SUPERTOKEN 中文文档

Codex 中文使用手册

桌面 App、CLI、IDE、团队定制、实战与排错

SuperToken 中文手册

本机官方 CLI 实际输出

OpenAI Codex CLI

```
$ codex --version
codex-cli 0.144.4

$ codex --help
Codex CLI

If no subcommand is specified, options will be forwarded to the interactive CLI.

Usage: codex [OPTIONS] [PROMPT]
       codex [OPTIONS] <COMMAND> [ARGS]

Commands:
  exec            Run Codex non-interactively [aliases: e]

Selected commands and options:
  exec            Run Codex non-interactively [aliases: e]
  review          Run a code review non-interactively
  mcp             Manage external MCP servers for Codex
  doctor          Diagnose local Codex installation, config, auth, and runtime health
  resume          Resume a previous interactive session (picker by default; use --last to continue
  -s, --sandbox <SANDBOX_MODE>
      --search
```

来源：OpenAI Codex CLI codex-cli 0.144.4

核对日期：2026-07-16

OpenAI Codex

基于 OpenAI 官方文档与当前官方 CLI 整理，非 OpenAI 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

Codex 中文使用手册

Codex 不只是一个终端命令。它可以在 ChatGPT 桌面 App、CLI、IDE 扩展和 Cloud 环境中理解项目、修改代码、运行命令并审查结果。本手册先帮你选对入口，再完成一条可验证的真实开发流程。

SuperToken 中文整理版 · 核对日期 2026-07-15 · 当前模型 GPT-5.6 · 实测 CLI `0.144.3` · 非 OpenAI 官方译本

> **阅读边界**
>
> 产品能力以当前 OpenAI Codex 官方文档 和你的账号界面为准。界面、模型和实验功能可能分批开放；手册不会把未验证能力写成所有账号都能使用。

> **当前模型基线**
>
> 当前推荐系列是 GPT-5.6：默认 Power 使用 5.6 Sol，Terra 面向日常工作，Luna 面向清晰、重复和高吞吐任务。ChatGPT 登录场景中的 `gpt-5.2` 与 `gpt-5.3-codex` 已弃用；模型选择与迁移细节见 桌面 App 完整操作。

先选对入口

入口	最适合	你会直接接触什么	不适合
ChatGPT 桌面 App 中的 Codex	同时管理多个任务、可视化审查 diff、使用集成终端和 Worktree	项目侧栏、任务、Review、终端、Local / Worktree / Cloud	纯脚本流水线
Codex CLI	终端优先、本地仓库、可复现命令和非交互任务	当前目录、终端会话、sandbox、approval、JSONL 输出	依赖大量鼠标操作的界面工作
IDE 扩展	围绕当前选区、文件和编辑器上下文快速迭代	Codex 侧栏、选中代码、当前文件、编辑器 diff	大量并行后台任务
Cloud	让任务在已配置的远程环境中执行	云端仓库快照、setup、网络策略和远程结果	必须依赖未上传本地文件或本机进程的任务

最简单的选择规则：

- 第一次使用，想看清过程：从 **桌面 App + Local** 开始。
- 已经习惯终端：从 **CLI + workspace-write** 开始。

- 问题只涉及当前文件或选区：使用 **IDE 扩展**。
- 想并行处理独立任务，且远程环境已经配好：使用 **Worktree 或 Cloud**。

Subagents 是跨入口的并行能力

当前 Codex 在桌面 App、CLI 和 IDE 扩展中都可以运行 subagent workflow。它不是第五个独立入口，而是由主任务把边界清楚的工作分给多个 agent thread，再汇总结果。

- 最适合并行：代码库探索、测试、日志分析、按维度审查和资料汇总。
- 谨慎并行：多个 agent 同时修改相同文件，容易产生覆盖、冲突和额外协调成本。
- 子代理继承父任务当前权限；非交互任务无法弹出新审批时，需要额外权限的动作会失败并回报主任务。
- 每个子代理都会单独使用模型和工具，因此通常比单代理消耗更多 Token。
- CLI 用 `/agent` 或 `/subagents` 查看和切换 agent thread；App 与 IDE 在当前界面提供子代理活动入口时，也应逐个检查结果。

10 分钟完成第一次任务

这条最短路径使用一个你熟悉、可以运行测试的 Git 仓库。第一次不要拿生产密钥仓库、个人主目录或尚未备份的项目练习。

1. 确认工作区

在终端进入项目并检查当前状态：

```bash
cd /path/to/your-project
git status --short --branch
```

预期结果：你能说清当前分支和已有未提交修改。已有修改不用清空，但必须知道哪些不是本次任务产生的。

2. 打开 Codex

桌面 App：打开 ChatGPT 桌面 App，在产品下拉框选择 **Codex**，用 `Cmd/Ctrl + O` 打开项目，运行位置选择 **Local**。

CLI：

```bash
codex -C /path/to/your-project
```

预期结果：任务明确绑定到正确目录。若界面显示的项目或 CLI 当前目录不对，先退出并修正，不要靠提示词补救错误工作区。

3. 先做只读调查

输入：

```text
先不要修改文件。请检查这个项目并告诉我：
1. 它解决什么问题，主要入口在哪里；
2. 安装、开发、测试和构建命令分别来自哪个文件；
3. 当前 Git 状态中哪些改动已经存在；
4. 如果只改 README 中的一处错别字，最小验证方法是什么。

不要读取或输出 .env、密钥和账号信息。结论尽量引用具体文件。
```

预期结果：Codex 只读文件和仓库状态，没有产生 diff；命令来自 `package.json`、任务文件或项目文档，而不是凭空猜测。

4. 做一个最小修改

确认调查结果无误后再输入：

```text
只修复 README 中刚才指出的那一处错别字。
不要改写段落，不调整格式，不修改其他文件。
完成后显示 diff，并运行最小验证；最后说明实际执行了什么。
```

审批时只允许与当前任务匹配的文件写入和命令。陌生命令、仓库外路径、网络访问或凭据读取都应先拒绝并追问原因。

5. 用证据验收

至少完成四项检查：

1. Review 面板或 `git diff -- README.md` 只显示目标改动。
2. `git status --short` 没有意外文件。
3. Codex 报告的验证命令确实执行成功，而不是只建议你运行。
4. 原有未提交改动没有被覆盖、还原或混入本次结果。

这才算任务完成。模型说“已经修复”本身不是证据。

第一次不要这样做

- 不要从主目录、下载目录或包含多个项目的上级目录启动。
- 不要把 `danger-full-access` 或危险绕过审批作为默认配置。
- 不要把真实 Key、`.env` 内容、客户数据或私人会话贴进提示词。
- 不要让 Codex 在没有复现问题时连续试错式修改生产代码。
- 不要在没看 diff、没跑验证时直接提交或推送。

- 不要把 Worktree 当成数据库、端口、云账号或外部服务的隔离层。

手册怎么读

你的目标	阅读
在桌面 App 完成打开项目、修改、Review 和验证	桌面 App 完整操作
掌握交互式 CLI、图片输入、恢复会话和自动化	CLI 使用
在编辑器中工作，或把任务交给 Cloud	IDE 与 Cloud
配置 AGENTS.md、权限、MCP、Skills、Plugins 和 Hooks	项目与团队定制
跟着完整案例练习 Bug、功能、重构、审查和前端验收	项目实战
遇到目录、认证、权限、网络、MCP 或 Worktree 问题	故障排除与速查

一条成熟的日常工作流

1. **定界：** 确认仓库、分支、已有修改、数据和权限范围。
2. **调查：** 先读入口、相似实现、测试和项目规则。
3. **计划：** 写清方案、改动文件、失败状态和验证命令。
4. **实现：** 做最小纵向闭环，避免顺手重构和升级依赖。
5. **审查：** 检查完整 diff，而不是只看最终回答。
6. **验证：** 运行测试、类型检查、lint、构建或人工界面检查。
7. **交付：** 列出改动、证据、未验证项和剩余风险。
8. **沉淀：** 把稳定约定放进 `AGENTS.md`、配置或复用流程，不把临时对话当长期规则。

产品边界速记

- Codex 可以操作文件和命令，但实际范围由工作区、sandbox、approval、组织策略和外部系统权限共同决定。
- CLI、IDE 和桌面 App 共享一部分 `config.toml` 行为；编辑器外观和 `chatgpt.*` 设置属于 IDE 自己。
- Local 与 Worktree 都在本机运行；Cloud 使用远程环境，不会自动拥有本机未提交文件、进程或凭据。
- Review 面板展示的是 Git 仓库状态，其中可能同时包含你、Codex 和其他工具产生的修改。
- Worktree 隔离 Git 文件和分支，不隔离数据库、端口、缓存、浏览器登录或云资源。
- Subagents 适合并行读和独立验证，不是并行写同一工作区的默认方案；委派前先划分文件所有权和完成标准。

第 2 部分

桌面 App

Codex 桌面 App 完整操作

桌面 App 的价值不是把终端换成聊天框，而是把项目、任务、diff、终端和并行工作区放进同一条可审查流程。

核对日期 2026-07-15 · 依据 OpenAI 当前官方 Codex 手册 · 当前模型基线 GPT-5.6

开始之前

准备以下条件：

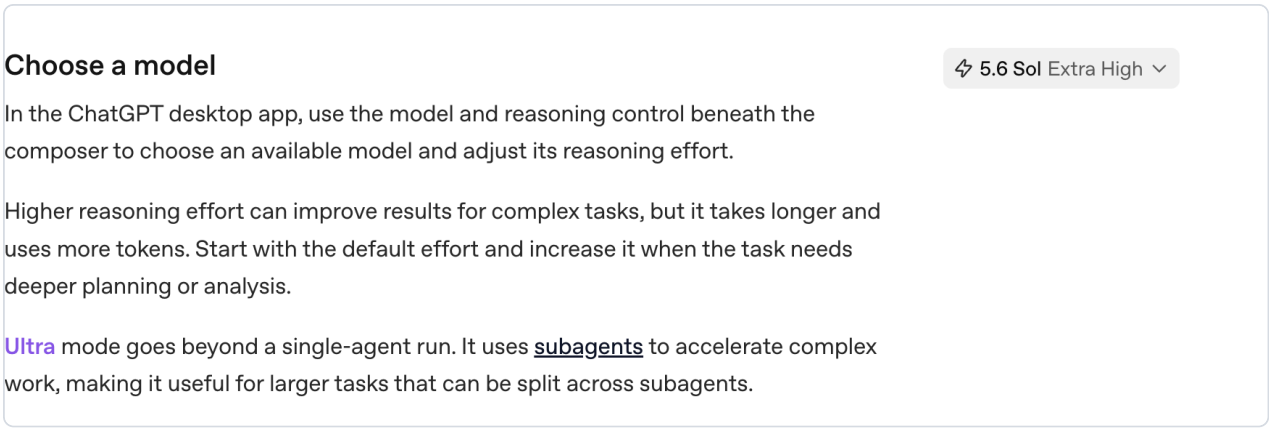
- ChatGPT 桌面 App 中可以选择 **Codex**。
- 项目是你信任的本地目录；需要 Review 或 Worktree 时应是 Git 仓库。
- 你知道项目的安装、测试和构建命令，或允许 Codex 先从仓库文件中确认。
- 工作区没有未识别的敏感文件；真实 Key 不出现在截图和提示词中。

> **App 不等于无限权限**
>
> App 能看到哪些目录、能否写入、是否能运行命令和访问网络，仍受当前项目、sandbox、approval、系统权限和组织策略限制。

当前模型：GPT-5.6

OpenAI 当前模型页在 ChatGPT 桌面 App、Codex CLI 和 IDE 扩展中推荐 GPT-5.6 系列。默认 **Power** 档使用 `gpt-5.6-sol` 和 Medium reasoning；需要更快或更低成本时再向 Faster 调整，只有明确需要时才进入 **Advanced** 固定具体模型。

