— 技术 · 实践

WARP

终端形状的元工具

不只是更漂亮的终端 —— 一台带 agent、接 MCP 的元工具:10 分钟首胜开始,逐章跑通一件真事。

目录.

引子 · 这台元工具,你只用了一成.

你把 Warp 当一个更好看的终端用 —— 敲命令、看输出、再敲。这本书从一组对照开始:同一件真事,逐条手敲 vs 一句交办。结论不替你下,由你自己得出;然后下一章带你 10 分钟装好,拿到第一次成功。

你大概是这样用 Warp 的:打开它,敲一条命令,看输出成一块漂亮的 block,再敲下一条。比上一个终端顺手,配色也好看。一年下来几千条命令,几乎全是你一个字一个字敲进去的。那个输入框很好用,好用到你忘了它只是这台机器的一个入口。同一个窗口后面还连着一个能自己动手的 agent、一个把做过的事存下来复用的 Warp Drive、一个接到你终端之外的 MCP 插口,和能把活丢到后台去跑的云端 agent。这本书讲的,就是这些你付了钱、却只当壁纸用的部分 —— 而且不讲道理给你听,直接带你各跑通一遍。

— I

它是元工具,不是终端.

普通终端跑你打的字;Warp 在这之上多了三样,把「你亲手开的工具」变成「你能交办的工具」。哪三样:一个能就着你终端的上下文、自己跑命令并纠错的 **agent**;一个把跑通的命令、文档、变量存下来复用的地方(**Warp Drive**);一个把它接到你终端之外的数据和工具上的插口(**MCP**)—— 再加上能把活儿丢到后台和云端跑的 agent。底下仍然是那台终端,但有了这三样,用法从「逐条敲」变成「说清要什么,让它做完,你来审」。[3] 「元工具」不是我替它贴的标签,是从它的结构里读出来的;2026 年 Warp 把自己从「更快的终端」重定位成 agentic development environment、并把客户端开源(AGPL,OpenAI 是创始赞助方),只是这条结构的官方注脚。[1] [2] 这本书不给它写注脚 —— 下一节直接让你看一眼差别。

|| Warp 是 | Warp 不是 | | :-- | :-- | | 终端形状的 agentic workbench | 官网功能翻译或快捷键大全 | | 命令、文件、脚本、系统、开发环境的主入口 | 替代所有 IDE、浏览器、Notebook、运维平台的万能层 | | 让命令行工作流可见、可复用、可派发的系统 | 让你不用审命令、不管权限、不算成本的自动驾驶 |

· · ·

– II

同一件事,两种走法.

「为什么是元工具」不该靠论证,该靠对照。挑一件谁都躲不掉的杂事 —— 磁盘又满了,找出那些一个月没动过、单个超过 1GB 的大文件,确认没用就清掉。同一件事,两种走法:

普通终端：你来当发动机

```
$ find ~ -size +1G          # 参数是 -size +1G 还是 -s?搜一下
$ find ~ -size +1G -mtime +30   # 跑完 200 行输出,分不清哪个能删
$ du -sh ~/Downloads/* | sort -rh | head   # 换个思路再试
$ rm ~/Downloads/big-backup.zip   # 赌一把,没有回收站
# 全程约 20 分钟;下周磁盘再满,从头再来
```

边搜边拼，错了重来，下周再来一遍

命令你其实都会,只是每次现拼参数;输出你自己读;判断你自己下;做完什么也没留下。

WARP：一句话交办，你只管审

```
> 找出我 home 目录下 30 天没动过、超过 1GB 的文件,
  按大小列出来,标出哪些大概率能删(缓存、旧下载、
  安装包),先别删任何东西,等我确认。

  agent: 先跑只读的 find + du,给你一张带建议的清单
  你:    确认两个能删,它执行,报告释放了 14.2 GB
  你:    把这套「月度磁盘清理」存成 Workflow,
         下个月一句话再来一次
```

交办 → 审命令 → 跑 → 存成下周能再用的东西

你下的功夫只剩两步:把活说清楚,把它要跑的命令审一遍。跑通的那一次被存进 Drive,从一次性变成可复用。

右边那一幕就是这本书对「元工具」的全部论证:不是它替你打字快了多少,而是同一件事从「你亲手拼」变成「你交办、它执行、成果留下」。读到这里如果你想去装,直接去第二章 —— 十分钟内你会亲手跑出右边这一幕。

• • •

— III

这本书怎么用.

这本书写给两种人。**还没装的人**:第二章是给你的一条 10 分钟路径 —— 安装、登录、三个必要设置、第一次交办一件真实的小事、把结果存下来,跑完再决定要不要继续。**装了只当好看终端用的人**:从第三章开始,每章一条完整工作流,把你停在的那一层逐层往上抬 —— 底座、交办、派活、沉淀、接外部、管控、上云。[4] 每一项核心能力,只回答三个问题:省什么、什么活用、什么时候别用。每章的结尾不是总结,是一张验收单 —— 你做得出那几件事,这章才算读完。官网叙事在这本书里只当脚注,不当论据;每个价格、额度、版本都带日期,过期了你能自己去核。

> — 补充
>
> 立场说在前面: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;我不接 affiliate,推荐一样东西会告诉你为什么,也会告诉你它的代价。

动手之前先照一次镜子 —— 看看你这台元工具到底用到了第几成:

01 **往回翻两周,挑 10 条命令**

打开你的命令历史,往回翻两周,挑出 10 条你手动敲、而且不是第一次敲的命令。

02 **给每条标一列:本可以怎么省**

对每条问一句:这件事本可以让 agent 一句话做完吗?能的话,本可以存成一个 workflow 下次直接调吗?把答案记成「能 / 不能」两列。

03 **数一数有几条本可以不手敲**

数第一列有几个「能」。这个数就是你现在浪费掉的那部分元工具,也是读完这本书之后的对照组。

做完你多半会发现,不是 Warp 少了什么能力,是你一直只站在那个输入框前面,一个字一个字地敲。

- [ ] 我知道 Warp 终端部分不登录就能用,agent 部分需要什么,第二章会带我配好。
- [ ] 我数过自己有多少条命令本可以交办或沉淀 —— 那个数字是我读这本书的对照组。
- [ ] 我准备好下一章真装真跑:10 分钟,拿到第一次交办成功。

— 引用与参考

参考.

