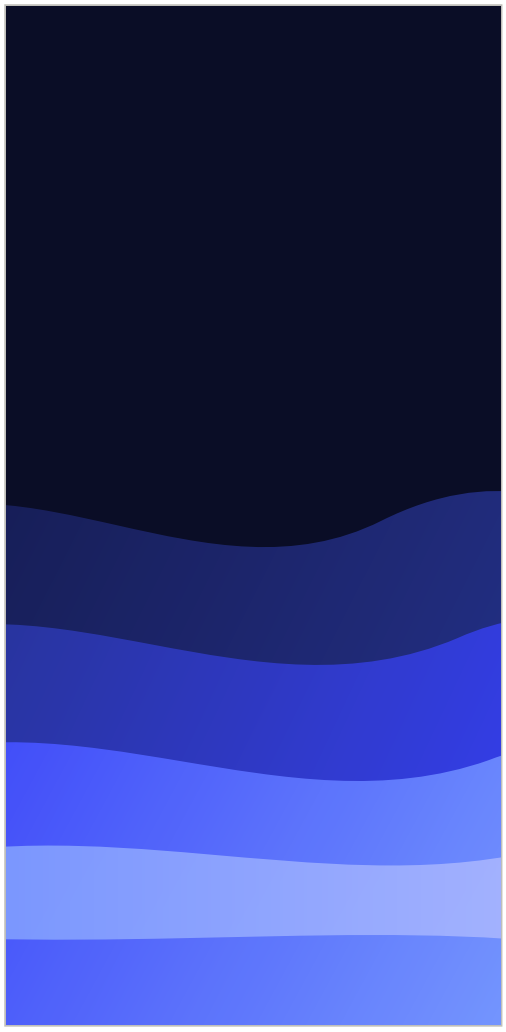
— 技术 · 实践

Codex

# 一个 agent，四面开工

「ChatGPT 里写代码的」只是它最小的入口。它其实是一个派到云端、四面开工、交回 PR 的异步编码 agent。



# 目录.

# 引子 · 不止 ChatGPT 里那个写代码的.

你付着 ChatGPT 的订阅,对 Codex 的用法大概停在:贴一个报错,看它改,复制走。但你的订阅里装着的,是一个四面开工的异步编码 agent —— 终端 CLI、IDE 扩展、ChatGPT 应用、云端沙箱连着同一台引擎;它最大的样子,是把一批活派到云端并行跑,回来是一摞待审的 PR。这一章讲清楚这本书的立场:量 Codex 的尺子不是「它能写什么代码」,而是「有多少活可以不守着做完」—— 以及为什么停在聊天框那层,等于把订阅里的后一半白扔。

---

你大概是这样用 Codex 的:在 ChatGPT 里切到它,贴一个报错、一段函数,看它给你改好的代码,复制走。偶尔它跑了个命令、开了个 PR,你觉得挺方便。一天下来,它对你来说是 ChatGPT 里一个更会写代码的模式 —— 比普通聊天多省下的,只是你自己落地那几步。这没错,但这是它最小的一层。你的 ChatGPT 订阅里本来就含着 Codex,和 ChatGPT 的其它 agentic 功能共用一个额度池[1] —— 也就是说,你从没为「会写代码的聊天框」单独付费,你付的钱里装着一台大得多的机器。这本书的立场一句话说完:Codex 最小的样子是 ChatGPT 里一段会改代码的对话,最大的样子,是一个**四面开工的异步编码 agent** —— 把一批活派到云端并行跑,回来是一摞待你审的 PR。停在聊天框那层,等于把订阅里的后一半白扔。

– I

## 一个 agent,四处开工.

Codex 文档给自己的一句话定位是「用 Codex CLI、IDE 扩展与云端自动化加速开发」。[2] 注意它没说「一个聊天功能」—— 它说的是四个面。同一个 agent,连着同一台引擎,在四处开工:

| 面 | 它在哪开工 | 哪章 |
| --- | --- | --- |
| CLI | 终端与脚本,进得了管道与 CI | 第 3、8 章 |
| IDE 扩展 | 编辑器里陪你改、盯每一步 diff | 第 2 章 |
| ChatGPT 应用 | 桌面、网页、手机,一个窗口切换本地与云 | 第 2、7 章 |
| 云端 | OpenAI 的隔离容器,批量并行、交回 PR | 第 7 章 |

2026 年 7 月 9 日起,原本独立的 Codex app 并入了 ChatGPT 桌面应用 —— Codex 保留自己的编码体验,和 Chat、Work 并排。[2] 四个面里最有分量的是云端:它的一句话定位是「把活委派给隔离的云端环境里的 Codex」,每个任务在 OpenAI 托管的容器里跑,检出你的仓库、装依赖、干活,收尾交回一份 diff,一键开 PR。[3] 这不是「在别的地方也能聊天」,这是**活离开你这台机器、在你不看着的时候完成**。

· · ·

— II

# 边界重画:能派多少,验得动多少.

> **Codex 最省的不是敲代码的时间,而是你守着 AI 干活的时间。**
>
> 本书的第一条判断

所以量它的尺子得换。旧尺子问「它能写什么代码」—— 按这把尺子,它和 Claude Code、Cursor、Copilot 挤在同一条跑道上。新尺子问「有多少活可以不守着做完」—— 按这把尺子,一次能派一批:五个 bug 各开一个云端任务,并行跑,你去干别的,回来挨个收;写代码只是它顺手的一类活。四面开工的意思,是同一件事在四个面各有最省的入口:要陪跑的留在编辑器,要批量的上云,要进脚

本的走 CLI。注意新尺子的后半句。派活是容易的一半;难的一半是确认它真做对了。官方最佳实践反复讲的不是「写更好的 prompt」,而是把 Codex 当队友:让它写测试、跑检查、审自己的 diff,再由你接受 —— 原话是「在 OpenAI,Codex 审 100% 的 PR」。[4] 活越是在你看不见的地方完成,验收就越是唯一能兜底的动作。这本书把「验收」单独设为第 9 章,也是全书的书眼:别的教程教你怎么派活,这本书还要你带着凭据把活收回来。[5]

> — 补充
>
> 立场照旧说在前面:我是付费用它的人,不是它的产品经理。官网自己能写的话,这里不写。Codex 变得很快 —— 模型名、命令、档位以周为单位在动,书里每个事实都标了核对日期(截至 2026-07-17),过期请以官方文档为准。它也不是所有活的最优解:哪些活该留给 Claude Code、Cursor 或 Copilot,第 10 章照实摆。

• • •

— III

## 先量一次你的活.

往下读之前,先给自己的活拍一张底片:

01

### 翻两周,挑 5 件你亲手陪着 AI 做完的活

不限写代码:改过的一个 bug、加过的一个小功能、清理过的一批依赖、写过的一份发布说明 —— 挑 5 件当时你一句句盯着、看它每一步的活。

02

### 给每件写一条「它能自己跑的检查」

想象把这件活派到云端、你不在场:你能不能写出一条测试、一条命令、一个可对比的产出,来判断它做没做对?写得出、写不出、拿不准,各自标上。

03

### 数「写得出」的有几件

那几件是你现在就能派出去、不用守着的活;「拿不准」的部分,是这本书接下来九章要补的。读完回来重标一遍 —— 这张表就是你的前后对照。

# 参考.


01 OpenAI 帮助中心 · Using Codex with your ChatGPT plan —— Codex 是帮你写、审、修、交付代码的 AI agent,包含在 ChatGPT 计划里(Free / Go / Plus / Pro / Business / Enterprise),各档额度不同;Codex、ChatGPT Work、以及其它 agentic 功能在可用时共用同一个 usage / credit 池。截至 2026-07-17。OpenAI · Codex pricing

02 OpenAI Codex 文档 · Quickstart / CLI —— Codex 的入口:ChatGPT 桌面应用与网页、Codex CLI(终端与脚本)、Codex IDE 扩展、Codex cloud;llms.txt 的一句话定位是「用 Codex CLI、IDE 扩展与云端自动化加速开发」。2026 年 7 月 9 日起,原独立的 Codex app 并入 ChatGPT 桌面应用(macOS / Windows),Codex 保留独立的编码体验。截至 2026-07-17。OpenAI · Codex quickstart

03 OpenAI Codex 文档 · Codex cloud —— 一句话定位是「把活委派给隔离的云端环境里的 Codex」。每个云端任务在 OpenAI 托管的隔离容器里跑:检出你的仓库、带网跑 setup 脚本、然后 agent 阶段默认断网;跑完交回一份 diff,你可以开 PR 或追问。截至 2026-07-17。OpenAI · Codex cloud

04 OpenAI Codex 文档 · Best practices —— 建议把 Codex 当成一个你逐步配置、逐步改进的队友,而不是一次性助手;可靠性来自测试与复审:让它写 / 更新测试、跑该跑的检查、确认结果、审 diff 里的 bug 与回归,再由你接受。原话:「在 OpenAI,Codex 审 100% 的 PR。」截至 2026-07-17。OpenAI · Codex best practices

05 OpenAI Codex 文档 · Code review —— 用 ChatGPT 或 Codex 在提交 / 推送前审代码;本地 /review 起一个专门的 reviewer 读选定的 diff、报出按优先级排序的发现,不改你的工作区;GitHub 上 @codex review 只标 P0 / P1 高优先级问题,可开自动审查让每个 PR 都过一遍。截至 2026-07-17。OpenAI · Code review


---

**聊天框是它最小的样子,**
**四面开工才是最大的.**

# 四面 · 一个 agent,四处开工.

CLI 在终端和脚本里,IDE 扩展在编辑器里,ChatGPT 应用在桌面、网页和手机上,云端在 OpenAI 的容器里 —— 四个面,一台引擎:config.toml、AGENTS.md、skills、MCP 走到哪跟到哪。2026 年 7 月起,Codex app 并入 ChatGPT 桌面应用,本地、worktree、云端在同一个窗口里切换,Handoff 把一场对话连代码一起搬来搬去。这一章把四面各自的主场讲清楚:活跟人走 —— 要陪跑的去编辑器,要批量的上云,要进脚本的走 CLI。

---

第 1 章把 Codex 说成「四面开工的 agent」。这一章把四个面各自的主场讲清楚,再说清一件容易被忽略的事:它们不是四个各管各的工具,是同一台引擎的四个入口 —— 你写的一份 AGENTS.md、配的一次 config、接的一个 MCP,四面通用。选对面,不是学四套东西,是同一件活挑最省的那个入口。

— I

## 四个面,各有主场.

官方的分法很干脆:日常用 ChatGPT 就开桌面应用或网页;想在终端或编辑器里用 Codex,就装 CLI 或 IDE 扩展。[1] 但「用哪个」不该按身份分,该按**这件活是什么形状**分。四个面各自最擅长的:

| 面 | 它最擅长 | 代价 / 边界 |
| --- | --- | --- |
| IDE 扩展 | 陪你改,盯每一步 diff | 你得在场 |
| CLI | 进脚本、进管道、进 CI | 要自己搭上下游 |
| ChatGPT 应用 | 桌面 / 网页 / 手机,一处切本地与云 | 重活仍要挑对环境 |
| 云端 | 批量、并行、不用你守着 | 要先配环境,回来才是 PR |

2026 年 7 月起 ChatGPT 应用这一面变重了:原本独立的 Codex app 并进了 ChatGPT 桌面应用,Codex 保留自己的编码体验,和 Chat、Work 并排;新版带 diff 内联编辑、侧栏里的 PR 复审、多仓库项目。[2] 手机也在这一面里 —— ChatGPT 应用的 Remote 能连回你桌面上的 host,在路上派活、看 diff、留行内评论。[2]