模型	适合
5.6 Sol	复杂、开放式、高价值任务，需要更强分析、判断和交付质量
5.6 Terra	日常开发、代码探索和工具调用，在能力与成本间取平衡
5.6 Luna	边界明确、可重复、高吞吐任务，优先速度与成本



图：当前官方模型选择区域已显示 GPT-5.6 Sol。 ChatGPT 桌面 App / Codex GPT-5.6 模型页；截图获取于 2026-07-15；来源：OpenAI 官方 Models 页面实时渲染截图，界面英文保持原样；核对日期：2026-07-15；官方公开文档界面；不含账号、API Key、私人项目或会话数据

> **旧模型边界**
>
> 使用 ChatGPT 登录时，`gpt-5.2` 和 `gpt-5.3-codex` 已被 Codex 标记为弃用。本版不再发布显示 `GPT-5.2-Codex` 的旧 App 图。API Key 或自定义模型提供方是否仍支持旧 ID 是另一套可用性判断，不能从 ChatGPT 登录状态直接外推。

认识四个核心区域

区域	主要用途	你要检查什么
项目与任务侧栏	切换项目、搜索任务、管理并行工作	当前项目、任务名称、是否进错仓库
任务记录与输入框	描述目标、查看工具动作、追加约束	Codex 正在读什么、改什么、运行什么
Review 面板	查看 Git diff、行内反馈、stage、unstage 或 revert	改动是否越界，哪些修改原本就存在
底部面板与终端	运行验证、查看本地服务或命令输出	命令目录、退出码、错误是否被忽略

常用快捷键：

动作	macOS / Windows
打开文件夹	`Cmd/Ctrl + O`
新任务	`Cmd/Ctrl + N`
搜索历史任务	`Cmd/Ctrl + G`
任务内查找	`Cmd/Ctrl + F`
打开 Review	`Ctrl + Shift + G`

动作	macOS / Windows
切换底部面板	`Cmd/Ctrl + J`
切换终端	`Ctrl + ``
打开设置	`Cmd/Ctrl + ,`

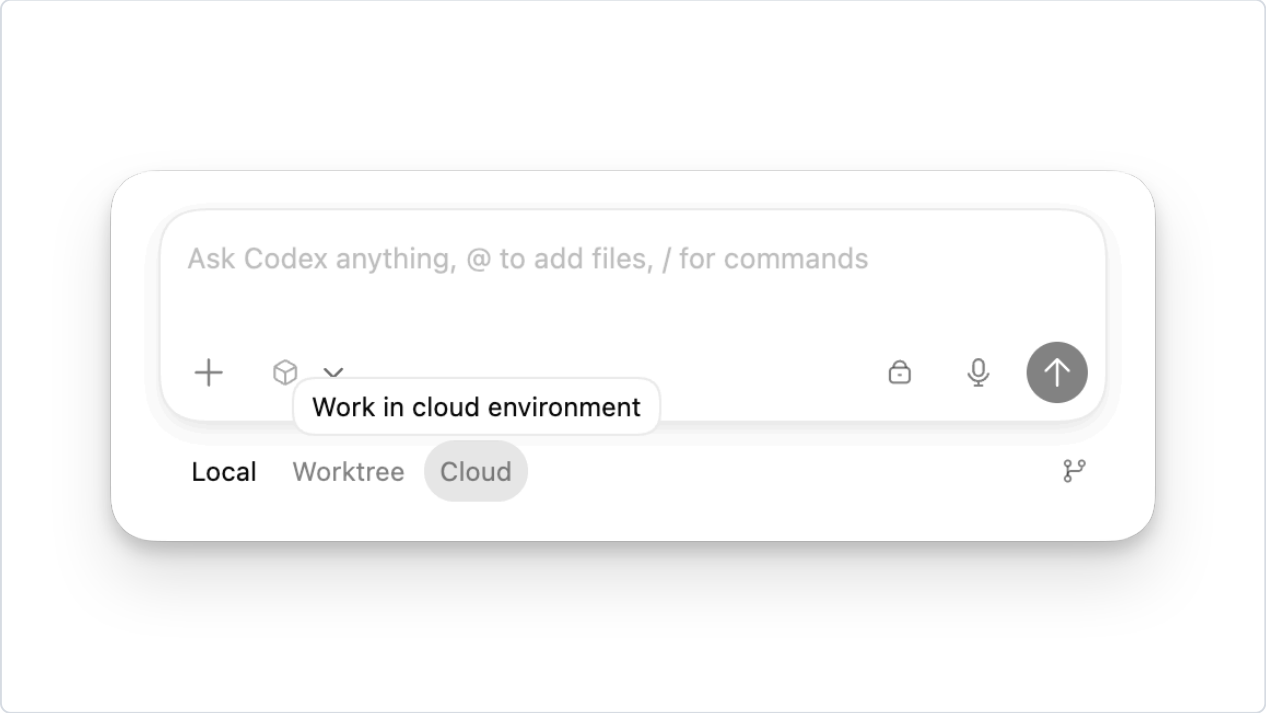
快捷键可以在 **Settings > Keyboard Shortcuts** 中搜索、修改或恢复默认值。

流程一：在 Local 完成一次小改动

1. 打开正确项目

1. 在 ChatGPT 桌面 App 的产品下拉框选择 **Codex**。
2. 按 `Cmd/Ctrl + O`，选择仓库根目录，而不是它的上级目录。
3. 新任务输入框下方选择 **Local**。
4. 开始前让 Codex 报告当前目录、Git 分支和未提交文件。

选择前先分清运行位置：**Local** 直接在当前项目目录工作，**Worktree** 在本机 Git worktree 中隔离修改，**Cloud** 在已配置的云环境中远程运行；Local 和 Worktree 都在你的电脑上运行。



图：新任务先选对 Local、Worktree 或 Cloud。 OpenAI Codex App 官方文档当前界面；客户端版本未标注；来源：OpenAI 官方 Codex environments；核对日期：2026-07-14；官方示例画面；不含真实账号、API Key 或私人项目数据

预期结果：任务绑定到本地 checkout；Review 能识别仓库状态。

失败分支：

- 看不到 Review：项目可能不是 Git 仓库，或打开的不是仓库目录。
- 项目路径不对：关闭当前任务并重新打开目录，不要继续授权写入。
- 存在陌生未提交修改：暂停，让 Codex 只读说明它们，不要自动清理。

2. 先调查，再让它动手

```text
先不要修改。检查登录页提交逻辑、对应请求封装和现有测试。
复现“连续点击登录会发出重复请求”的问题，给出：
- 已确认事实；
- 仍待验证的假设；
- 最小修改方案；
- 需要运行的验证。
```

预期结果：先出现文件读取、搜索和必要的只读命令；在你确认方案前没有源代码 diff。

如果 Codex 过早修改，直接输入：

```text
停下。先说明已经改了哪些文件，恢复到调查阶段；不要覆盖我原有的修改。
```

3. 审批工具动作

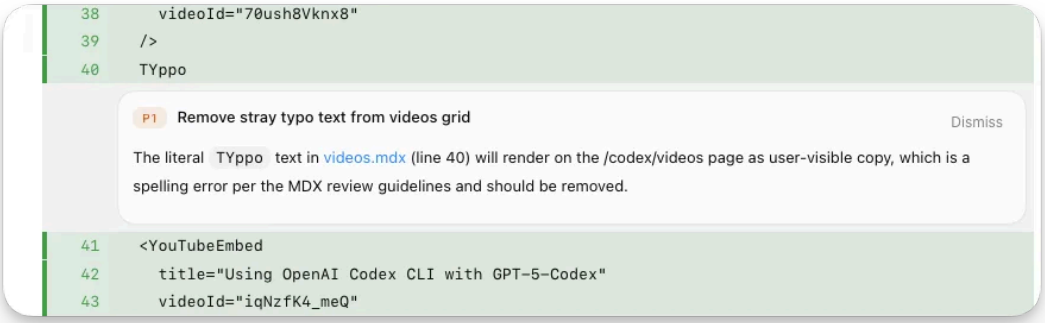
审批前逐项看：

1. **动作是什么：** 读文件、写文件、运行命令还是访问网络。
2. **范围在哪里：** 是否仍在当前仓库，是否扩大到上级或其他目录。
3. **为什么需要：** 是否直接服务于当前任务。
4. **后果是什么：** 会不会删除、覆盖、提交、推送、安装全局软件或访问账号。

对 `git status`、读取项目文件和已知测试命令可以更快确认；对删除、回滚、仓库外写入、发布、推送和凭据访问保持人工确认。

4. 在 Review 面板检查完整 diff

1. 打开 Review 面板。
2. 先看 **Unstaged**；需要时切换 **Staged**、**Commit**、**Branch** 或 **Last turn**。
3. 展开每个文件，检查是否有无关格式化、依赖更新、调试日志或敏感内容。
4. 对具体行悬停并选择 `+`，留下行内反馈。
5. 发送明确跟进：`处理行内评论，保持改动范围不变；完成后重新验证。`



图：把反馈留在对应代码行，再让 Codex 定点处理。 OpenAI Codex App 官方文档当前界面；客户端版本未标注；来源：OpenAI 官方 Code review；核对日期：2026-07-14；官方示例代码与审查意见；不含真实账号、API Key 或私人仓库数据

预期结果：你能把每一处改动对应到原始需求。Review 展示整个仓库的 Git 状态，并不只包含 Codex 产生的修改。

> **Revert 前先辨认所有者**
>
> Review 支持按整个 diff、文件或 hunk stage、unstage 和 revert。Revert 会丢弃对应修改；如果其中包含你原先的工作，不能直接执行。

5. 在集成终端验证

从右上角终端按钮或 `Ctrl + `` 打开终端。终端跟随当前 Local 或 Worktree 项目。

```bash
git status --short --branch
npm test -- --runInBand
npm run typecheck
npm run lint
```

