worktree`）
`--bg`，`--background`	以后台 agent 形式启动并立即返回
`--exec <cmd>`	配合 `--bg`，把一条 shell 命令作为后台任务运行，而非启动 Claude 会话
`--agent <name>`	指定本次会话使用的 subagent
`--forward-subagent-text`	🆕 v2.1.211 新增：把 subagent 的思考过程也带进输出（也有同名环境变量）
`--teammate-mode`	设置 agent team 队友的展示方式：`in-process` / `auto` / `tmux` / `iterm2`
`--teleport`	把一个 web 会话拉到本地终端继续
`--remote-control`，`--rc [name]`	启动交互式会话并开启远程控制，可从 claude.ai / Claude App 继续操作

目录、配置与 MCP

参数	说明
`--add-dir <path...>`	额外授权 Claude 读写的工作目录（不会自动发现这些目录下的 `.claude/` 配置，`.claude/skills/` 除外）
`--settings <path|json>`	指定一份 settings JSON 文件或内联 JSON 字符串，覆盖同名配置项
`--setting-sources <user,project,local>`	指定加载哪些来源的 settings
`--mcp-config <file...>`	从 JSON 文件/字符串加载 MCP 服务器配置
`--strict-mcp-config`	只使用 `--mcp-config` 里的 MCP 服务器，忽略其它来源
`--plugin-dir <path>`	仅本次会话从指定目录/ `.zip` 加载一个插件，可重复传参
`--plugin-url <url>`	仅本次会话从 URL 拉取插件 `.zip`
`--disable-slash-commands`	本次会话禁用全部 skills 与命令

调试与其它

参数	说明
`--debug ["cat1,cat2"]`	开启调试模式，可选按分类过滤（如 `"api,hooks"`，`!` 前缀表示排除）
`--debug-file <path>`	把调试日志写到指定文件（隐式开启调试模式）
`--verbose`	展示完整的逐轮输出
`--version`， `-v`	输出版本号
`--ide`	启动时若刚好只有一个可用 IDE，则自动连接
`--init`	在非交互模式下，会话开始前运行 `init` 匹配的 Setup hooks
`--init-only`	只运行 Setup / SessionStart hooks 然后退出，不进入对话
`--chrome` / `--no-chrome`	启用/禁用 Chrome 浏览器自动化集成
`--cloud <task>`	用给定任务描述在 claude.ai 上新建一个 web 会话（旧名 `--remote`）
`--ax-screen-reader`	输出无装饰边框/动画的纯文本，适配屏幕阅读器

示例

```
# 交互式启动，带初始 prompt
claude “解释一下这个仓库的整体架构”

# 非交互模式，管道输入，输出 JSON
git log --oneline -20 | claude -p “总结这些提交” --output-format json

# 在独立 worktree 里并行修一个 bug
claude --worktree fix-auth-bug

# 计划模式起步，并允许后续切到 bypassPermissions
claude --permission-mode plan --allow-dangerously-skip-permissions

# CI 场景：限制轮数、限制预算、禁用交互确认
claude -p --max-turns 5 --max-budget-usd 2.00 --dangerously-skip-permissions “跑一遍
```

常见误用提醒

- `claude --dangerously-skip-permissions daemon <id>` 在 v2.1.199 之前会被当成新会话的 prompt 文本（因为 `daemon <id>` 被解释为消息内容，而不是子命令），v2.1.199 起才正确路由到 `daemon` 子命令。只有紧跟在最前面的 `--dangerously-skip-permissions` / `--allow-dangerously-skip-permissions` 才会触发这个特殊路由。
- `--bg` 不能和 `-p` / `--print` 同时使用（v2.1.198 起会直接报错拒绝，而不是静默产生一个无法 attach 的会话）。
- 子命令拼错时（如 `claude udpate`）会提示“你是不是想输入 xxx”，并直接退出而不会误当成新会话的 prompt。

环境变量

Claude Code 的绝大多数运行时行为都能用环境变量控制，且效果等价于在 `settings.json` 的 `env` 字段里写同名键值——写进 `settings.json` 的好处是不用改 shell 配置、且能提交到仓库让团队共享。

settings.json

```
{
  "env": {
    "API_TIMEOUT_MS": "1200000",
    "DISABLE_AUTOUPDATER": "1"
  }
}
```

优先级：同一个行为如果既有环境变量又有 settings 字段，环境变量优先（例如 `ANTHROPIC_MODEL` 优先于 `model` 设置）。环境变量与 CLI 参数、`/` 命令的优先级则因功能而异，例如 `--model` / `/model` 优先于 `ANTHROPIC_MODEL`，而 `CLAUDE_CODE_EFFORT_LEVEL` 优先于 `/effort`。环境变量在启动时读取一次，运行中修改需要重启 `claude` 才生效。

下面按用途分类列出常用变量；完整的长尾列表（约 250 个）见官方 Environment variables 文档。

认证与 API 路由

变量	说明
`ANTHROPIC_API_KEY`	直接用 API Key 鉴权，优先于已登录的订阅账号。非交互模式（`-p`）下始终生效；交互模式下首次使用会提示确认。`unset ANTHROPIC_API_KEY` 可临时切回订阅账号
`ANTHROPIC_AUTH_TOKEN`	自定义 `Authorization` 头的值（会自动加 `Bearer` 前缀）
`ANTHROPIC_BASE_URL`	把请求路由到自建代理/网关，而非 `api.anthropic.com`
`ANTHROPIC_CUSTOM_HEADERS`	附加自定义请求头，格式 `Name: Value`，多个用换行分隔
`ANTHROPIC_BETAS`	逗号分隔的额外 `anthropic-beta` header，任何鉴权方式（含订阅）都生效
`CLAUDE_CODE_USE_BEDROCK` / `CLAUDE_CODE_USE_VERTEX` / `CLAUDE_CODE_USE_FOUNDRY` / `CLAUDE_CODE_USE_ANTHROPIC_AWS`	切换到 Amazon Bedrock / Google Cloud's Agent Platform / Microsoft Foundry / Claude Platform on AWS
`AWS_BEARER_TOKEN_BEDROCK`	Amazon Bedrock 的 API Key 鉴权
`CLAUDE_CODE_OAUTH_TOKEN`	claude.ai 的 OAuth access token，适合 SDK/自动化环境，可用 `claude setup-token` 生成
`CLAUDE_CODE_OAUTH_REFRESH_TOKEN` / `CLAUDE_CODE_OAUTH_SCOPES`	用 refresh token 免浏览器完成登录，常用于容器/CI 预置认证
`CLAUDE_CONFIG_DIR`	覆盖配置目录（默认 `~/.claude`），便于多账号并存，如 `alias claude-work='CLAUDE_CONFIG_DIR=~/.claude-work claude'`
`VERTEX_REGION_CLAUDE_*`	一组变量，分别覆盖各代 Claude 模型在 Google Cloud's Agent Platform 上的区域（如 `VERTEX_REGION_CLAUDE_5_SONNET`）

模型选择与推理

变量	说明
`ANTHROPIC_MODEL`	设置使用的模型，优先于 `model` 设置，但会被 `--model` / `/model` 覆盖
`ANTHROPIC_DEFAULT_SONNET_MODEL` / `_OPUS_MODEL` / `_HAIKU_MODEL` / `_FABLE_MODEL`	分别覆盖 `/model` 里 `sonnet` / `opus` / `haiku` / `fable` 别名对应的具体模型 ID，常用于网关/自定义路由场景
`CLAUDE_CODE_SUBAGENT_MODEL`	subagent 使用的模型，与主会话模型独立配置
`CLAUDE_CODE_EFFORT_LEVEL`	设置推理强度（`low` / `medium` / `high` / `xhigh` / `max` / `auto`），优先于 `/effort` 与 `effortLevel` 设置
`MAX_THINKING_TOKENS`	扩展思考的 token 预算上限；设为 `0` 在 Anthropic API 上禁用思考（Fable 5 除外，它无法关闭思考）
`CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING`	禁用 Opus 4.6 / Sonnet 4.6 的自适应推理，回退到固定思考预算（对 Fable 5、Sonnet 5、Opus 4.7+ 无效）
`CLAUDE_CODE_MAX_OUTPUT_TOKENS`	单次请求的最大输出 token 数
`CLAUDE_CODE_MAX_CONTEXT_TOKENS`	覆盖 Claude Code 假定的上下文窗口大小，用于自定义模型路由场景
`CLAUDE_CODE_AUTO_COMPACT_WINDOW` / `CLAUDE_AUTOCOMPACT_PCT_OVERRIDE`	自定义自动压缩触发的上下文容量阈值及百分比
`DISABLE_AUTO_COMPACT`	禁用自动压缩，保留手动 `/compact`
`DISABLE_COMPACT`	连手动 `/compact` 一起禁用
`CLAUDE_CODE_DISABLE_1M_CONTEXT`	禁用 1M 上下文窗口支持，常用于有合规要求的企业环境

网络、代理与超时

变量	说明
`API_TIMEOUT_MS`	单次 API 请求超时（默认 600000ms，即 10 分钟）
`HTTP_PROXY` / `HTTPS_PROXY` / `NO_PROXY`	标准代理配置，`NO_PROXY` 支持逗号分隔的域名/IP 白名单
`CLAUDE_CODE_CERT_STORE`	TLS 信任的 CA 证书来源，`bundled`（内置 Mozilla CA 集）和/或 `system`，默认两者都用
`CLAUDE_CODE_CLIENT_CERT` / `CLAUDE_CODE_CLIENT_KEY` / `CLAUDE_CODE_CLIENT_KEY_PASSPHRASE`	mTLS 客户端证书鉴权
`CLAUDE_ENABLE_STREAM_WATCHDOG` / `CLAUDE_STREAM_IDLE_TIMEOUT_MS`	流式响应的空闲看门狗开关与超时时间，防止连接静默卡死
`CLAUDE_CODE_PROXY_RESOLVES_HOSTS`	允许代理而非本机做 DNS 解析

Bash 工具行为

变量	说明
`BASH_DEFAULT_TIMEOUT_MS`	长时间运行命令的默认超时（默认 120000ms）
`BASH_MAX_TIMEOUT_MS`	模型可为单条命令设置的超时上限（默认 600000ms）
`BASH_MAX_OUTPUT_LENGTH`	输出超过该字符数时落盘保存，Claude 只拿到文件路径 + 预览
`CLAUDE_CODE_SHELL`	指定 Bash 工具使用的 shell 路径（仅支持 `bash` / `zsh`）
`CLAUDE_CODE_SHELL_PREFIX`	给所有 Claude Code 派生的 shell 命令套一层前缀命令，常用于审计/日志
`CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR`	每条 Bash/PowerShell 命令执行后都恢复到最初的工作目录
`CLAUDE_CODE_USE_POWERSHELL_TOOL`	Windows/Linux/macOS 上启用 PowerShell 工具（需要 `pwsh` 在 PATH 上）
`CLAUDE_CODE_GIT_BASH_PATH`	Windows 下手动指定 Git Bash 路径

更新、遥测与隐私

变量	说明
`DISABLE_AUTOUPDATER`	关闭后台自动更新检查，`claude update` 仍可手动更新
`DISABLE_UPDATES`	连手动更新一起彻底禁止，适合自建分发渠道
`CLAUDE_CODE_PACKAGE_MANAGER_AUTO_UPDATE`	让 Homebrew/WinGet 安装也能在后台自动跑升级命令
`DISABLE_TELEMETRY` / `DO_NOT_TRACK`	退出遥测收集（不含代码、文件路径、bash 命令等用户数据），两者等价
`DISABLE_ERROR_REPORTING`	退出 Sentry 错误上报
`DISABLE_GROWTHBOOK`	禁用 feature flag 拉取，统一使用代码内置默认值
`CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC`	一次性等价于同时设置 `DISABLE_AUTOUPDATER` + `DISABLE_FEEDBACK_COMMAND` + `DISABLE_ERROR_REPORTING` + `DISABLE_TELEMETRY`，适合内网/隔离环境
`DISABLE_COST_WARNINGS`	关闭费用提醒
`DISABLE_INSTALLATION_CHECKS`	关闭安装方式警告（仅在你手动管理安装路径时使用）
`IS_DEMO`	演示模式：隐藏邮箱/组织名，跳过 onboarding，适合直播/录屏