• • •

– II

## 一台引擎:配置走到哪跟到哪.

四个面驱动的是同一台引擎的同一套定制,这是最省心也最容易被忽略的一点。定制分几层协同:`AGENTS.md` 写持久的项目指令(第 5 章),`config.toml` 写本地默认 —— 模型、沙箱、审批、MCP,skills 打包可复用的流程,MCP 接外部工具。[3] 其中 `config.toml` 这套配置层**在 CLI 与 IDE 扩展间共用** —— 你在 CLI 里配好的一个 MCP 服务器,切到编辑器里照样在,配好一次不用重做。[3] 这条「一台引擎」的性质,决定了这本书的读法:后面讲 AGENTS.md(第 5 章)、MCP(第 6 章)、沙箱与审批(第 4 章),讲的都是引擎级的东西,四个面同时受益。你不是在学四个工具,是在配一台机器,然后决定从哪个面驱动它。

— III

# 本地、worktree、云端:一个窗口,三种去处.

在 ChatGPT 桌面应用里开一个 Codex 对话,先选它在哪跑:**Local** 直接在你当前项目目录里改;**Worktree** 在一个 Git worktree 里隔离改动;**Cloud** 在配好的云端环境里远程跑 —— Local 和 Worktree 都在你这台电脑上。[4] 这三种去处是「四面」在一个窗口里的落地:同一个 agent,你决定它的活留在前台、挪到后台、还是送上云。worktree 是这里的关键零件:它基于 git worktree,让 Codex 在同一个项目里并行跑多个互不干扰的对话 —— 各自一份检出,共享 .git 元数据,所以你能同时铺开好几件活而不打架。[5] 搬动用 Handoff:把一场对话连同它的 Git 状态在 Local 与 Worktree 之间安全移动,底层那些容易出错的 Git 操作由 Codex 替你处理。[5] 前台想专心时把活挪进 worktree 后台跑,要亲手验时再 Handoff 回 Local —— 这正是「不守着做完」在本地的样子,云端只是把它推到极限(第 7 章)。

> — 实践提示
>
> 一个实用默认:要陪着看、要立刻验的活,留在 IDE 扩展或 Local;方向可信、能自动验的活,挪进 worktree 或送上云。别用一个面硬扛所有活 —— 编辑器里跑一夜的批处理,和云端里逐字盯的小改,都是把活放错了面。

动手 · 把同一件活在两个面上各跑一次:

- [ ] 挑一件中等改动,先在编辑器 / Local 里陪着跑一遍,感受「盯每一步」的成本。
- [ ] 同一类活,下次开一个 worktree 让它后台跑,你去干别的,回来只看 diff —— 对比你省下的时间。
- [ ] 确认一次:你在一个面里配的 AGENTS.md 或 MCP,切到另一个面还在。共用一台引擎不是口号。

# 参考.


01 OpenAI Codex 文档 · Quickstart —— 用 ChatGPT 的日常界面选桌面应用或网页;开发者想在终端或编辑器里用 Codex,则用 Codex CLI 或 Codex IDE 扩展。四个入口连着同一台引擎。截至 2026-07-17。OpenAI · Codex quickstart

02 OpenAI Codex 文档 · What's new(2026-07-06–10)—— 7 月 9 日 Codex app 并入 ChatGPT 桌面应用(macOS / Windows),Codex 保留独立的编码体验,和 Chat、Work 并排;新增 diff 内联编辑、侧栏里的 PR 复审、由 GPT-5.6 驱动的更快 Computer Use、多仓库项目。桌面版对所有 ChatGPT 档位开放,含 Free。截至 2026-07-17。OpenAI · Codex what's new

03 OpenAI Codex 文档 · Config basics / Customization —— Codex 的定制来自几层协同:AGENTS.md 是持久的项目指令、config.toml 是本地默认(模型、沙箱、审批、MCP)、skills 是可复用工作流、MCP 接外部工具。config-basic 原话:「The CLI and IDE extension share the same configuration layers.」配好后在两端之间切换无需重做。截至 2026-07-17。OpenAI · Config basics

04 OpenAI Codex 文档 · Codex environments —— 在 ChatGPT 桌面应用里开始一个 Codex 对话时选它在哪跑:Local(直接在你当前项目目录)、Worktree(在 Git worktree 里隔离改动)、Cloud(在配置好的云端环境里远程跑);Local 与 Worktree 都在你的电脑上运行。截至 2026-07-17。OpenAI · Codex environments

05 OpenAI Codex 文档 · Worktrees —— worktree 让 Codex 在同一个项目里并行跑多个互不干扰的对话(基于 git worktree,各自一份检出、共享 .git 元数据);Handoff 把一场对话连同 Git 状态在 Local 与 Worktree 之间安全搬移,由 Codex 处理底层的 Git 操作。仅在 ChatGPT 桌面应用的 Codex 里可用。截至 2026-07-17。OpenAI · Worktrees


---

## 不是四个工具,
## 是一台引擎的四个入口.

# 上手 · 装 CLI,跑通第一个本地循环.

一条 npm 或 brew 命令装上,codex login 用 ChatGPT 账号登录 —— 订阅额度直接用,不用碰 API key。然后跑通第一个循环:用「目标、上下文、约束、完成标准」四件套说清一件活,先 /plan 看它的方案,放它动手,/diff 看结果。中途想改主意不用等:回车是当场纠偏,Tab 是排队下一轮。这一章还有会话的日常:/status 看配置、/compact 压上下文、/fork 岔开试另一条路、/side 问句闲话不弄脏正题。

---

四个面里,CLI 是理解 Codex 最快的入口 —— 装一条命令、登一次录、跑通一个循环,你就摸到了引擎本身,而不是某个界面。这一章带你走完这三步:用 ChatGPT 账号登录(不用碰 API key),用「目标、上下文、约束、完成标准」四件套说清第一件活,先看它的计划再放它动手;末了是会话的日常手感 —— 中途改主意、压上下文、岔开试另一条路。

— I

## 装上、登录:两条路,先走订阅这条.

CLI 的一句话定位是「跑在你终端里的轻量编码 agent」,一条命令装上: 1

```
npm install -g @openai/codex      # 或
brew install --cask codex         # 或
curl -fsSL https://chatgpt.com/codex/install.sh | sh
```

三选一;之后 CODEX 启动交互界面

登录有两条路,选错会多花冤枉钱。用 `codex login` 走浏览器流程、以 ChatGPT 账号登录 —— 直接吃你的订阅额度,这是已经付着 ChatGPT 的人该走的路。另一条是 API key(把 key 管道给 `codex login --with-api-key`):按 token 计费、走标准 API 价,且**不含云端功能**(GitHub 审查、Slack 那些),它是为 CI 这类自动化环境准备的(第 8 章),不是日常。[2] 还有一条硬边界记牢:**Codex cloud 只认 ChatGPT 登录** —— 想用第 7 章的云端,API key 不行。[2]

> — 注意
>
> 凭据缓存在 `~/.codex/auth.json`,里面是访问令牌 —— 当密码看待:别提交进仓库、别贴进工单、别在聊天里发。想更稳,让 Codex 把凭据存进系统钥匙串而不是明文文件。[2]

• • •

— II

# 第一个循环:说清目标,先看计划,再放它动手.

Codex 就算你 prompt 不完美也能干活,但在大仓库、高风险的活上,给它对的上下文和清晰的结构,结果稳得多。官方推荐的 prompt 有四件套:**目标**(要改 / 建什么)、**上下文**(哪些文件、文档、报错相关 —— 可以 @ 提及)、**约束**(要守的规范、架构、安全线)、**完成标准**(什么条件下算完:测试通过、行为变了、bug 不再复现)。[3] 这四件里,完成标准最容易漏,却最要紧 —— 它是第 9 章验收的基线。

> 提示词 — 第一件活的四件套模板
>
> 目标:`<一句话说清要改什么、达到什么效果>`。
> 上下文:相关的是 `<目录 / 文件 / 报错>`,可以 @ 提及。
> 约束:沿用现有风格;不要碰 `<不许动的目录 / 依赖>`;不要顺手重构无关代码。
> 完成标准:`<一条能检验的标准 —— 做完要能通过>`。
> 拿不准的地方列成问题问我,别自己猜。

> 目标:给用户导出加一个 CSV 选项,默认仍是 JSON。
> 上下文:相关的是 src/export/、src/api/export.ts,可以 @ 提及。
> 约束:沿用现有风格;不要碰鉴权和数据库层;不要顺手重构无关代码。
> 完成标准:CSV 导出的用例全部通过,已有 JSON 导出测试仍然全绿。
> 拿不准的地方列成问题问我,别自己猜。

复杂或模糊的活,先让它计划:/plan 或 Shift+Tab 进入计划模式,它会先收集上下文、把打算怎么做讲给你听,你批了它才动手。[3] 动手之后,/diff 随时看它改了什么(连未跟踪的新文件也在)。[5] 这就是最小的循环:说清目标 → 看计划 → 放它动手 → 看 diff。它是第 1 章那个「派活 + 验收」的单机版,后面所有章节都在放大它。

· · ·

— III

## 中途改主意,和会话的日常手感.

它正在跑,你想到一句要补的,不用等它停:回车是 **steer** —— 把你这句并进当前这轮,改方向、补细节都行;Tab 是 **queue** —— 把这句留到下一轮再处理。[4] 这两个键区分了「现在就拐」和「这轮先跑完」,是长任务里最省心的两个动作:方向对就 queue、方向偏就 steer,不用打断重来。会话的其余日常靠斜杠命令,先认几个每天都用的:/status 看当前配置和 token 用量(撞额度前先看这个);/compact 把跑长了的对话压成摘要、腾出上下文,留给任务之间的自然断点用;/fork 从当前对话岔出一条新的、去试另一种方案而不丢原路;/side(或 /btw)开一个临时侧聊,问句闲话、查个小事,不弄脏正题的记录。[5] 这些不是花活,是让一场长会话不至于越跑越乱的卫生习惯。动手 · 半小时跑通第一个循环:

01

### 装上,用 ChatGPT 账号登录

三条安装命令挑一条,codex login 走浏览器流程。用 /status 确认登的是订阅、不是 API key —— 别让第一件活就走了按量计费。

02

### 用四件套派一件真活,先 /plan

挑一件跨两三个文件的中等改动,照上面的模板写。先 /plan 看它的方案,至少改一处再放它动手 —— 感受「审方案」和「审代码」的差别。

03

### 故意 steer 一次,再 /diff 收尾

跑到一半回车补一句「顺便把这个也改了」,看它当场并进去;完事 /diff 通读它改的每一处。这套手感,就是你之后在四个面上派活的底子。

# 参考.


01 openai/codex 仓库(GitHub)—— Codex CLI 的一句话定位是「跑在你终端里的轻量编码 agent」,主体 Rust 写成;安装:npm install -g @openai/codex,或 brew install --cask codex,或 curl 官方安装脚本。仓库同时指路:要在 VS Code / Cursor / Windsurf 里用,装 IDE 扩展;要云端 agent,去 chatgpt.com/codex。截至 2026-07-17。GitHub · openai/codex

02 OpenAI Codex 文档 · Authentication —— 两种登录:用 ChatGPT 账号登录走订阅额度,用 API key 登录走按量计费。本地三面(桌面应用 / CLI / IDE)两种都支持;Codex cloud 必须用 ChatGPT 登录。CLI 用 codex login(浏览器流程),或把 key 通过 stdin 管道给 codex login --with-api-key;凭据缓存在 ~/.codex/auth.json(当密码看待)或系统钥匙串。API key 走标准 API 价、不含云端功能(GitHub 审查、Slack 等)。截至 2026-07-17。OpenAI · Authentication

03 OpenAI Codex 文档 · Best practices —— 大 / 复杂仓库里,最大的提升是给 Codex 对的上下文加清晰的结构。推荐 prompt 含四件:Goal(要改 / 建什么)、Context(哪些文件 / 文档 / 报错相关,可 @ 提及)、Constraints(要遵守的规范、架构、安全要求)、Done when(什么条件下算完 —— 测试通过、行为变了、bug 不再复现)。复杂或模糊的任务,先让它计划:/plan 或 Shift+Tab。截至 2026-07-17。OpenAI · Best practices

04 OpenAI Codex 文档 · Prompting —— Codex 正在干活时,你不用等它跑完就能再发一条:Steer 把消息并进当前这轮(改方向、补细节),Queue 把消息留给下一轮。Codex CLI 里,干活时按回车 steer 当前轮,按 Tab 把消息 queue 到下一轮。截至 2026-07-17。OpenAI · Prompting

05 OpenAI Codex 文档 · Slash commands(CLI)—— 键盘优先的会话控制:/plan 切计划模式、/diff 看含未跟踪文件的 Git diff、/status 看会话配置与 token 用量、/compact 压缩可见对话腾出 token、/fork 从当前对话岔出一条新对话、/side(/btw)开一个不弄脏正题的临时侧聊、/model 换模型、/review 审工作区、/init 生成 AGENTS.md、/usage 看用量。截至 2026-07-17。OpenAI · Slash commands

装一条命令,登一次录,
跑通一个循环,你就懂了它.