只运行项目真实存在的命令。先从 `package.json`、Makefile、任务配置或项目文档确认，不要照抄示例。

验证完成后要求 Codex 报告：

- 实际执行的命令和退出结果；
- 哪些检查没有执行以及原因；
- diff 中每个文件的作用；

- 仍需人工检查的界面或外部系统状态。

流程二：用 Worktree 隔离并行任务

Worktree 适合后台开发、实验或当前 Local 已有未完成工作时的独立任务。

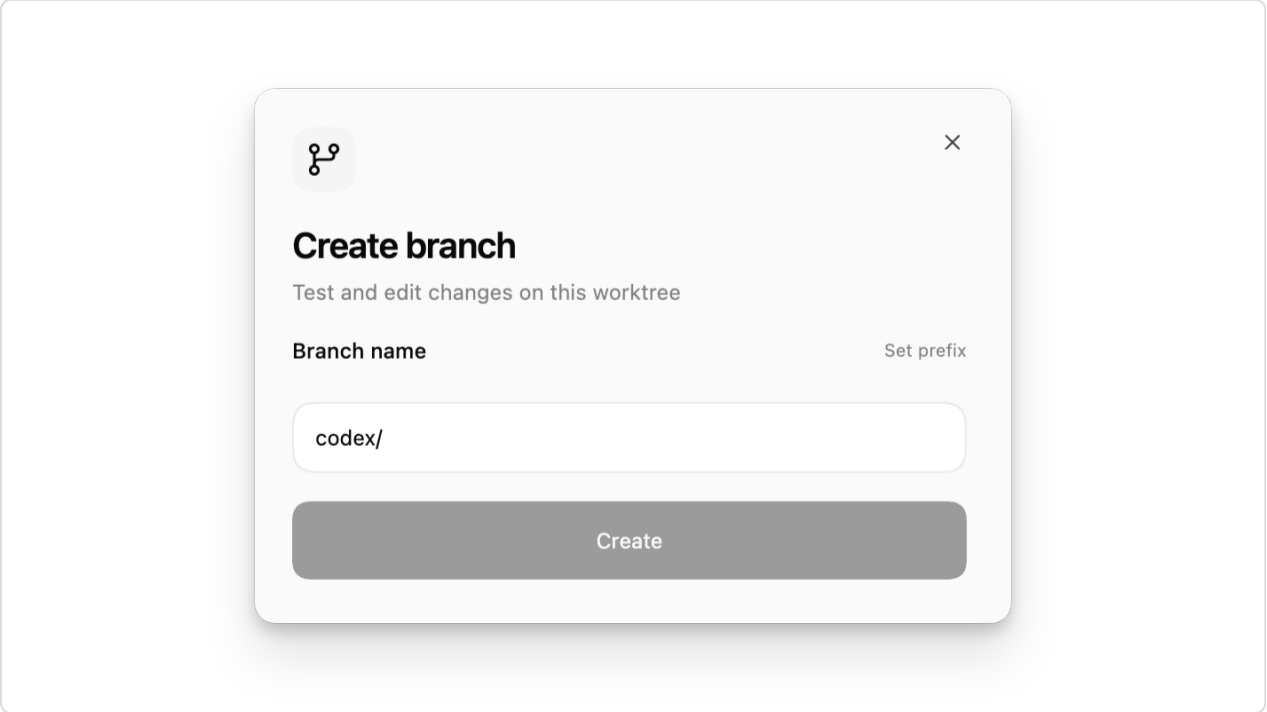
创建 Worktree 任务

1. 新任务输入框下选择 **Worktree**。
2. 选择起始分支；它可以是 `main`、功能分支或当前带未提交改动的分支。
3. 如项目需要依赖安装，选择已经配置好的 Local Environment。
4. 提交任务。Codex 默认在托管 worktree 的 detached HEAD 上工作。

预期结果：侧栏出现独立任务；Local checkout 不直接出现该任务的文件修改。

在 Worktree 创建分支

1. 先在 Worktree 的集成终端或 IDE 中验证改动。
2. 确认要继续保留这份工作后，在任务头部选择 **Create branch here**。
3. 在对话框中确认分支名并创建分支。
4. 继续在这个 Worktree 中提交、推送或创建 Pull Request；只在你确认改动和远端目标后执行这些动作。

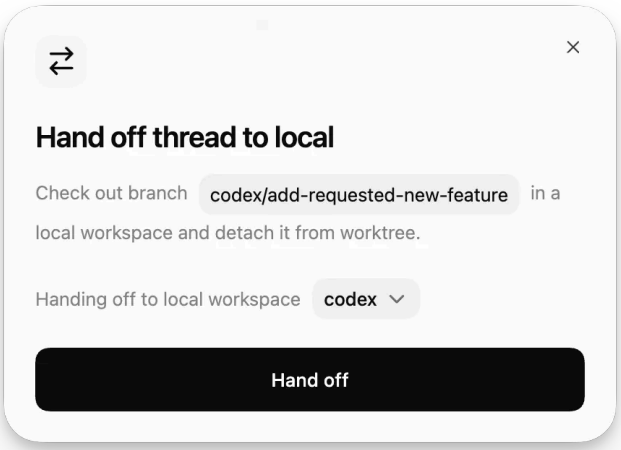


图：把 detached HEAD 上的 Worktree 工作保存到分支。 OpenAI Codex App 官方文档当前界面；客户端版本未标注；来源：OpenAI 官方 Worktrees；核对日期：2026-07-14；官方示例分支对话框；不含真实账号、API Key 或私人仓库数据

同一个 Git 分支不能同时签出到 Local 和另一个 Worktree。需要回到日常本地环境时，使用 Handoff，不要在两个 checkout 中强行切到同一分支。

Hand off 到 Local

1. 在 Worktree 任务头部选择 **Hand off**，目标选择 **Local**。
2. 在对话框中确认要接收任务的本地 workspace。
3. 完成交接后，在熟悉的本地 IDE、开发服务或终端中继续检查和验证。
4. 之后若 Hand off 回 Worktree，任务会回到原来关联的 Worktree。



图：把任务和代码从 Worktree 交接到 Local。 OpenAI Codex App 官方文档当前界面；客户端版本未标注；来源：OpenAI 官方 Worktrees；核对日期：2026-07-14；官方示例分支与 workspace 名称；不含真实账号、API Key 或私人仓库数据

Handoff 会处理在两个 checkout 之间移动任务和代码所需的 Git 操作。被 `.gitignore` 忽略的文件不会随任务移动，除非它们按官方规则通过 `.worktreeinclude` 复制到本地托管 Worktree。

常见失败

现象	原因	处理
项目不能选择 Worktree	不是 Git 仓库，或当前没有选择 Codex	确认 Git 根目录和产品入口
新 worktree 缺依赖或本地配置	Git 只带 tracked 文件	配置 Local Environment；只对确有必要的忽略文件使用 `.worktreeinclude`
分支无法在 Local checkout 切换	同一分支已经被另一个 worktree 使用	使用 Handoff，或让两个 worktree 使用不同分支

现象	原因	处理
两个任务仍互相影响	共用了数据库、端口、缓存或外部账号	分配独立端口、测试数据和外部资源

> **`.worktreeinclude` 不是密钥同步清单**
>
> 官方支持把必要的忽略文件复制到托管 worktree，但列入 `.env` 或 secret 文件会扩大暴露面。优先使用测试凭据、短期凭据或环境初始化方式，并确保文件永不进入 Git。

流程三：让 `/review` 做专门审查

在 Git 仓库任务中输入 `/review`，按需要选择：

- **Review against a base branch**：检查当前分支相对基线的完整变化。
- **Review uncommitted changes**：检查 staged、unstaged 和 untracked 修改。
- **Review a commit**：检查一个明确提交。
- **Custom review instructions**：只关注权限、并发、数据一致性等标准。

审查任务应要求每个发现包含严重程度、文件位置、触发条件和影响。对没有证据的风格偏好不做“问题”报告。

审查结果默认出现在当前任务；可在设置中把 Code review 调整为单独任务。无论哪种模式，审查本身不会自动修改工作树；要求修复后仍会遵守现有 sandbox 和 approval。

流程四：用 Subagents 并行调查

在 App 任务中明确要求 Codex 把互不依赖的读密集工作交给 subagents，例如一个检查安全、一个找测试缺口、一个核对文档。App 会显示子代理 thread，主任务在它们完成后汇总结果。

```text
把这次审查分给三个 subagents：安全与权限、行为回归、测试缺口。
只读，不修改文件。等全部完成后去重，并按严重程度汇总带文件位置的发现。
```

打开每个 thread 检查证据，不只看主任务摘要。子代理继承当前 permission mode 并各自消耗 Token；多个 agent 同时写同一 Worktree 容易冲突，所以 App 中也应优先并行探索、测试与审查，写入任务要先划分互不重叠的文件范围。

流程五：配置 Local Environment

Local Environment 只在 ChatGPT 桌面 App 的 Codex 中使用，通过 App 设置配置，并保存到项目根目录的 `.codex` 文件夹；需要团队复用时可以把不含凭据的配置提交到仓库。

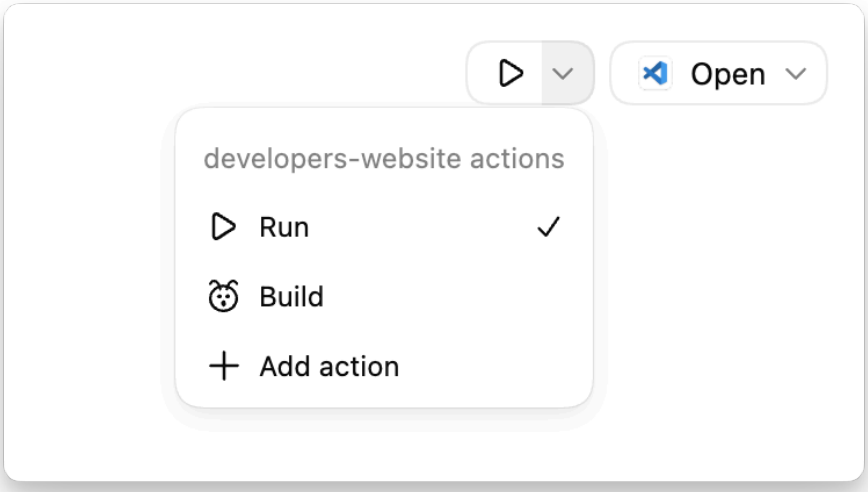
它包含两类行为，不能混为“每个 Local 任务都会自动运行”：

- **Setup scripts**：仅在 Codex 为新任务创建新 Worktree 时自动运行，用于安装依赖、生成代码或做初始构建。

- **Actions**：显示在 App 顶栏，由用户手动触发，在当前项目的集成终端中运行，例如启动开发服务或执行测试。

配置并运行一个 Action：

1. 确认桌面 App 当前选择的是 **Codex**，并打开项目根目录。
2. 打开 App 设置，为这个项目配置 Local Environment；配置文件会保存在项目根目录的 `.codex` 文件夹。
3. 在 **Actions** 中添加常用命令，例如把项目真实存在的启动命令配置为 `Run`，把测试命令配置为 `Test`。
4. 如果命令在 macOS、Windows 和 Linux 上不同，分别配置对应平台脚本；为 Action 选择容易辨认的图标。
5. 回到任务，从顶栏选择该 Action。它会在当前项目的集成终端中运行；检查命令、工作目录、输出和退出结果。



图：把重复命令做成可手动触发的 Local Environment Action。 OpenAI Codex App 官方文档当前界面；客户端版本未标注；来源：OpenAI 官方 Local environments；核对日期：2026-07-14；官方示例项目名与 Action；不含真实账号、API Key 或私人项目数据

设置原则：

1. setup 只做可重复、可审查的准备工作。
2. 不把长期密钥硬编码进脚本或文档。
3. 常用测试、开发服务和格式化命令做成明确 Action，需要时人工调用。
4. setup 失败时保留原始输出，不让任务在半配置状态继续修改。

终端和 Codex agent 的环境可能不同；出现“终端能运行、任务里找不到命令”时，检查 shell 初始化、PATH 和 Local Environment 设置。

流程六：创建 Scheduled 任务

Scheduled 适合周期性、结果可复核的工作，例如每天汇总失败测试、每周检查依赖公告或定期生成只读报告。

创建前先确认：

- 任务是否可以在无人即时监督时安全运行。
- 本地 Git 项目使用 Local 还是 Worktree，以及依赖是否可复现；不要把 Cloud 当成本地计划任务的运行位置。
- 运行时电脑是否保持开机、ChatGPT App 是否保持运行、项目路径是否仍可访问。
- 需要的网络、插件和账号权限是否最小化。
- 失败后输出保存在哪里，谁负责处理。
- 是否可能重复写入、重复通知或产生费用。

从 App 的 **Scheduled** 入口创建任务，写清频率、工作项目、目标、允许动作和成功标准。第一次先手动运行同一提示词并检查完整结果，再启用周期执行。

> WARNING
>
> 本地 Scheduled 在 Git 项目中只选择 **Local** 或独立 **Worktree**；非 Git 项目直接在项目目录运行。它会无人值守地使用默认 sandbox。组织允许时通常采用 `approval_policy = "never"`，超出 sandbox 的动作会失败，不会停在那里等待一次性人工审批。网页 Scheduled 可以使用上传内容、连接工具和插件，但不能直接访问电脑上的本地文件夹。
>
> 账号套餐、工作区策略和项目类型仍可能影响可用性。涉及发布、付款、删除或生产变更的任务不应只靠定时提示词控制。