会话、上下文与运行时

变量	说明
`CLAUDE_CODE_DISABLE_AUTO_MEMORY`	禁用/强制开启自动记忆（`1` 禁用，`0` 强制开启）
`CLAUDE_CODE_DISABLE_CLAUDE_MDS`	不加载任何 CLAUDE.md 记忆文件
`CLAUDE_CODE_MAX_TURNS`	未显式传 `--max-turns` 时的默认轮数上限
`CLAUDE_CODE_MAX_TOOL_USE_CONCURRENCY`	只读工具与 subagent 的最大并行数（默认 10）
`CLAUDE_CODE_ENABLE_TASKS`	控制是否使用结构化 Task 工具而非旧版 `TodoWrite`（2.1.142 起默认开启）
`CLAUDE_CODE_SKIP_PROMPT_HISTORY`	不写会话记录到磁盘，无法被 `--resume` / `--continue` 找到，适合一次性脚本化会话
`CLAUDE_CODE_DISABLE_FILE_CHECKPOINTING`	禁用文件检查点，`/rewind` 将无法回滚代码改动
`CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS`	从系统提示词里移除内置的 commit/PR 工作流指引
`CLAUDECODE`	Claude Code 派生的子进程里会被设为 `1`，用于脚本判断“我是不是被 Claude Code 调用的”
`CLAUDE_CODE_CHILD_SESSION`	类似 `CLAUDECODE`，但只有 Claude Code 自己派生子进程时才设置，可用来区分嵌套会话
`CLAUDE_CODE_SESSION_ID`	子进程/hook 里的当前会话 ID，用于日志关联
`CLAUDE_CODE_SIMPLE`	精简系统提示词 + 仅 Bash/文件读写工具，等价于 `--bare`
`CLAUDE_ENV_FILE`	一个 shell 脚本路径，每条 Bash 命令执行前会先 source 它，用于跨命令保持 virtualenv/conda 的激活状态

后台任务与并行

变量	说明
`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS`	禁用全部后台任务能力（`run_in_background`、自动转后台、`Ctrl+B`）
`CLAUDE_AUTO_BACKGROUND_TASKS`	强制开启长任务自动转后台（约运行 2 分钟后）
`CLAUDE_ASYNC_AGENT_STALL_TIMEOUT_MS`	后台 subagent 的停滞超时（默认 10 分钟无进展则判定失败）
`TASK_MAX_OUTPUT_LENGTH`	subagent 输出截断前的最大字符数（默认 32000）
`CLAUDE_CODE_DISABLE_AGENT_VIEW`	关闭 `claude agents` / `--bg` / `/background` 等后台 agent 能力
`CLAUDE_CODE_TASK_LIST_ID`	多个 Claude Code 实例共享同一个任务列表

MCP 相关

变量	说明
`MCP_TIMEOUT`	MCP 服务器启动超时（默认 30000ms）
`MCP_TOOL_TIMEOUT`	单次 MCP 工具调用超时（默认约 28 小时，几乎不设限）
`MCP_CONNECTION_NONBLOCKING`	设为 `0` 恢复旧版“启动时阻塞等待 MCP 连接”行为（2.1.142 起默认非阻塞）
`MCP_CONNECT_TIMEOUT_MS`	阻塞模式下等待 MCP 连接批次完成的超时（默认 5000ms）
`MCP_SERVER_CONNECTION_BATCH_SIZE` / `MCP_REMOTE_SERVER_CONNECTION_BATCH_SIZE`	本地（stdio）/远程（HTTP、SSE）MCP 服务器启动时的并行连接数
`MAX_MCP_OUTPUT_TOKENS`	MCP 工具响应允许的最大 token 数（默认 25000）
`CLAUDE_CODE_MCP_ALLOWLIST_ENV`	只给 stdio MCP 服务器传递安全基线环境变量，而不是继承整个 shell 环境
`CLAUDE_CODE_MCP_TOOL_IDLE_TIMEOUT`	远程 MCP 工具调用的空闲超时（默认 5 分钟，超时则报错而非无限等待）
`ENABLE_TOOL_SEARCH`	控制 MCP 工具是否延迟加载（按需检索）还是一次性全部载入上下文

插件与技能

变量	说明
`CLAUDE_CODE_PLUGIN_CACHE_DIR`	覆盖插件根目录（默认 `~/.claude/plugins`）
`CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS`	安装/更新插件时 git 操作的超时（默认 120000ms）
`CLAUDE_CODE_PLUGIN_PREFER_HTTPS`	插件仓库用 HTTPS 而非 SSH 克隆，适合没配 SSH key 的 CI/容器环境
`CLAUDE_CODE_DISABLE_BUNDLED_SKILLS`	禁用随 Claude Code 自带的技能与工作流（不影响自定义/插件技能）
`CLAUDE_CODE_SYNC_SKILLS`	非交互模式下，首次查询前把你在 claude.ai 上启用的技能同步到本地
`SLASH_COMMAND_TOOL_CHAR_BUDGET`	技能元数据展示给 Skill 工具时的字符预算，技能很多时可调大避免描述被裁剪

终端 UI 与可访问性

变量	说明
`CLAUDE_CODE_ACCESSIBILITY`	保持原生终端光标可见，便于屏幕放大镜类工具跟踪光标
`CLAUDE_AX_SCREEN_READER`	输出无装饰边框/动画的纯文本，适配屏幕阅读器
`CLAUDE_CODE_DISABLE_MOUSE` / `CLAUDE_CODE_DISABLE_MOUSE_CLICKS`	全屏渲染下禁用鼠标追踪 / 只禁用点击拖拽（保留滚轮）
`CLAUDE_CODE_DISABLE_ALTERNATE_SCREEN`	禁用全屏渲染，回退到传统主屏渲染器（终端原生回滚缓冲区可用）
`CLAUDE_CODE_NO_FLICKER`	启用全屏渲染（减少闪烁、长会话内存更平稳的实验性渲染器）
`CLAUDE_CODE_TMUX_TRUECOLOR`	允许 tmux 内使用 24 位真彩色（需先在 `~/.tmux.conf` 开启 `Tc`）
`CLAUDE_CODE_SCROLL_SPEED`	全屏渲染下的鼠标滚轮倍率
`CLAUDE_CODE_HIDE_CWD`	启动 Logo 里隐藏工作目录路径，适合截图/录屏
`CLAUDE_CODE_SYNTAX_HIGHLIGHT`	设为 `false` 关闭 diff 输出的语法高亮

OpenTelemetry 监控

变量	说明
`CLAUDE_CODE_ENABLE_TELEMETRY`	启用 OpenTelemetry 数据采集，是配置其它 OTel 变量的前提
`OTEL_LOG_USER_PROMPTS` / `OTEL_LOG_ASSISTANT_RESPONSES`	是否在日志事件中包含用户提示词/模型回复原文（默认脱敏）
`OTEL_LOG_TOOL_CONTENT` / `OTEL_LOG_TOOL_DETAILS`	是否包含工具调用的输入输出内容与详细参数（默认关闭以保护敏感信息）
`OTEL_METRICS_INCLUDE_SESSION_ID` / `_ACCOUNT_UUID` / `_VERSION`	控制指标里包含哪些附加维度
标准 OTel 变量	`OTEL_METRICS_EXPORTER`、`OTEL_LOGS_EXPORTER`、`OTEL_EXPORTER_OTLP_ENDPOINT`、`OTEL_EXPORTER_OTLP_PROTOCOL`、`OTEL_EXPORTER_OTLP_HEADERS`、`OTEL_RESOURCE_ATTRIBUTES` 等标准导出器配置同样支持

调试

变量	说明
`DEBUG`	设为 `1` / `true` / `yes` / `on` 开启调试模式，等价于 `--debug`
`CLAUDE_CODE_DEBUG_LOGS_DIR`	调试日志文件路径（默认 `~/.claude/debug/<timestamp>.txt`）
`CLAUDE_CODE_DEBUG_LOG_LEVEL`	日志级别：`verbose` / `debug`（默认）/ `info` / `warn` / `error`

以上覆盖了日常最常用的变量类别。企业级场景（网关鉴权、Perforce 集成、屏幕阅读器细节、agent team 相关变量等）数量繁多且更新频繁，建议直接查阅官方 Environment variables 页面获取权威、实时的完整列表。

配置文件

Claude Code 的配置分散在几类文件里，各自管不同的东西：`settings.json` 管行为开关与权限，`CLAUDE.md` 管“给 Claude 的说明书”，`.mcp.json` 管项目级 MCP 服务器，`~/.claude.json` 管登录状态等杂项。理解它们各自的位置和优先级，能解决大部分“为什么我改了配置却没生效”的问题。

作用域总览

Claude Code 用统一的“作用域”体系决定一份配置对谁生效、要不要进代码仓库：

作用域	位置	影响范围	是否随仓库共享
Managed（企业管控）	服务端下发 / MDM（plist、注册表）/ 系统级 `managed-settings.json`	组织全体成员或本机全部用户	是（IT 部署）
User（个人）	`~/.claude/` 目录	你自己，跨所有项目	否
Project（项目）	仓库内 `.claude/` 目录	该仓库的全部协作者	是（提交进 git）
Local（本机项目级）	`.claude/settings.local.json`	你自己，仅当前仓库	否（Claude Code 自动创建时会加进 `.gitignore`）

不同功能对应的具体文件路径：

功能	User 位置	Project 位置	Local 位置
settings.json	`~/.claude/settings.json`	`.claude/settings.json`	`.claude/settings.lo`
CLAUDE.md	`~/.claude/CLAUDE.md`	`./CLAUDE.md` 或 `./.claude/CLAUDE.md`	`./CLAUDE.local.md`
Subagent	`~/.claude/agents/`	`.claude/agents/`	无
MCP 服务器	`~/.claude.json`	`.mcp.json`	`~/.claude.json`（按项
插件	`~/.claude/settings.json`	`.claude/settings.json`	`.claude/settings.lo`

Windows 上 `~/.claude` 对应 `%USERPROFILE%\.claude`。

优先级与合并规则

同一个配置项在多处出现时，按下面顺序生效（从高到低）：

1. Managed（企业管控）—— 任何其它层级都无法覆盖，包括命令行参数
2. 命令行参数（如 `--settings`）—— 仅本次会话临时覆盖
3. Local（`.claude/settings.local.json`）
4. Project（`.claude/settings.json`）
5. User（`~/.claude/settings.json`）—— 兜底

标量值（字符串/数字/布尔）是“高优先级覆盖低优先级”；数组类配置（如 `permissions.allow`）则是跨层级拼接去重，而不是覆盖——例如企业策略里 `allowWrite` 设了 `["/opt/company-tools"]`，你个人又加了 `["~/.kube"]`，最终两条路径都生效。`fallbackModel`（有序降级链）和 managed 层设置的 `availableModels` 是例外，由最高优先级那一份文件整体决定，不做合并。

想知道当前会话实际加载了哪些配置来源，运行 `/status`，查看 `Setting sources` 一行；配置文件本身有 JSON 格式错误或校验失败，`/doctor` 会列出具体文件与错误详情。