01 Warp Docs · Getting started with Warp and Oz —— 截至 2026-08-06,官方定义 Warp 为开源 Agentic Development Environment:现代高性能终端加 agent,用于 build / test / deploy / debug;Warp agent 由 Oz 提供本地与云端编排。Warp Docs · Warp and Oz

02 GitHub · warpdotdev/warp —— 截至 2026-08-06,Warp 客户端源码公开在 GitHub;README 称 Warp 是「born out of the terminal」的 agentic development environment;UI framework crate 采用 MIT,其余仓库采用 AGPL v3。GitHub · warpdotdev/warp

03 Warp Docs · Warp Agents overview —— 本地 agent 嵌在终端里,能写代码、debug、跑命令、自动化任务,带代码库、Warp Drive 与连接工具上下文;你仍控制每个动作。Warp Docs · Agents overview

04 Warp quickstart —— 官方快速上手把全程压在约 10 分钟:安装、跑第一条命令看 Blocks、试输入编辑、向 agent 提第一个问题;终端部分不登录也能用,账户为可选。本书第二章的首胜路径在这一官方骨架上,补上登录、初始设置与「第一次交办真实任务」的完整闭环。Warp Docs · Quickstart

---

终端只是它的形状,
元工具才是它的里子.

首胜 · 10 分钟拿到第一次成功.

安装、登录、三个必要设置,然后把第一件真实的活交办出去 —— 让 agent 只读地分析你机器上最占地方的东西,跑通、看懂、存下来。10 分钟后你不再是装过 Warp 的人,是用它做成过一件事的人。每一步精确到按什么键;说不准的地方标「待核」,不编。

这一章没有概念,只有动作。目标:10 分钟后,你的机器上多一个装好的 Warp,和一个你亲手交办成功的任务。跟着做,别跳步。

— I

装上它(约 2 分钟).

按你的系统选一条路。装完打开,你会看到一个可以直接输入的终端 —— 到这里为止不需要账号,终端的全部基本功(Blocks、补全、历史)开箱即用。[1]

MACOS

```
# 有 Homebrew:
brew install --cask warp

# 没有的话:从 warp.dev 下载,把 Warp 拖进 Applications
```

WINDOWS

```
# 有 WinGet:
winget install Warp.Warp

# 没有的话:从 warp.dev 下载 .exe 安装包运行
```

LINUX

```
# Debian / Ubuntu:从 warp.dev 下载 .deb 后
sudo apt install ./warp-terminal.deb

# 其他发行版:下载页还有 .rpm / .tar.zst / AppImage
```

三条安装路径,任选其一;包管理器方式将来升级最省心

打开后先做一件 10 秒钟的事,确认底座活着:敲 ls 回车。输出会收进一个独立的 block —— 命令和它的结果包在一起,可以单独复制、单独回看。这就是 Warp 与传统终端的第一眼差别,也是后面一切的地基。

• • •

— II

登录,和 agent 的三条路(约 2 分钟).

终端不用登录,但 **agent 是账户功能**:每一次交办都走 Warp 的云端 harness,credits 记在你的账号上,所以用 agent 之前先点右上角登录(首次会开浏览器走授权,回来就登上)。[1] [2] 登录之后,让 agent 跑起来有三条路,按你的现状选:

|| 你的现状 | 走哪条 | 怎么做 | | :-- | :-- | :-- | | 手里有 OpenAI / Anthropic / Google 的 API key | BYOK(最顺) | 打开 Settings(⌘,),搜 API keys,把 key 贴进去;模型选择器里带钥匙图标的模型就会走你的 key,provider 直接计费,不耗 Warp credits | | 没有 key,想先零门槛试 | 按量充值(PAYG) | Free 档支持按 pay-as-you-go 价格买 credits;在 Settings > Billing and usage 里充一小笔,够你跑完这本书的前几章 | | 确定会常用 | Build 档(正经答案) | 月付 $20 / 年付折算 $18,含 1500 credits/月;同一个 Billing and usage 页点 Upgrade(价格截至 2026-08-06,选型章细讲) |

> — 注意
>
> 一个常见的误会先掐掉:**你订阅了 ChatGPT 或 Claude,不等于你有能接进 Warp 的东西**。官方明示:OpenAI 和 Anthropic 的消费级订阅不允许接进第三方客户端,要用自己的账号只能走 API key(BYOK);唯一的消费订阅通道是 xAI 的 SuperGrok。[3] 两条补充:pricing 页 Free 档那行「Bring your own AI inference*」的星号脚注在页面上不展开,个别边界(如组织规模)以 BYOK 文档页为准并标「待核」;按量充值的实际入口文案以你账号里看到的为准。

• • •

— III

三个必要设置(约 2 分钟).

设置项很多,只有三个和「第一次就交办成功」直接相关。其余的 —— 尤其是主题 —— 跑通之后再回来玩。

01

确认 auto-detection 开着,认识 (autodetected) 标记

Settings > Agents > Warp Agent > Input 里, `Autodetect agent prompts in terminal input` 新用户默认开。它的作用:你在普通输入框打一句人话,提交前 Warp 用 magenta 色标 `(autodetected)` 告诉你这句会发给 agent;打的是 shell 语法就直接当命令跑。认错了就按 `⌘I` (Windows / Linux: `Ctrl+I`)手动改判,或加 `!` 前缀强制当命令跑。[5]

02

把命令面板和 Drive 的键位按进手指

只记四个: `⌘P` 命令面板(搜一切功能、workflow、设置)、`⌘\` 开关 Warp Drive、`^R` 命令搜索(历史 / workflow / agent 对话统一搜)、`⌘,` 设置。Windows / Linux 上对应 `Ctrl+Shift+P` 系,完整表在 Settings > Keyboard shortcuts。[4]

03

主题:选一个暗色或亮色,然后关掉设置页

认真的 —— 主题不改变你做事的速度。挑一个顺眼的(<Kbd>^⌘T</Kbd> 打开主题选择器),然后回来继续。这本书后面不会再提外观。

• • •

— IV

第一次交办(约 3 分钟).