搜索、恢复和管理任务

- `Cmd/Ctrl + G` 搜索历史任务；当前版本可能同时匹配任务内容和 Git 分支名。
- `Cmd/Ctrl + F` 只在当前任务内查找。
- 重要任务可以置顶；完成任务应归档，避免侧栏失去可读性。
- 已归档任务可在 **Settings > Archived tasks** 中恢复。
- 任务名称写结果而不是写工具，例如“修复登录重复提交”，比“Codex 测试”更容易搜索。

Appshots：把前台窗口交给任务

macOS 的 Appshots 可以把当前最前面的 App 窗口作为附件加入任务。适合展示错误、设置页、设计稿或预览状态。

1. 把需要分享的窗口放到最前面。
2. 按两个 Command 键，或使用你在设置中定义的 Appshots 快捷键。
3. 检查系统的 Screen Recording 和 Accessibility 授权请求。
4. 发送前确认画面和可读取文本没有邮箱、账号、密钥或私人数据。

Appshot 只应提供当前任务真正需要的范围。CLI 可以继续包含 Appshot 的历史任务，但不能创建新的 Appshot。

App 交付检查清单

- 当前交互任务使用了正确项目和 Local / Worktree / Cloud 环境；本地 Scheduled 只使用 Local / Worktree。
- 所有权限请求都能解释为什么需要。
- Review 中没有无关文件、密钥、生成物或调试代码。
- 测试命令在当前环境真实运行，退出结果被记录。
- Worktree 任务说明了分支、Handoff 和外部资源隔离状态。
- 没有把 stage、commit、push 或发布误当作“代码已完成”的必要自动步骤。

第 3 部分

CLI 使用

Codex CLI 使用

CLI 适合把工作目录、权限、输入和输出写得明确。它既能进行交互开发，也能用 `exec`、`review` 和结构化输出接入脚本。

当前实测版本：`codex-cli 0.144.3` · 模型基线 `gpt-5.6` · 核对日期 2026-07-15

安装、更新与诊断

使用 npm：

```bash
npm install -g @openai/codex
codex --version
```

macOS 使用 Homebrew：

```bash
brew install --cask codex
codex --version
```

更新前先确认你原来的安装方式；不要同时混用多个全局安装来源。

```bash
codex update
codex doctor --summary
```

`doctor` 检查安装、配置、认证和运行时状态；需要机器可读且脱敏的结果时使用 `codex doctor --json`。

登录与计费边界

交互式登录默认打开 ChatGPT 登录流程：

```bash
codex login
codex login status
```

无浏览器或远程终端可以使用 device auth。该登录方式当前为 Beta，流程可能继续调整；失败时回到交互式登录或按组织批准的认证方式处理：

```bash
codex login --device-auth
```

使用 OpenAI Platform API Key 时通过标准输入传递，不把 Key 写进 shell 历史：

```bash
printenv OPENAI_API_KEY | codex login --with-api-key
codex login status
```

- **ChatGPT 登录**：使用 ChatGPT 订阅与工作区能力，遵循对应的管理员、保留和数据策略。
- **API Key 登录**：按 OpenAI Platform API 标准用量计费，遵循 API 组织的数据设置；部分 ChatGPT 能力不可用。
- **Codex Cloud**：只支持 ChatGPT 登录。API Key 可以用于本地 App、CLI 和 IDE 工作，但不自动获得 Cloud、连接器或 ChatGPT 工作区能力。

共享电脑使用结束后运行 `codex logout`。不要在文档、截图、命令参数或仓库文件中出现 Key。

选择当前模型

交互会话中使用 `/model`；启动时可以用 `--model` / `-m` 一次性覆盖：

```bash
codex --model gpt-5.6
codex exec -m gpt-5.6 "审查当前未提交修改并列出证据"
```

`gpt-5.6` 是指向 `gpt-5.6-sol` 的当前别名。默认 Power 使用 Sol 和 Medium reasoning；日常任务可选 Terra，清晰且高吞吐的任务可选 Luna。先从默认推理强度开始，只有复杂任务确实需要更深规划时再提高。

使用 ChatGPT 登录时，`gpt-5.2` 和 `gpt-5.3-codex` 已弃用。API Key、自定义 `model_provider` 或第三方网关可能有不同模型清单；这时应使用该提供方实际公布并验证可用的 ID，不要假设别名映射完全相同。

第一次交互会话

从仓库根目录启动：

```bash
cd /path/to/project
codex
```

或从任何位置明确目录：

```bash
codex -C /path/to/project
```

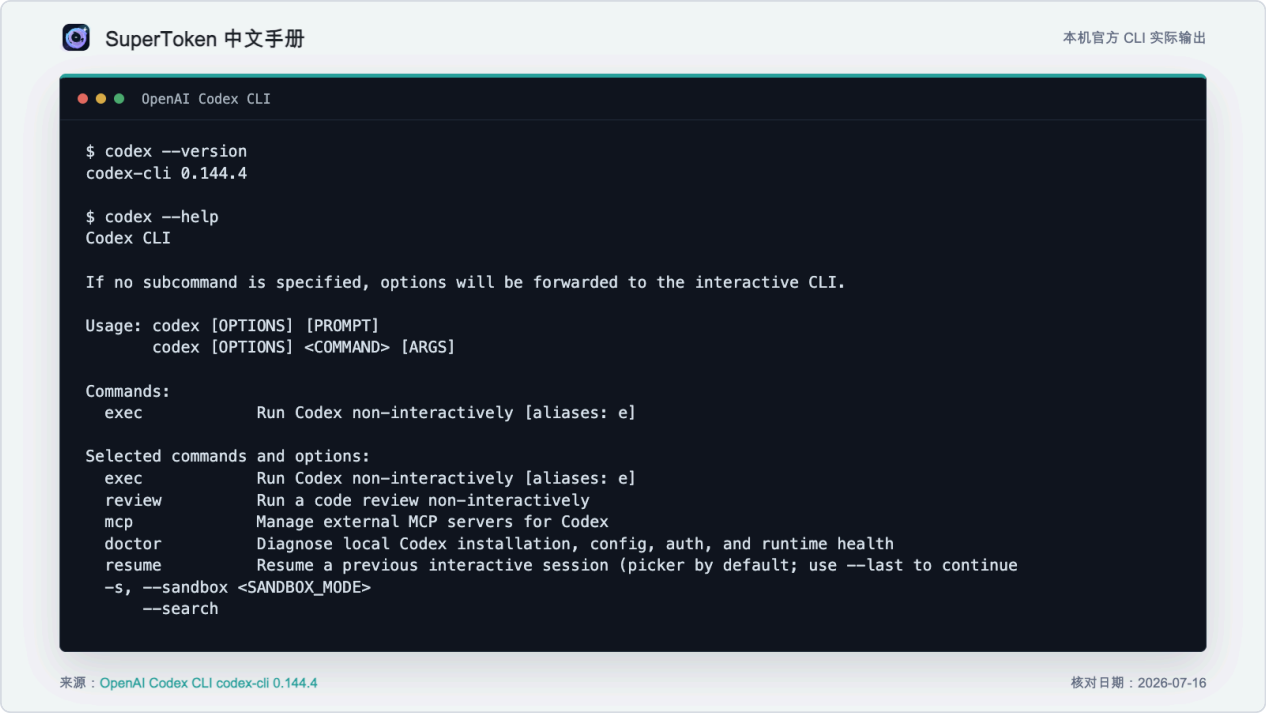
建议第一个任务只读：

```text
先不要修改文件。读取 AGENTS.md 和项目入口，报告：
- 当前目录、分支和未提交修改；
- 安装、测试、类型检查和构建命令的来源；
- 本任务最小需要访问的文件；
- 仍不能确认的前提。
```

正常结果应包含具体文件证据，没有新增 diff。若 Codex 把 README 当成唯一事实，要求它继续检查实际配置和测试文件。

当前命令地图

命令	适用场景
`codex`	启动交互会话
`codex exec`	非交互执行一次任务，可输出 JSONL 或结构化结果
`codex review`	非交互审查当前仓库的未提交、分支或提交 diff
`codex resume`	恢复已有会话；`--last` 继续最近一次
`codex fork`	从历史会话复制上下文，开始独立分支
`codex mcp`	管理 MCP 服务
`codex plugin`	管理插件
`codex app`	启动桌面 App；缺失时按当前 CLI 行为打开安装流程
`codex cloud`	浏览 Codex Cloud 任务并把改动应用到本地，当前 CLI 标为实验能力
`codex doctor`	检查安装、配置、认证和运行时健康状态



图：Codex CLI 版本与高频命令实测。 OpenAI Codex CLI 0.144.3；来源：SuperToken 本机官方 CLI 实际输出，非 OpenAI 宣传图；核对日期：2026-07-14；脱敏状态：不含账号与密钥

工作目录与附加目录

`-C` 决定主要工作根目录；`--add-dir` 会额外开放可写目录。

```bash
codex -C ./apps/web
codex -C ./apps/web --add-dir ./packages/shared
```

只在任务确实跨目录时增加范围。不要为了省一次审批直接开放仓库上级目录、整个用户目录或包含多个客户项目的位置。

Sandbox 与 Approval

两个设置解决不同问题：

- **sandbox**：命令可以访问哪些文件，以及是否能使用网络。
- **approval**：什么时候必须由你确认。

Sandbox	适合	风险
`read-only`	调研、架构梳理、审查	不能直接修改；某些工具仍会尝试写缓存
`workspace-write`	日常开发	可以写工作区，仓库范围仍需检查
`danger-full-access`	外部已经严格隔离的临时环境	可访问范围大，不适合普通本机默认值

Approval	行为
`untrusted`	只有受信任的读取类命令可直接运行，其他动作请求确认
`on-request`	Codex 按动作和上下文决定何时申请确认
`never`	不弹审批，失败直接返回给 Codex；不等于自动获得更高权限

一次性覆盖示例：

```bash
codex -s read-only -a untrusted
codex -s workspace-write -a on-request
```

`workspace-write` 下 spawned commands 默认不能联网；确需网络时可以在配置中明确开启：

```toml
sandbox_mode = "workspace-write"

[sandbox_workspace_write]
network_access = true
```

Web search 是另一套控制面：普通本地任务默认使用 OpenAI 维护的 cached 搜索结果；`codex --search` 或 `web_search = "live"` 开启实时搜索，并且不会为每次搜索单独弹审批。例外是 `--yolo` 或其他 full-access sandbox 设置，此时 Web search 默认改为 live；若仍要限制为缓存结果，必须显式设置 `web_search = "cached"`。不要把“命令不能联网”误解为“Web search 已关闭”，也不要为了联网直接切到 `danger-full-access`。

> **DANGER**
>
> `--dangerously-bypass-approvals-and-sandbox` 同时绕过审批和 sandbox。只有容器、虚拟机或专用 runner 已经隔离文件、网络与凭据时才考虑使用，不能写进普通开发机的别名或团队快速开始。

会话中的高频操作

输入 `/` 查看当前版本实际支持的命令。常用入口包括：

命令	作用
`/status`	检查模型、目录、权限和上下文状态
`/model`	选择当前可用模型和推理设置
`/permissions`	查看或调整当前权限策略
`/plan`	先调查和规划，再决定是否实现

命令	作用
`/diff`	查看工作区修改
`/review`	选择范围并启动专门代码审查
`/compact`	压缩长会话，保留关键事实和未完成项
`/mcp`	查看 MCP 服务和工具状态
`/init`	生成或完善项目 `AGENTS.md` 初稿
`/agent` / `/subagents`	查看并切换主任务和子代理线程

命令会随版本和已安装能力变化。手册列的是高频入口，不代替你本机的 `/` 菜单。

图片输入

启动时附加截图：

```bash
codex -i screenshot.png \
  "解释这个错误，先定位对应组件和日志，再提出最小修复"
```

对比两个状态：

```bash
codex --image before.png,after.png \
  "比较两个界面，只报告可验证的布局和交互回归"
```