# 放权 · 沙箱、审批与规则.

Codex 的安全模型是两个旋钮:沙箱定它「技术上能碰什么」(read-only / workspace-write / danger-full-access),审批策略定它「什么时候要问你」(untrusted / on-request / never)。默认的 Auto 档 —— 工作区随便写、越界才问 —— 加上默认断网和 .git 等保护路径,是多数活的甜点位。往细处走有规则文件:用 prefix_rule 给命令前缀定白灰黑名单,连 bash -lc 里藏的复合命令都会被拆开逐条审。最后一层最有意思:把审批本身交给一个 reviewer agent —— auto_review 替你看每个越界请求,拦下外传数据、探凭据、破坏性动作。

---

放它上云、放它无人值守之前,得先搞清楚它「能碰什么、什么时候要问你」。如果你记忆里 Codex 的放权是「suggest / auto-edit / full-auto」三档,那已经过时了 —— 那套旧命名如今只剩 full-auto 作为一个弃用兼容 flag 残留在 codex exec 里(会打印告警,官方让你改用显式的 --sandbox)。现在的安全模型是两个正交的旋钮:沙箱定它技术上能做什么,审批定它什么时候必须停下来问 —— 再加上一层规则,和一个能替你把关的 reviewer agent。这一章讲怎么把这套调到「信方向就松、信不过的点钉死」。

— I

## 两个旋钮:能做什么,要不要问.

Codex 的安全控制来自两层,分开转:**沙箱模式**定它技术上能做什么 —— 在哪写、能不能联网;**审批策略**定它什么时候必须先问你。[1] 两个旋钮各三档:

| 旋钮 | 从紧到松 |
| --- | --- |
| 沙箱 | `read-only · workspace-write · danger-full-access` |
| 审批 | `untrusted · on-request · never` |

沙箱作用的是它派生的每一个命令 —— git、包管理器、测试器,统统继承同一条边界,靠操作系统原生强制(macOS 的 Seatbelt、Linux / WSL2 的 bubblewrap、Windows 的原生沙箱)。[2] 这一点很重要:沙箱不是「模型答应不越界」,是「操作系统不让它越界」—— 你信的是强制的边界,不是它的意图。两个旋钮组合出你要的自由度:read-only + on-request 是最安全的调研档,danger-full-access + never 是全放行(只在隔离环境用)。

• • •

– II

## 默认与甜点位:Auto 档.

多数活的甜点位是 **Auto 预设**:`workspace-write` 加 `on-request` —— 工作目录里随它读、改、跑命令,只有越界(工作区外写、要联网)才停下问你。[1] Codex 启动时还会替你猜:目录在版本控制里就推荐 Auto,不在就推荐 read-only —— 因为有 git 兜底的地方,试错成本低。[2] 想临时收紧,`/permissions` 一键切 read-only,只聊不改。[1] Auto 档有两条默认的硬边界,值得记牢。其一:**workspace-write 下网络默认是关的** —— 要放开得显式在 config 里写 `network_access = true`,断网是默认,不是意外。[1] 其二:就算在可写的工作区里,也有几条**保护路径**:`.git`、`.agents`、`.codex` 递归只读 —— 它能改你的源码,但改不了版本库元数据和它自己的配置。[1] 这两条是「放权但不失控」的地基:你放开的是工作区,不是整台机器。

• • •

— III

# 规则:给命令前缀定白灰黑名单.

旋钮是粗调,规则是细调。规则(实验特性)控制 Codex 能在沙箱外跑哪些命令:一个 `.rules` 文件里用 `prefix_rule` 给命令前缀定裁决 —— `allow` 直接放行、`prompt` 每次问、`forbidden` 直接拦;多条命中,最严的赢。[3]

~/.codex/rules/default.rules

```
prefix_rule(
    pattern = ["gh", "pr", "view"],
    decision = "prompt",
    justification = "看 PR 可以,但要我点头",
    match = ["gh pr view 7888", "gh pr view --repo openai/codex"],
    not_match = ["gh pr merge 7888"],
)
```

一条规则:GH PR VIEW 允许但每次问,带内联测试

规则最要紧的一条性质,是它拆得开复合命令。有人把危险动作藏在安全命令后面 —— `git add . && rm -rf /`。只要这串命令是普通词加安全操作符,Codex 就用 tree-sitter 把它拆成两条逐个判:就算你允许了 `git add`,后面那条 `rm -rf /` 会被单独评估,整条命令因此不会被自动放行。[3] 写完规则用 `codex execpolicy check` 拿具体命令测一测,别等它到线上才发现规则写漏了。[3]

• • •

— IV

# 把审批本身交给一个 reviewer.

最后一层最有意思:审批不一定要你亲自点。`approvals_reviewer` 默认是 `user`(问你),改成 `auto_review`,越界请求就先送给一个独立的 reviewer agent 判 —— 它只看那些本来就要停下问人的动作:沙箱升权、被拦的网络请求、可写根外的改动、有副作用的 MCP 调用。[4] 说清楚:这是**审查者的替换,不是权限的放开** —— 它不扩大可写范围、不开网络、不动保护路径,只改「谁来审这个越界」。[4] reviewer 的策略默认拦几类真危险的动作:把私有数据或凭据外传、探凭据、持续削弱安全、有重大不可逆风险的破坏。[4] 它还带熔断:同一轮里连续拒绝 3 次(或滚动窗口内 10 次),Codex 直接中止本轮 —— 免得 agent 在越权尝试上死循环;判断出错时 fail closed(宁可拦错,不放过)。[4] 被误拦了也有退路:`/approve` 对某个被拒动作放行一次重试。这一层是无人值守(第 8 章)真正的价值 —— 没有人在场时,得有个东西替你说不。

> — 注意
>
> 反过来的纪律:auto_review 不是把安全外包,别用它替代好的沙箱设计。它只评估「已经要越界」的动作 —— 待在沙箱内的常规操作根本不经它;它也会犯错,尤其在对抗性场景里。真正的地基仍是把沙箱和审批调对,规则钉死那几个不能碰的前缀,auto_review 是这之上的一层,不是这之下的替身。

动手 · 把放权从直觉调成配置:

- [ ] 在一个有 git 的项目里确认默认落在 Auto(workspace-write + on-request),用 /status 看当前档。
- [ ] 给一个你不想让它随便跑的命令前缀写一条 prompt 规则,用 codex execpolicy check 验它真拦。
- [ ] 在一个长任务上试一次 auto_review,读它拦下 / 放行的记录 —— 判断它的口味和你的是否一致。

# 参考.


01 OpenAI Codex 文档 · Agent approvals & security —— 安全控制来自两层:Sandbox mode 定它技术上能做什么(在哪写、能否联网),Approval policy 定它什么时候必须先问你。默认 agent 断网;本地用 OS 沙箱把它限制在工作区。Auto 预设(--sandbox workspace-write --ask-for-approval on-request)让它在工作目录里自动读、改、跑命令,越界(工作区外写、要联网)才问。可用 /permissions 切 read-only。默认 workspace-write 下网络关闭,需显式 [sandbox_workspace_write] network_access = true 打开。可写根里仍有保护路径:.git、.agents、.codex 递归只读。截至 2026-07-17。OpenAI · Agent approvals & security

02 OpenAI Codex 文档 · Sandbox —— 常见沙箱模式:read-only(只读,改文件 / 跑命令要批)、workspace-write(读 + 在工作区内写 + 跑常规本地命令,是低摩擦默认)、danger-full-access(无沙箱限制)。常见审批策略:untrusted(非信任集里的命令先问)、on-request(默认在沙箱内工作,越界才问)、never(不停下来问)。沙箱作用于派生命令(git、包管理器、测试器都继承同一边界),OS 原生强制(macOS Seatbelt、Linux/WSL2 bubblewrap、Windows 原生沙箱)。启动时检测:版本库目录推荐 Auto,非版本库推荐 read-only。截至 2026-07-17。OpenAI · Sandbox

03 OpenAI Codex 文档 · Rules(实验特性)—— 用规则控制 Codex 能在沙箱外跑哪些命令。.rules 文件用 Starlark 写 prefix_rule(pattern 命令前缀、decision = allow / prompt / forbidden、justification、match / not_match 内联测试);多条匹配时最严的赢(forbidden > prompt > allow)。放 ~/.codex/rules/ 或项目 .codex/rules/(仅信任项目加载)。对 bash -lc 里 && 串起的复合命令,若只含普通词 + 安全操作符,Codex 会用 tree-sitter 拆开逐条判 —— 所以 git add . && rm -rf / 不会因你允许了 git add 就整条放行。codex execpolicy check 可测试规则。截至 2026-07-17。OpenAI · Rules

04 OpenAI Codex 文档 · Auto-review —— 把沙箱边界上的人工审批换成一个独立的 reviewer agent:approvals_reviewer = "user"(默认)或 "auto_review"。它只评估那些本来就要停下问人的越界请求(沙箱升权、被拦的网络请求、可写根外的文件改动、有副作用的 MCP / app 调用),是审查者的替换,不是权限的放开。策略默认拦:外传私有数据 / 凭据、探凭据、持续削弱安全、有重大不可逆风险的破坏性动作。带熔断:同一轮连续 3 次拒绝(或滚动窗口内 10 次)就中止本轮;出错 fail closed。/approve 可对某个被拒动作放行一次重试。截至 2026-07-17。OpenAI · Auto-review

信方向就松档，
信不过的点用规矩钉死。

# 交代 · AGENTS.md、config 与 Skills.

同一句话你跟它说了十遍:测试用这条命令、别碰那个目录、PR 要带说明。AGENTS.md 治这个 —— 它不是 Codex 私产,是六万多个开源仓库在用的开放标准,全局、仓库根、子目录逐层叠加,离当前目录最近的赢。config.toml 管机器上的默认:模型、沙箱、审批、MCP,一份配置三个本地面共用。重复的流程再往上沉一层:存成 Skill,$名字 唤起,用到才加载正文。这一章讲三层交代怎么分工 —— 事实进 AGENTS.md,默认进 config,流程进 skill,以及为什么「它自己攒的 memories」不能替你立规矩。

---

同一句话你跟它说了十遍:测试跑这条命令、别碰那个目录、PR 要带说明。模型不跨会话记事,每个新对话都是一张白纸 —— 除非你把交代写下来。Codex 有三层能写:事实和规范进 AGENTS.md,机器上的默认进 config.toml,重复的流程沉成 Skill。这一章讲三层怎么分工,以及一条容易被高估的边界:它自己攒的 memories,不能替你立规矩。