settings.json

这是最核心的配置文件，支持权限规则、环境变量、hooks、状态栏、插件开关等几乎所有行为定制。

.claude/settings.json

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "allow": ["Bash(npm run lint)", "Bash(npm run test *)"],
    "deny": ["Bash(curl *)", "Read(./.env)", "Read(./.env.*)", "Read(./secrets/**)
  },
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1"
  },
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh"
  }
}
```

顶部的 `$schema` 指向官方 JSON Schema，在 VS Code / Cursor 等支持 JSON Schema 校验的编辑器里能获得自动补全和内联校验（schema 更新有滞后，新字段偶尔会被误报，不代表配置真的错了）。

何时生效

Claude Code 会监听配置文件变化并自动重新加载，`permissions`、`hooks`、`apiKeyHelper` 等大多数字段改完立刻在当前会话生效（`ConfigChange` hook 会为每次检测到的变化触发一次）。只有两个字段例外，改完要下次重启或用对应命令才生效：

- `model` —— 用 `/model` 临时切换
- `outputStyle` —— 属于系统提示词的一部分，只有 `/clear` 或重启会话时才会重建

常用字段精选

`settings.json` 支持的字段有几百个（权限、hooks、沙箱、插件、企业策略……），这里只列日常最常用的一批；完整列表见官方 Settings 文档。

字段	说明
`permissions.allow` / `ask` / `deny`	权限规则数组，按 `Tool` 或 `Tool(specifier)` 语法匹配，如 `*)"`、`"Read(./.env)"`。规则按 deny → ask → allow 顺序计
`permissions.defaultMode`	默认权限模式（`default` / `acceptEdits` / `plan` / `auto` / `dontAsk` / `byp` `auto` 在 project/local 里会被忽略，只能在 `~/.claude/sett` 止仓库自我授权
`permissions.additionalDirectories`	等价于持久化的 `--add-dir`
`model`	覆盖默认模型，如 `"claude-sonnet-5"`
`fallbackModel`	主模型过载时按顺序尝试的降级模型链，最多 3 个
`effortLevel`	跨会话保留的推理强度（`low` / `medium` / `high` / `xhigh`），/ 里
`env`	应用到每个会话及其子进程的环境变量，等价于逐条设置环境变
`hooks`	生命周期事件（`PreToolUse`、`PostToolUse`、`Stop` 等）见自定义与扩展
`statusLine`	自定义状态栏命令，见 `/statusline`
`outputStyle`	系统提示词风格，如 `"Explanatory"`
`autoUpdatesChannel`	更新渠道，见安装与升级
`autoMemoryEnabled` / `autoMemoryDirectory`	自动记忆开关与自定义存储目录
`fileCheckpointingEnabled`	是否为 `/rewind` 做文件快照（默认开启）
`cleanupPeriodDays`	会话记录本地保留天数（默认 30）
`editorMode`	输入框按键模式：`"normal"` 或 `"vim"`
`disableBundledSkills` / `disableWorkflows`	关闭随附技能/动态工作流
`enabledPlugins`	插件启用状态，格式 `"插件名@marketplace名": true/fal`
`extraKnownMarketplaces`	注册额外的插件 marketplace 源

字段	说明
`attribution`	自定义 git commit / PR 的署名文案
`claudeMdExcludes`	排除指定路径的 `CLAUDE.md`，常用于 monorepo 中跳过其它团

> ⚠ `.claude/settings.json` 是项目共享文件，会被提交进仓库——不要在里面放 API Key、Token 等敏感信息。个人专属的临时配置放 `.claude/settings.local.json`（Claude Code 创建时会自动加进 `.gitignore`，自己手建则要记得手动加）。

权限规则速查

`permissions.allow` / `ask` / `deny` 是最常用的一组字段，规则格式为 `Tool` 或 `Tool(specifier)`：

规则	效果
`Bash`	匹配全部 Bash 命令
`Bash(npm run *)`	匹配以 `npm run` 开头的命令
`Read(./.env)`	匹配读取 `.env` 文件
`WebFetch(domain:example.com)`	匹配对 example.com 的 fetch 请求
`mcp__*`	匹配全部 MCP 工具

排除敏感文件是最常见的用法之一：

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

CLAUDE.md：项目记忆

`CLAUDE.md` 是一份纯文本 markdown 文件，每次会话开始时都会被加载，用来存放“你不想每次重新口述一遍”的项目知识：构建/测试命令、代码规范、架构决策、目录约定。

存放位置与加载顺序

作用域	位置	用途
企业策略	系统级目录（见下方管控配置）	组织级强制指令，所有用户在所有仓库都会加载
用户	`~/.claude/CLAUDE.md`	你的个人偏好，对所有项目生效
项目	`./CLAUDE.md` 或 `./.claude/CLAUDE.md`	团队共享的项目说明，进版本控制
本机项目级	`./CLAUDE.local.md`	个人在这个项目里的私有偏好，建议加进 `.gitignore`

Claude Code 从当前工作目录往上逐级查找 `CLAUDE.md` / `CLAUDE.local.md` 并全部拼接进上下文（离工作目录越近的排在越后面，即“越具体的指令越靠后被读到”）；子目录下的 `CLAUDE.md` 则是按需加载——只有 Claude 读取该子目录下的文件时才会带上。

用 `/init` 可以自动生成初版 `CLAUDE.md`（已存在时则给出改进建议而非覆盖）；建议控制在 200 行以内，太长的文件会占用上下文并降低指令遵循度，长内容应该拆到 `.claude/rules/` 里按需加载，而不是塞进主文件。

`@` 导入与 AGENTS.md 兼容

CLAUDE.md 里可以用 `@path/to/file` 语法导入其它文件，启动时一并展开进上下文（最多 4 层递归，反引号包裹的 `` `@path` `` 不会被当作导入）：

```
参见 @README 了解项目概况，@package.json 查看可用的 npm 命令。
# 个人偏好
- @~/.claude/my-project-instructions.md
```

如果仓库已经在用 `AGENTS.md` 给其它编码工具用，不用重复维护两份，导入一下即可：

CLAUDE.md

```
@AGENTS.md

## Claude Code 专属说明

`src/billing/` 下的改动一律先进计划模式。
```

用 `.claude/rules/` 拆分大型说明

大项目建议把 `CLAUDE.md` 拆成多个主题文件放进 `.claude/rules/`，每个文件一个主题，可选用 `paths` frontmatter 限定只在操作匹配文件时才加载，节省上下文：

.claude/rules/api-design.md

```
---
paths:
  - "src/api/**/*.ts"
---

# API 开发规则

- 所有接口必须包含入参校验
- 统一使用标准错误响应格式
```

不带 `paths` 的规则和 `.claude/CLAUDE.md` 一样，每次都会加载。`~/.claude/rules/` 则是个人级、跨项目生效的规则目录。

自动记忆（Auto Memory）

除了你手写的 `CLAUDE.md`，Claude 还会在工作过程中自己积累笔记——构建命令、调试心得、代码风格偏好等，不需要你手动维护。

- 存放位置：`~/.claude/projects/<project>/memory/`，以 `MEMORY.md` 为索引入口，配合若干主题文件
- 每次会话只加载 `MEMORY.md` 的前 200 行或 25KB（以先到者为准），主题文件按需读取
- 默认开启，可在 `settings.json` 里设 `"autoMemoryEnabled": false` 关闭，或在会话内用 `/memory` 切换
- 同一个 git 仓库的所有 worktree 共享同一份自动记忆，但不跨机器同步

自动记忆是纯 markdown，随时可以用 `/memory` 打开查看、编辑或删除。

.mcp.json：项目级 MCP 服务器

项目要用到的 MCP 服务器配置写在仓库根目录的 `.mcp.json` 里，和团队共享：

.mcp.json

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${GITHUB_TOKEN}"
      }
    }
  }
}
```

出于安全考虑，仓库自带的 `.mcp.json` 不会被自动信任启动——首次使用需要你在会话里手动批准。想提前在 `settings.json` 里声明批准/拒绝名单，用 `enabledMcpjsonServers` / `disabledMcpjsonServers`：

.claude/settings.local.json

```
{
  "enabledMcpjsonServers": ["github", "memory"]
}
```

个人级、只在你本机生效的 MCP 服务器配置（以及 OAuth 会话）则存放在 `~/.claude.json` 里，通常通过 `claude mcp add` / `/mcp` 管理，不建议手动编辑。

~/.claude.json：其它状态

这个文件保存的是“不属于团队共享配置”的杂项状态：OAuth 登录会话、用户/本机作用域的 MCP 服务器配置、每个项目的信任状态与已批准工具缓存等。Claude Code 会自动为配置文件创建带时间戳的备份并保留最近 5 份，避免误操作导致数据丢失。日常不需要手动改这个文件，出问题时优先用 `claude doctor` / `/doctor` 诊断，而不是直接编辑。

企业级管控配置（简述）

面向组织统一下发的策略配置，不属于个人日常配置范围，这里只做提要：

- 服务端下发：通过 claude.ai 管理后台或自托管的 Claude apps 网关，登录时远程下发
- MDM / 系统策略：macOS 用 `com.anthropic.claudecode` 配置描述文件，Windows 用 `HKLM\SOFTWARE\Policies\ClaudeCode` 注册表项
- 文件下发：`managed-settings.json`（以及可选的 `managed-settings.d/` 目录做策略分片）放在系统级目录（macOS `/Library/Application Support/ClaudeCode/`、Linux `/etc/claude-code/`、Windows `C:\Program Files\ClaudeCode\`）

这一层配置任何用户/项目/本机配置都无法覆盖，常用于强制安全策略（禁用 `bypassPermissions`、限制可用模型、企业级 `CLAUDE.md` 等），具体格式和字段与个人 `settings.json` 完全一致。

排查配置问题

- `/status` —— 查看当前会话实际加载了哪些配置来源（`Setting sources`），配置文件有语法/校验错误时也会在这里列出受影响的文件
- `/doctor` —— 给出每个错误的详细信息与来源，按 `f` 可以让 Claude 直接修复
- `/memory` —— 确认 `CLAUDE.md` / `CLAUDE.local.md` /rules 是否被正确加载
- 指令没被遵守，优先检查：文件是否在会被加载的位置、指令是否够具体、多份 `CLAUDE.md` 之间是否有冲突

> Claude 把配置分成两层：settings 是硬性约束（不管 Claude 怎么“决定”，客户端都会强制执行，比如 `permissions.deny`），CLAUDE.md 是软性指导（会影响 Claude 的行为倾向，但不保证严格遵守）。需要“无论如何都要拦住”的场景，用 settings/hooks；需要“教会 Claude 怎么做”的场景，用 CLAUDE.md。

上手使用

这一页是日常开发中最常用的一批操作套路（prompt recipes），照搬即可，按项目实际情况替换措辞。它们都不依赖任何特殊配置，在任何 Claude Code 界面（终端、IDE 插件、Web）都适用。

快速摸清一个新代码库

刚接手一个新项目，想尽快搞清楚它的结构：

```
cd /path/to/project
claude
```

```
给我讲讲这个代码库的整体情况
```

```
这里主要用了哪些架构模式？
核心的数据模型是什么样的？
认证是怎么做的？
```

技巧：先问宽泛问题再逐步收窄；顺便让 Claude 讲讲项目里的编码约定，以及列一份项目特有术语表。如果仓库很大，给对应语言装一个代码智能插件，能让 Claude 拿到精确的“跳转到定义/查找引用”能力。

定位相关代码

需要找到和某个功能相关的代码：

```
找到处理用户认证的文件
这些认证相关文件是怎么协作的？
把登录流程从前端到数据库完整梳理一遍
```

技巧：描述要尽量具体，多用项目里实际使用的术语，而不是泛泛的说法。

高效修 bug

遇到报错，需要定位并修复：

```
运行 npm test 时报了这个错
```

```
针对 user.ts 里的 @ts-ignore 建议几种修法
```

```
按你建议的方案，给 user.ts 加上 null 检查
```

技巧：告诉 Claude 复现问题的命令，让它自己拿到堆栈信息；说明复现步骤；告诉它这个错误是偶发还是必现。

重构代码

需要把老代码换成更现代的写法：

```
找一下代码库里用了已废弃 API 的地方
针对 utils.js，给出用现代 JavaScript 特性重构的建议
在保持行为不变的前提下，把 utils.js 重构成 ES2024 写法
跑一下重构后代码对应的测试
```

技巧：让 Claude 解释新写法比旧写法好在哪；需要时明确要求保持向后兼容；重构分成小步骤、每步都可测试。

补测试

给没有测试覆盖的代码补测试：

```
找一下 NotificationsService.swift 里没有测试覆盖的函数
给通知服务补上测试
针对通知服务的边界条件补充测试用例
跑一下新测试，修复失败的部分
```

Claude 会参考项目已有测试文件的风格、框架和断言方式来生成新测试。描述清楚要验证的具体行为，效果会更好；也可以直接让 Claude 帮忙找容易漏掉的边界情况、错误路径、非预期输入。

创建 Pull Request

可以直接让 Claude 代劳（“帮我创建一个 PR”），也可以分步引导：

```
总结一下我在认证模块做的改动
create a pr
针对安全性改进，把 PR 描述写得再详细一些
```

用 `gh pr create` 创建 PR 后，当前会话会自动与该 PR 关联，之后可以用 `claude --from-pr 123` 或在 `/resume` 选择器里粘贴 PR URL 找回这个会话。提交前记得看一眼 Claude 生成的 PR 内容，让它顺便指出潜在风险。

补文档

给代码补充或更新文档：

```
找一下 auth 模块里没写 JSDoc 的函数
给 auth.js 里没文档的函数补上 JSDoc
结合更多上下文和示例，完善生成的文档
检查一下这些文档是否符合我们的项目规范
```

技巧：明确想要的文档风格（JSDoc、docstring 等）；要求带示例；优先覆盖公共 API、接口和复杂逻辑。

在非代码目录里使用

Claude Code 在任何目录都能工作，不只是代码仓库。在笔记库、文档目录、任意一堆 markdown 文件所在的目录里运行 `claude`，同样可以搜索、编辑、重新组织内容。`.claude/` 目录和 `CLAUDE.md` 可以和其它工具的配置目录并存不冲突。