现在把一件真实的活交出去。选一件只读、低风险、人人都有的:让 agent 看看你磁盘上最占地方的是什么。这个任务故意只读 —— 第一次建立信任的方式,是看它干活,而不是让它改东西。

01

按 ⌘↩ 开一个 agent 会话

在终端输入框直接按 `⌘↩`(Windows / Linux: `Ctrl+Shift+Enter`)。输入框会展开成一个带模型选择器的对话视图 —— 这就是 Agent Mode。[1]

02

把这句话贴进去,回车

用下面这个 PromptBox 的示例 tab 里那句(复制按钮直接拷)。它要求 agent 只读、给命令先给你看 —— 第一次交办的两个好纪律,从第一天就养成。

03

看它跑:每条命令你都有机会说不

agent 会拆步骤:先跑 du / df 这类只读命令,读输出,再决定下一步。每条命令跑前都会摆出来等你确认 —— 看一眼,认得就放行,不认得就拒绝并问它「这条是干什么的」。这个过程本身就是教学。

04

验收:你得到了一张清单,和一个判断

成功的样子:它给你一张按大小排的清单,标出哪些大概率能清(缓存、旧下载、安装包),哪些别碰 —— 并且没有删任何东西。你 3 分钟前还不知道答案的问题,现在有答案了。

提示词

第一次交办:只读、给命令先看

```
只读分析,不要改任何文件:
1. 列出 <目录> 下最占空间的 10 个文件/目录,按大小排序;
2. 每条打算跑的命令先给我看,我确认再跑;
3. 标出哪些大概率可以清理(缓存/旧下载/安装包),哪些别碰;
4. 最后给我一段 100 字以内的结论。
```

```
只读分析,不要改任何文件:
1. 列出 ~ 下最占空间的 10 个文件/目录,按大小排序;
2. 每条打算跑的命令先给我看,我确认再跑;
3. 标出哪些大概率可以清理(缓存/旧下载/安装包),哪些别碰;
4. 最后给我一段 100 字以内的结论。
```

没有 WARP 的这 3 分钟

打开访达或资源管理器,按大小排序,逐层点进去看;或者打开搜索引擎查「du 按大小排序」「mac 查看大文件」,拼出一条命令,跑,发现输出不对,再查。

刚才这 3 分钟

你用一句话换来了:一张排好序的清单、每条命令的解释权、和一个没被改动过的文件系统。这就是「交办」的最小完整形态 —— 后面十一章都在给它加宽加深。

• • •

— V

存下来(约 1 分钟).

跑通了别让它过去 —— 让它变成资产。在 agent 给出的那段结果上,用 Block Actions(块右键菜单)或对话里的 `Save as Workflow` 把它存成一个带参数的 Workflow;按 ⌘\ 打开 Warp Drive,确认它躺在里面。下个月磁盘再满,你不用再写那句话 —— 从命令面板调出这个 workflow 就行。这个「跑通 → 存下 → 复用」的动作是第六章的主题,今天先完成第一次。[1]

> — 实践提示
>
> 如果 agent 给出的命令你不认识:别放行,直接在同一个对话里问「逐行解释这条命令,特别是 - 开头的参数」。它读着自己写的命令解释给你听,这是 Agent Mode 最便宜也最常被人忽略的用法。

到这里,验收这一章 —— 逐条都能勾上,你才算拿到了首胜:

- [ ] Warp 装好了,ls 的 output 收进了一个独立的 block。
- [ ] 我登录了账号,并且知道我的 agent 走哪条路:BYOK、按量充值,还是 Build。
- [ ] 我认得 (autodetected) 标记,会用 ⌘I 改判、会用 ! 前缀强制当命令跑。

- [ ] 我用 ⌘↩ 开了第一个 agent 会话,交办了一个只读任务,逐条审了它要跑的命令。

- [ ] 我拿到了那张「最占空间」的清单,并且文件系统没被改动。

- [ ] 这次交办存进了 Warp Drive,我能用 ⌘\ 再看到它。

参考.

01 Warp quickstart —— 截至 2026-08-06,官方上手路径:安装(brew install --cask warp / winget install Warp.Warp / Linux 各包)→ 跑第一条命令看 Blocks → 试输入编辑(Shift+Enter 多行)→ 自动补全(→ 接受)→ ⌘↩(macOS)/ Ctrl+Shift+Enter(Windows/Linux)向 agent 提第一个问题;「可以不注册账户,直接开始用」—— 终端功能开箱即用。Warp Docs · Quickstart

02 Warp Docs · Credits —— 截至 2026-08-06:Free 档不含 Warp Agent 自带用量,要用 agent 需 BYOK / custom endpoint / SuperGrok / X Premium,或升付费档;credits 分 AI / compute / platform 三桶,同一额度池;普通 shell 命令不消耗 credits;余额与用量在 Settings > Billing and usage。Warp Docs · Credits

03 Warp Docs · Bring Your Own API Key —— BYOK 支持 OpenAI / Anthropic / Google,Free 与符合条件的付费档可用(个人及 10 人以内组织);key 只存本机;官方明示:ChatGPT / Claude 的消费级订阅不能像 SuperGrok 那样接入 Warp,要用自己的 OpenAI / Anthropic 账号需走 API key 的 BYOK。Warp Docs · BYOK

04 Warp Docs · Keyboard Shortcuts —— 截至 2026-08-06:⌘P 命令面板、⌘\ Warp Drive、^R Command Search、^⇧R Workflows、^` Generate(自然语言生成命令);自定义键位在 Settings > Keyboard shortcuts,或按 keysets 仓库的文件方式批量导入。Warp Docs · Keyboard Shortcuts

05 Warp Docs · Terminal and Agent modes —— auto-detection 对新用户默认开启(老用户默认关闭),提交前以 magenta 的 (autodetected) 标注分流方向;两个开关在 Settings > Agents > Warp Agent > Input;⌘↩ 起新 agent 会话,⌘Y 继续上一段。Warp Docs · Terminal and Agent modes

---

10 分钟前你装的是一个终端,
现在你有一个能交办的搭档.

底座 · 留下,还是换回普通终端.