— I

## AGENTS.md:不是 Codex 私产,是开放标准.

先纠正一个常见误解:AGENTS.md 不是 OpenAI 发明的私有格式。它是「一个引导编码 agent 的简单开放格式」,被六万多个开源项目使用,Cursor、Copilot、Devin、Warp、Gemini CLI 等一大票 agent 都读它。[1] 这意味着你为 Codex 写的这份交代,换个工具照样管用 —— 你投的时间不绑死在一家。规则也简单:离被改文件最近的 AGENTS.md 赢,而你当场在对话里说的话压过文件里的一切。[1] Codex 动手前先读它,发现链是分层叠加的:全局的 `~/.codex/AGENTS.md` 打底,然后从项目根往当前目录逐层走,每个目录取一个文件、从根往下拼 —— 越靠近你正在改的文件,越晚出现、覆盖越前面的。[2] 所以一个 monorepo 里,根目录放全仓通用规矩,子服务目录放它自己的特殊规矩,离得近的自动赢。起步不用手写:CLI 里 `/init` 生成一份起草稿。[6]

AGENTS.md

```
## 怎么跑
- 装依赖用 pnpm,不是 npm
- 测试:pnpm test;lint:pnpm run lint,开 PR 前必过

## 别碰
- 不要改 migrations/,不要动格式化配置
- 生产配置从只读副本读,不写回

## Review guidelines
- 不要记录 PII
- 确认鉴权中间件包住了每条路由
```

一份短而准的项目交代 —— 只写它猜不到的

写什么、写多少,官方的判断很干脆:短而准胜过长而空 —— 只收它读代码猜不到的东西(构建 / 测试命令、和默认不同的规范、禁区、踩过的坑)。[6] 什么时候加一行?一条好用的触发律:**Codex 同一个错犯第二次,就让它做个复盘、把教训写进 AGENTS.md**。[6] 文件长到臃肿就拆 —— 主文件保持精简,专门的细则放进子目录里更近的文件。上面那段 `Review guidelines` 不是摆设:第 9 章 GitHub 上的自动审查,读的正是它。

• • •

— II

## config.toml:机器上的默认.

AGENTS.md 管「这个项目怎么回事」,config.toml 管「这台机器上 Codex 默认怎么做」—— 模型、沙箱档、审批策略、思考力度、MCP 服务器,都在这里定死一次,省得每次开会话重设。[3] 个人默认放 `~/.codex/config.toml`,项目特有的行为放项目里的 `.codex/config.toml`(且只在你信任那个项目时才加载)。优先级从高到低是:命令行标志 → 项目 config(越近越优先)→ profile → 用户 config → 系统 config → 内置默认。[3] 和第 2 章那条「一台引擎」呼应:**CLI 与 IDE 扩展共用同一套配置层** —— 你在 `config.toml` 里定的模型和沙箱档,两个本地面同时生效。[3] 所以调

它是一次性投资:配一次,两端通用。别把该长期钉住的默认反复在命令行里临时传 —— 命令行标志留给一次性的例外,持久的偏好写进 config。

• • •

— III

## Skills:把重复的流程沉成一个词.

有些交代不是一句事实,是一整套多步流程:发版的检查清单、跑一份数据管道、按团队口吻写 changelog。这些沉成 **Skill**:一个目录加一份 `SKILL.md`(必含 name 和 description),旁边可放脚本和参考。[4] 它建立在开放的 agent skills 标准上,和 AGENTS.md 一样不绑死一家。关键在加载方式 —— progressive disclosure:Codex 平时只拿每个技能的名字和描述(初始清单至多占 2% 上下文),真决定用某个技能时才读它完整的正文。[4] 一份几百行的发版手册,平时几乎不占地方。存好之后有两条入口:你打 $名字 显式唤起,或 Codex 按描述判定相关就隐式加载 —— 所以描述要写清「什么时候该用我」。[4] 位置分层,和记忆、配置一样:仓库里的 `.agents/skills` 随版本库共享,用户级 `~/.agents/skills` 跟着你走。[4] 如果你记得旧的「自定义 prompts」:已经弃用,一律改用 skills;要跨团队分发,再往上打包成 plugin。[4]

> — 注意
>
> 还有第四样东西也在记事 —— 但它不能替你立规矩。本地记忆(默认关闭)让 Codex 把先前工作里有用的上下文带进后续对话,方便,但官方把话说死了:**必须始终生效的团队指令,留在 AGENTS.md 或签入的文档里;记忆是有用的回忆层,不是那些必须永远适用的规则的唯一来源。**[5] 换句话说:记忆帮它「想起来」,AGENTS.md 让它「必须照办」—— 两件事,别指望记忆兜底规矩。

动手 · 给一个真项目铺三层交代:

01

### 跑 /init,再手动删到「短而准」

让它生成起草稿,然后逐行问「删掉这行它会不会犯错」—— 不会就删。顺手加一段 Review guidelines,给第 9 章用。

02

### 把一个反复设的默认写进 config.toml

模型、沙箱档、思考力度,挑你每次都要调的那个,定死在 ~/.codex/config.toml。之后切到编辑器确认它也生效 —— 一台引擎。

03

### 把重复第三次的流程存成第一个 skill

找一套你这周贴过两遍的多步流程,放进 .agents/skills/<名字>/SKILL.md,描述写清触发时机。下次 $名字 唤起。

# 参考.


01 agents.md(标准站)—— 一句话定位「一个引导编码 agent 的简单开放格式」;称被六万多个开源项目使用,支持方包括 OpenAI Codex、Cursor、Jules、Devin、GitHub Copilot Coding Agent、Warp、Zed、opencode、Gemini CLI、Aider 等。规则:「离被改文件最近的 AGENTS.md 赢;显式的用户对话指令压过一切。」就是普通 Markdown,用什么标题都行。截至 2026-07-17。agents.md

02 OpenAI Codex 文档 · Custom instructions with AGENTS.md —— Codex 动手前先读 AGENTS.md。发现链:全局 ~/.codex/AGENTS.md(或 AGENTS.override.md)→ 项目根往当前目录逐层走,每目录取一个文件(override > AGENTS.md > 回退名),从根往下拼接,越靠近当前目录越晚出现、因此覆盖更早的。默认上限 project_doc_max_bytes 32 KiB;撞上限可抬高上限或把指令拆到嵌套目录。截至 2026-07-17。OpenAI · AGENTS.md

03 OpenAI Codex 文档 · Config basics —— 个人默认在 ~/.codex/config.toml,项目覆盖在 .codex/config.toml(仅信任项目加载)。优先级(高到低):CLI 标志 / -c 覆盖 > 项目 config(越近越优先)> profile 文件 > 用户 config > 系统 config > 内置默认。常改项:model、approval_policy、sandbox_mode、model_reasoning_effort、personality、[features] 开关(MCP 服务器的完整字段见 config-reference)。原话:「The CLI and IDE extension share the same configuration layers.」截至 2026-07-17。OpenAI · Config basics

04 OpenAI Codex 文档 · Build skills —— 技能建立在开放的 agent skills 标准(agentskills.io)上:一个目录加一份 SKILL.md(必含 name 与 description)加可选脚本 / 参考。progressive disclosure:Codex 先只拿每个技能的名字、描述、路径(初始清单至多占 2% 上下文或 8000 字符),决定用某个技能时才读它完整的 SKILL.md。触发:$名字 显式提及或 /skills,或按描述隐式匹配。位置分仓库 .agents/skills、用户 ~/.agents/skills、admin、系统内置。旧的自定义 prompts 已弃用,改用 skills;要跨团队分发则打包成 plugin。截至 2026-07-17。OpenAI · Build skills

05 OpenAI Codex 文档 · Memories —— 记忆让 Codex 把先前工作里有用的上下文带进后续工作;本地 Codex 记忆默认关闭,可在设置或 [features] memories = true 打开,/memories 按对话控制,存 ~/.codex/memories/ 的 markdown。原话:「把必须始终生效的团队指令留在 AGENTS.md 或签入的文档里,把记忆当成有用的回忆层,而

不是那些必须永远适用的规则的唯一来源。」截至 2026-07-17。OpenAI · Memories

06 OpenAI Codex 文档 · Best practices —— 一旦某个 prompt 模式好用,就用 AGENTS.md 固化它。CLI 里 /init 是脚手架命令,生成一份起步 AGENTS.md,再照你团队实际的构建 / 测试 / 审查 / 发布方式改。官方判断:短而准的 AGENTS.md 胜过又长又空的;「当 Codex 同一个错犯第二次,让它做个复盘并更新 AGENTS.md」;文件太大就让主文件精简、把计划 / 代码审查 / 架构等拆成任务专用的 markdown 文件来引用。截至 2026-07-17。OpenAI · Best practices

---

## 事实进 AGENTS.md,默认进 config,
## 流程沉成一个词.

# 接线 · MCP 把上下文接进来.

你的上下文大半不在仓库里 —— 在 Figma 的稿、Sentry 的报错、Linear 的工单、文档站的页面里。MCP 让 Codex 直接读它们、调它们:codex mcp add 一条命令接上,CLI、IDE、桌面应用共用同一份配置。Codex 给了比「接不接」更细的闸门:每个服务器能开白名单、关掉危险工具、按工具定审批档 —— 标了破坏性的调用永远要问你。这一章还算一笔多数教程不算的账:每接一个服务器,它的工具清单就常驻你的上下文 —— 官方省额度建议里专门有一条「少接 MCP」。

---

你的上下文大半不在仓库里 —— 需求在 Linear 的工单,报错在 Sentry,设计稿在 Figma,文档在别的站点。于是你的日常变成搬运:开网页、复制、粘进对话、再解释一遍这是什么。MCP 把搬运换成接线:让 Codex 直接读它们、调它们。这一章讲怎么接、闸门怎么收紧,以及一条比「能不能接」更要紧的线 —— 每接一个,它的读写半径和你的额度都大一圈。

— I

## 接线:一条命令,两种形态.

MCP(Model Context Protocol)是把模型接到工具与上下文的开放协议 —— 第三方文档、浏览器、Figma 都能接进来。[1] Codex 把配置存在 `config.toml` 里,和第 5 章那条「一台引擎」一致:**CLI、IDE、桌面应用共用同一份 MCP 配置**,配一次几端通用。接入一条命令,分两种形态:本地进程走 STDIO,远程服务走 Streamable HTTP(支持 Bearer token 或 OAuth):

```
# 本地进程(STDIO):-- 之后是服务器自己的启动命令
codex mcp add context7 -- npx -y @upstash/context7-mcp

# 看已配的服务器 / 需要 OAuth 的远程服务器在这里登录
codex mcp list
codex mcp login <server>
```

加一个本地服务器，或直接写 CONFIG.TOML；/MCP 看连接

远程服务器直接写进 config.toml 更省事,一个 [mcp_servers.名字] 表配好 URL 和认证:

~/.codex/config.toml

```
[mcp_servers.figma]
url = "https://mcp.figma.com/mcp"
default_tools_approval_mode = "prompt"
enabled_tools = ["get_file", "get_comments"]
```

一个远程 HTTP 服务器，带按工具的审批档

文档自己举的常用服务器就那几个,覆盖多数人的搬运:OpenAI Docs(查官方文档)、Context7(查最新的第三方库文档)、Figma、Playwright、Sentry、GitHub。[1] TUI 里 /mcp 随时看哪些在连着。

• • •

— II

## 比「接不接」更细的闸门.

接一个服务器不是「全接或全不接」。Codex 给了按服务器、按工具的闸门:enabled_tools 只放行你要的那几个工具,disabled_tools 关掉危险的,default_tools_approval_mode 定这

台服务器的默认审批档(auto 自动、prompt 每次问、writes 只对非只读工具问、approve 全要批),还能对单个工具再单独定 approval_mode。[1] 也就是说,你可以接一个能查也能改的服务器,但只放行「查」,把「改」单独设成必须弹窗。还有一条 Codex 替你兜底的硬规矩:**标了破坏性注解的 app / MCP 工具调用永远要审批** —— 即便那个工具同时声称自己只读,只要它宣告了破坏性,Codex 就一定停下问你。[2] 这是「能接但别失控」的最后一格:再松的档,也拦不住的那类动作,由协议层替你按住。