处理图片

分析截图、设计稿、图表：

```
把截图拖进 Claude Code 窗口
或者复制图片后用 Ctrl+V 粘贴进 CLI(注意不是 Cmd+V)
或者直接告诉 Claude 图片路径:“分析这张图片:/path/to/image.png”
```

```
这张图展示的是什么?
描述一下这个截图里的 UI 元素
根据这个设计稿生成对应的 CSS
```

用 @ 引用文件与目录

```
解释一下 @src/utils/auth.js 里的逻辑      # 引用单个文件，带上完整内容
@src/components 的结构是怎样的?           # 引用目录，给出文件列表（不含内容）
```

`@` 文件引用会把该文件所在目录及其父目录的 `CLAUDE.md` 一并带入上下文；可以在一条消息里引用多个文件。MCP 资源同样支持，格式为 `@server:resource`。

恢复之前的会话

任务跨越多次操作时，不用每次重新讲一遍背景：

```
claude --continue
```

这会恢复当前目录下最近一次会话；如果没有历史会话，会提示 `No conversation found to continue` 并退出。想从列表里挑选，用 `claude --resume`，或者在会话内用 `/resume`。

并行开发用 Worktree

在一个终端里做一个功能，另一个终端里让 Claude 修 bug，互不干扰。每个 worktree 是同一仓库下独立分支的独立检出：

```
claude --worktree feature-auth
```

在第二个终端换个名字重复上面的命令，即可开启一个隔离的并行会话。要在一个界面里统一监控多个并行会话，用 `claude agents`（见并行与后台协作）。

先计划、再落地

想在改动落盘前先审查方案，切到计划模式。Claude 只读文件、给出计划，在你确认前不做任何修改：

```
claude --permission-mode plan
```

会话中途也可以按 `Shift+Tab` 切入计划模式。

把探索工作甩给 subagent

探索一个大代码库会把很多文件内容塞进上下文。把探索工作委托出去，只让结论回到主对话：

```
用一个 subagent 调查一下我们的认证系统是怎么处理 token 刷新的
```

subagent 会在自己独立的上下文窗口里读文件，只把总结报告返回主会话。更多定义与配置见 Agents。

把 Claude 接入脚本管道

在 CI、pre-commit hook、批处理场景下非交互地运行 Claude。stdin/stdout 和普通 Unix 工具一样使用：

```
git log --oneline -20 | claude -p “总结这些最近的提交”
```

更多输出格式、权限参数、扇出模式见启动命令行选项中的非交互模式部分。

会话与上下文管理

`/` 只有出现在消息开头才会被识别为命令，后面的文字会作为该命令的参数。输入 `/` 可以看到当前可用的全部命令，输入 `/` 加字母可以过滤。

目录与工作区

命令	作用
`/add-dir <path>`	为当前会话额外授权一个工作目录的文件读写权限。注意：这只授予文件访问，不会自动发现该目录下的 `.claude/` 配置（`.claude/skills/` 除外）
`/cd <path>`	把当前会话切换到新的工作目录，同时保留 prompt 缓存——新目录的 `CLAUDE.md` 会作为一条消息追加，而不是重建整个系统提示词（v2.1.169+）

上下文管理

命令	作用
`/compact [instructions]`	总结当前对话以释放上下文空间，可传入聚焦说明，例如 `/compact 只保留和认证模块相关的内容`
`/context [all]`	用彩色网格可视化当前上下文占用情况，给出优化建议；传 `all` 展开每一项明细
`/clear [name]`	开启一个全新对话（项目记忆仍保留），之前的对话仍可通过 `/resume` 找回；可传名称标注旧对话。别名 `/reset`、`/new`

> `/compact` 是“总结当前对话、继续同一个任务”；`/clear` 是“另起一个任务”。上下文快满时优先用前者，任务已经完成、准备做别的事情时用后者。

分支、检查点与回滚

命令	作用
`/branch [name]`	在当前节点创建一个对话分支，尝试不同方向而不丢失原对话；原对话仍可通过 `/resume` 找回
`/rewind`	把对话和/或代码回滚到之前的某个检查点，或者从某条消息开始重新总结。别名 `/checkpoint`、`/undo`

恢复与导出

命令	作用
`/resume [session]`	按 ID/名称恢复对话，或打开会话选择器；后台会话会标记 `bg`。别名 `/continue`
`/rename [name]`	重命名当前会话，显示在 prompt bar 上；不传名称则根据对话内容自动生成
`/export [filename]`	把当前对话导出为纯文本；不传文件名则弹出选择“复制到剪贴板”或“保存文件”
`/copy [N]`	复制最近一条（或倒数第 N 条）assistant 回复；含代码块时会弹出选择器

其它

命令	作用
`/btw <question>`	问一个不计入对话历史的旁支问题
`/recap`	手动生成当前会话的一句话摘要
`/exit`	退出 CLI；如果当前是 attach 到后台会话，则只是 detach，会话继续在后台跑。别名 `/quit`

典型时序

```
claude                          # 进入项目，开始一次会话
...(工作一段时间，上下文变长)
/context                        # 看看上下文都花在哪了
/compact 聚焦在登录模块的改动
...(继续工作)
/clear 完成登录模块重构         # 任务切换，给旧对话留个名字
...(第二天)
claude --resume “完成登录模块重构”   # 或交互式 /resume 选择器
```

模型、效率与权限

模型与推理

命令	作用
`/model [model]`	切换模型并保存为新会话默认值；支持左右方向键调整推理强度。不带参数打开选择器，在某一行按 `s` 表示“仅本次会话切换”
`/effort [level|auto]`	设置推理强度：`low` / `medium` / `high` / `xhigh` / `max` / `ultracode`（可用档位取决于模型）。`ultracode` 是 `xhigh` 推理 + 自动工作流编排的组合。不带参数打开滑块交互
`/fast [on|off]`	切换 fast mode（用更高单价换取更快响应速度）
`/advisor [model|off]`	启用/关闭顾问工具：在任务关键节点额外咨询第二个模型的意见，支持 `opus` / `sonnet` / `fable` 或完整模型 ID（v2.1.98+）

权限与执行模式

命令	作用
`/permissions`	管理 allow/ask/deny 规则的交互界面，可按作用域查看、增删规则、管理工作目录、查看 auto mode 最近的拒绝记录。别名 `/allowed-tools`
`/plan [description]`	直接进入计划模式（Claude 只读文件、给出方案，不做任何修改，需你确认后才真正动手）；可带任务描述，如 `/plan 修复认证 bug`
`/sandbox`	切换沙箱模式（仅部分平台支持）

会话级配置

命令	作用
`/config [key=value ...]`	打开设置界面调整主题、模型、输出风格等；也可以直接 `/config thinking=false`、`/config model=sonnet` 一步到位设置，非交互模式（`-p`）和 Remote Control 也支持这种写法。别名 `/settings`
`/goal [condition|clear]`	设定一个目标条件，Claude 会持续跨轮工作直到条件满足；不带参数查看当前/最近达成的目标，传 `clear` / `stop` / `off` 提前取消

权限模式说明

Claude Code 有一组可以互相切换的权限模式（用 `--permission-mode` 启动指定，或会话中按 `Shift+Tab` 循环切换）：

模式	行为
`default`（即 `manual`）	每个有副作用的工具调用都需要你手动确认
`acceptEdits`	自动接受文件编辑，其它操作仍需确认
`plan`	只读：Claude 探索代码并给出计划，不执行任何修改
`auto`	用内置分类器自动放行“看起来安全”的操作
`dontAsk`	减少确认提示的更宽松模式
`bypassPermissions`	跳过全部权限确认（高风险，建议只在隔离/一次性环境使用）

> ⚠ `bypassPermissions`（对应 `--dangerously-skip-permissions`）会跳过几乎所有确认步骤。在生产代码库或有写权限的 CI 环境中使用前，务必先了解清楚它*不会*跳过什么（见官方 permission modes 文档），并优先考虑限定作用域的 `--allowedTools` 规则。

示例

```
/model sonnet           # 切到 sonnet 系列
/effort high             # 提高本次任务的推理强度
/plan 重构订单模块的状态机
/permissions             # 打开权限规则管理
```

并行与后台协作

Claude Code 支持把工作“甩到后台”或“拆成多份并行执行”，这一页汇总相关命令。更完整的场景化讲解见上手使用 · 并行开发。

后台会话

命令	作用
`/background [prompt]`	把当前会话转入后台运行并释放当前终端；可先补一条指令再转后台。别名 `/bg`。用 `claude agents` 监控
`/agents`	提示你去让 Claude 创建/管理 subagent，或直接编辑 `.claude/agents/`（旧版本 `/agents` 是交互式管理界面，已在 v2.1.198 移除）
`/tasks`	查看并管理当前会话后台运行的一切（命令、subagent 等）。别名 `/bashes`
`/stop`	停止当前 attach 的后台会话（仅在 attach 状态下可用）；会话记录与 worktree 都会保留
`/fork <directive>`	🆕 v2.1.212 起语义变更：把当前对话复制成一个独立的后台会话（可单独继续跑），不再是会话内 subagent（更早版本 `/fork` 曾是 `/branch` 的别名）
`/subtask <directive>`	🆕 v2.1.212 新增：派生一个后台 subagent，继承完整对话上下文，你继续手头工作、等它做完把结果带回来（即 v2.1.212 之前 `/fork` 的行为）

大规模并行：/batch

命令	作用
`/batch <task>`	技能。先研究代码库，把大改动拆成 5~30 个独立单元并给出计划；确认后为每个单元在独立 git worktree 里派生一个后台 subagent，各自实现、跑测试、开 PR。需要 git 仓库

```
/batch 把 src/ 下的组件从 Vue 迁移到 React
```

工作流编排

命令	作用
`/workflows`	打开工作流进度视图，查看/暂停/恢复/保存正在运行或已完成的动态工作流

动态工作流（dynamic workflow）是 Claude 根据任务自动生成、跨数十到数百个 subagent 并行编排的后台执行方案，适合体量远超单会话能处理的大型任务（详见 Agents 与版本 Changelog）。

远程与跨端

命令	作用
`/remote-control`	让当前会话可以被 claude.ai / Claude App 远程控制。别名 `/rc`
`/teleport`	把一个 Claude Code on the web 会话拉到本地终端继续（拉取分支 + 对话）。别名 `/tp`
`/desktop`	把当前会话转到 Claude Code Desktop 客户端继续（需 macOS/Windows + 订阅账号）。别名 `/app`
`/remote-env`	选择云端 agent 的默认运行环境
`/web-setup`	用本地 `gh` CLI 凭证把 GitHub 账号接入 Claude Code on the web

定时与循环

命令	作用
`/loop [interval] [prompt]`	技能。会话保持打开的前提下，按固定或自适应间隔重复执行一个 prompt，例如 `/loop 5m 检查部署是否完成`。别名 `/proactive`

`/loop` 只在当前会话存活期间有效，关闭会话就停止。需要“电脑关机也能跑”的定时任务，应使用 Routines（云端托管）或 GitHub Actions，而不是 `/loop`——具体取舍见官方 Common workflows 文档的调度选项对比表。

与 subagent 的关系

日常对话里，Claude 也会自主派生 subagent 去做代码探索、并行实现等工作（不需要你手动敲上面任何命令）。Subagent 的定义、`.claude/agents/` 与监控方式见 Agents。手动命令的意义在于：

- `/fork`、`/batch`：你明确知道要并行拆分，主动触发。
- `/background`：你想腾出终端去做别的事，让当前任务继续跑。
- `/tasks`、`/agents`、`claude agents`：你想看清楚“现在到底有哪些东西在后台跑”。

代码评审与 Git 工作流

查看变更

命令	作用
`/diff`	打开交互式 diff 查看器，展示未提交的改动以及每一轮对话产生的 diff；左右方向键切换“当前 git diff”与“某一轮的改动”，上下方向键切换文件。v2.1.198 起，仓库状态在会话外变化（比如切分支、别的终端提交）时视图会自动刷新