Blocks、Command Search、命令面板、补全 —— 九成用户停在这一层。这章用一条工作流把底座用明白:找回昨天跑过的那条命令,复用它,顺手给高频命令加书签。最后老实回答:什么情况下 iTerm、tmux、普通终端反而更对。

这一章讲你已经在用的那层 —— Blocks、现代编辑、补全、命令面板。九成 Warp 用户只用到这里就停了。它确实比上一个终端顺手,但这章不罗列功能,而是用一条完整工作流把底座用明白:找回你昨天跑过的那条命令,复用它,把高频的几条固定到手指边。然后老实回答:这层里哪些值得改习惯,以及什么情况下普通终端反而更对。

— I

主线:找回昨天那条命令.

场景:昨天下午你跑通了一条命令 —— 一串带参数的 rsync,或者一条拼了很久的 ffmpeg。今天你要再跑一次,只记得大概。传统终端里你要么翻 scrollback,要么 history | grep 猜关键词。在 Warp 里这件事有一个确定的动作序列,练一遍就会:

01

按 ^R,输入你记得的任何碎片

Command Search 是模糊匹配、按相关度排序的 —— 打 `rsync` 加目录名的一个音节,通常前几条就是它。它搜的不只是命令历史:保存的命令、workflows、连 agent 对话历史都在同一个框里。[2]

02

搜太宽就用过滤器收口

前缀 `history:`(或 `h:`)只看终端历史,`prompts:`(或 `p:`)只看存下的自然语言任务,`ai_history:`(或 `a:`)只看 agent 对话。昨天那条是亲手敲的,所以 `h: rsync` 最准。

03

回车前决定:原样重跑,还是改完再跑

选中那条命令回车,它进输入框而不是直接执行 —— 这是关键设计:你可以先改路径和参数,确认无误再跑。找旧命令最常见的坑就是原样重跑了一条参数已经过时的命令。

04

一周用三次以上的,当场加书签

跑完后选中那个 block,按 `⌘B` 加书签;下次 ^R 或命令面板里它排在最前。到这里工作流闭环:找回 → 复用 → 固定。还不值得为它建 workflow —— 那是第六章的事,等它重复到烦再说。[1]

传统终端里的同一幕

history | grep rsync,出来四十条;肉眼筛出像的那条,复制,粘贴,发现路径是上周的旧目录;改完再跑,又忘了当时那个关键参数是什么。前后十分钟,还不敢确定跑的就是昨天那条。

刚才这一分钟

^R,两个关键词,回车,改一个参数,跑。命令进了输入框让你审过,跑完顺手 ⌘B —— 下次连搜都不用搜。省的不是十秒钟,是「不确定自己跑的对不对」那份心虚。

– 实践提示

把 Block 当抓手用:一块跑完,选中它,右键菜单(或快捷键)能单独复制输出(⌥⇧⌘C,不带你打的那行命令)、单独复制命令(⇧⌘C)、带格式分享(⇧⌘S)。给 agent 喂上下文时,喂一块输出远比贴整屏 scrollback 有效 —— 这是底座层和后八章之间的桥。[1]

• • •

– II

快捷键只背一张表.

底座层的键位很多,值得进肌肉记忆的只有这些 —— 按 2026-08-06 官方页面逐字核过;Windows / Linux 上 ⌘ 系对应 Ctrl+Shift 系,完整表在 Settings > Keyboard shortcuts,也可以改:

键	动作	什么时候用
^R	Command Search	找任何跑过、存过、聊过的东西
⌘P	Command Palette	找功能、设置、动作本身
^⇧R	Workflows	直接进已沉淀的参数化命令(第六章)
^`	Generate	记得要干什么、不记得语法,让 AI 拼命令
⌘\	开关 Warp Drive	看你的 workflow、notebook、prompt 库存
⌘D	向右分屏	一边看日志一边跑命令
⌘B	给选中 block 加书签	本节主线第四步
⇧⌘C / ⌥⇧⌘C	复制命令 / 复制输出	喂文档、喂同事、喂 agent

[3]

> — 注意
>
> 一个键位陷阱先说清:⌘I **不是全局键,它的含义随语境变**。选中一个 block 时它是「重新输入这条命令」;在输入框里它是命令与 agent 模式的切换开关;在一个跑着的交互式会话里,它是「把 agent 拉进来接管」。三处都是官方文档写的,谁也覆盖不了谁 —— 所以别背「⌘I = agent」,要背「⌘I = 看语境」。[3]

表里没有主题和外观,是故意的:输入编辑的语法高亮和多光标、补全、命令面板这三样每天都省你的时间,值得练;主题不改变你做事的速度,挑一个顺眼的就该收工 —— 第二章已经说过了,这里只是再确认一遍。

– III

普通终端就够的时候.

如果你只用到这一层,一个普通终端、iTerm 或 tmux 就够 —— 不必为底座纠结。说句公道话:底座这层 Warp 不是全面赢。论纯定制深度,iTerm2 的触发器、smart selection、Python 脚本 API 攒了多年,Warp 给不齐;论会话持久化和多路复用,tmux 仍是标准,Warp 的 tmux 支持只能算够用。[4] 真正把 Warp 和它们拉开的是下一章的发动机,不是这层底座。所以判断很简单:只为好看的 block 和主题换工具,不值;为后面那台发动机换,才值。

场景	先用	原因
远程服务器上一条状态命令	普通终端 / SSH	少依赖、少同步、默认可用;若要在远程跑 agent,第十章有更新的答案
本地项目反复跑测试并读输出	**Warp Blocks**	输出可定位、可引用、可丢给 agent
知道命令,只是忘了参数	^R / 补全	不需要多步 agent
记得目的、忘了整条语法	^` Generate	单条命令的生成,不必开 agent 会话
需要解释一段复杂输出	**Warp Agent**	开始需要语义判断
重复命令链且参数固定	**Workflow**	别让 agent 每次现编

什么时候不需要 Agent:命令你已经记得;输出你一眼能看懂;动作是只读、小步、能立即重跑;或者这台机器不能承受 agent 误判。此时底座层就够了。验收这一章 —— 主线走通,键位上桌,边界认清:

- [ ] 我用 ^R 找回了一条昨天跑过的命令,先在输入框里审过再跑。
- [ ] 我会用 history: / prompts: / ai_history: 三个过滤器收口搜索结果。
- [ ] 我给一条高频命令加了书签(⌘B),下次不用搜。
- [ ] 我说得清 ⌘I 在三种语境下分别是干什么的,不会把它当全局 agent 键。
- [ ] 我单独复制过一块输出喂给别人或 agent,而不是贴整屏。
- [ ] 我写下一个我宁可用普通终端的场景(远程 / 离线 / 容器 / 重定制),记住那条线。

参考.