• • •

– III

## 能接,不等于该接.

现在说那条更要紧的线,它有两头。一头是**额度**:每接一个服务器,它的工具清单就要占你每条消息的上下文 —— 官方省额度建议里明写着一条「限制你用的 MCP 服务器数量,不用时禁用它们」。[3] 接十个服务器却只用一个,是拿真金白银养着九份闲置上下文。另一头是**安全**:会抓取外部内容的服务器可能带来 prompt injection —— 它替你读回的网页或工单里,可能藏着写给模型看的指令,诱它做你没让它做的事。[2] 接线前过三问,比接完再后悔便宜:这条数据真需要直读吗 —— 一次性的贴进来更快,反复要的才值得接;只读够不够 —— 能查和能改是两个半径,从只读开始;写操作有没有闸 —— 危险动作用上一节的按工具审批档单独按住。接线是为了少搬运,不是为了收集服务器。动手 · 接第一条线,按信任梯度来:

01

### 挑你每周复制最多的那个系统

工单、报错监控或文档站,选一个。codex mcp add 接上,/mcp 确认连通,先只开只读工具。

02

### 给它定一个 prompt 审批档,跑一周

default_tools_approval_mode 设 prompt,让每次调用都过你一眼;观察它把数据用得对不对,再决定要不要放开某几个自动。

03

### 月底数一遍你真在用的服务器

把一个月没调用过的服务器 disabled 掉 —— 每个闲置服务器都在吃你的上下文和额度。少接,是省额度纪律里最实的一条。

# 参考.


01 OpenAI Codex 文档 · Model Context Protocol —— MCP 把模型接到工具与上下文,让 ChatGPT 或 Codex 用上第三方文档、浏览器、Figma 等。Codex 把 MCP 配置存在 config.toml([mcp_servers.名字]),CLI / IDE / 桌面应用共用同一份。加服务器:codex mcp add …(STDIO 本地进程,或 Streamable HTTP 远程地址,支持 Bearer / OAuth);codex mcp list 看已配、codex mcp login 走 OAuth;TUI 里 /mcp 看在连的服务器。每服务器可配 enabled_tools 白名单、disabled_tools 黑名单、default_tools_approval_mode(auto / prompt / writes / approve)与按工具的 approval_mode。文档举的常用服务器:OpenAI Docs、Context7、Figma、Playwright、Sentry、GitHub。截至 2026-07-17。OpenAI · Model Context Protocol

02 OpenAI Codex 文档 · Agent approvals & security —— Codex 也会为宣告有副作用的 app(连接器)工具调用征求审批;标了破坏性注解的 app / MCP 工具调用永远要审批,即便它同时标了只读等其它提示。默认 agent 断网;会抓取外部内容的场景可能带来 prompt injection —— 它替你读回的网页或工单里,可能藏着写给模型看的指令。截至 2026-07-17。OpenAI · Agent approvals & security

03 OpenAI Codex 文档 · Pricing(省额度建议)—— 每接一个 MCP 服务器都会给你的每条消息加更多上下文、多吃额度;官方省额度建议里明列一条:「限制你用的 MCP 服务器数量……不用时禁用它们。」截至 2026-07-17。OpenAI · Codex pricing


---

## 上下文不该靠你搬运,
## 但每根线都记在账上.

# 派上云 · 云端沙箱、并行与 PR 流.

这是 Codex 和同类拉开差距的一章。每个云端任务在 OpenAI 的隔离容器里跑:检出你的仓库、带网跑 setup 脚本、然后断网进入 agent 阶段 —— secrets 在动手前撤走,收尾交回一个 diff,一键开 PR。你可以同时派一批:五个 bug 各开一个任务,互不干扰地并行,你去干别的,回来挨个收。派活的入口也不止网页:GitHub 里 @codex 一句评论就能把 PR 上的活派进云端,手机上也能派能收。这一章讲环境怎么配、联网白名单怎么开、以及哪类活值得上云 —— 可验证、批量、不需要你陪跑的。

---

这是 Codex 和同类拉开距离的一章。前六章讲的活,不管在终端、编辑器还是本地 worktree,都还在你这台机器上、你多少看得见。云端不一样:活离开你的机器,进 OpenAI 的隔离容器,跑完回来是一份 diff、一个 PR。它最狠的地方不是「也能在别处跑」,是你能一次派一批,并行跑,回来挨个收。这一章讲一个云端任务怎么跑、环境怎么配、从哪几处派、以及哪类活值得上云。

— I

## 一个云端任务怎么跑.

云端的一句话定位是「把活委派给隔离的云端环境里的 Codex」。[1] 你提交一个任务后,大致这样走:创建一个容器、按你选的分支或 SHA 检出仓库、跑你的 setup 脚本,然后进 agent 循环 —— 跑命令、改代码、验证(仓库里有 AGENTS.md 就用它找 lint / test 命令),完成后给出答案和一份改动 diff,你可以一键开 PR,或追问再改。[2] 安全模型是这一步的关键,分两阶段:**setup 阶段**跑在 agent 之前、可以联网装依赖;然后**agent 阶段默认断网**。[1] 更要紧的一条:你为云端环境配的 secrets **只在 setup 期间可用,在 agent 阶段开始前就被移除** —— 也就是说,真正干活、可能被 prompt injection 诱导的那个阶段,手里已经没有你的密钥了。[1] 这是「把活交出去」敢成立的底层:容器隔离你的主机,断网隔离外传,撤密钥隔离最坏情况。

– II

## 环境:装什么,连什么.

云端跑得对不对,一半看环境配得对不对。默认镜像叫 universal,预装了常见语言和工具,你可以 pin 具体版本、加 setup 脚本装额外依赖。[2] 两类值得分清:**env vars** 全程可用,**secrets** 只在 setup 可用(前面说过,agent 阶段前撤走)。容器还会缓存至多 12 小时,新任务和追问都快 —— 改了 setup 脚本或 secrets 会自动失效重建。[2] 联网是最该谨慎的一格。agent 阶段默认断网;要开,按环境选 On,再用域名白名单收窄 —— 预设有 None(从零加)、Common dependencies(一份常见包管理与源码域名)、All(无限制),还能限定只放 GET / HEAD / OPTIONS 这类只读方法。[3] 为什么这么小心?文档给的注入例子很直白:你让它去修一个 GitHub issue,issue 文本里藏了一条 curl 外传命令,联网又不设防的 agent 可能就把你的数据发去了攻击者的服务器。[3] 默认断网不是麻烦,是护栏 —— 只在真需要装依赖、真信任目标域名时才开一条缝。

– III

## 一次派一批,从四处派入.

云端最能拉开差距的用法,是**并行**:五个 bug 各开一个云端任务,它们在各自的容器里同时跑,你去干别的,回来挨个看 diff、挨个开 PR。这不是「一个 agent 跑快点」,是「一个人同时监工五件活」—— 而验收成了你唯一还需要亲手做的动作(第 9 章)。派入的口子也不止网页一个。**网页**(chatgpt.com/codex)是主入口;**GitHub** 上一句评论就能派 —— 在 PR 里 @codex 加一句话,它用这个 PR 作上下文起一个云端对话,有权限时把修复推回分支:

```
# 在一个 PR 评论里(GitHub)
@codex fix the CI failures

# 在终端:把某个云端对话最新的 diff 应用到本地工作树(别名 codex a)
codex apply <TASK_ID>
```

GITHUB 上派活,或把云端 DIFF 拉回本地

还有**手机**:ChatGPT 应用的 Remote 连回你桌面的 host,在路上也能派、也能收。[4] 收回来的路径同样多:网页直接开 PR,或在终端用 `codex apply`(别名 `codex a`)把某个云端对话最新的 diff 拉到本地工作树、在你熟悉的环境里验;`codex cloud` 命令组(`codex cloud exec` 派任务、`codex cloud list` 列最近对话)则从终端浏览、执行云端对话。[5] 派活的口子和收活的口子都铺开了,四个面在这里合流成一句话:活在哪都能派,回来都是可审的 diff。

• • •

— IV

## 哪类活值得上云.

不是所有活都该上云。云端的甜点位有三个特征:可验证 —— 有测试、有构建、有能对比的产出,你才敢让它在你看不见时跑完;可并行 —— 一批彼此独立的活,并行才划算,一件串行的活上云只是多绕一圈;不需要你陪跑 —— 要你逐字盯着改的活,留在编辑器更快。反过来,一件需要你反复口头拉扯、边看边改的探索型任务,上云是把沟通成本翻倍,那种活属于第 2、3 章的本地面。

> — 补充
>
> 一条诚实的短板:云端把活推得很远,也把「验收」推成了刚需 —— 一批 PR 涌回来,如果你只会扫两眼就合,并行省下的时间会被漏掉的 bug 连本带利收走。云端越强,第 9 章越是这本书的书眼:能派一批的人不缺,能收一批的人才稀缺。

动手 · 派第一批云端任务:

- [ ] 配一个云端环境:选默认 universal 镜像,加一段最小 setup 脚本,联网先设 Off。
- [ ] 挑三件彼此独立、可验证的小活,各派一个云端任务,并行跑 —— 感受「同时监工三件」。
- [ ] 回来后用 codex apply 把其中一份 diff 拉到本地,在你熟悉的环境里跑一遍它交回的检查,再决定合不合。

# 参考.


01 OpenAI Codex 文档 · Codex cloud —— 一句话定位「把活委派给隔离的云端环境里的 Codex」。云端在 OpenAI 托管的隔离容器里跑,防止碰到你的主机或无关数据;采用两阶段:setup 阶段在 agent 之前跑、可联网装依赖,然后 agent 阶段默认离线(除非你为该环境开联网)。为云端环境配置的 secrets 只在 setup 期间可用、在 agent 阶段开始前被移除。截至 2026-07-17。OpenAI · Codex cloud

02 OpenAI Codex 文档 · Cloud environments —— 每个云端对话:创建容器、按选定分支 / SHA 检出仓库,跑 setup 脚本(+ 缓存容器复用时可选的 maintenance 脚本),应用联网设置,agent 循环里跑命令、改代码、验证,若仓库有 AGENTS.md 就用它找 lint / test 命令;完成后给出答案和一份改动 diff,你可开 PR 或追问。默认镜像 universal(预装常见语言 / 工具,可 pin 版本);env vars 全程可用,secrets 只在 setup 可用;容器缓存至多 12 小时。截至 2026-07-17。OpenAI · Cloud environments

03 OpenAI Codex 文档 · Agent internet access —— agent 阶段默认断网;setup 脚本仍可联网装依赖。按环境开启,可选择 Off / On,On 时用域名白名单(None / Common dependencies 预设 / All 无限制)加允许的 HTTP 方法(为多一层保护,可只放 GET / HEAD / OPTIONS)。风险:prompt injection、代码 / secrets 外泄、拉进恶意或受限依赖。文档给的注入例子:issue 文本里藏 curl 外传命令,诱 agent 执行。截至 2026-07-17。OpenAI · Agent internet access

04 OpenAI Codex 文档 · Codex code review in GitHub —— GitHub 上 PR 评论里 @codex(非 review)会用该 PR 作上下文起一个云端对话,例如「@codex fix the CI failures」,它有权限时能把修复推回分支。截至 2026-07-17。OpenAI · Codex in GitHub

05 OpenAI Codex 文档 · Command line options(CLI reference)—— 顶层命令 codex apply <TASK_ID>(别名 codex a)把某个云端对话最新的 diff 应用到你的本地工作树。codex cloud 是从终端浏览 / 执行云端对话的命令组:默认开交互选择器,codex cloud exec 直接派任务、codex cloud list 列最近对话,不必打开 TUI。截至 2026-07-17。OpenAI · Command line options

活离开你的机器，
回来是一份可审的 diff.