代码评审

命令	作用
`/code-review [low|medium|high|xhigh|max|ultra] [--fix] [--comment] [target]`	技能。评审当前 diff 的正确性 bug，以及可复用性、简化、效率方面的清理建议。`--fix` 直接应用到工作区，`--comment` 以行内评论发到 GitHub PR，`ultra` 触发云端深度多 agent 评审
`/simplify [target]`	技能。只做清理不找 bug：四个评审 agent 并行分析代码复用、简化、效率、抽象层级是否合适，并直接应用修复（v2.1.154+；更早版本等价于 `/code-review --fix`）
`/review [PR]`	只读地评审一个 GitHub PR；不传参数则列出可选的 open PR
`/security-review`	分析当前分支的 pending 改动，识别注入、鉴权问题、数据泄露等安全风险
`/ultrareview [PR]`	在云端沙箱里跑一次深度多 agent 评审；推荐改用 `/code-review ultra`，`/ultrareview` 作为别名保留

PR 与协作流程

命令	作用
`/install-github-app`	为仓库安装 Claude GitHub App，可选顺带配置 GitHub Actions workflow 与 secrets
`/autofix-pr [prompt]`	派生一个 Claude Code on the web 会话，持续盯着当前分支的 PR:CI 失败或有人留评论时自动推送修复。默认修复全部 CI 失败与评论，可传自定义指令覆盖默认行为
`/ultraplan <task>`	在 ultraplan 会话里起草一份计划，在浏览器里评审，然后选择远程执行或发回本地终端

典型流程

```
# 改完代码后
/diff                          # 先自己看一眼改了什么
/code-review --fix             # 让 Claude 找 bug 并直接修
/security-review               # 过一遍安全检查
# 满意后直接让 Claude 创建 PR
"create a pr"
```

用 `gh pr create` 创建 PR 后，该会话会自动与这个 PR 关联，之后可以用 `claude --from-pr 123` 或在 `/resume` 选择器里搜索 PR URL 找回这个会话。

自定义与扩展

项目记忆：CLAUDE.md

命令	作用
`/init`	探索项目并生成一份 `CLAUDE.md`。设置 `CLAUDE_CODE_NEW_INIT=1` 后会先交互式询问要生成哪些文件（CLAUDE.md、skills、hooks），再动手
`/memory`	编辑 `CLAUDE.md` 记忆文件，开关自动记忆（auto-memory），查看自动记忆已经记了什么

一个新仓库的标准起手式是 `/init` 生成初版 `CLAUDE.md`，再用 `/memory` 逐步精修——把项目约定、常用命令、代码风格写成 Claude 每次都会看到的“常识”，而不是每次对话重复口述。`CLAUDE.md` 的存放位置、加载顺序、`@` 导入语法等文件层面的细节见配置文件。

技能（Skills）

技能是“你反复复制粘贴的一段指令/清单/多步骤流程”的固化产物：写一个 `SKILL.md`，Claude 就能在合适时机自动使用，或者你用 `/skill-name` 直接调用。旧版的 `.claude/commands/*.md` 自定义命令已经并入技能体系，写法兼容，行为等价。完整说明见 Skills 与 Commands。

MCP(Model Context Protocol)

命令	作用
`/mcp [reconnect <server>|enable|disable [<server>|all]]`	管理 MCP 服务器连接与 OAuth 登录状态；不带参数打开交互列表

命令行侧对应 `claude mcp`（增删配置）、`claude mcp login/logout <server>`（单独走 OAuth，v2.1.186+）。MCP 服务器暴露的 prompt 会以 `/mcp__<server>__<prompt>` 的形式出现在命令列表里。

插件

命令	作用
`/plugin [subcommand]`	管理插件；不带参数打开插件菜单，或直接传 `list` / `install` / `enable` / `disable`
`/reload-plugins [--force]`	重新加载全部已启用插件，应用未生效的变更；如果会改变已加载的 MCP 工具集（从而让 prompt 缓存失效），需要加 `--force` 才会真正执行
`/reload-skills`	重新扫描技能/命令目录，让磁盘上新增或修改的技能无需重启即可生效（v2.1.152+）

插件可以打包技能、subagent、hooks、MCP 服务器、输出风格，统一分发给团队；完整的组成、目录结构、Marketplace 与作者侧打包见 Plugins。安装示例：

```
/plugin install skill-creator@claude-plugins-official
```

Hooks

命令	作用
`/hooks`	查看当前生效的 hook 配置（按工具事件分类）

Hooks 用于在特定生命周期事件（如 `SessionStart`、`PreToolUse`、`Stop`）上运行确定性的脚本逻辑，弥补“技能只能靠模型自觉遵守指令”的不足，适合强制校验、审计、自动格式化等场景。

界面与体验

命令	作用
`/theme`	切换配色主题，含跟随终端明暗的 `auto`、色盲友好主题、ANSI 主题，以及 `~/.claude/themes/` 自定义主题
`/statusline`	配置状态栏；描述你想要的效果，或不带参数从当前 shell prompt 自动生成
`/keybindings`	打开快捷键配置文件
`/color [color|default]`	设置当前会话 prompt bar 颜色，连接 Remote Control 时会同步到 claude.ai/code
`/focus`	切换聚焦视图：只显示最后一条 prompt、一行工具调用摘要与最终回复
`/tui [default|fullscreen]`	设置终端 UI 渲染模式并重新进入（`fullscreen` 为无闪烁 alt-screen 渲染器）

项目专属能力

命令	作用
`/run`	技能。启动并驱动你的应用，用真实运行效果而非只靠测试来验证改动（v2.1.145+）
`/verify`	技能。构建并运行项目，用观察到的实际行为确认一次代码改动是否达到预期（v2.1.145+）
`/run-skill-generator`	技能。从一次干净环境的启动过程中，记录安装命令、环境变量、启动脚本，固化成项目专属技能 `.claude/skills/run-<project>/`，供 `/run`、`/verify` 及其它 agent 复用（每个项目建议跑一次）
`/dataviz [request]`	技能。图表/仪表盘设计指导：自动挑选图表形式、按角色分配颜色、校验色盲友好度与对比度（v2.1.198+）
`/claude-api [migrate|managed-agents-onboard]`	技能。加载 Claude API 参考资料（按项目语言自动匹配），或帮你把已有 Claude API 代码升级到新模型
`/design-sync [hint]`	技能。把仓库里的 React 设计系统同步上传到 Claude Design，让生成的设计使用你的真实组件（仅 Anthropic API 可用）
`/design-login`	为 `/design-sync` 授权 claude.ai 账号访问设计系统

建议的自定义投入顺序

1. 先写好 `CLAUDE.md`（`/init` + `/memory`），这是成本最低、收益最直接的一步。
2. 把“经常口述的多步骤指令”沉淀成个人/项目技能。
3. 需要跨项目共享或团队统一分发时，再打包成插件。
4. 需要“确定性强制执行”（而不是靠模型自觉）时，才引入 hooks。

账户、诊断与其它命令

账户与登录

命令	作用
`/login`	登录 Anthropic 账号
`/logout`	登出
`/status`	打开设置界面的 Status 标签页，显示版本、模型、账号、连接状态（Claude 响应过程中也能打开）
`/upgrade`	打开升级页面切换到更高的订阅套餐
`/privacy-settings`	查看/修改隐私设置（仅 Pro/Max 订阅用户可见）
`/passes`	分享一周免费试用给朋友（仅符合条件的账号可见）

用量与费用

命令	作用
`/usage`	查看会话花费、套餐用量上限、活跃度统计；Pro/Max/Team/Enterprise 套餐下还能按技能、subagent、插件、MCP 服务器细分。别名 `/cost`、`/stats`
`/usage-credits`	配置用量额度，在触达套餐上限后仍可继续工作（原 `/extra-usage`）

诊断与反馈

命令	作用
`/doctor`	诊断并校验安装与配置状态，结果附带状态图标；按 `f` 可以让 Claude 直接修复报告出的问题
`/debug [description]`	技能。为当前会话开启调试日志并分析问题；调试日志默认关闭（除非以 `claude --debug` 启动），中途运行 `/debug` 只会从此刻开始记录
`/feedback [report]`	提交反馈、报 bug 或分享会话。别名 `/bug`、`/share`
`/insights`	生成一份分析报告：你的会话都花在哪些项目区域、交互模式与卡点
`/team-onboarding`	基于过去 30 天的会话、命令、MCP 使用情况，生成一份团队 onboarding 指南
`/release-notes`	在交互式版本选择器里查看 changelog，可选查看某个具体版本或全部版本
`/heapdump`	生成 JS 堆快照与内存分析，写到桌面（或 Linux 无桌面环境时的主目录），用于排查高内存占用

IDE 与浏览器集成

命令	作用
`/ide`	管理 IDE 集成并查看连接状态
`/chrome`	配置 Claude in Chrome 浏览器自动化设置
`/terminal-setup`	配置 Shift+Enter 等终端快捷键（仅在需要的终端，如 VS Code、Cursor、Alacritty、Zed 中出现）

语音与移动端

命令	作用
`/voice [hold|tap|off]`	开关语音听写，或指定特定模式（需要 claude.ai 账号）
`/mobile`	显示下载 Claude 移动端 App 的二维码。别名 `/ios`、`/android`
`/install-slack-app`	安装 Claude Slack App，走浏览器完成 OAuth

第三方 API 提供方配置

命令	作用
`/setup-bedrock`	交互式向导配置 Amazon Bedrock 的鉴权、区域、模型钉版（需先设置 `CLAUDE_CODE_USE_BEDROCK=1`）
`/setup-vertex`	交互式向导配置 Google Cloud's Agent Platform 的鉴权、项目、区域、模型钉版（需先设置 `CLAUDE_CODE_USE_VERTEX=1`）

杂项

命令	作用
`/help`	显示帮助与可用命令列表
`/scroll-speed`	交互式调整鼠标滚轮速度（仅全屏渲染模式）
`/radio`	在浏览器打开 Claude FM lo-fi 电台
`/stickers`	订购 Claude Code 周边贴纸

> 并非所有命令对所有用户都可见，取决于平台、套餐与运行环境——例如 `/desktop` 只在 macOS/Windows 且已登录订阅账号时出现，`/upgrade` 只在 Pro/Max 套餐下出现。命令列表以你本机 `/help` 或 `/` 菜单的实际展示为准。

Commands

Commands（自定义命令） 是你用 Markdown 写的一段固定 prompt，在会话里以 `/命令名` 调用。v2.1 起，旧版 `.claude/commands/*.md` 已并入 Skills 体系——文件格式兼容、行为等价，新内容应写在 `.claude/skills/<name>/SKILL.md`。本篇说明两种写法的差异、存放位置与日常用法。

> 技能（Skills）的完整机制见 Skills。本篇聚焦「命令」视角：如何命名、调用、传参，以及从旧目录迁移。

两种形态，一套机制

形态	路径	状态
旧版自定义命令	`.claude/commands/<name>.md` 或 `~/.claude/commands/<name>.md`	仍兼容，建议迁移到 skills
技能（推荐）	`.claude/skills/<name>/SKILL.md` 或 `~/.claude/skills/<name>/SKILL.md`	当前标准写法

两者都会出现在 `/` 补全菜单里；Claude 也可根据 `description` 自动判断是否调用（除非设置了 `disable-model-invocation`）。

最小自定义命令（旧版路径）

在项目根创建 `.claude/commands/commit.md`：

```
---
description: 按 Conventional Commits 规范提交当前改动
---

1. 运行 `git status` 与 `git diff` 查看变更
2. 起草简洁的 commit message（英文，focus on why）
3. 执行 `git add` 与 `git commit`
4. 不要 push，除非用户明确要求
```

保存后执行 `/reload-skills`（或重启会话），即可在终端输入 `/commit` 调用。

推荐：写成 SKILL.md

同一逻辑迁移到 `.claude/skills/commit/SKILL.md`：

```
---
name: commit
description: 按 Conventional Commits 规范提交当前改动
disable-model-invocation: true
allowed-tools: Bash(git status) Bash(git diff) Bash(git add *) Bash(git commit *)
---

1. 运行 `git status` 与 `git diff` 查看变更
2. 起草简洁的 commit message（英文，focus on why）
3. 执行 `git add` 与 `git commit`
4. 不要 push，除非用户明确要求
```

相比纯 `.md` 命令文件，SKILL 额外支持：

能力	作用
`allowed-tools`	声明工具白名单，减少反复点确认
`disable-model-invocation`	仅手动 `/commit` 触发，避免 Claude 误提交
`arguments` / `$ARGUMENTS`	支持 `/commit fix: login bug` 传参
`context: fork` + `agent`	在独立 Agent 里执行

存放位置与作用域

位置	影响范围	是否进 git
`~/.claude/commands/` 或 `~/.claude/skills/`	你所有项目	否
`.claude/commands/` 或 `.claude/skills/`	当前仓库协作者	是（推荐团队共享）
插件内置	安装插件的用户	由插件分发