发送前裁掉邮箱、账号、文件路径、浏览器标签、通知和 Key。图片能提供可见状态，但不能证明网络请求、数据库或不可见组件内部状态。

恢复、分叉和清理会话

```bash
codex resume
codex resume --last
codex fork
codex fork --last
codex archive SESSION_ID
codex unarchive SESSION_ID
```

- **resume** 继续同一任务，适合目标和仓库没有变化。
- **fork** 复制已有上下文后走另一条方案，避免两个方向混在一个历史里。
- **archive** 从活跃列表中归档会话但保留记录；**unarchive** 将它恢复。

- **delete** 永久删除会话记录及其后代会话。只有确认不再需要追溯时才运行 `codex delete SESSION_ID`；不要用 `--force` 做日常清理。
- 目标、仓库或信任边界已经改变时，直接开新会话通常更清晰。

恢复后先要求 Codex 重报当前目录、分支、diff 和未完成步骤，防止历史上下文与现实工作区脱节。

非交互执行

只读摘要：

```bash
codex exec -s read-only \
  "概括仓库入口、测试命令和三个主要风险；引用文件，不修改"
```

从标准输入追加日志：

```bash
npm test 2>&1 | codex exec -s read-only \
  "找出最早根因，区分实现、测试和环境问题"
```

输出事件流：

```bash
codex exec --json -s read-only \
  "分析项目并输出过程事件" > codex-events.jsonl
```

保存最终消息：

```bash
codex exec -s read-only \
  -o codex-summary.md \
  "输出架构摘要和风险，不修改文件"
```

需要稳定机器接口时，用 `--output-schema schema.json` 限定最终结果结构；脚本仍需校验退出码、JSON 解析和必需字段，不能只相信自然语言内容。

非交互代码审查

```bash
codex review --uncommitted
```

```bash
codex review --base main
codex review --commit COMMIT_SHA
codex review "只报告可验证的 Bug、安全问题和缺失测试，按严重程度排序"
```

`--uncommitted`、`--base`、`--commit` 和自定义 prompt 是互斥的审查范围。每个发现应包含文件位置、触发条件、影响和修复方向。高风险代码不要只把 `git diff` 管道给模型；应让审查在仓库内读取调用方、规则和测试。

自动化安全基线

非交互任务至少满足：

- 固定可信工作目录，并在运行前检查 Git 状态。
- 默认 `read-only`；确需修改时使用 `workspace-write` 和最小附加目录。
- 使用短期、最小权限凭据，日志和产物不得输出 Token。
- 限制网络目标和外部服务权限。
- 校验退出码、结构化输出和最终 diff。
- 不让任务自动提交、推送或发布，除非该外部 runner 已有独立审批和回滚机制。

第 4 部分

IDE 与 Cloud

Codex IDE 与 Cloud

IDE 让 Codex 靠近当前选区和文件；Cloud 让任务离开本机并行执行。两者最容易出错的地方不是提示词，而是上下文和运行环境并不相同。

核对日期 2026-07-15 · 依据 OpenAI 当前 IDE、Cloud environment 与 developer settings 文档

什么时候用 IDE

优先用 IDE 的场景：

- 解释当前选区或文件中的逻辑。
- 围绕一个组件、测试或错误快速迭代。
- 需要在编辑器中立即跳转、比较和人工修改。
- 想把选区或整个文件明确加入当前任务上下文。

改用桌面 App 的场景：

- 同时管理多个项目或后台任务。
- 需要完整 Review、集成终端、Worktree、Handoff 或 Scheduled。
- 任务跨越大量模块，更适合保持独立任务窗口。

IDE 第一次任务

1. 打开仓库而不是单个文件

在 VS Code 或兼容编辑器中打开仓库根目录，再打开 Codex 侧栏。项目级 `AGENTS.md`、`.codex/config.toml`、Git 状态和测试命令都依赖正确 workspace。

预期结果：Codex 侧栏关联当前项目；在 Git 仓库中可以使用 `/review`。

2. 明确加入上下文

通过命令面板可以：

- `chatgpt.addToThread`：把选中文本加入当前任务。
- `chatgpt.addFileToThread`：把整个文件加入当前任务。
- `chatgpt.newChat`：新建任务。
- `chatgpt.newCodexPanel`：打开新的 Codex 面板。
- `chatgpt.openCommandMenu`：打开 Codex 命令菜单。
- `chatgpt.openSidebar`：聚焦 Codex 侧栏。

选区只提供局部事实。涉及调用链、状态管理或接口契约时，要求 Codex 继续搜索定义、调用方和测试，不要让它只根据片段猜整个系统。

3. 先解释，再改动

```text
解释当前选中函数的输入、输出、副作用和调用方。
先不要修改。指出它与同目录测试之间的可验证契约，以及你仍缺少的上下文。
```

确认后：

```text
只修复刚确认的空值边界。保持公开 API、日志字段和错误类型不变。
先补回归测试，再做最小修改；完成后运行相关测试并审查完整 diff。
```

预期结果：改动集中在目标实现和测试；最终回答列出真实命令与结果。

IDE 设置分两层

层	管什么	放在哪里
Codex agent 设置	模型、推理、sandbox、approval、MCP、个性化	`config.toml` 和 Codex Settings

层	管什么	放在哪里
编辑器设置	侧栏启动、输入框行为、Review 交付、字体、Windows WSL	VS Code 的 `chatgpt.*` 设置

从 Codex 侧栏齿轮进入 **Codex Settings**；常用设置可以在面板中调整，完整配置通过 **Open config.toml** 编辑。

不要把 `chatgpt.*` 键写入 `config.toml`。它们属于编辑器设置系统。

高频编辑器设置：

设置	作用
`chatgpt.openOnStartup`	编辑器启动后是否聚焦 Codex 侧栏
`chatgpt.followUpQueueMode`	运行中发送消息时排队还是立即 steer 当前任务
`chatgpt.composerEnterBehavior`	Enter 与 Cmd/Ctrl+Enter 的发送行为
`chatgpt.reviewDelivery`	`/review` 在当前任务内运行，还是创建 detached 任务
`chatgpt.runCodexInWindowsSubsystemForLinux`	Windows 项目和工具链位于 WSL2 时让 Codex 在 WSL 运行

IDE、App 与子代理

当前官方资料明确了 IDE 的选区/文件上下文、IDE 中的 subagent activity，以及 App 自己的项目与任务界面，但没有承诺所有版本和账号都能在 IDE 与 App 之间自动共享每个活跃任务，也没有把 **IDE context** 作为通用跨端开关。不要依赖未在当前界面验证的自动续接。

使用原则：

- IDE 需要明确上下文时使用 `chatgpt.addToThread` 或 `chatgpt.addFileToThread`，并要求 Codex继续搜索调用方和测试。
- 从 IDE 切到 App，或从 App 切到 IDE 后，重新确认真实路径、Git HEAD、当前 diff 和任务完成标准。
- 同名项目的不同 checkout 或 Worktree 不能仅靠目录名判断。
- 适合并行探索、测试或审查时可以明确要求 subagents；IDE 出现 background-agent 面板时，可展开查看状态、停止活动线程或打开单个线程。
- 子代理继承父任务当前权限并额外消耗 Token；多个子代理不要同时改同一文件集合。

Cloud 与本机有什么不同

Cloud 任务在配置好的远程环境运行。它不是把当前电脑完整复制过去。

项目	Local / Worktree	Cloud
文件来源	当前本地 checkout 或托管 worktree	远程可用的仓库快照与任务输入

项目	Local / Worktree	Cloud
未提交文件	Local 可见；Worktree 按创建与 Handoff 规则处理	不应假设自动存在
依赖	使用本机环境或 Local Environment	依赖 setup 脚本和缓存
网络	受本机 sandbox、approval 和策略限制	setup 阶段可安装依赖；agent 阶段默认网络策略更严格，可按环境配置
凭据	来自本机已授权范围	必须单独、安全地配置，不能假设继承本机 shell
运行中的服务	可以连接本地端口	不能直接依赖你电脑上的进程

创建可复现的 Cloud Environment

实际入口与闭环：

1. 打开 Codex environments，确认目标仓库已有环境；没有时在这里创建。
2. 选择仓库并配置运行时版本、setup 脚本、可选 maintenance 脚本、环境变量、setup-only secrets 和 agent 网络范围，然后保存。环境配置变化后缓存会自动失效，也可以在环境页手动 **Reset cache**。
3. 在 Codex cloud 新任务中选择目标仓库与分支或提交基线，确认匹配到预期环境，再提交一个只读检查提示词。
4. 任务完成后先看回答和完整 diff。结果可继续追问、从 Cloud 打开 PR，或在本地目标仓库使用 `codex apply` 应用最近的 Cloud diff；应用前仍要核对分支、已有修改和冲突。

账号、工作区或仓库权限不具备 Cloud 时，环境入口或可选仓库可能不可见。这不是本地 API Key 能自动解锁的能力。

一个可靠环境至少定义：

1. **仓库与基线：** 任务从哪个仓库、分支或提交开始。
2. **Setup：** 安装依赖、生成代码和准备测试数据的可重复命令。
3. **运行时：** 语言、包管理器、系统依赖和必要环境变量。
4. **网络：** agent 阶段真正需要访问的域名和方法。
5. **验证：** 测试、lint、类型检查、构建和产物检查。
6. **凭据：** 区分普通环境变量与 setup-only secrets，不写进仓库和日志。

Cloud 的变量生命周期很容易配错：

类型	Setup 阶段	Agent 阶段	适用
Environment variable	可用	可用	agent 运行时确实需要的非敏感配置

类型	Setup 阶段	Agent 阶段	适用
Secret	可用	会被移除	仅安装依赖或 setup 下载私有资源时使用

Setup 在独立 Bash 会话中运行，因此仅在脚本里 `export NAME=value` 不会延续到 agent 阶段。需要贯穿任务的值应在 Environment settings 中配置；只给 setup 使用的凭据才放 Secrets。不要为了让 agent 读取凭据而把 secret 写入仓库、缓存产物或日志。

第一次使用 Cloud 时先跑只读任务：

```text
不要修改文件。报告当前提交、可用运行时、依赖安装状态、环境变量名称（不要输出值）、网络限制和可执行的验证命令。
```

预期结果：你能把远程环境与本机差异列出来，再决定是否适合实现。

Cloud 失败分支

现象	常见原因	下一步
本机通过，Cloud 找不到命令	setup 未安装依赖或 PATH 不同	从锁文件和项目脚本重建 setup，记录版本
找不到本机未提交文件	Cloud 没有当前 checkout 临时状态	提交到任务分支，或把必要补丁明确交给任务
下载依赖成功，任务阶段访问 API 失败	setup 与 agent 阶段网络策略不同	只允许必要域名；不要直接开放无限网络
测试连接到错误服务	远程环境变量或 endpoint 不同	输出变量名与目标环境，禁止打印 secret 值
Setup 能认证，Agent 阶段却是 401	凭据放在 Secrets，任务阶段已被移除	判断 agent 是否真的需要凭据；需要时改用受控环境变量和最小权限账号
Setup 中 `export` 后任务仍找不到变量	Setup 与 Agent 是独立 Bash 会话	在 Environment settings 配置，不依赖临时 shell export
结果无法在本机应用	基线变化、冲突或生成物差异	先检查 patch、基线提交和锁文件，再人工解决冲突

选择 Local、Worktree 还是 Cloud

- 需要当前本地服务、数据库、设备或未提交文件：选 **Local**。
- 想在同一台电脑隔离 Git 修改并行工作：选 **Worktree**。
- 任务可从清晰仓库基线开始、依赖可脚本化、适合远程并行：选 **Cloud**。
- 无法说清环境差异时，先不要上 Cloud；把本地流程做成可复现脚本后再迁移。