01 Warp Docs · Blocks —— 截至 2026-08-06,Block 把一条命令和它的输出合成可定位单元,可单独复制命令(SHIFT-CMD-C)或输出(ALT-SHIFT-CMD-C)、跳到输出开头、重新输入、分享(SHIFT-CMD-S)、加书签(CMD-B)、搜索、过滤、作为 agent 上下文。Warp Docs · Blocks

02 Warp Docs · Command Search —— 截至 2026-08-06,^R 打开统一搜索:终端输入历史、保存的命令、workflows、Terminal/Agent 对话历史;支持 history: / h:、prompts: / p:、ai_history: / a: 过滤器,模糊匹配、按相关度排序;Settings > Features 里有「Show Global Workflows」开关。Warp Docs · Command Search

03 Warp Docs · Keyboard Shortcuts —— 本章键位按 2026-08-06 页面逐字核:^R Command Search、^⇧R Workflows、^` Generate、⌘\ Warp Drive、⌘P Command Palette、⌘D 向右分屏、⌘B 给 block 加书签、⇧⌘S 分享 block、⌥⇧⌘C 复制输出、⇧⌘C 复制命令。注意 ⌘I 是语境键:选中 block 时是「Reinput Selected Commands」,在输入框 / 编辑器里是「Inspect Command」与 agent 模式切换,在交互式会话里是把 agent 拉进来 —— 本书按语境分别描述,不把它当一个全局键。Warp Docs · Keyboard Shortcuts

04 Warp Docs · Supported shells / Downloads —— Warp 支持 macOS、Linux、Windows,终端层依赖 shell integration 与本机 shell;远程、容器、离线环境仍要保留普通终端判断(第十章会修正这条建议的一半:Warp Agent CLI 现在能随 SSH 带进远程机器)。Warp Docs · Supported shells

---

底座让你留下,
发动机才让你不走.

发动机 · 把一个真实报错修到好.

Agent Mode 不是更聪明的补全,是一个会读你终端状态、会多步动手的发动机。这章从头到尾跑通一件事:dev server 起不来,从交办开始,看它读报错、提命令、你审、它跑、出错自己改,直到修复验收。附任务合同模板和喂上下文的方法。

上一章的底座只能到「找回命令」为止。这一章开始是 Warp 真正的发动机:一个会读你终端状态、把活拆成多步、跑命令、读输出、出错自己改的 agent。光讲机制没用,所以这章从头到尾跑通一件真事 —— dev server 起不来,报 EADDRINUSE,从交办开始,到修复验收结束。你盯着看:它每一步干什么,你每一步审什么。

— I

主线:把 EADDRINUSE 修到好.

场景:`npm run dev` 起不来,终端里躺着一块报错。传统路径是:复制错误、开浏览器、搜、在三个 Stack Overflow 答案里挑一个、回来试、不行再搜。这次我们把它交办出去,一步一步看。[1]

01

⌘↩ 开会话,@ 指上报错那块

在终端里按 ⌘↩ 起一个新 agent 会话。第一件事不是打字,是喂上下文:用 `@` 把刚才那块 EADDRINUSE 输出指进 prompt —— 让它就着真实报错答,而不是凭印象答。然后贴上下面 PromptBox 示例 tab 里那段话。

02

审它的计划,不审它的措辞

它先回一份计划:查谁占了 3000 端口、判断进程身份、给你建议、等你确认再动手、最后重跑验证。你审三件事:方向对不对、第一步是不是只读、它有没有打算跳过你直接 kill。都对,才放行。

03

每条命令摆出来,逐条放行

它要跑 `lsof -nP -i :3000` —— 只读,放行。它读完输出,告诉你占端口的是上周那个没关干净的 node 进程,然后提议 `kill 41782` —— 这一步是写操作,它会停下来等你。你确认那是自己的残留进程,放行。这就是「交办」的真实手感:方向盘在你手里,油门在它脚上。

04

出错它自己改,改完你要复验

如果 kill 之后端口仍占着,它会自己读新报错、换 `kill -9` 或换思路再试 —— 自纠错是发动机的标配。[2] 但「它说好了」不算好:最后它重跑 `npm run dev`,你要亲眼看到 `Ready on http://localhost:3000` 才算验收通过。验收标准在交办那句话里就写死了。

提示词

EADDRINUSE 交办词:只读先行,动手先问

```
@ <报错那块输出> dev server 起不来。
1. 只读排查:找出是什么占了端口,它是什么进程;
2. 每条命令先给我看我确认再跑;
3. 杀进程之前必须问我,我说杀才杀;
4. 修好后重跑启动命令,把「起起来了」的证据给我看。
```

```
@上面那块 EADDRINUSE 输出 dev server 起不来。
1. 只读排查:找出是什么占了 3000 端口,它是什么进程;
2. 每条命令先给我看我确认再跑;
3. 杀进程之前必须问我,我说杀才杀;
4. 修好后重跑 npm run dev,把 Ready on localhost:3000 给我看。
```

一次真实交办的样子(删节版 transcript —— 注意每条命令前的确认点,和最后那条验收证据):

```
$ npm run dev
> my-app@0.1.0 dev
> next dev
Error: listen EADDRINUSE: address already in use :::3000
    at Server.setupListenHandle [as _listen2] (node:net:1940:16)

──── Agent 会话(⌘↩)──────────────────────────────────────────
你: @上面的报错输出 dev server 起不来。只读排查,杀进程前问我。

Agent: 计划 —
  1. lsof 查 3000 端口占用(只读)
  2. 判断进程身份
  3. 给你建议,等你确认再动手
  4. 重跑 npm run dev 验收

Agent: 执行 ⟶ lsof -nP -i :3000          [你: 放行 ✓]
  COMMAND  PID  USER   FD  TYPE  NAME
  node    41782  you   23u  IPv6  TCP *:3000 (LISTEN)

Agent: 执行 ⟶ ps -p 41782 -o pid,etime,command   [你: 放行 ✓]
  PID    ELAPSED  COMMAND
  41782  6-04:12  node /Users/you/proj/node_modules/.bin/next dev

Agent: 结论 — 是 6 天前启动的 next dev 残留进程,占着 3000。
  建议 kill 41782。这是写操作,等你确认。

你: 杀。

Agent: 执行 ⟶ kill 41782                  [你: 已确认 ✓]
Agent: 执行 ⟶ npm run dev                  [你: 放行 ✓]
  > next dev
  ✓ Ready on http://localhost:3000

Agent: 验收 — Ready on http://localhost:3000,端口已释放,服务正常。
```