项目级命令应提交进仓库，让团队共用同一套 `/review`、`/deploy` 等工作流。

传参

在 SKILL frontmatter 声明参数：

```
---
name: explain
description: 用通俗语言解释指定文件或概念
argument-hint: [文件路径或概念名]
---

请用中文向初级开发者解释：$ARGUMENTS
```

调用：`/explain src/auth/token.ts`

旧版 `.claude/commands/*.md` 同样支持 `$ARGUMENTS` 与位置参数 `$0`、`$1`。

堆叠调用

v2.1.199+ 支持在一次输入里串联多个技能，例如：

```
/deploy-staging /run-tests 确认通过后部署
```

最多可一次性加载 5 个前置技能的上下文（详见 Skills · 堆叠调用）。

相关斜杠命令

命令	作用
`/skills`	列出全部技能/命令，按名称过滤，按 `t` 按 token 排序，按 `Space` 切换可见性
`/reload-skills`	热重载磁盘上的技能与命令目录（v2.1.152+），无需重启会话
`/disable-slash-commands`	本次会话禁用全部 skills 与命令（排查问题时用）

命令行等价：`claude --disable-slash-commands`。

与内置命令的区别

类型	例子	谁维护
内置斜杠命令	`/clear`、`/model`、`/memory`	Anthropic 随 Claude Code 发布
随附技能（bundled skills）	`/batch`、`/code-review`	内置，可用 `disableBundledSkills` 关闭
自定义命令 / 项目技能	`/commit`、`/deploy`	你或团队写在 `.claude/`
插件技能	`skill-creator:...`	第三方插件

自定义命令不会覆盖内置命令名；重名时以命名空间或加载顺序为准，建议避免与内置 `/init`、`/help` 等冲突。

常见误区

1. 「改了文件却没出现在菜单里」——执行 `/reload-skills`，或检查 frontmatter 是否缺少 `description`（影响自动调用，不影响手动 `/name`）。
2. 「命令能执行危险操作」——对 `/deploy`、`/rm` 类命令务必加 `disable-model-invocation: true`，并用 `permissions.deny` 限制 Bash。
3. 「`.claude/commands` 和 `skills` 二选一」——可共存，但同一逻辑只维护一份，优先 skills。
4. 「`--add-dir` 会自动加载额外目录的命令」——只额外发现 `.claude/skills/`，不会加载其它目录下的完整 `.claude/` 配置（见 会话管理命令）。

建议沉淀顺序

1. 把每周重复口述的多步骤流程写成项目技能（如 `/pr-checklist`）。
2. 有副作用的操作加 `disable-model-invocation` + `allowed-tools`。
3. 需要跨项目复用时，放到 `~/.claude/skills/`。
4. 需要团队统一分发时，打包成插件。

延伸阅读

- Skills——frontmatter 全表、`context: fork`、插件技能命名空间

- Agents——让技能在独立 subagent 中执行
- Plugins——把命令/技能打包成可分发插件
- 自定义与扩展——MCP、Hooks、插件一览

← 上一篇：账户、诊断与其它命令 · 下一篇：Skills

Skills

Skills（技能） 是 Claude Code 里复用 prompt 的一等公民：把反复使用的检查清单、评审流程、部署步骤写成 `SKILL.md`，Claude 可在合适时机自动调用，你也可以用 `/skill-name` 手动触发。旧版的 `.claude/commands/*.md` 自定义命令已并入同一体系（见 Commands）。

核心概念

```
你写 SKILL.md
    ↓
Claude Code 扫描 ~/.claude/skills/ 与 .claude/skills/
    ↓
出现在 / 菜单 + 可被 Claude 根据 description 自动选用
    ↓
激活时把正文注入上下文（可带 allowed-tools 白名单）
```

技能不是独立进程；除非设置 `context: fork`，否则与主会话共享同一上下文窗口。

文件结构

```
.claude/skills/
└── deploy/
    └── SKILL.md
```

或单文件技能（较少用）：

```
.claude/skills/quick-lint/SKILL.md
```

个人全局技能：`~/.claude/skills/<name>/SKILL.md`。

SKILL.md 完整示例

```
---
name: deploy
description: 将应用部署到 staging 或 production；用户提到发布、上线、deploy 时使用
disable-model-invocation: true
allowed-tools: Bash(git status) Bash(git diff) Bash(npm run build) Bash(npm run te
argument-hint: [staging|production]
---

将 $ARGUMENTS 环境部署到目标：

1. 确认当前分支干净，`git status` 无未提交改动
2. 运行 `npm run test` 与 `npm run build`
3. 按项目 README 中的部署脚本执行（先读 README 再动手）
4. 部署后做最小冒烟验证并汇报 URL
```

Frontmatter 字段速查

字段	作用
`name`	技能 ID，对应 `/name` 调用名
`description`	强烈建议填写——Claude 判断是否自动使用该技能的主要依据
`disable-model-invocation: true`	仅用户手动 `/name` 可触发（适合 deploy、commit 等有副作用流程）
`user-invocable: false`	不出现在 `/` 菜单，仅 Claude 可自动触发（背景知识类）
`allowed-tools`	技能激活期间免确认可用的工具列表
`context: fork`	在独立 subagent 中执行技能正文（见 Agents）
`agent`	与 `context: fork` 联用，指定 subagent 类型
`argument-hint`	`/` 补全时提示参数格式
`arguments`	声明结构化参数（与 `$0`、`$1`、`$ARGUMENTS` 配合）

存放位置与优先级

位置	作用域	典型用途
`~/.claude/skills/`	个人，跨项目	你的 commit 模板、个人评审习惯
`.claude/skills/`	项目，可提交 git	团队统一的 `/verify`、`/run`
插件携带	安装插件的用户	`插件名:技能名` 命名空间

插件技能示例：`/plugin install skill-creator@claude-plugins-official`。

管理与调试

命令 / 操作	作用
`/skills`	交互列表：过滤、按 token 排序、切换对 Claude/菜单的可见性
`/reload-skills`	热重载磁盘变更（v2.1.152+）
`/usage`	按技能、subagent、插件细分用量（套餐用户）
`SLASH_COMMAND_TOOL_CHAR_BUDGET`	技能很多时调大，避免 description 被裁剪（见环境变量）

排查「技能不出现」：

1. 路径是否为 `.claude/skills/<dir>/SKILL.md`
2. YAML frontmatter 是否合法
3. 执行 `/reload-skills`
4. 是否被 `--bare`、`--safe-mode` 或 `disableBundledSkills` 禁用

自动调用 vs 手动调用

模式	配置	适用
双向	默认（有 `description`）	评审清单、测试流程——Claude 见机行事，你也可 `/name`
仅手动	`disable-model-invocation: true`	部署、删库、发 PR——避免误触发
仅自动	`user-invocable: false`	编码规范、领域术语表——当背景知识注入

堆叠调用

v2.1.199+ 支持一条输入串联多个技能，最多 5 个前置技能上下文一并加载：

```
/code-review --fix /simplify 对 src/api/ 做一轮清理
```

适合「先评审再简化」等固定组合；注意 token 消耗会叠加。

在独立 Agent 中执行（context: fork）

需要隔离上下文、避免污染主会话时：

```yaml
---
name: deep-research
description: 对指定模块做深度代码调研并只返回摘要
context: fork
agent: Explore
---

调研范围：$ARGUMENTS

要求：只返回结构化摘要（文件列表、调用链、风险点），不要把全文贴进主会话。
```

`agent: Explore` 使用内置的 Haiku 探索型 subagent，省主上下文 token。自定义 agent 见 Agents。

内置技能（Bundled Skills）精选

随 Claude Code 附带、无需自建文件即可 `/` 调用，例如：

技能	用途
`/batch`	大任务拆成多 worktree 并行 subagent
`/code-review`	多维度代码评审，可 `--fix`
`/run`、`/verify`	实际跑应用验证改动（v2.1.145+）
`/run-skill-generator`	从启动过程生成项目专属 `/run` 技能
`/fewer-permission-prompts`	扫描历史会话生成 Bash 白名单

可用 `disableBundledSkills` 或环境变量 `CLAUDE_CODE_DISABLE_BUNDLED_SKILLS` 关闭（不影响自定义技能）。

与 CLAUDE.md、插件的关系

1. CLAUDE.md——常驻项目记忆；技能是按需加载的流程模板。先写好 CLAUDE.md（`/init`），再把重复流程沉淀为技能。
2. 插件——可打包多个技能 + agents + hooks + MCP，适合团队分发。见 Plugins。
3. Hooks——确定性脚本拦截（如 PreToolUse）；技能靠模型遵守指令。副作用强制的步骤用 hooks 兜底。

常见误区

1. 不写 `description`——技能只能手动 `/name`，Claude 几乎不会自动选用。
2. 把所有 CLAUDE.md 内容塞进一个巨大技能——应拆分：常识放 CLAUDE.md，流程放技能。
3. 忘记 `allowed-tools`——技能里反复 `Bash` 会点到手软；把只读命令写进白名单。
4. 以为 `/reload-skills` 会重载插件——插件变更用 `/reload-plugins [--force]`。

延伸阅读

- Commands——`.claude/commands` 兼容写法与迁移
- Agents——subagent 文件格式与后台并行
- Plugins——把技能打包成可分发插件
- 配置文件——`settings.json` 中的 `disableBundledSkills`

← 上一篇：Commands · 下一篇：Agents

Agents

Agents（代理） 在 Claude Code 里主要指 subagent（子代理）：拥有独立上下文窗口、可并行运行的 Claude 实例。主会话把子任务派给 subagent，只收回摘要结果，从而节省主上下文、提高探索与并行开发的效率。v2.1.198 起，subagent 默认在后台运行，主会话可继续其它工作。

> 与「Claude Agent SDK / 第三方 Agent 平台」不同，本篇讲的是 Claude Code 内置的 subagent 机制（`.claude/agents/`、Task 工具、`claude agents` 命令）。

三种使用方式

方式	谁触发	典型场景
自动派生	Claude 自主决定	代码库探索、并行读多文件
技能 fork	`/skill` + `context: fork`	隔离执行某段 SKILL 正文
手动命令	`/fork`、`/batch`、`/background`	你明确要并行或腾终端

日常开发多数时候不用手动管——Claude 会在需要时自己开 subagent；你要做的是在复杂任务时知道如何监控和显式编排。

自定义 Subagent：.claude/agents/

与技能类似，subagent 用 Markdown + frontmatter 定义，放在：

位置	作用域
`~/.claude/agents/<name>.md`	个人，跨项目
`.claude/agents/<name>.md`	项目，可提交 git

示例 `.claude/agents/security-reviewer.md`：

```
---
name: security-reviewer
description: 专注 OWASP Top 10、密钥泄露、依赖 CVE 的安全评审
tools: Read, Grep, Glob, Bash
model: sonnet
---

你是一名安全评审专家。收到代码路径或 diff 描述后：

1. 列出潜在漏洞与误报可能
2. 标注严重级别（critical / high / medium / low）
3. 给出可操作的修复建议，不要直接改代码除非用户要求
```

创建或修改后，让 Claude「创建一个 security-reviewer agent」或直接编辑文件即可；旧版交互式 `/agents` 管理界面已在 v2.1.198 移除，改为直接编辑 `.claude/agents/` 或在对话中描述需求由 Claude 生成。

内置 Subagent

Agent	模型倾向	用途
Explore	Haiku（轻量）	快速搜索代码库、查引用、省主会话 token
Plan 等	随版本迭代	任务规划、调研（以你本机 `/agents` 提示为准）

在对话中说「用 Explore subagent 查一下认证模块」即可触发；Explore 自 v2.0.17 起内置。

手动编排命令

命令 / CLI	作用
`/fork <directive>`	🆕 v2.1.212 起语义变更：把当前对话复制成一个独立的后台会话（可单独继续跑），而不再是会话内 subagent
`/subtask <directive>`	🆕 v2.1.212 新增：派生后台 subagent，继承完整对话上下文，做完把结果带回（即 v2.1.212 之前 `/fork` 的行为）
`/batch <task>`	技能：拆成 5–30 单元，每单元独立 git worktree + 后台 subagent + PR
`/background [prompt]`	当前会话转后台，释放终端；别名 `/bg`
`/tasks`	查看本会话所有后台任务（命令、subagent 等）；别名 `/bashes`
`/workflows`	查看/暂停/恢复动态工作流进度
`claude agents`	终端 UI：监控与派发并行后台会话
`claude --agent <name>`	指定本次会话使用的 subagent
`claude --bg`	启动后立即转后台