# 无人值守 · exec、SDK、CI 与定时任务.

到这里,派活还都要你亲手发一句话。这一章把那句话也自动化:codex exec 让它在脚本里无人值守地跑一次,JSONL 事件流和 JSON Schema 约束的结构化输出让脚本能直接消费结果;SDK 把同一台引擎装进你的 TypeScript 或 Python 程序;openai/codex-action 让它长在 GitHub CI 里;定时任务让它按日程自己开工。无人时的规矩也在这一章:exec 默认只读、越界即停,hooks 在生命周期的固定时点执行你的脚本 —— 且每条 hook 都要你审过、按哈希信任过才许跑。

---

到这里,派活还都要你亲手发一句话 —— 打开一个面、说清一件活、看它跑。这一章把那句话也自动化:让 Codex 在脚本里无人值守地跑一次,长进你自己的程序,守在每个 PR 上,按日程半夜上班。这是「不守着做完」的极限形态:活不在你这台机器、不在你醒着的时候完成。也因此,这一章的另一半讲规矩 —— 它跑得越自动,边界越要在出发前写死。

— I

## codex exec:让那句话进脚本.

把交互会话变成一条 Unix 命令,靠的是 `codex exec`:跑完就退,进度走 stderr、最终消息走 stdout,天然能接进管道。[1] 它把 Codex 从「一个你打开的应用」变成「一个脚本里的工序」:

```
# 把构建日志灌进去,要一句根因
cat build-error.txt | codex exec "一句话解释这次构建失败的根因"

# 结构化输出:按 schema 约束,写最终 JSON 到文件
codex exec "列出 src/api 里的全部路由" \
  --output-schema ./routes.schema.json \
  -o ./routes.json

# 要它改文件:默认只读,得显式放开
codex exec --sandbox workspace-write "修好失败的那条测试"
```

灌上下文、要结构化结果；默认只读，要改文件才放开

三个细节是无人化的关键。`--json` 让 stdout 变成 JSONL 事件流,脚本能逐条读它干了什么;`--output-schema` 更进一步,让最终响应符合你给的 JSON Schema,脚本直接消费结构化字段、不用解析散文。[1] 还有一条安全默认:**exec 默认是只读沙箱** —— 要它改文件,必须显式 `--sandbox workspace-write`。[1] 第 9 章说的「回归集用 `codex exec` 重跑、拿 JSON 对一对」,零件就是这几个 flag。它还要求在 Git 仓库里跑,防止无人时的破坏性改动无从回滚。[1]

• • •

— II

## SDK: 同一台引擎,做成库.

脚本再长,也只是把命令串起来;要把 Codex 嵌进你自己的服务 —— 一个自动分诊 bug 的机器人、一个替客服读日志的后台 —— 用 Codex SDK 编程控制它。[2] TypeScript 装 `@openai/codex-sdk`,一个 `startThread()` 就是一个带全套工具的对话,`run(prompt)` 拿结果、`resumeThread(id)` 续跑:

```
triage.ts
```

```
import { Codex } from "@openai/codex-sdk";

const codex = new Codex();
const thread = codex.startThread();
const result = await thread.run("诊断并修好 CI 里失败的用例,做一个计划");
console.log(result.finalResponse);
```

一个最小的 SDK 线程;PYTHON 有对应的 OPENAI-CODEX(BETA)

Python(beta)装 openai-codex,Codex() 或 AsyncCodex() 经 app-server 控制,还能按轮切 Sandbox 预设(read_only / workspace_write / full_access)—— 让「先改、再只读复审」这种两步走在代码里表达出来。[2] CLI 和 SDK 怎么分工很清楚:交互开发、一次性活用 CLI;要嵌进 CI/CD、定制应用、生产自动化,用 SDK。这本书讲到这条线为止 —— 再往里,是《Agent 实战》整本书的地界。

• • •

— III

## 长在 CI 和日程里.

第三种形态不属于任何一台机器。**GitHub Action**(openai/codex-action@v1)在 CI 里跑 Codex:它装好 CLI、为你的 OpenAI key 起一个 Responses API 代理、跑 codex exec;输入里有 prompt、model、sandbox、output-file,最终消息作输出给后续步骤。[3] 默认的 safety-strategy: drop-sudo 在跑前移除 sudo、保护内存里的 secrets,安全清单还专门提醒:来自 PR、commit、issue 的 prompt 输入要先清洗,防注入。[3] 定时的活交给**定时任务**:在 ChatGPT 桌面应用或网页里建、在后台按日程跑(CLI / IDE 不提供管理界面,但能先把 prompt 试好)。[4] 两种形态:standalone 每次开一个新对话、支持 RRULE 自定义节奏,适合独立的定期巡检;in-chat 回到同一对话、带着它已有的上下文,适合盯一件长活。可以跑在本地项目或 worktree,无人值守时用你的默认沙箱 —— 组织策略允许的话,用 approval_policy = never 全程不停下问。[4] 夜间依赖巡检、晨会前的 PR 摘要,都是它的班次。

— IV

# 无人时的规矩:hooks 与信任.

交互会话里,审批弹窗是最后的安全网;无人值守时没有这张网 —— 边界要前置。第 4 章那套沙箱、审批、规则照样在;这一章再加一层 **hooks**:把你的脚本注入 agentic 循环的固定时点(工具调用前后、会话结束、收工时),用来记录对话、拦下贴出的 API key、收工前跑一遍校验。[5] 配在 `hooks.json` 或 `config.toml` 的 `[hooks]` 里,事件有 PreToolUse、PostToolUse、Stop、SessionStart 这些,matcher 按工具名筛。[5]

> — 注意
>
> hooks 有一条 Codex 特有的信任闸,别踩:**非托管的命令 hook 必须先经你审阅、按它当前的哈希信任过才许跑** —— 脚本一改,就重新标记为待审、在你重新信任前跳过。[5] 这既是保护(别人塞进仓库的 hook 不会自动在你机器上执行),也是个坑:你在 CI 里换了 hook 脚本却忘了它需要重新信任,它就会被静默跳过。`/hooks` 里审阅与信任,无人值守出发前先确认它们真的在跑。

动手 · 把一件活送出你的视线:

01

### 把一个重复检查做成 codex exec 一行

挑你每周手动做的一个检查(错字、日志异常、路由清单),写成 codex exec,加 --output-schema 让结果能被脚本消费,挂进 package.json 或 Makefile。

02

### 给团队仓库装上 GitHub Action

配 openai/codex-action@v1 在 PR 上跑一次小活,把 API key 放进 secrets、safety-strategy 保持 drop-sudo,看它从触发到留言走完全程。

03

### 设一个夜班定时任务,先确认 hook 在跑

用定时任务给明早设一个巡检,出发前在 /hooks 里确认你的校验 hook 已被信任 —— 无人时,规矩必须先于活出发。

# 参考.


01 OpenAI Codex 文档 · Non-interactive mode —— codex exec 从脚本里跑 Codex,不开交互界面;进度走 stderr,最终消息走 stdout,便于管道。默认只读沙箱,要改文件加 --sandbox workspace-write。--json 让 stdout 变 JSONL 事件流(thread.started / turn.completed / item.* 等);--output-schema schema.json 让最终响应符合 JSON Schema,-o 写最终消息到文件。可 codex exec resume --last 续跑。要求在 Git 仓库里跑(--skip-git-repo-check 可绕过);CODEX_API_KEY 仅 exec 支持。截至 2026-07-17。OpenAI · Non-interactive mode

02 OpenAI Codex 文档 · Codex SDK —— 用 SDK 编程控制 Codex(嵌进 CI/CD、自建 agent、自家工具)。TypeScript: npm install @openai/codex-sdk,new Codex().startThread(),thread.run(prompt) 拿 result.finalResponse,resumeThread(id) 续跑。Python(beta): pip install openai-codex,Codex() / AsyncCodex() 经 app-server 控制,thread_start(model, sandbox) + run;Sandbox 预设 read_only / workspace_write / full_access 可按轮切。截至 2026-07-17。OpenAI · Codex SDK

03 OpenAI Codex 文档 · Codex GitHub Action —— openai/codex-action@v1 在 GitHub Actions 里跑 Codex: 安装 CLI、为 OpenAI key 起一个 Responses API 代理、跑 codex exec。输入含 prompt / prompt-file、codex-args、model / effort、sandbox、output-file,final-message 作输出。默认 safety-strategy: drop-sudo(跑前移除 sudo,保护内存里的 secrets);安全清单提醒: 清洗来自 PR / commit / issue 的 prompt 输入以防注入。截至 2026-07-17。OpenAI · Codex GitHub Action

04 OpenAI Codex 文档 · Scheduled tasks —— 在后台按日程跑任务;在 ChatGPT 桌面应用与网页的 Work / Codex 里创建管理(CLI / IDE 不提供管理界面,但能先准备测试 prompt)。standalone(每次新对话跑,支持 RRULE 自定义节奏)vs in-chat(回到同一对话、用它已有上下文,支持分钟级循环)。可跑在本地项目或 worktree。无人值守,用你的默认沙箱;组织策略允许时用 approval_policy = never。截至 2026-07-17。OpenAI · Scheduled tasks

05 OpenAI Codex 文档 · Hooks —— 把你的脚本注入 agentic 循环(记录对话、拦贴出的 API key、自动生成记忆、收工时跑校验)。配 hooks.json 或 config.toml 的 [hooks],位置 ~/.codex/ 与项目 .codex/(信任项目才加载)。事件含 PreToolUse、PermissionRequest、PostToolUse、UserPromptSubmit、Stop、SubagentStart /

Stop、SessionStart。JSON 进 stdin；matcher 按工具名过滤(Bash、apply_patch / Edit|Write、mcp__*)。关键：非托管命令 hook 必须先经你审阅、按当前哈希信任过才许跑，改了就重新标记待审；/hooks 里审阅与信任。GA 于 2026-05，[features] hooks = true 默认。截至 2026-07-17。OpenAI · Hooks

---

**活可以不在你眼前跑，**
**规矩必须先于它出发.**

# 验收 · 怎么收 agent 交回的活.

四面开工的代价,是四面都有活等你收。扫两眼 diff、看它说「测试过了」、合并 —— 那不是验收,是祈祷。这一章是全书的书眼:验收标准在派活时就写死,交付要带证据;本地 /review 用一个独立的 reviewer 审 diff,应用里的 review pane 逐 hunk 收放、行内评论直接喂回下一轮;GitHub 上 @codex review 只报 P0/P1,automatic reviews 让每个 PR 都过一道 —— OpenAI 自己 100% 的 PR 都由 Codex 审。最后是沉淀:同一个错第二次出现,写进 AGENTS.md 的 Review guidelines,让下一次自动被拦住。它直接接上《Agent 实战》的评测那章。

---

四面开工的代价,是四面都有活等你收:编辑器里一份 diff、云端一摞 PR、CI 里一条自动改动、定时任务的一份夜间班报。扫两眼、看它说「测试过了」、合并 —— 那不是验收,是祈祷。这一章是全书的书眼:会派活的人已经太多,能系统性验收 agent 的人还太少。它讲怎么把「看一眼就合」升级成一套真正的收活工作流,并接上《Agent 实战》里评测那一章 —— Codex 是你派活的地方,评测是你确认没被坑的地方。

— I

## 底线:一条能复跑的检查.

没有检查的时候,「看起来做完了」是它唯一的停机信号,而你是唯一的验证回路 —— 每个错误都躺在 diff 里,等你亲眼看见。官方最佳实践把这条说得很直:别停在让它改,让它写 / 更新测试、跑该跑的检查、确认结果、审自己 diff 里的 bug 与回归,再由你接受 —— 但它得知道「好」长什么样,这靠你的 prompt 或 AGENTS.md 告诉它。[1] 有了能返回过 / 不过的信号,回路才闭合:它干活、跑检查、修到过为止。纪律有两条。其一:**验收标准在派活时就写死** —— 第 3 章那个四件套里的「完成标准」,就是为这一刻准备的。其二:**要证据,不要断言**。让它交测试输出、贴跑过的命令和返回,而