EADDRINUSE 从交办到验收的完整对话(删节)

浏览器路径的同一晚

复制错误、新开标签页、搜「EADDRINUSE 3000」;第一个答案说 killall node —— 差点把你另一个项目的 dev server 也带走;第二个答案给的 lsof 参数在你的系统上报错;四十分钟后服务是起来了,你没搞懂刚才哪一步真正起了作用,下周再犯还是从头搜。

刚才这四分钟

它读的是你那块真实报错,查的是你这台机器的端口,杀的是你能确认的进程,验收证据摆在你面前。全程你没有离开终端,并且每一步你都说了「可以」才发生 —— 这就是发动机和补全的区别:补全猜你这一行的下半句,agent 接你整件事的意图。

– 要点

可截图判断:Warp Agent 省的是把意图翻成命令和排障步骤的成本;它增加的是你审查命令的责任。

• • •

– II

喂上下文,决定它准不准.

主线里它一次到位,不是它神,是那块 @ 进去的报错喂得准。它不会自动读你整个会话 —— 你喂多少准上下文,它就回多准。三层上下文要会喂: 一是**当场指**:@ 把文件、某块输出、某段选区贴进 prompt,让它就着具体的东西答。二是**代码库索引**:Warp 索引你 Git 跟踪的代码帮它理解项目,不在 Warp 服务器存你的代码,尊重 `.gitignore` / `.warpindexingignore` 等忽略文件;注意它消耗 credits,且 SSH / WSL 会话暂不支持。[4] 三是**Rules**:项目里放全大写的 `AGENTS.md`(`/init` 生成),写清栈、规范、禁区;子目录的 `AGENTS.md` 自动生效且优先于根目录;已有的 `CLAUDE.md` / `.cursorrules` 可以直接链接,不用重写。命中的规则会在对话里标来源,你看得见它遵守了哪条。[3]

— 实践提示

卡壳时先看你喂了什么,再换措辞。一条好 prompt 的骨架:@ 指上相关文件或输出 + 一个明确目标 + 一句验收标准 + 「先给命令,我确认再跑」。同一件事,「光打一句话」和「@ 文件 + AGENTS.md」跑两次对比,那个差就是你以后信不信它的依据。

• • •

— III

任务合同:交办大活的模板.

EADDRINUSE 是小活,一句话够。活一大 —— 要写入、跨目录、有禁区 —— 就把下面这份合同填进去。它把成熟工作流的四个阶段(生成 → 审查 → 执行 → 验收)压成一段 prompt:

提示词

Warp Agent 任务合同模板

```
目标:<一句话说清要完成什么>
目录:<只允许在这个目录工作>
上下文:@<文件/Block/日志/截图>
写入边界:<只读 / 可改哪些文件 / 禁止改哪些文件>
禁止命令:rm -rf、sudo、curl | sh、生产数据库写入、deploy、secret 输出
执行方式:先给计划和命令;需要写入、删除、网络、sudo、生产操作时等我确认
验收标准:<测试命令 / 统计结果 / diff 要求 / dry-run 输出>


目标:把过期 30 天以上的临时上传文件清理掉,释放磁盘
目录:只允许在 ./storage/tmp 里工作
上下文:@scripts/cleanup.sh @df -h 的输出
写入边界:只能删 ./storage/tmp 下的文件,禁止改其他目录、禁止改任何源码
禁止命令:rm -rf、sudo、curl | sh、生产数据库写入、deploy、secret 输出
执行方式:先给计划和命令;需要写入、删除、网络、sudo、生产操作时等我确认
验收标准:先用 find 列出待删清单和总大小,我确认后再删;删完给出释放的空间和 df -h 对比
```

阶段	你要它做	你审什么
生成	先解释计划与命令,不执行	目标是否对、目录是否对、会不会写入
审查	列出风险命令和 dry-run 方式	是否碰生产、密钥、删除、网络、sudo
执行	小范围先跑,通过再全量	退出码、输出、diff、样本
验收	按你给的标准证明完成	测试、统计、文件 diff、回滚方案

• • •

— IV

它会在哪儿跑偏.

它会自纠错,但也会自信地跑错 —— 意图越模糊、会话越长、动作越不可逆,越容易出事。

危险信号	为什么危险	默认处理
`rm -rf`, `find ... -delete`	批量删除不可逆	必须人工确认,先列清单
`sudo`, `chmod -R`, `chown -R`	扩大权限或改坏系统状态	解释影响范围,禁止自动跑
`curl ...	sh`, `wget ...	bash`	影响真实数据	SELECT 样本、事务、备份、人工批准
生产数据库 UPDATE / DELETE / DROP	影响真实数据	SELECT 样本、事务、备份、人工批准
deploy / force-push / secret 输出	外部后果或泄密	每次停下解释

三种情形要收住:意图含糊(「优化一下」它会乱猜方向)、长会话漂移(开太久它会忘了最初要什么)、不可逆操作(批量删、改生产、强推)。「能自己再试」也意味着它会在错误方向上多走几步。[2] 对策两层:这一章的 —— 把验收标准写进 prompt、别开漫无目的的长会话;危险动作那层 —— 让它自动跑什么、绝不让它碰什么 —— 在**「管住它」**那章用 profile、allowlist / denylist 配。什么时候不如自己敲?你已经记得的一行、只读低风险的一步、或者你比 agent 更清楚现场状态时,直接敲比开口快。验收这一章 —— 主线那四分钟,你自己也要能复现:

- [ ] 我用 ⌘↩ 开会话、@ 指上一块真实报错,交办成功了一次只读排障。
- [ ] 它的每条命令我都先看到再放行,写操作那条我让它停下来等过我。
- [ ] 我用自己写的验收标准(不是它说的「好了」)确认了修复。
- [ ] 我的项目里有一个 AGENTS.md,写清了栈、规范、禁区。
- [ ] 我对比过「光打一句话」和「@ 上下文 + AGENTS.md」两次的结果差。
- [ ] 任务合同模板我存下来了,下次大活直接填。

参考.