详见 并行与后台协作。

动态工作流（Dynamic Workflow）

v2.1.154 引入：Claude 把超大任务自动编排成数十到数百个 subagent 的后台执行图，适合：

- 全仓库迁移（框架升级、语言改写）
- 大规模文档生成
- 跨模块一致性重构

用 `/workflows` 查看进度；`/config` 里可调整「动态工作流规模」建议值（非硬限制）。关闭：`disableWorkflows` in settings 或 `CLAUDE_CODE_DISABLE_BUNDLED_SKILLS` 相关开关。

后台 Agent 与 claude agents

v2.1.198 起 subagent 默认后台化后的典型工作流：

```
# 终端 1：主会话继续交互
claude

# 终端 2：统一监控
claude agents
```

`claude agents` 可查看多个并行会话状态、attach 到后台任务。禁用后台能力：`CLAUDE_CODE_DISABLE_AGENT_VIEW=1`（见环境变量）。

并行开发还可为每个任务开独立 worktree（`claude --worktree feature-a`），在上手使用 · 并行开发有逐步说明。

技能 + Agent：context: fork

在 Skills 里设置：

```
context: fork
agent: Explore
```

技能正文会在指定 subagent 内执行，结果摘要回到主会话——适合深度调研而不撑爆主上下文。

Task 工具与任务追踪

v2.1.142 起默认用结构化 Task API 替代旧版 `TodoWrite`：

- `TaskCreate` / `TaskUpdate` / `TaskGet` / `TaskList`

Claude 在复杂多步任务中用来维护任务板；与 subagent 并行互补——Task 管「计划」，subagent 管「谁去干活」。

环境变量精选

变量	作用
`CLAUDE_CODE_SUBAGENT_MODEL`	subagent 使用的模型（可与主会话不同）
`CLAUDE_CODE_MAX_TOOL_USE_CONCURRENCY`	只读工具与 subagent 最大并行数（默认 10）
`CLAUDE_ASYNC_AGENT_STALL_TIMEOUT_MS`	后台 subagent 无进展超时（默认 10 分钟）
`TASK_MAX_OUTPUT_LENGTH`	subagent 输出截断前最大字符（默认 32000）
`CLAUDE_CODE_DISABLE_AGENT_VIEW`	关闭 `claude agents` / `/background` 等

与插件、MCP 的关系

插件可打包 agents + skills + hooks + MCP，一次安装全团队对齐编排能力：

```
/plugin install my-team-pack@company-marketplace
```

MCP 服务器暴露的工具对 subagent 同样可用（受 manifest 权限约束）。

常见误区

1. 「每个 subagent 有无限上下文」——仍有模型窗口上限；Explore 用 Haiku 是为了又快又省，不是魔法。
2. 「后台 = 电脑关机也能跑」——本地后台会话依赖本机进程；长期无人值守用 Claude Code on the web、Routines 或 CI。
3. 「/agents 打不开管理界面」——v2.1.198 后改为编辑 `.claude/agents/` 或口述让 Claude 创建。
4. 「subagent 输出总是完整文件内容」——受 `TASK_MAX_OUTPUT_LENGTH` 截断；应要求 subagent 只返回摘要。

建议实践

1. 探索型问题交给 Explore，主会话只收结论。
2. 大重构用 `/batch` 或动态工作流，不要在一个会话里硬扛。

3. 团队专用评审/安全/agent 放进 `.claude/agents/` 并提交 git。
4. 用 `claude agents` 或 `/tasks` 养成「先看后台在跑什么」的习惯。

延伸阅读

- 并行与后台协作——命令速查表
- Skills——`context: fork` 与 bundled skills
- Plugins——把 agents 打包分发给团队
- 配置文件——agents 路径与作用域
- 版本 Changelog——subagent 与动态工作流演进时间线

← 上一篇：Skills · 下一篇：Plugins

Plugins

Plugins（插件） 是 Claude Code 的打包与分发层：把 Commands、Skills、Agents，连同 hooks、MCP 服务器、LSP 服务器等，装进一个自包含目录，做成可版本化、可一键安装、可跨项目与团队共享的单元。前三篇讲「怎么写单个能力」，这一篇讲「怎么把它们打成一个包发出去、别人怎么装」。

> 单独放在 `.claude/` 下的技能/命令/agent 只在当前项目可用、名字是 `/hello`；打成插件后可通过 Marketplace 分发、带版本、跨项目复用，代价是技能名带命名空间前缀 `/插件名:hello`。先用 `.claude/` 快速试，成型了再打包成插件。

独立配置 vs 插件

维度	独立配置（`.claude/`）	插件（**Plugin**）
技能名	`/hello`	`/插件名:hello`（带命名空间，避免多插件重名冲突）
作用域	单个项目或个人	可跨项目、跨团队分发
分发	手动复制	`/plugin install`，走 Marketplace
版本	无	有版本号或 git commit SHA
适合	个人工作流、项目定制、快速试验	团队统一、社区发布、versioned 发布

插件能打包什么

一个插件目录里，除清单外的组件都放在插件根目录（不是 `.claude-plugin/` 里）：

目录 / 文件	组件
`.claude-plugin/plugin.json`	插件清单（名字、描述、版本）
`skills/<name>/SKILL.md`	技能（推荐新写法）
`commands/*.md`	旧版扁平命令文件（兼容，新插件用 `skills/`）
`agents/`	自定义 subagent 定义
`hooks/hooks.json`	生命周期 hooks
`.mcp.json`	MCP 服务器配置
`.lsp.json`	LSP 语言服务器（代码智能）
`monitors/monitors.json`	后台监视器（watch 日志/文件，有事件就通知 Claude）
`bin/`	可执行文件，启用插件期间加入 Bash 工具的 `PATH`
`settings.json`	启用插件时应用的默认设置（目前仅 `agent`、`subagentStatusLine` 两个键生效）

⚠ 最常见的错误：把 `commands/`、`agents/`、`skills/`、`hooks/` 放进了 `.claude-plugin/` 目录里。`.claude-plugin/` 里只放 `plugin.json`，其余全部放在插件根目录。插件根是插件自己的目录（你 `--plugin-dir` 指向的那个），永远不是 `~/.claude/`。

安装与使用（用户侧）

`/plugin` 打开插件管理面板，四个 tab 用 `Tab` 循环切换：

- Discover —— 浏览已添加 Marketplace 里的插件
- Installed —— 查看/启用/禁用/卸载已装插件
- Marketplaces —— 增删、刷新 Marketplace
- Errors —— 插件加载错误

官方 Marketplace `claude-plugins-official` 开箱即用，直接装：

```
/plugin install github@claude-plugins-official
```

常用命令（交互式 `/plugin ...` 与非交互式 `claude plugin ...` 一一对应，后者别名 `claude plugins`，适合脚本/CI）：

命令	作用
`/plugin install <名>@<市场>`	安装某插件
`/plugin list [--enabled|--disabled]`	列出已装插件
`/plugin enable` / `disable <名>@<市场>`	启用/禁用（不卸载）
`/plugin uninstall <名>@<市场>`	彻底卸载
`/reload-plugins [--force]`	不重启会话即应用插件变更；若会改动已加载的 MCP 工具集（使 prompt 缓存失效），需 `--force`

> ⚠ 插件会以你的用户权限执行任意代码（hooks、MCP 服务器、`bin/` 脚本都是）。只安装你信任来源的插件、只添加你信任的 Marketplace。Anthropic 不审核第三方插件里打包了什么。企业可用托管设置的 `strictKnownMarketplaces` 限制可添加的来源。

Marketplace（插件市场）

Marketplace 是一份插件目录，本质是一个含 `.claude-plugin/marketplace.json` 的 git 仓库（或本地路径、远程 URL）。用它分两步：先 `add` 注册目录，再 `install` 挑装。

```
# 从 GitHub owner/repo 添加(可 @ref 固定分支/tag);别名 /plugin market
/plugin marketplace add anthropics/claude-code
# 从任意 git host(带 https:// 与 .git),或本地路径
/plugin marketplace add https://gitlab.com/company/plugins.git
/plugin marketplace add ./my-marketplace
# 刷新 / 移除(别名 rm)
/plugin marketplace update [名字]
/plugin marketplace remove <名字>
```

Anthropic 维护三个公共 Marketplace：

Marketplace	添加方式	内容
`claude-plugins-official`	自动可用	官方精选：代码智能 LSP（`typescript-lsp`、`pyright-lsp`、`gopls-lsp`、`rust-analyzer-lsp` 等）、外部集成 MCP（`github`、`gitlab`、`linear`、`notion`、`figma`、`slack`、`sentry` 等）、`security-guidance`、`commit-commands`、`pr-review-toolkit`、`plugin-dev`
`claude-community`	`/plugin marketplace add anthropics/claude-plugins-community`	通过审核的第三方插件，装为 `<名>@claude-community`
`claude-code-plugins`（示例）	`/plugin marketplace add anthropics/claude-code`	官方演示插件，展示能做什么

> “`claude-plugins-official`、`claude-community` 等名字为官方保留，第三方 Marketplace 不能占用。”

写一个插件（作者侧）

最小插件 = 一个清单 + 一个技能。以下从零建一个 `my-first-plugin`：

```
mkdir -p my-first-plugin/.claude-plugin my-first-plugin/skills/hello
```

清单 `my-first-plugin/.claude-plugin/plugin.json`：

```
{
  "name": "my-first-plugin",
  "description": "A greeting plugin to learn the basics",
  "version": "1.0.0",
  "author": { "name": "Your Name" }
}
```

技能 `my-first-plugin/skills/hello/SKILL.md`：

```
---
description: Greet the user with a friendly message
disable-model-invocation: true
---

Greet the user warmly and ask how you can help them today.
```

本地加载测试（不必先发布），然后调用带命名空间的技能：

```
claude --plugin-dir ./my-first-plugin      # 可重复传多次;也接受 .zip(v2.1.128+)
# 会话内:
/my-first-plugin:hello
```

清单 `plugin.json` 字段：

字段	说明
`name`	唯一标识，也是技能命名空间前缀（`/name:skill`）
`description`	插件管理器里展示的描述
`version`	可选。设了它，用户仅在你 bump 版本号时才收到更新；省略且用 git 分发时，每个 commit 的 SHA 都算新版本
`author` / `homepage` / `repository` / `license`	可选，元数据

改动后 `/reload-plugins` 热重载，无需重启。想验证清单/YAML 是否合法，跑 `claude plugin validate .`（或会话内 `/plugin validate .`）。

> *“🆕 skills-dir 插件：`claude plugin init my-tool` 会在 `~/.claude/skills/my-tool/` 脚手架一个带清单的插件，下次启动直接以 `my-tool@skills-dir` 加载，无需 Marketplace 也无需 install。适合个人自用又想要插件目录结构的场景。”*

发布时，再写一份 `.claude-plugin/marketplace.json` 目录、推到 GitHub，别人就能 `/plugin marketplace add 你的仓库` + `/plugin install`。marketplace.json 顶层是 `name` / `owner` / `plugins[]`，每个插件项至少给 `name` 与 `source`（相对路径、`github`、`url`、`git-subdir`、`npm` 之一）。完整 schema 见官方文档。

团队分发与 settings

要让协作者 clone 仓库、信任目录后被自动提示安装，在项目 `.claude/settings.json` 里声明 Marketplace 与默认启用的插件：

```
{
  "extraKnownMarketplaces": {
    "company-tools": {
      "source": { "source": "github", "repo": "your-org/claude-plugins" }
    }
  },
  "enabledPlugins": {
    "code-formatter@company-tools": true,
    "deployment-tools@company-tools": true
  }
}
```

安装作用域三选一：user（自己所有项目）、project（写进仓库 `.claude/settings.json`，全体协作者）、local（仅自己、仅本仓库）。企业管理员可用托管设置的 `strictKnownMarketplaces` 锁定/allowlist 允许添加的来源（`[]` 完全锁死），用户无法覆盖。Marketplace 与缓存状态存在 `~/.claude/plugins/`（`known_marketplaces.json`、`cache/`）。

版本与更新

Claude Code 按这个顺序解析插件版本：① `plugin.json` 的 `version` → ② Marketplace 条目里的 `version` → ③ 插件源的 git commit SHA。所以要么每次发布 bump `version`，要么干脆省略 version、靠 commit SHA 自动区分——两者都设且忘了改，老版本用户会一直停在缓存里。