不是一句「搞定了」;看证据比你亲手重跑一遍快,也是你敢不盯着它的前提。这两条尤其在无人值守和云端(第 7、8 章)时救命 —— 那些活你根本没在场看它跑。

**提示词**

派活时把验收一起派下去

这件活的验收标准:<一条可检验的标准>。
做完后跑 <检查命令>,把完整输出贴给我。
没验证过的部分明说「没验证」,不要写「应该可以」。
检查跑不过就修到跑过再交,别把失败的输出藏起来。
最后列出你改过的每个文件和一句为什么。

这件活的验收标准:CSV 导出的用例全部通过,已有 JSON 导出测试仍然全绿。
做完后跑 pnpm test -- src/export,把完整输出贴给我。
没验证过的部分明说「没验证」,不要写「应该可以」。
检查跑不过就修到跑过再交,别把失败的输出藏起来。
最后列出你改过的每个文件和一句为什么。

• • •

— II

## 对着计划审,换个脑子审.

检查证明「它能跑」,证明不了「是你要的」。跑得过测试的代码,也可能顺手重构了你没让它动的模块、绕开了你点名的方案。第二层验收要有基线 —— 最现成的基线,是第 3 章计划模式里你批准过的那份计划:把 diff 对着它逐条核,要求都实现了吗、点名的边界情况有测试吗、范围之外的东西动了没有。[1] 这层审最好换个脑子来。Codex 内置了这套:本地 /review 起一个**专门的 reviewer** 读选定的 diff、报按优先级排序的发现,而且**不改你的工作区** —— 它只挑毛病,不顺手动手。[2] 审的范围能选:对基线分支审(PR 式)、审未提交改动、审某个 commit,或按你给的自定义指令审。[1] 应用里还有 review pane:按 hunk stage 或 revert,行内评论会作上下文喂回下一轮 —— 你圈一行说「这里漏了空值」,它下一轮就知道去哪改。[2]

– 注意

一条反向纪律:被派去找碴的 reviewer,几乎总会交出几条碴 —— 哪怕活本身是好的,因为找碴就是它接到的任务。逐条照办会把你推向过度工程。GitHub 上 Codex 的做法值得抄:它只标 P0 和 P1 高优先级问题,把噪音挡在外面。[3] 给你自己的 reviewer 也划同一条线:只报影响正确性和既定要求的,其余当可选建议。这不是把审查放松,是把「找什么」说清楚。

• • •

– III

## 在 GitHub 上收一批,再把错固化.

第 7 章一批 PR 涌回来时,你需要的不是逐个手审,是一道自动底审。在 chatgpt.com/codex 的设置里为仓库开启后,PR 评论里 `@codex review` 就触发一次:Codex 先给个 👀、再像队友一样贴出审查,只标 P0 / P1。[3] 开 Automatic reviews,每个新 PR 都自动过一遍,不用你 @ —— 这正是 OpenAI 自己「审 100% 的 PR」的做法。[1] 而且它跟着你最近的 AGENTS.md 里的 `Review guidelines` 走(第 5 章那段不是摆设);看到问题,一句 `@codex fix the P1 issue` 就让它起云端对话把修复推回分支。[3] 验收做到第三遍,就该问:哪部分能固化?按约束力从软到硬,有三个台阶。最软是 AGENTS.md 的 `Review guidelines`:把「同一个错」写成一条审查规矩,下次自动审时就替你盯着 —— 但它是请求,不是保证(第 5 章那条记忆的边界)。硬一点是第 8 章的 hook:收工前必跑的校验,harness 执行、跳不过。最硬的一格,是把犯过的错写成一条测试:从此那个错不归你拦,归测试拦。

• • •

— IV

# 一次对了不算数:接上评测.

单件活的验收到此闭环。但四面开工意味着你迟早在重复派同一类活 —— 每周的依赖升级、每个 PR 的安全审、每天的日志巡检。这时候该换单位了:单件问「这次对不对」,批量要问「这类活它多稳」。τ-bench 给这个直觉起了正式的名字:**pass@k** 量「k 次里至少对一次」,是能力上限;**pass^k** 量「k 次全对」,是可靠性。实测扎心:当时最强的 function calling agent 任务成功率不到 50%,retail 域连续 8 次全对的 pass^8 掉到 25% 以下。[5] 跑对一次和次次跑对之间,隔着「你敢不敢把它设成定时任务」的全部距离。第二个坑在「改进」上。你换了模型、改了 AGENTS.md、加了个 skill,感觉变好了 —— 感觉不算数。评测是实验,实验自带噪音:报成功率要带标准误;比较改动前后,用同一批任务的逐题配对差,而不是两个总分相减;想在 80% 功效下检出 3 个点的差距,大约要 1000 道题。[6] 日常尺度记一条就够:**小样本上的个位数点差,不构成「变好了」的结论** —— 它只构成「再跑几遍」的理由。落地不用自建 harness。把 Codex 坑过你的每个案例攒下来 —— 十几二十条,每条带上第 I 节那样的可检验标准,就是你的回归集;每次动配置,用 `codex exec` 把这批活重跑一遍、`--output-schema` 拿结构化结果对一对。[4] 这正是《Agent 实战》里 golden set 加回归的家用版,也是两本书真正合流的地方:那本书教你把评测建成产线,这本书把它接到你每天派活的四个面上 —— Codex 是你派活的地方,评测是你确认没被坑的地方。动手 · 把这章装进你的下一次派活:

- [ ] 给你最近派出的一件活补一条能复跑的检查,并要它交证据(输出、命令)—— 不收「搞定了」。
- [ ] 挑一份你本来打算直接合的 diff,先用 /review 对着计划审一遍,数数它找出几条你没看到的。
- [ ] 把 Codex 最近一次坑你的错沉淀掉:进 AGENTS.md 的 Review guidelines、变成一条 hook,或写成一条测试。

# 参考.


01 OpenAI Codex 文档 · Best practices —— 别停在让它改;让它写 / 更新测试、跑该跑的检查、确认结果、审 diff 里的 bug 与回归,再由你接受 —— 但它得知道「好」长什么样,这靠 prompt 或 AGENTS.md。/review 给几种审法:对基线分支审(PR 式)、审未提交改动、审某个 commit、按自定义指令审。原话:「在 OpenAI,Codex 审 100% 的 PR。」截至 2026-07-17。OpenAI · Best practices

02 OpenAI Codex 文档 · Code review —— 提交 / 推送前审代码。本地 /review 起一个专门的 reviewer 读选定的 diff、报按优先级排序的发现,不改你的工作区。应用里有 review pane:按 Unstaged / Staged / Commit / Branch / Last turn 看,可按文件 / 按 hunk stage 或 revert,行内评论会作上下文喂回下一轮 Codex。可设 review_model 让审查用不同模型,或设 detached 起独立审查对话。截至 2026-07-17。OpenAI · Code review

03 OpenAI Codex 文档 · Codex code review in GitHub —— 在 chatgpt.com/codex 的 code-review 设置里为仓库开启后,PR 评论里 @codex review 触发一次审查:Codex 先给 👀、再像队友一样贴一条审查,GitHub 上只标 P0 与 P1 高优先级问题。开 Automatic reviews 则每个新 PR 自动审、无需 @。审查跟随最近的 AGENTS.md 里的「Review guidelines」;可「@codex fix the P1 issue」让它起云端对话推修复。截至 2026-07-17。OpenAI · Codex in GitHub

04 OpenAI Codex 文档 · Non-interactive mode —— codex exec 非交互跑一次;--json 出 JSONL 事件流,--output-schema 让最终响应符合 JSON Schema,-o 写最终消息到文件;文档场景为脚本与 CI/CD。适合把一批攒下的回归用例重跑、拿结构化结果对比。截至 2026-07-17。OpenAI · Non-interactive mode

05 Yao et al. · 「τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains」(2024,arXiv:2406.12045)—— 提出 pass^k「度量 agent 行为在多次试验下的可靠性」(k 次全部成功的概率,区别于 pass@k 的至少一次);实测当时最强的 function calling agent(gpt-4o)任务成功率不到 50%,retail 域 pass^8 低于 25%。arXiv · τ-bench / pass^k (2406.12045)

06 Miller · 「Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations」(Anthropic,2024-11,arXiv:2411.00640)—— 把评测当实验:报成功率要带标准误;比较两个配置,用同一批题上的题级配对差做推断,而不是两个总分相

减;动手前先算最小可检测效应(MDE)—— 论文算例:80% 功效下检出 3 个点的差距约需 1000 题。arXiv · Adding Error Bars to Evals (2411.00640)

---

## 派活的人已经过剩,
## 验收的人还稀缺.

# 节奏 · 档位、模型、成本与同类分工.

选档不是看价格,是看你撞哪堵墙:额度、模型,还是速度。这一章把 Plus、Pro(5× / 20× 两档)、API key 旁路摆开,讲清 GPT-5.6 三兄弟的分工 —— Sol 管复杂开放的活,Terra 是日常主力,Luna 跑量大清晰的活 —— 以及 credits 怎么算、fast mode 什么时候值 2.5 倍的价、官方那五条省额度纪律为什么条条都在讲上下文。再划同类分工线:Claude Code、Cursor、Copilot、Warp 各自的主场,已经付着 ChatGPT 的人什么时候该加第二份订阅 —— 以及为什么多数时候不该。

---

选档的错误问法是「哪档划算」,正确问法是「我撞的是哪堵墙」。额度不够,是一堵墙;默认模型不够强,是另一堵;等不起响应速度,是第三堵 —— 三堵墙对应三个不同的开关,只有一个真需要花更多钱。这一章把档位、模型、力度、成本四个旋钮摆开,最后划一条和同类的分工线:已经付着 ChatGPT 的人,什么时候该加第二份订阅 —— 以及为什么多数时候不该。

— I

## 档位与三堵墙.

先把价目摆平:Free $0、Go $8、Plus $20、Pro 从 $100 起(按比 Plus 多 5× 还是 20× 用量分两档)、Business $20/人·月、Enterprise。另有一条 API key 旁路,按 token 计费、不含云端功能(GitHub 审查、Slack 那些)。[1] 要紧的一条:Codex 和 ChatGPT Work **共用同一个 usage 池** —— 你在 Work 里跑掉的和在 Codex 里跑掉的,是同一笔额度。[1]

| 你撞的墙 | 症状 | 开关 |
| --- | --- | --- |
| **额度** | 5 小时窗口内就见底,活排着队 | 升 Pro(5× / 20×),或先做第 III 节的省额度纪律 |
| **模型** | 难题上明显吃力 | 换 Sol、升思考力度(第 II 节) |
| **速度** | 交互时等不起 | /fast(走 credits,另一本账) |

额度墙的形状值得看清:本地消息按模型和任务大小,每 5 小时给一个**范围** —— Plus 下 Sol 大概 15–90 条、Terra 20–110、Luna 50–280(任务越大越复杂,吃得越多);Pro 的 5× / 20× 就是把这个范围整体抬 5 倍、20 倍。[1] 还有一条容易忽略的:**本地和云端共享同一个 5 小时窗口** —— 你在云端并行派一批,吃的和本地是同一格额度。[1] 判断顺序照表走:先确认撞的是哪堵墙,再动钱包 —— 三堵墙里只有额度墙必须用钱解决。

• • •

– II

## 三兄弟与力度:两个旋钮.

模型的日常分工是三个 GPT-5.6 兄弟。**Sol** 管细节与打磨 —— 复杂、开放、高价值的活(旗舰,拿不准就用它);**Terra** 是务实全能的日常主力,原先给 GPT-5.5 的活自然落到它;**Luna** 跑清晰可重复、量大的活:抽取、分类、转换。[2] 默认的 Power 设置就是 Sol 加 medium 推理。[2] 最顺手的用法不是主对话换来换去,而是第 7 章那招:难活主力用 Sol,批量的读 / 扫 / 整理派给 Terra 或 Luna 的并行任务。第二个旋钮是**推理力度**:从 Light(应用)/ Low(CLI)、Medium、到 High、Extra High,控制它每一步想多深 —— 难题升档、机械活降档,同一个模型,力度对了常比换模型便宜。[2] 再上面有两个特殊档:**Max** 给单个任务更多推理时间,啃最硬的问题;**Ultra** 用 subagents 并行拆解,把一件能分块的复杂活铺开来跑 —— 多数活两个都用不上。[2] 还有一个 Pro 专属的速度型模型 **Codex-**