01 Warp Docs · Terminal and Agent modes —— 截至 2026-08-06:Terminal(默认,跑 shell)与 Agent(多轮对话视图)两种模式;auto-detection 新用户默认开,提交前以 magenta 的 (autodetected) 标注分流;⌘↩ 起新 agent 会话,⌘Y 继续上一段,⌥⌘↩ 起云端 agent 会话;⌘I 是语境键(见第三章)。Warp Docs · Terminal and Agent modes

02 Warp Docs · Warp Agents overview —— 本地 agent 嵌在终端里,能写代码、debug、运行命令、自动化开发任务;用户保持控制权;对话可带代码库、Warp Drive、连接工具与上下文。Warp Docs · Agents overview

03 Warp Docs · Rules for agents —— 截至 2026-08-06:项目规则文件是全大写 AGENTS.md(WARP.md 仍支持;两个都存在时 WARP.md 优先);子目录 AGENTS.md 自动生效,优先级:子目录 > 根目录 > 全局;/init 生成 AGENTS.md 并可链接 CLAUDE.md、.cursorrules、GEMINI.md 等;命中的规则会作为对话引用显示来源。Warp Docs · Rules

04 Warp Docs · Codebase Context —— 截至 2026-08-06:索引 Git 跟踪的代码(worktree 算独立仓库),不在 Warp 服务器存代码;尊重 .gitignore / .warpindexingignore / .cursorignore 等忽略文件;所有档每仓库至少索引 5,000 文件;SSH / WSL 会话暂不支持;消耗 credits。Warp Docs · Codebase Context

---

你喂的是上下文,
它接的是意图.

派活 · 运维、文件、数据、系统各跑通一件.

发动机不挑活。这章把不同形状的手动活跑通:300 张图批量转格式(含 dry-run 和回滚)是主线,运维、文件、数据、系统四个域各一条可抄的紧凑变体,外加进数据库只读核数的一幕。最后划清:什么时候该切回脚本、cron 和专用工具。

这是全书论点落地的一章。官网把 Agent Mode 演示成改 bug、提 PR,于是你默认它是给写代码的人的。但发动机不挑活:运维、文件、数据、系统的手工活,剥开底下都是命令行和脚本,照样能一件件交办。这章主线跑通一件真实的批处理 —— 300 张图转 webp,含 dry-run 和回滚;然后四个域各给你一条可抄的紧凑变体;最后划清什么时候该切回脚本、cron 和专用工具。

— I

主线:300 张图转 webp,带 dry-run 和回滚.

场景:设计给了你一个 `./assets/raw/` 目录,300 张 PNG 要统一转成 80% 质量的 webp 上网。手动路径是找在线转换工具、分批上传、等、下载、改名 —— 或者自己拼一条 `find` 加 `cwebp` 的参数,拼错一次就得重来。这次交办,但守一条纪律:**批量写入的活,先 dry-run 再真跑**。[2]

01

交办时就写死 dry-run 纪律

⌘↩ 开会话,贴上下面 PromptBox 示例 tab 那段话。注意它没让 agent 直接转换 —— 它先要求列出前 20 个会被处理的文件和源 → 目标路径,声明会不会覆盖。这一步是整件事的保险丝。

02

审 dry-run 输出,而不是审它的信心

它跑完只读分析,给你:300 个文件匹配、目标目录 ./assets/webp/、不覆盖原图、预计总大小。你抽查三条路径映射对不对 —— 对,才说「跑」。它怎么说不重要,那张清单对不对才重要。

03

真跑,盯着计数器

转换是批量写,但写进的是新目录,原图还在 —— 这就是为什么这个任务敢让它自动跑完。几分钟后它报:成功 298、失败 2,两个失败的是带特殊字符的文件名,它已经单独列出来等你处理。

04

验收 + 确认回滚路径

验收三条:输出目录文件数对得上、抽查一张图能正常打开、总大小从 1.2G 降到 90M。回滚路径也在交办时就想好了 —— 原图未动,删掉 `./assets/webp/` 就回到起点。批量活敢交办的前提,从来不是信它,是回滚成本够低。

提示词

文件批处理 dry-run 模板(主线同款)

目标:批量处理 <目录/文件模式>,但先不要改任何文件。
要求:
1. 先列出会被处理的前 20 个文件;
2. 给出 dry-run 命令或只打印源路径 → 目标路径;
3. 说明是否会覆盖、删除、移动;
4. 我确认后才执行真实写入;
5. 执行后给出处理数量、失败数量、可回滚方式。

```
目标:批量处理 ./assets/raw/*.png,统一转成 80% 质量的 .webp,输出到
./assets/webp/,但先不要改任何文件。
要求:
1. 先列出会被处理的前 20 个文件;
2. 给出 dry-run 命令或只打印源路径 → 目标路径;
3. 说明是否会覆盖、删除、移动;
4. 我确认后才执行真实写入;
5. 执行后给出处理数量、失败数量、可回滚方式。
```

手动路径的那个下午

在线转换工具一次限 20 张,你分了 15 批;第三批传完发现免费版加水印;换工具,重来;下载回来的文件名全被改了,再写个脚本对应回去。或者你自己拼 find + cwebp,第一次参数错了覆盖了三张原图 —— 没有 dry-run 的习惯,批量活每一步都在赌。

刚才这十五分钟

一段话交办,一张清单审过,一次真跑,三条验收。失败的两张它单独列出来了,原图一张没动。你做的唯一判断是抽查路径映射 —— 剩下的时间你回了三封邮件。

• • •

— II

四个域,各一条可抄的.