官方 Marketplace 默认开启后台自动更新（启动后随机延迟至多 10 分钟拉取，不打断当前会话，更新后提示你 `/reload-plugins`）；第三方与本地开发 Marketplace 默认关闭。手动刷新用 `/plugin marketplace update`。

常见误区

1. 组件放错层级 —— `commands/`、`skills/`、`agents/`、`hooks/` 必须在插件根，不能塞进 `.claude-plugin/`（只放 `plugin.json`）。
2. 改了插件却没生效 —— 跑 `/reload-plugins`；涉及 MCP 工具集变化时加 `--force`。`/reload-skills` 不会重载插件。

3. 推了新 commit 用户却没更新 —— `plugin.json` 里写死了 `version` 又没 bump；要么 bump，要么删掉 version 用 SHA。
4. URL 直链 Marketplace 里用了相对路径 source —— 只下载 `marketplace.json` 本身，相对路径插件文件取不到；改用 `github` / `npm` /git URL source，或把 Marketplace 托管成完整 git 仓库。
5. 引用了插件目录外的文件（如 `../shared-utils`） —— 插件安装时会被复制进缓存，外部文件不会跟着走；用 `${CLAUDE_PLUGIN_ROOT}` 引用包内文件，跨插件共享用 symlink。

延伸阅读

- Commands、Skills、Agents——插件打包的三类主要组件
- 自定义与扩展 · 插件—— `/plugin` 、 `/reload-plugins` 命令速查
- 配置文件—— `settings.json` 里的 `extraKnownMarketplaces` 、 `enabledPlugins`

← 上一篇：Agents · 回到：专栏首页

版本 Changelog

官方 CHANGELOG.md 有近 5000 行、覆盖每一个补丁版本。这里只挑对日常使用有实际影响的里程碑事件，而不是逐条搬运。想看某个具体版本的完整变更，用会话内的 `/release-notes` 或直接翻官方文件。

当前最新版本：v2.1.212。 下面「v2.1.205 → v2.1.212」一节按版本逐条梳理，标记约定：🆕 = 该版本引入的新特性 / 新命令 / 新配置；⚠️ = 行为变更（可能影响你现有的用法）；其余为修复与体验优化。更早的版本只保留速览与主题线索。

模型发布节点

版本	事件
1.0.0	Claude Code 正式 GA，随附 Sonnet 4 与 Opus 4
2.0.51 附近	Opus 4.5 发布，推出 Claude Code for Desktop
2.1.111	Opus 4.7 上线
2.1.154	Opus 4.8 fast mode 大幅降价（2 倍单价换 2.5 倍速度）
2.1.170	Fable 5 发布——Mythos 级模型，能力超过此前任何正式发布的模型
2.1.197	Sonnet 5 发布并成为默认模型，原生 1M token 上下文窗口，发布初期促销价 2/10 每 Mtok

协作与编排能力的演进

这条线索大致反映了 Claude Code 从“单会话问答工具”演变为“可编排的并行 agent 平台”的过程：

- 自定义 subagent（约 1.0.60）：可以创建专用 subagent 处理特定任务，`/agents` 首次上线（交互式管理界面，后在 2.1.198 简化为提示语，改为直接编辑 `.claude/agents/`）。
- Explore subagent（2.0.17）：内置一个基于 Haiku 的探索型 subagent，专门用来高效检索代码库、节省主上下文。
- 检查点与回滚（2.0.0）：引入 `/rewind`，可以撤销代码改动。
- 插件体系（2.0.12 附近）：`/plugin install`、`enable` / `disable`、marketplace 管理陆续完善，支持打包分发技能、subagent、hooks、MCP 服务器。

- 技能取代自定义命令（2.1.0 前后）：`context: fork` 让技能可以派生到独立 subagent 执行；技能支持热重载，新增/修改后无需重启会话。
- 动态工作流（2.1.154）：让 Claude 把一个大任务编排成跨数十到数百个 subagent 的后台执行方案，`/workflows` 用于查看进度——这是目前处理“体量远超单会话”任务的核心机制。
- 结构化 Task 工具默认开启（2.1.142）：`TaskCreate` / `TaskUpdate` / `TaskGet` / `TaskList` 取代旧版 `TodoWrite` 成为默认任务追踪方式。
- 后台 subagent 默认开启（2.1.198）：subagent 默认在后台运行，Claude 可以一边等待结果一边继续做别的事；同版本 Claude in Chrome 正式 GA。
- Claude Code on the web / Remote Control 生态持续加固：`/teleport`、`/remote-control`、`/desktop`、`claude agents` 等命令让本地终端、Web 会话、移动端可以互相流转与监控（集中体现在 2.1.19x 系列的大量修复与体验优化中）。

v2.1.205 → v2.1.212（逐版本详解）

这是手册上一版（截至 v2.1.204）之后的全部版本。🆕 新特性、⚠️ 行为变更单独标出，其余为修复/优化的精选。

v2.1.212（当前最新）

- 🆕 `/fork` 语义变了：现在把对话复制成一个独立的后台会话（而不是原来的会话内 subagent）。想要原来「会话内派生 subagent」的行为，用新命令 `/subtask`。
- 🆕 给失控加了两道会话级上限：`WebSearch` 调用默认封顶 200 次、subagent 派生默认封顶 200 个（都可用环境变量调整），防止无限搜索 / 无限委派烧钱。
- 🆕 `claude auto-mode reset`：一条命令（带确认）把 auto mode 设置恢复默认。
- ⚠️ Task 工具的 `mode` 参数弃用：subagent 现在直接继承父会话的权限，不再单独指定。
- MCP 工具调用超过 2 分钟自动转后台，避免卡死整个会话；`/resume` 的选择器现在把已删除的会话也列出来（作为后台会话恢复）。
- 安全修复：计划模式（plan mode）不再会未经确认就执行会改文件的 Bash 命令。
- 会话 transcript 现在按每条 assistant 消息记录推理强度（reasoning effort）；`/ultrareview` 一批修复（接受 `#123` / `PR 123`、会拉远端分支、`/clear` 后仍要计费确认）。

v2.1.211

- 🆕 `--forward-subagent-text` 标志与同名环境变量：把 subagent 的思考过程也带进输出。

- Vim 模式的 `s` / `S`（替换）现在在 NORMAL 模式下按 vim 语义工作；「always allow」规则改存到仓库根，让 git worktree 之间也持久生效；`/usage-credits` 发管理员请求前需确认。

v2.1.210

- 🆕 安全加固：Agent 工具加固，抵御经由 subagent 内容的间接提示注入；`ultracode` 关键词不再被非人类的 webhook / 转发的 PR 评论误触发。
- 🆕 折叠的工具摘要行现在带实时经过时间计数；内置 dataviz 技能升级（感知均匀配色校验 + 色盲阈值）。
- 启动时对 `Write` / `NotebookEdit` / `Glob` 的权限规则给出告警（建议更好的替代写法）；屏幕阅读器模式会朗读权限模式变化。（注：Fable 因服务端问题一度不可用。）

v2.1.209 —— 修复后台会话里 `/model` 等对话框被挡住的问题。

v2.1.208

- 🆕 屏幕阅读器模式：纯文本渲染，可用标志 / 环境变量 / 设置开启。
- 🆕 `vimInsertModeRemaps` 设置：把 `jj` 这类两键序列映射成 Escape；🆕 `CLAUDE_CODE_PROCESS_WRAPPER`：满足企业启动器（launcher）的包装需求。
- 多选菜单与「Other」输入行支持鼠标点击；超过 200 行的大 Markdown 表格显示截断提示（不再拖垮渲染）；一大批内存泄漏 / 性能修复（编辑读缓存降到 16 MB 等）。
- 安全：带子 shell 的「毁灭性删除」命令即使在 `--dangerously-skip-permissions` 下也会弹确认。

v2.1.207

- 🆕 auto mode 在 Bedrock / Vertex / Foundry 上免 opt-in 直接可用（可在设置里关）。
- ⚠️ Bedrock / Vertex / AWS 上的 Claude Platform 默认模型改为 Claude Opus 4.8。
- 修复终端在流式输出很长的列表 / 表格 / 代码块时卡死；修复远端托管设置被记为「已同意」却没弹安全对话框。

v2.1.206

- 🆕 `/cd` 现在像 `/add-dir` 一样给目录路径建议；🆕 `/commit-push-pr` 对配置好的 push remote 自动放行 `git push`。
- `/doctor` 新增检查：建议精简签入仓库的 `CLAUDE.md`；`EnterWorktree` 进入 `.claude/worktrees/` 以外的 worktree 前会先确认；Claude Code 更新后，后台 agent 在后台自行升级。

v2.1.205

- 🆕 auto mode 新增规则：阻止篡改会话 transcript。
- `--json-schema` 遇到非法 schema 现在报错，不再静默产出非结构化输出；`/review <pr>` 改回快速单轮（多 agent 深度评审用 `/code-review <level>`）；`/workflows` 的 agent 列表改进（更宽标题、独立时间列、更短模型名）。

v2.1.190 → v2.1.204（速览）

这一段是手册上一版覆盖的最后一批版本，只保留对日常使用影响较大的变化，细节以官方 CHANGELOG 为准：

- v2.1.204 —— 修复无头（headless）会话里 `SessionStart` hook 期间事件流中断，可能导致远程 worker 被误判空闲而回收的问题。
- v2.1.203 —— 登录即将过期时提前预警，避免后台会话被打断；`/mcp` 的 `roots/list` 现在会带上本次会话新增的工作目录；修复多个 `claude agents` 相关的稳定性问题（卡死、状态丢失、worktree 隔离失效等）。
- v2.1.202 —— 新增 `/config` 里的“动态工作流规模”设置（建议性质，不是硬限制）；修复 `/rename` 在后台会话重启后被还原的问题；`/review <PR>` 改回快速单轮评审，多 agent 深度评审改用 `/code-review <pr#>`。
- v2.1.200 —— `AskUserQuestion` 对话框默认不再自动继续（需要在 `/config` 里主动开启超时）；默认权限模式的展示名从 “default” 改为 “Manual”（`--permission-mode manual` 与 `default` 等价可互换）。
- v2.1.199 —— 堆叠式技能调用（`/skill-a /skill-b 继续`）现在最多可以一次性加载 5 个前置技能；`CLAUDE_CODE_RETRY_WATCHDOG` 把非容量类瞬时错误的默认重试次数提到 300 次，并解除 `CLAUDE_CODE_MAX_RETRIES` 原本 15 次的上限。
- v2.1.198 —— subagent 默认后台化；Claude in Chrome 正式 GA；新增 `/dataviz` 技能；`/run`、`/verify`、`/run-skill-generator` 三件套上线，用“实际跑起来看效果”替代“只看测试通过”。
- v2.1.197 —— Sonnet 5 发布，成为默认模型。
- v2.1.196 —— 支持组织级默认模型；安全加固：`.mcp.json` 中仓库自行批准的 MCP 服务器不再被 `claude mcp list` / `get` 静默启动；流式响应空闲看门狗（`CLAUDE_ENABLE_STREAM_WATCHDOG`）默认对所有提供方开启。

- v2.1.193 —— 新增 `autoMode.classifyAllShell`、OTel `assistant_response` 日志事件、bash 模式下的实时路径自动补全；新增后台 shell 因系统内存压力自动回收的机制。
- v2.1.187 —— 新增 `sandbox.credentials` 设置，阻止沙箱化命令读取凭证文件与密钥环境变量；远程 MCP 工具调用新增空闲超时（`CLAUDE_CODE_MCP_TOOL_IDLE_TIMEOUT`），避免无响应时无限期挂起。
- v2.1.186 —— `claude mcp login/logout <server>` 支持单独对某个 MCP 服务器走 OAuth，无需打开完整的 `/mcp` 面板。

更早的重要节点

- 2.0.0：全新原生 VS Code 插件；引入 `/rewind` 检查点回滚。
- 1.0.81 前后：推出输出风格（output styles），含内置的 “Explanatory”、“Learning” 教学向风格（该功能后续经历过一次弃用又恢复的反复，最终保留）。
- 0.2.x 系列：项目早期阶段，OTel 支持初具雏形，`CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` 等隐私类开关陆续加入。

> Claude Code 的发布节奏很快（经常一天多个补丁版本），`autoUpdatesChannel` 设为 `stable` 可以避免第一时间踩到重大回归——细节见安装与升级 · 配置发布渠道。

← 回到：专栏首页 · 进阶：Agents

道雾轩

《Claude Code 中文手册》

本电子书由道雾轩制作,免费分享,欢迎转发给需要的人。

阅读全部专栏,请访问官网 daiw.org。