第 5 部分

项目与团队定制

Codex 项目与团队定制

定制的目标不是让 Codex“更自由”，而是让项目事实、允许动作和完成标准在不同任务与入口中保持一致。

核对日期 2026-07-15 · 适用于当前 Codex CLI、IDE 与桌面 App 的共享配置模型

先用最小表面

需求	应该放在哪里
只对当前任务生效	当前提示词或任务上下文
仓库长期命令、目录和完成标准	`AGENTS.md`
模型、sandbox、approval、MCP 等项目默认设置	`.codex/config.toml`
对 sandbox 外命令前缀做实验性 allow / prompt / forbidden	`.codex/rules/*.rules`
重复的专业工作流和参考资料	Skill
并行探索、测试或专项审查	Subagents / custom agents
可安装、可分发的 skills、MCP servers、apps/connectors、hooks 和展示资产集合	Plugin
连接外部实时数据或执行动作	MCP server / app connector
工具调用前后的审计、提示和后置检查	Hook

不要把所有约束复制到每一层。规则重复后很快会冲突，也会增加上下文和排错成本。

用 AGENTS.md 保存项目事实

仓库根目录示例：

```md
# AGENTS.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

## 修改约束
- 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、覆盖率或临时截图。
- Bug 修复不顺手升级依赖或重构无关模块。
- 金额、权限和时间逻辑先增加回归测试。

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

子目录可以放更具体的 `AGENTS.md`；更接近目标文件的规则作用于对应子树。个人临时覆盖不要污染团队共享文件。

维护原则：

- 只写仓库里可以验证、团队愿意长期维护的事实。
- 命令要对应真实脚本，避免“运行测试”这种无法执行的描述。
- 不写真实密钥、内部账号或私人偏好。
- 项目变化后同步更新，否则错误规则会稳定地产生错误结果。

配置层与作用域

Codex 的有效配置从高到低依次为：

1. CLI flags 与 `-c` / `--config` 一次性覆盖。
2. 受信任项目中的 `.codex/config.toml`，从项目根到当前目录逐层加载，离当前目录最近者优先。
3. `--profile NAME` 选择的 `$CODEX_HOME/NAME.config.toml`。
4. 用户 `~/.codex/config.toml`。
5. 系统 `/etc/codex/config.toml`（存在时）。
6. 内建默认值。

管理员 `requirements.toml` 是约束上限，例如禁止 `approval_policy = "never"`；它不是普通的第七层覆盖配置。项目未被信任时，项目 `.codex` 配置、Hooks 和 Rules 都会跳过。

一个保守示例：

```toml
model = "gpt-5.6"
approval_policy = "on-request"
sandbox_mode = "workspace-write"

[shell_environment_policy]
inherit = "core"
```

这是旧版 sandbox 配置路径，适合仍以 `sandbox_mode` 为基线的环境。不要把它与下文的 Beta permission profiles 混在同一套有效配置里。

OpenAI 当前默认示例使用 `gpt-5.6`；该别名在 OpenAI 模型页中指向 `gpt-5.6-sol`。自定义 `model_provider` 或第三方网关应改用提供方实际支持的 ID。提交项目配置前，确认模型对团队账号可用，也不要把个人路径、代理、Token 或组织内部地址带给所有成员。

启动时校验未知配置键：

```bash
codex --strict-config
```

权限设计：默认最小，按需放宽

把风险拆成四层：

1. **文件：** 读哪些目录、写哪些目录、是否碰受保护路径。
2. **命令：** 只读检查、构建测试、安装、删除、Git 写操作。
3. **网络：** 访问哪些域名、是否需要登录态或上传数据。
4. **外部系统：** GitHub、工单、云平台、数据库和发布权限。

推荐基线：

- 陌生仓库：`read-only` + `untrusted`。
- 熟悉仓库日常开发：`workspace-write` + `on-request`。
- CI：外部容器隔离 + 精确工具和网络策略，不用危险参数代替隔离。
- 生产、发布和删除：保留独立人工审批，不只依赖模型判断。

`approval_policy = "never"` 只表示不弹出确认；权限不足时动作会失败。它不自动升级 sandbox，也不代表任务更安全。

`workspace-write` 默认关闭 spawned commands 的网络。确需命令联网时只开启这一项：

```toml
[sandbox_workspace_write]
network_access = true
```

Web search 独立配置：普通本地任务默认是 cached，`web_search = "live"` 或 CLI `--search` 才是实时搜索；但 `--yolo` 或其他 full-access sandbox 设置会把默认值改为 live。命令网络、Web search、Browser/连接器权限必须分别检查。

Beta permission profiles：与旧 sandbox 二选一

Codex `0.138.0` 及以上提供 permission profiles，把文件和网络边界组合成可选择的最小权限策略。该能力当前为 Beta，配置结构可能变化。必须二选一：

- 旧路径：`sandbox_mode` 与 `[sandbox_workspace_write]`。
- 新路径：`default_permissions` 与 `[permissions.<name>]`。

只要任一已加载配置、CLI `--sandbox` 或所选 profile 仍设置 `sandbox_mode`，Codex 就使用旧路径，`default_permissions` 不会按预期生效。迁移前先从用户、项目和系统层删除旧 sandbox 设置，再使用类似配置：

```toml
default_permissions = "project-edit"

[permissions.project-edit.filesystem]
":minimal" = "read"

[permissions.project-edit.filesystem.":workspace_roots"]
"." = "write"
"**/*.env" = "deny"

[permissions.project-edit.network]
enabled = false
```

`deny` 可从较宽的 read/write 范围中排除敏感路径；它属于 permission profile 的 filesystem 规则，不是 Rules 的命令决策。配置后用 `/permissions` 确认实际选中的策略，并在无敏感数据的演示仓库验证读、写和网络边界。

Rules：控制可执行命令

Rules 当前仍是 Experimental，主要控制哪些命令前缀可以在 sandbox 外运行。文件放在活动配置层旁的 `rules/` 目录，例如个人 `~/.codex/rules/default.rules` 或受信任项目的 `.codex/rules/default.rules`。

```python
prefix_rule(
    pattern = ["gh", "pr", "view"],
    decision = "prompt",
    justification = "查看远程 PR 前需要确认",
    match = ["gh pr view 123"],
    not_match = ["gh issue view 123"],
)
```

决策只有 `allow`、`prompt`、`forbidden`；多条匹配时取最严格结果。修改后重启 Codex，并在真正放行前测试：

```bash
codex execpolicy check --pretty \
  --rules ~/.codex/rules/default.rules \
  -- gh pr view 123
```

设计步骤：

1. 从真实审批记录找高频、稳定、可重复的命令。
2. 使用足够具体的前缀，不允许过宽 shell 通配。
3. 对删除、发布、凭据、远程写入和未知脚本保留 `prompt` 或 `forbidden`。
4. 给规则加正反例并在更新后测试。

规则不能判断业务语义。即使 `npm test` 被允许，测试本身仍可能访问共享数据库；环境隔离和项目说明必须同时存在。

连接 MCP

添加远程 HTTP MCP：

```bash
codex mcp add sentry --url https://mcp.sentry.dev/mcp
codex mcp login sentry
codex mcp list
```

只有服务要求 OAuth 时才运行 `codex mcp login <name>`；需要限定授权范围时先看 `codex mcp login --help` 的 `--scopes`。使用 bearer token 的服务应从环境变量读取凭据，不把 Token 写进命令、配置示例或截图。

添加本地 stdio MCP：

```bash
codex mcp add my-server -- command-that-starts-the-server
codex mcp list
```

接入前检查：

- 服务发布者、包名和启动命令是否可信。
- 工具是只读查询还是可以创建、修改、删除外部数据。
- Token 存储在哪里，是否最小权限、可轮换、可撤销。
- 项目级配置是否会要求所有成员安装相同工具。
- 日志、错误和最终回答是否可能泄露外部数据。

验证不要只看“connected”：让 MCP 执行一次无副作用查询，并检查返回范围、来源和账号边界。

Skills：复用成熟流程

适合做 Skill 的信号：

- 同一流程已经稳定重复至少三次。
- 步骤、输入、产物和完成标准清楚。
- 需要附带参考资料、脚本或模板。
- 使用者需要一致的质量门槛，而不是一段万能提示词。

不适合：一次性任务、仍在频繁变化的探索流程、只需一条 `AGENTS.md` 规则的约束。

Skill 内容应说明何时触发、需要什么前提、按什么顺序工作、如何验证以及遇到什么情况停止。仓库级 Skill 的最小结构是 `.agents/skills/<name>/SKILL.md`，并且 frontmatter 必须包含 `name` 与 `description`：

```md
---
name: release-check
description: 核对发布前的构建、测试、产物和变更记录；只在准备发布或要求发布验收时使用。
---

1. 读取仓库发布说明和真实脚本。
2. 运行项目规定的最小发布检查。
3. 报告结果、失败和未验证项，不自动推送或发布。
```

Codex 会按 `description` 判断是否加载完整 Skill；描述要写清触发条件和边界。修改后通常会自动发现，未出现时重启客户端再检查。

Subagents：把独立工作移出主线程

当前 Codex 默认启用 multi-agent，桌面 App、CLI 和 IDE 都能显示子代理活动。适合把探索、测试、日志分析和不同审查维度拆成独立任务，再让主线程汇总。

```text
使用三个 subagents 并行检查当前分支：
1. 安全与权限；
2. 行为回归与测试缺口；
3. 可维护性。
全部完成后由主线程去重，按严重程度列出带文件位置的发现。不要修改文件。
```

- CLI 使用 `/agent` 或 `/subagents` 查看和切换线程。
- 子代理继承父任务当前 sandbox 和 permission mode；无交互环境中需要新审批的动作会失败。
- 每个线程都会使用模型、上下文和工具，Token 与资源消耗高于单代理。
- 读密集任务优先并行；写密集任务应按互不重叠的文件或模块分工，最终由一个主线程合并和验证。
- 需要稳定角色时，可在 `~/.codex/agents/` 或项目 `.codex/agents/` 创建 custom agent TOML；项目文件只在受信任仓库加载。

最小 custom agent 文件必须包含 `name`、`description` 和 `developer_instructions`。例如 `.codex/agents/reviewer.toml`：

```toml
name = "reviewer"
description = "只读审查当前改动中的行为回归、安全问题和测试缺口。"
developer_instructions = """
不要修改文件。按严重程度报告有证据的问题，并引用文件位置。
没有发现时明确说明剩余测试缺口。
"""
```

文件名只是约定，`name` 才是代理标识。custom agent 会作为 spawned session 的配置层继承父会话设置；官方明确提示该格式仍可能随作者体验成熟而演进。

Plugins：分发一组能力

Plugin 适合把 skills、MCP servers、apps/connectors、hooks 和展示资产打包分发。工具通过 MCP server 或 app/connector 暴露；当前官方结构没有可直接声明的一等 `commands` 或 `tools` manifest 字段。团队采用前检查：

- manifest、来源和许可证。
- 安装后新增的 Skills、MCP tools、connectors、Hooks 和网络连接。
- 更新和回滚方式。
- 每次启动增加的上下文或 Token 成本。
- 工作区管理员是否允许安装。

只有一个简单流程时先用 Skill；需要一组可安装能力和生命周期管理时再做 Plugin。最小 Plugin 至少包含 manifest 和一个实际能力：

```text
my-first-plugin/
├── .codex-plugin/
│   └── plugin.json
└── skills/
    └── hello/
        └── SKILL.md
```

`.codex-plugin/plugin.json`：