**Spark**(研究预览):near-instant 的纯文本迭代,和 fast mode 不是一回事,它是独立模型、有自己的限额。[3]

• • •

— III

## 成本:credits、fast,和五条纪律.

撞了额度墙,Codex 让当前这轮先跑完(公平使用范围内),然后 Plus / Pro 可以买 credits 续用,不必升档。[1] credits 是把 token 折成的统一单位,按模型算 —— 大致上 Luna 最省、Terra 居中、Sol 最贵,这也和三兄弟的分工对得上:量大的活交便宜模型,是省钱也是省额度。[1] **Fast mode** 是用钱换等待,想清楚再开:它把支持的模型提速 1.5×,代价是按更高倍率吃 credits —— GPT-5.6 / 5.5 是标准的 2.5×,GPT-5.4 是 2×;`/fast on` 切。[3] 注意它是 credit 特性、只有 ChatGPT 登录能用;API key 走的是 API token 价、没有这个倍率。[3] 正确用法窄而清晰:赶工的交互时段开(快速迭代、现场调试),长的自动化任务与 CI 一律关。最省钱的其实不是任何一个开关,是纪律。官方那五条省额度建议,条条都在讲同一件事 —— **管住上下文**:prompt 写精、只喂相关文件、把 AGENTS.md 拆小(第 5 章)、少接 MCP(第 6 章)、常规活降到便宜模型。[1] 前面几章的每个习惯,在账单上都有回声:上下文纪律,就是额度纪律。

• • •

— IV

## 同类分工:入口跟着活走.

最后划分工线 —— 不是黑稿,是各自的主场。**Claude Code**(Pro $20 / Max 从 $100,均含在订阅里)的主场在终端里的编排:计划、记忆、subagents、hooks 拼成的那套「派活 + 验收」循环,是它写得

最透的地方。[4] **Cursor**(Pro $20 起,顶上还有 Pro+ / Ultra)的主场在编辑器:你要「陪着改、盯每一步 diff」的工作节奏时,它的 agent 工作台仍是最顺的入口。[5] **Copilot**(Pro $10 起)的主场在 GitHub 本体:issue 直接派、收回一个 PR,甚至能把活转派给 Claude、Codex 这些第三方 agent —— 你的团队整个活在 GitHub 流程里时,它是那条流程自带的手。[6] **Warp** 自我定位「面向 agentic coding 的现代终端」(Build $20/月):活以终端会话本身为中心时,它贴得更近。[7] Codex 自己的主场,是这本书的整条主线:**四面开工 + 云端异步** —— 已经付着 ChatGPT、又常做「派一批上云、回来收 PR」的人,它是那个不用额外付费就能用满的选项(和 Work 共用额度池)。[1] 组合的原则一句话:**入口跟着活走,订阅别叠着买** —— 每加一份订阅,都该对应一类你真在做、且 Codex 干得不如它的活;先把手里这份 ChatGPT 用满,再谈第二份。

> – 补充
>
> 本章全部价格、档位、模型名核对于 2026-07-17;这些是全书更换最勤的数字,读到时请以各家官网为准。判断框架 —— 三堵墙、三兄弟加力度、上下文即额度 —— 比数字活得久。

动手 · 用一周数据代替直觉:

- [ ] 记录一周你每次「用不了」的原因:额度见底、模型吃力、还是嫌慢 —— 月底对着三堵墙的表决定动不动钱包。
- [ ] 把批量的读 / 扫活降到 Luna、难活留给 Sol,跑一周看额度曲线的变化。
- [ ] 只在赶工的那一小时开 /fast,完事就关 —— 月底看 credits 那本账,判断这个「快」值多少钱。

# 参考.


01 OpenAI Codex 文档 · Pricing —— 档位:Free $0、Go $8、Plus $20、Pro 从 $100(选比 Plus 多 5× 或 20× 用量)、Business $20/人·月(年付,月付 $25)、Enterprise & Edu;API key 旁路按 token 计费、不含云端功能。Codex 与 ChatGPT Work 共用同一 usage 池。本地消息按模型和任务大小每 5 小时给一个范围:Plus 下 Sol 15–90、Terra 20–110、Luna 50–280;Pro 5× 约为 5 倍,Pro 20× 约为 20 倍;本地与云端共享同一个 5 小时窗口。撞额度时活能跑完当前轮,Plus/Pro 可买 credits 续用。截至 2026-07-17。OpenAI · Codex pricing

02 OpenAI Codex 文档 · Models —— Codex 有三个 GPT-5.6 模型:Sol(细节与打磨,复杂开放 / 高价值任务的旗舰)、Terra(务实全能,日常主力,原先给 GPT-5.5 的活的自然起点)、Luna(清晰可重复、量大的活:抽取、分类、转换)。默认 Power 设置 = Sol + medium 推理。推理力度:Light(应用)/ Low(CLI)、Medium、High、Extra High;Max 给单任务更多推理时间,Ultra 用 subagents 并行拆解复杂任务。gpt-5.2 与 gpt-5.3-codex 对 ChatGPT 登录已弃用。云端对话默认模型不可改。截至 2026-07-17。OpenAI · Models

03 OpenAI Codex 文档 · Speed —— Fast mode 把支持的模型提速 1.5×,按更高倍率吃 credits:GPT-5.6 / 5.5 为标准的 2.5×,GPT-5.4 为 2×;/fast on|off|status 切,或 config 里 service_tier = "fast"。仅 ChatGPT 登录可用(是 credit 特性);API key 走 API token 价、无 credit 倍率。GPT-5.3-Codex-Spark 是另一个更快、能力较弱的独立模型(研究预览,仅 ChatGPT Pro),有自己的用量限额,和 fast mode 不是一回事。截至 2026-07-17。OpenAI · Speed

04 Claude · Pricing —— Pro:月付 $20,年付折合每月 $17(一次预付 $200);Max:从 $100 起,可选比 Pro 多 5× 或 20× 用量两档;Pro 与 Max 均含 Claude Code。截至 2026-07-17。Claude · Pricing

05 Cursor · Pricing —— Hobby 免费;个人付费从 Pro $20/月起(页面另有 Pro+ / Ultra 更高档);Teams $40/座·月,Enterprise 定制。卖点为 Agent 请求、code review、cloud agents。截至 2026-07-17。Cursor · Pricing

06 GitHub Copilot · Plans —— Free / Pro $10 / Pro+ $39 / Max $100(每人·月),另有 Business 与 Enterprise;卖点原话「把活派给 Copilot,它创建一个 pull request」,且

Pro+ / Max 可把任务转派给 Claude、OpenAI Codex 等第三方编码 agent。截至 2026-07-17。GitHub Copilot · Plans

07 Warp · Pricing —— 自我定位「面向 agentic coding 的现代终端」;Free / Build $20 / Max $200(月付),另有 Business $50/人·月(年付 $45)与 Enterprise。截至 2026-07-17。Warp · Pricing

---

## 入口跟着活走,
## 订阅别叠着买.

# 收束 · 一个人,四面开工.

把整本书压成一句:Codex 的问题从来不是「它能不能写这段代码」,是「有多少活可以不守着做完 —— 而你还能确认它做对了」。聊天框是它最小的样子;四个面、云端沙箱、exec 与 CI、验收工作流把它撑成一个人指挥的异步车间。剩下的,是知道那条边界画在哪 —— 哪些活陪跑、哪些活上云、哪一步切走 —— 然后让它在你不看着的地方把活干完,而你只做机器替不了的两件事:决定派什么,以及确认它真做成了。

---

把十章压成一句:Codex 的问题从来不是「它能不能写这段代码」,是「有多少活可以不守着做完 —— 而你还能确认它做对了」。ChatGPT 里那个会写代码的模式,是它最小的样子;你付费买到的,是一个四面开工、能把一批活派上云再收回 PR 的异步编码 agent。

– I

## 回扣:四面与一台引擎.

回扣一遍这台机器怎么拼起来的。四个面各有主场 —— IDE 陪跑、CLI 进脚本、应用切本地与云、云端批量并行(第 2 章);底下是同一台引擎,AGENTS.md、config、skills、MCP 走到哪跟到哪(第 5、6 章)。[1] 第一个循环在 CLI 里跑通:说清目标、看计划、放它动手、看 diff(第 3 章)。放权靠两个旋钮加规则,再加一个替你把关的 reviewer(第 4 章)。差异点在云端:活离开你的机器、进隔离容器、回来是 PR,还能一次派一批(第 7 章);无人值守把这个推到极限 —— exec、SDK、CI、定时任务(第 8 章)。而托住这一切的是验收:检查、复审、回归,凭据在手才算完(第 9 章)。选对节奏,比选贵的档更省(第 10 章)。[1]

• • •

— II

## 你只做机器替不了的两件事.

四面成形之后,你的工作只剩两件,恰好是机器替不了的两件:**决定派什么**,和**确认它真做成了**。前者是品味和取舍 —— 哪件活值得派、派到哪一面、边界画在哪;后者是这本书反复回来的那句 —— 给它一个能自己跑的检查,要证据,不要断言。[2] 派活的人已经太多,这两件事上的功夫,才是四面开工的时代里你真正的手艺。

• • •

— III

## 边界,和一把不过期的尺.

剩下的是画边界。哪些活自己盯:不可逆的、高风险的、要品味的。哪些活派出去:可验证的、可并行的、重复的 —— 这些正是云端的甜点位。哪一步切走:该在编辑器里陪着改的、长在 GitHub 流程里的、以终端本身为中心的,入口跟着活走(第 10 章)。四面不是让你四处忙,是让你在对的面上派对的活。最后说句实话:这本书会过期,而且很快 —— Codex 以周为单位在变,模型名、命令、档位,书里每个数字都标了核对日期,过期请以官方文档为准。但这把尺子不过期:拿到任何一个新特性、新入口,还是那三问 —— **它省什么,什么活用它,怎么验收**。问完这三问,你就不需要下一本教程了。

# 参考.

01 OpenAI Codex 文档 · Quickstart / Codex cloud —— Codex 的四个入口(ChatGPT 桌面应用与网页、CLI、IDE 扩展、云端)连着同一台引擎,AGENTS.md、config、skills、MCP 跨入口通用;云端把活委派给隔离环境、跑完交回可开 PR 的 diff。截至 2026-07-17。OpenAI · Codex cloud

02 OpenAI Codex 文档 · Best practices —— 别停在让它改;给它一个它能自己跑的检查、要证据不要断言,让它写测试、跑检查、审自己的 diff,再由你接受。原话:「在 OpenAI,Codex 审 100% 的 PR。」截至 2026-07-17。OpenAI · Best practices

---

## 一个人,四面开工:
## 你派活,你验收,其余交给它们.

Codex

# 一个 agent，四面开工。

「ChatGPT 里写代码的」只是它最小的入口。它其实是一个派到云端、四面开工、交回 PR 的异步编码 agent。

扫码在线阅读

**作者** · AKLMAN

Aklman · 文库 —— 写给已经订阅 AI 工具、却只用到一小部分的人；每本短书讲清一份订阅里该学、该跳过、该换入口的部分。写代码，也写字。

这是静态 PDF 快照；网页版阅读体验更佳，内容持续更新、数据更及时准确。

library.aklman.com · x.com/aklman2018 · github.com/aklmans

| 章节 | 字数 | 更新 | 版本 |
| --- | --- | --- | --- |
| 11 | 1.4 万字 | 2026.07 | PDF 1.0 |