```json
{
  "name": "my-first-plugin",
  "version": "1.0.0",
  "description": "Reusable team workflow",
  "skills": "./skills/"
}
```

先把 Plugin 加入受信任 marketplace，再按 `codex plugin list` 显示的准确标识安装：

```bash
codex plugin marketplace add ./local-marketplace-root
codex plugin marketplace list
codex plugin list
codex plugin add my-first-plugin@MARKETPLACE_NAME
```

把 `MARKETPLACE_NAME` 替换为 `codex plugin marketplace list` 显示的实际名称。安装后启动新任务，确认 Skill 或 MCP tool 真正出现，再做最小无副作用验证。不要只以“安装成功”作为能力可用证据。

Hooks：审计与后置检查，不是写入阻断器

Hook 适合：

- 写入后运行格式化或静态检查。
- 在工具调用前记录或提示组织规则。
- 记录工具调用结果，供交付或合规检查。

当前 `PreToolUse` 不支持用 `continue: false` 或 `stopReason` 阻止工具；返回这些字段会让 Hook 标记失败，但工具调用继续。因此，密钥路径、受保护文件和危险命令的硬阻断必须交给旧 sandbox，或单独采用 permission profile 的 filesystem `deny`、管理员策略或 Rules，不能只靠 Hook，也不能把两套权限配置混用。

Hook 不适合替代业务测试，也不应悄悄修改大量文件。每个 Hook 都要有明确超时、错误输出、失败策略和跳过条件；后置 Hook 停止后续流程也不能撤销已经发生的写入。

启用来源不明的 Hook 前先审查脚本。危险的 `--dangerously-bypass-hook-trust` 只适合外部已经审查并隔离的自动化环境。

团队落地顺序

1. 先补准确的 `AGENTS.md` 和真实验证命令。
2. 使用保守的项目 `config.toml`，把个人设置留在用户层。
3. 从审批记录提炼少量 Rules。
4. 只连接必要的 MCP，并做最小权限验证。
5. 重复流程成熟后再做 Skill。
6. 多能力分发时再做 Plugin。
7. 只有审计或后置检查需要自动化时才加 Hook；硬阻断仍交给 sandbox、permissions、管理员策略或 Rules。
8. 每次变更都在 CLI、IDE 和 App 中各做一次冒烟检查。

定制故障隔离

出现异常时按层关闭：

1. 新任务中用最小提示词复现。

2. 检查 `AGENTS.md` 是否过时或作用域错误。
3. 用 `--strict-config` 检查配置键。
4. 暂停项目 MCP、Skill、Plugin 和 Hook，逐个恢复。
5. 运行 `codex doctor --summary`。
6. 比较 CLI、IDE 和 App 是否读取同一个项目与 `CODEX_HOME`。

一次只改变一层，才能知道根因。把所有定制同时删除会失去可复现证据，也可能误删用户原有配置。

第 6 部分

项目实战

Codex 项目实战

真正稳定的用法不是“给一句需求，等它写完”，而是让 Codex 按探索、计划、实现、验证、审查、交付的顺序闭环。

核对日期 2026-07-15 · CLI 示例实测于 `codex-cli 0.144.3` · 配套阅读：入口与快速开始 · 桌面 App · CLI

> **入口说明**
>
> 下面的提示词可以在 App、CLI 或 IDE 中使用。需要可视化 diff、集成终端和 Worktree 时优先用 App；需要脚本化输入输出时用 CLI；只围绕当前文件或选区时用 IDE。无论入口如何，验证命令和 Git diff 必须来自同一个实际工作区。

实战总流程

阶段	Codex 要做什么	你要确认什么
探索	阅读代码、文档、Git 状态和测试	它是否找对入口、没有凭空猜测
计划	拆分步骤、风险和验证方式	方案是否符合业务与边界
实现	做范围明确的修改	diff 是否集中、是否引入多余依赖
验证	跑测试、lint、类型检查或构建	命令是否真实执行、失败是否说明
审查	检查回归、安全和遗漏	结论是否按严重程度、有文件依据
交付	汇报结果与剩余风险	你能否快速复核并继续工作

复杂任务使用 `/plan` 先做前两步。计划得到确认后，再明确说“按刚才的计划实现并验证”。

实战一：快速读懂陌生项目

先用只读模式启动：

```bash
codex -s read-only
```

输入：

```text
先不要修改文件。请像接手项目的高级工程师一样阅读这个仓库。

请输出：
1. 项目目标、技术栈和启动入口；
2. 主要目录及职责；
3. 一次用户请求经过的关键调用链；
4. 安装、开发、测试、构建命令，注明依据文件；
5. 当前 Git 状态和明显风险；
6. 你无法从代码确认的问题。

不要只复述 README；所有结论尽量引用具体路径。
```

接着让它补一份项目规则草案：

```text
根据刚才确认的信息，起草一份简短 AGENTS.md。
只写能从仓库验证的命令和约束；不确定的内容放到“待团队确认”，不要编造。
```

人工核对后再保存。项目说明一旦写错，会让后续每个任务都稳定地走错方向。

实战二：修复一个可复现的 Bug

一个可靠的修复请求需要“现象、复现、范围、验证”：

```text
修复：用户连续点击两次“提交订单”时，偶尔会创建两条订单。

复现：
1. 启动测试环境；
2. 在订单确认页快速双击提交按钮；
3. 可以看到两次 POST /orders。

请按以下顺序工作：
1. 先阅读相关页面、请求封装和现有测试；
2. 复现并说明根因，不要先猜修复；
3. 添加能失败的回归测试；
4. 做最小修复，不修改后端接口；
5. 运行相关测试、类型检查和 lint；
6. 审查 diff，确认没有破坏键盘操作和错误重试；
7. 最后列出根因、修改文件、验证命令和结果。
```

如果它没能复现，不要让它“凭经验修”。改为要求：

```text
停在诊断阶段。列出已经验证的事实、无法复现的原因，以及还需要哪一条日志或环境信息。
不要修改生产代码。
```

实战三：开发一个完整小功能

以“订单列表增加状态筛选”为例。

第一步：计划

输入 `/plan`，然后发送：

```text
为订单列表增加状态筛选：全部、待支付、已支付、已取消。
筛选条件要同步到 URL，刷新和前进后退后保持一致。

请先调查：
- 当前列表的数据获取和路由参数方式；
- 是否已有可复用筛选组件；
- 相关测试如何组织；
- 移动端布局约束。

输出实施步骤、会修改的文件、测试方案和风险。
现在不要改代码。
```

你需要检查：它是否复用了现有模式、是否理解 URL 是状态来源、是否覆盖浏览器历史和空结果。

第二步：实现

```text
按已确认的计划实现。

约束：
- 不新增状态管理依赖；
- 不修改服务端接口；
- 查询参数使用 status；
- 非法状态值回退到“全部”；
- 保持现有桌面和移动端布局。

完成前运行相关单元测试、类型检查、lint 和构建。
如果测试环境本身失败，保留原始错误并区分“本次引入”和“已有问题”。
```

第三步：验收

```text
先不要继续改代码。请对照原始需求审查当前 diff：
- 是否遗漏刷新、前进后退、非法参数、空结果；
- 是否有无关修改；
- 测试是否真正覆盖行为而不是实现细节；
- 是否存在可访问性或移动端回归。

发现问题按严重程度列出；确认无问题后给出最终验收摘要。
```

实战四：安全地做重构

重构最容易变成“改了很多，但无法证明行为没变”。提示词要锁住外部行为：

```text
重构 src/services/payment.ts，目标是拆分支付渠道选择和请求发送逻辑。

外部行为必须保持不变：
- 导出的函数名和参数不变；
- 错误类型和错误码不变；
- 重试次数、超时和日志字段不变；
- 不升级依赖，不修改调用方。

先列出现有可观察行为和测试缺口。
必要时先添加特征测试，再分小步重构。
每一步都运行相关测试，最后比较 diff 并说明为什么行为等价。
```

如果当前测试不足，先让 Codex 写“特征测试”，把已有行为固定下来，再动结构。

实战五：处理测试失败

把真实输出通过标准输入交给 Codex，比口述“测试挂了”更准确：

```bash
npm test 2>&1 | codex exec --ephemeral \
  "分析失败测试，区分根因和连带失败。只给诊断和最小修复建议，不修改文件。"
```

交互会话中可以这样要求：

```text
运行与当前改动最相关的测试。
如果失败：
1. 保留完整的首个根因错误；
2. 判断是实现、测试、环境还是依赖问题；
3. 不要通过删除断言、跳过测试或扩大超时来掩盖问题；
4. 修复后重新运行同一命令，再跑更广的检查。
```

实战六：代码审查

审查未提交内容：

```bash
codex review --uncommitted
```

相对主分支审查：

```bash
codex review --base main
```

审查某个提交：

```bash
codex review --commit COMMIT_SHA
```

需要按基线审查：

```bash
codex review --base main
```

需要自定义审查标准时单独使用 prompt：

```bash
codex review "重点检查权限绕过、数据竞争和缺失测试；按严重程度输出，只报告可验证的问题"
```

`--uncommitted`、`--base`、`--commit` 与自定义 prompt 互斥，不能写在同一条命令中。

高质量审查结果应该包含具体文件位置、触发条件、影响和修复方向，而不是泛泛的“建议优化”。

实战七：前端页面验收

不要只让 Codex 看组件代码。要求它运行页面并检查真实状态：

```text
完成页面修改后启动本地开发服务器，并验证：
- 1440×900 桌面视口；
- 390×844 移动视口；
- 加载、空数据、错误、长文本和禁用状态；
- 键盘可操作性与焦点状态；
- 控制台错误和失败请求。

对页面截图做视觉检查，确认没有文字溢出、遮挡、布局跳动或横向滚动。
发现问题后修复并重新验证。
```

如果当前环境没有浏览器能力，Codex 应明确说明缺少哪项验证，而不是声称“页面正常”。

入口要分清：

- **ChatGPT 桌面 App**：安装或启用 bundled Browser plugin 后，在提示词中使用 `@Browser`。它使用独立浏览器环境；提交表单、发送信息等敏感动作仍应人工确认。
- **CLI / IDE**：没有相同的内置 Browser 入口。项目应接入 Playwright 命令、测试脚本或经过审查的 browser MCP，再要求 Codex运行并保存证据。
- **Chrome extension / Computer Use**：会接触登录态网站或桌面 App，权限和数据面更大。先确认 profile、站点范围、文件下载、外部写入和敏感操作确认，不要默认复用个人浏览器状态。

浏览器能看到真实页面，不等于自动证明后端数据、邮件、付款或第三方写入成功；关键外部状态必须用对应系统的只读查询再次验证。

实战八：非交互自动化

`codex exec` 适合脚本、CI 和批处理。它默认以只读沙箱运行，自动修改时要显式给出工作区写权限。

生成仓库风险摘要：

```bash
codex exec --ephemeral \
  "阅读仓库并列出五个最高风险区域。引用文件路径，不修改文件。"
```

输出 JSONL 供脚本消费：

```bash
codex exec --json "总结当前仓库结构，不修改文件" > codex-events.jsonl
```

只保存最终消息：

```bash
codex exec -o review.md \
  "审查当前未提交改动，按严重程度输出可验证的问题，不修改文件"
```

允许在当前工作区修复：

```bash
codex exec --sandbox workspace-write \
  "修复现有 lint 错误，只修改直接相关文件，并重新运行 lint"
```

> **自动化边界**
>
> 不要在包含不可信仓库代码的同一进程环境中暴露长期 API Key。CI 中优先使用官方 Codex GitHub Action、短期凭据和最小仓库权限，并把“生成补丁”和“推送或开 PR”拆成不同权限阶段。

让结果更稳定的复盘法

任务完成后，可以追加一轮：

```text
复盘这次任务：
1. 哪些信息最晚才发现，导致了返工；
2. 哪条项目约定应该写进 AGENTS.md；
3. 哪个验证步骤应该变成固定命令；
4. 有没有不应长期保留的临时配置。

只提出有本次证据支持的改进。
```

把反复出现的问题写入规则，把稳定的多步骤流程做成技能或脚本。不要把一次性的特殊要求永久化。

交付前检查清单

- 目标与边界是否仍和最初一致。
- `git diff` 中是否只有相关修改。
- 新行为是否有回归测试或可重复验证步骤。
- lint、类型检查、测试、构建是否真实执行。
- 失败项是否保留原始错误并说明影响。
- 是否意外记录了密钥、日志、缓存或构建产物。
- 最终摘要能否让另一位工程师快速复核。

第 7 部分

故障排除与速查

Codex 故障排除与速查

排错先保存原始症状，再按安装、工作区、配置、权限、网络和外部服务逐层缩小范围。不要用扩大权限掩盖配置错误。

核对日期 2026-07-15 · 当前实测 CLI `0.144.3`

先运行这组只读检查

```bash
codex --version
codex doctor --summary
git status --short --branch
pwd
```

需要保存脱敏诊断：

```bash
codex doctor --json > codex-doctor.json
```

分享前仍要人工检查路径、用户名、仓库名和环境信息；“redacted”不等于可以直接公开整份文件。

按症状查找

症状	先检查	不要先做
`codex` 找不到	`which codex`、安装来源、shell PATH、重开终端	同时用 npm 和 Homebrew 反复安装
登录或认证失败	`codex login`、账号类型、系统时间、代理和组织策略	把 Token 粘到聊天或截图里
打开了错误项目	App 项目路径、CLI `pwd`、`git rev-parse --show-toplevel`	用 `--add-dir` 开放更大范围来“找文件”
配置没有生效	`CODEX_HOME`、用户/项目配置层、`--strict-config`	同时修改所有层
权限弹窗太多	当前 sandbox、approval、Rules 和动作原因	直接切危险全权限

症状	先检查	不要先做
命令在终端能跑，Codex 里不能	shell、PATH、Local Environment、工作目录	把个人 shell 配置全部注入任务
网络请求失败	sandbox、approval、目标域名、代理、组织策略	开放无限网络或输出代理凭据
MCP 显示失败	`codex mcp list`、启动命令、认证、作用域、服务日志	重复添加同名服务
Review 不出现	当前产品是否为 Codex、项目是否是 Git 仓库	初始化 Git 前不看已有文件
Worktree 缺文件	tracked 状态、setup、`.gitignore`、`.worktreeinclude`	把所有忽略文件和密钥复制过去
Cloud 与本机结果不同	基线、setup、运行时、网络和环境变量名	假设 Cloud 是本机镜像

安装与 PATH

```bash
which codex
codex --version
npm list -g @openai/codex --depth=0
brew list --cask codex
```

只保留你实际选择的安装渠道。更新后版本仍旧时，检查 `which codex` 指向的路径是否来自另一个 Node 版本管理器或旧安装。

目录与 Git 状态

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

如果 `pwd` 在仓库子目录，而任务需要整个仓库，可以用 `-C` 明确根目录。只有确实跨仓库共享包时才使用 `--add-dir`。

发现已有修改时：

```text
不要修改。先区分当前 diff 中哪些改动在本次会话开始前就存在，哪些是你产生的。不要还原任何无法确认所有者的内容。
```

配置解析与作用域

```bash
codex --strict-config
codex -c 'approval_policy="on-request"' -s read-only
```

一次性覆盖能帮助判断问题来自配置还是程序本身。按真实优先级从高到低检查：CLI flags 与 `-c` > 受信任项目中离当前目录最近的 `.codex/config.toml` > 选中的 profile > 用户 `$CODEX_HOME/config.toml` > 系统配置 > 内建默认值。

管理员 `requirements.toml` 与工作区策略是约束，不是普通覆盖层；即使高优先级配置请求了某个值，管理员仍可禁止它。未信任项目不会加载项目 `.codex` 配置、Hooks 或 Rules。

不要在排错记录中贴完整配置；保留相关键并遮盖 endpoint、路径和凭据。

Approval 与 Sandbox

权限被拒绝时先读失败信息：是文件不在 writable root、网络被禁、命令需要审批，还是组织策略禁止。

只读复现：

```bash
codex -s read-only -a untrusted
```

日常开发基线：

```bash
codex -s workspace-write -a on-request
```

如果某动作仍失败，解释它为什么需要更大范围，并只增加对应目录或规则。不要把 `danger-full-access` 当诊断开关，因为它会同时改变太多变量并扩大风险。

网络与代理

检查顺序：

1. 任务是否真的需要网络。
2. 当前 surface 是 Local、Worktree 还是 Cloud。
3. sandbox 和 approval 是否允许网络动作。
4. 组织是否限制目标域名或开发者模式。
5. 代理是否只在交互 shell 中设置，App/IDE 是否读取到。
6. 目标服务自身是否需要登录、VPN 或额外权限。

本地 `workspace-write` 中 spawned commands 默认不能联网。确需命令网络时使用 `[sandbox_workspace_write] network_access = true`，不要直接改成全权限。Web search 是独立控制面：普通本地任务默认 `cached`，`--search` 或 `web_search = "live"` 才会取实时结果，而且不会为每次搜索单独请求审批；`--yolo` 或其他 full-access sandbox 设置则默认 live。Browser、apps/connectors 和 Cloud agent 网络也分别受自己的设置控制。

不要让 Codex 输出代理 URL 中的用户名、密码或 Token。能用允许域名解决时，不开放任意网络。

MCP

```bash
codex mcp list
codex mcp --help
```

逐项确认：

- 名称与作用域是否正确。
- stdio 启动命令能否在同一环境独立运行。
- HTTP endpoint 是否可达，认证是否过期。
- App/IDE 是否需要重启或新任务才能读取新配置。
- 工具是否被组织策略、插件状态或当前任务权限禁用。

先用无副作用查询验证连接，再尝试写操作。

IDE 与 App 上下文不同

1. 确认 App 和 IDE 打开的是同一真实路径，而不是同名 Worktree。
2. 确认两个 surface 使用预期登录、`CODEX_HOME` 和受信任项目。
3. IDE 通过 `chatgpt.addToThread` / `chatgpt.addFileToThread` 明确加入上下文；不要假设 App 自动得到 IDE 的选区、未保存内容或活跃任务。
4. 每次切换入口后重新报告路径、Git HEAD、diff 和完成标准。
5. 重启扩展或 App 后新建最小测试任务，判断问题来自历史任务还是配置层。

Worktree 与 Handoff

查看 Git worktree：

```bash
git worktree list
git status --short --branch
```

错误 `branch is already used by worktree` 表示同一分支已在另一个 worktree checkout。使用 App 的 Handoff，或给两个工作区使用不同分支；不要强行删除仍在使用的 worktree 元数据。

依赖缺失时优先修 Local Environment setup。忽略文件确需复制时只列最小 `.worktreeinclude`，并检查 secret 生命周期。

Cloud

要求任务先输出环境事实：

```text
不要修改。报告当前提交、操作系统、运行时版本、依赖状态、可用验证命令、环境变量名称和网络限制。不要输出 secret 值。
```

对比本机与 Cloud：基线提交、锁文件、setup 输出、运行时、网络、时区和外部服务 endpoint。先解决环境差异，再判断代码是否有问题。

会话开始跑偏

使用短指令恢复控制：

```text
停下，不要继续修改。把当前内容分为已验证事实、仍是推测、已产生的 diff 和下一步最小实验。
```

```text
回到原始完成标准。列出哪些修改超出范围；不要还原我原有的未提交改动。
```

```text
压缩会话时只保留：原始需求、已确认根因、修改文件、验证结果、失败结果和未完成步骤。
```

目标或仓库已经改变时开新任务，比不断压缩旧上下文更可靠。

命令速查

```bash
# 健康检查
codex doctor --summary
codex doctor --json

# 交互会话
codex -C /path/to/repo
codex -s read-only -a untrusted
codex resume --last
codex fork --last

# 非交互
codex exec -s read-only "只读分析，不修改"
codex exec --json -s read-only "输出事件流"
codex review --uncommitted
codex review --base main

# 扩展
codex mcp list
codex plugin --help
```

请求支持前准备

提供：

- Codex CLI 和 App/IDE 版本。
- 操作系统、入口和最小复现步骤。
- 脱敏后的完整错误文本。
- `codex doctor --summary` 或经人工检查的 JSON 相关部分。
- 当前目录类型、Git 状态和是否使用 Worktree/Cloud。
- 已尝试的最小配置覆盖，以及结果有什么变化。

不要提供：API Key、浏览器 cookies、完整 `.env`、私人会话、客户仓库源码或未脱敏日志。

附录

来源与版本

来源与版本

基于 OpenAI 官方文档与当前官方 CLI 整理，非 OpenAI 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. OpenAI Codex documentation
https://developers.openai.com/codex/
2. Codex environments
https://learn.chatgpt.com/docs/environments/modes
3. Desktop app commands
https://learn.chatgpt.com/docs/reference/commands
4. Code review
https://learn.chatgpt.com/docs/code-review
5. Integrated terminal
https://learn.chatgpt.com/docs/integrated-terminal
6. Worktrees and Handoff
https://learn.chatgpt.com/docs/environments/git-worktrees
7. IDE settings
https://learn.chatgpt.com/docs/developer-settings?surface=ide
8. Configuration
https://learn.chatgpt.com/docs/config-file/config-reference
9. Approvals and security
https://learn.chatgpt.com/docs/agent-approvals-security
10. MCP
https://learn.chatgpt.com/docs/extend/mcp
11. Models
https://learn.chatgpt.com/docs/models
12. CLI options
https://learn.chatgpt.com/docs/developer-commands?surface=cli
13. Models
https://learn.chatgpt.com/docs/models?surface=app
14. Local environments
https://learn.chatgpt.com/docs/environments/local-environment
15. Scheduled tasks
https://learn.chatgpt.com/docs/automations
16. Appshots
https://learn.chatgpt.com/docs/appshots
17. Subagents
https://learn.chatgpt.com/docs/agent-configuration/subagents
18. Models
https://learn.chatgpt.com/docs/models?surface=cli
19. Non-interactive mode
https://learn.chatgpt.com/docs/non-interactive-mode
20. Authentication
https://learn.chatgpt.com/docs/auth
21. Permissions
https://learn.chatgpt.com/docs/permissions
22. Web search
https://learn.chatgpt.com/docs/web-search
23. IDE commands
https://learn.chatgpt.com/docs/developer-commands?surface=ide
24. Cloud environments
https://learn.chatgpt.com/docs/environments/cloud-environment
25. Agent internet access
https://learn.chatgpt.com/docs/cloud/internet-access
26. AGENTS.md
https://learn.chatgpt.com/docs/agent-configuration/agents-md
27. Config basics
https://learn.chatgpt.com/docs/config-file/config-basic
28. Rules
https://learn.chatgpt.com/docs/agent-configuration/rules
29. Build skills
https://learn.chatgpt.com/docs/build-skills
30. Build plugins
https://learn.chatgpt.com/docs/build-plugins
31. Hooks
https://learn.chatgpt.com/docs/hooks
32. Codex 最佳实践
https://learn.chatgpt.com/guides/best-practices
33. Codex 提示词指南
https://learn.chatgpt.com/docs/prompting
34. Codex GitHub Action
https://learn.chatgpt.com/docs/github-action
35. Browser
https://learn.chatgpt.com/docs/browser
36. Computer Use
https://learn.chatgpt.com/docs/computer-use
37. Chrome extension
https://learn.chatgpt.com/docs/chrome-extension
38. Troubleshooting
https://learn.chatgpt.com/docs/reference/troubleshooting