一件事适不适合 Warp,先看它能不能被命令、文件、脚本、日志、环境变量表达。四个域各一条紧凑交办词 —— 抄走,改尖括号里的部分就能用:

运维 · 日志

提示词

从日志里捞统计(只读)

从 <日志路径> 里统计 <时间范围> 每小时的 5xx 请求数,
按「小时 → 次数」输出,顺带列出 5xx 最多的前 5 个路径。
先把你打算跑的命令给我看,确认后再跑,别写任何文件。

从 ./logs/nginx/access.log 里统计昨天每小时的 5xx 请求数,
按「小时 → 次数」输出,顺带列出 5xx 最多的前 5 个路径。
先把你打算跑的命令给我看,确认后再跑,别写任何文件。

适合:日志、服务状态、端口、进程、磁盘、deploy 前检查。切走:正式告警、SLO、长期趋势 —— 那是监控平台的活。

文件 · 整理

提示词

批量重命名 + 归类(dry-run 先行)

把 <目录> 里的文件按 <规则> 重命名并归入子目录。
先只打印「现名 → 新名/去向」的前 20 条,不改任何文件;
说明冲突怎么处理(重名、非法字符);
我确认后再动;动完给成功/失败计数和回滚方式。

把 ~/Downloads 里的截图按「2026-08-06_项目名_序号」重命名,按月份归入子目录。
先只打印「现名 → 新名/去向」的前 20 条,不改任何文件;
说明冲突怎么处理(重名、非法字符);
我确认后再动;动完给成功/失败计数和回滚方式。

适合:重命名、搜索、压缩、转换、diff,可 dry-run 的。切走:视觉整理、靠人眼分类、不可恢复的删除。

数据 · 清洗

提示词

CSV 清洗(不覆盖原文件)

读取 `<input.csv>`。
先做只读分析:行数 / 字段名 / 缺失值统计;抽样 10 行;你建议的清洗规则。
然后给一个可复现脚本,输出写到 `<output.tmp>`,不要覆盖原文件。
最后用 diff / head / wc 验收。

读取 ./data/subscribers.csv。
先做只读分析:行数 / 字段名 / 缺失值统计;抽样 10 行;你建议的清洗规则。
然后给一个可复现脚本,把去掉重复邮箱、统一小写后的结果写到
./data/subscribers.clean.tmp,不要覆盖原文件。
最后用 diff / head / wc 验收。

适合:CSV/JSON 抽样、jq/sed/awk/Python 小脚本。切走:正经 ETL、BI、Notebook 探索、生产写库。

系统 · 诊断

提示词

环境诊断(只读)

只读诊断这台机器的 `<某方面>` 环境:
版本、路径、环境变量、端口、进程、磁盘、权限。
每条命令只读,先给我看我确认再跑;
最后给一段「正常 / 异常 / 建议」的结论。

只读诊断这台机器的 Node 开发环境:
版本、路径、环境变量、端口、进程、磁盘、权限。
每条命令只读,先给我看我确认再跑;
最后给一段「正常 / 异常 / 建议」的结论。

适合:环境诊断、依赖检查、权限排查、配置 diff。切走:内核、权限、安全策略变更 —— 那要专用工具和人工流程。

— III

一幕:进数据库,只读核数.

agent 不止能跑一次性命令 —— Full Terminal Use 让它看到实时终端缓冲、向 PTY 写入、应答提示,于是它能开 psql 进去待着、来回试。[1] 很多数据活不是「跑一条命令」,是「进去看一眼再决定」。看这一幕:

```
# agent 开了一个 psql 会话,先只读地确认范围
psql "$DATABASE_URL"
=> SELECT count(*) FROM orders WHERE status IS NULL;
 count
-------
   1843
=> SELECT id, created_at FROM orders WHERE status IS NULL
   ORDER BY created_at DESC LIMIT 5;
-- 1843 行待回填,样本看过了,现在它把打算跑的 UPDATE 摆出来等你点头
```

AGENT 进 PSQL 核一段数据(只读,先看再动)

注意纪律没变:它进得去,不代表它该闭眼改。先只读会话核范围、样本、行数,UPDATE 摆出来你点头才执行。两个开关值得知道:全局 FTU 权限默认「首次写入询问」,跑着的交互命令(比如你手开的 psql)可以按 `⌘I` 或点底部 Use Agent 把 agent 拉进来接管。[1]

— IV

什么时候别派给它.

稳定重复的活该沉淀:一件事每周都做且步骤固定,让 agent 每次现编就是浪费 —— 让它做一次,存成 Workflow 或写成脚本挂上 cron,下次直接跑。[3] 还有一类该交给专用工具:正经 ETL、备份、监控有成熟的东西,比 agent 每次重搭更稳。元工具的价值是覆盖那条长长的杂活尾巴,不是取代那几件值得专门工具的事。再往外,要去点别的 GUI app,那已经出了边界 —— 第十章的事。

> — 要点
>
> 可截图判断:**Warp 最适合的不是「所有工作」,而是所有能被命令、文件、脚本、日志和环境变量表达的工作。**

切到	信号
普通终端	只跑一两条你已经懂的命令、远程服务器默认环境、离线容器
脚本 / cron / CI	流程稳定、重复、需要审计和失败重试
ETL / BI / Notebook	数据量大、需要可视化探索、统计假设要反复验证
专用运维工具	SLO、告警、长期趋势、权限审计
浏览器 / GUI agent	核心动作是点击网页或桌面软件

验收这一章 —— 主线你自己也要能复现:

- [ ] 我把一件真实的批量活交办出去了,交办词里写死了 dry-run 先行。
- [ ] 我审的是 dry-run 清单(路径映射、覆盖声明),不是它的信心。
- [ ] 验收时我核了计数、抽查了产物,并确认回滚路径还在。

- [ ] 四个域里至少一条交办词我抄走并跑通了一件自己的活。
- [ ] 碰数据库我先让它只读核范围,UPDATE 摆出来我点头才跑。
- [ ] 我写下一件该沉淀成脚本 / cron 的重复活,留给第六章。

— 引用与参考

参考.

01 Warp Docs · Full Terminal Use —— 截至 2026-08-06:agent 能看到实时终端缓冲、向 PTY 写入、应答提示,在运行中的进程里继续工作(psql / mysql、gdb、vim、Python REPL、dev server);跑着的交互命令可以用底部 Use Agent 按钮或 ⌘I 把 agent 拉进来;全局 FTU 权限默认三档:首次写入询问 / 总是询问 / 总是允许;Secret Redaction 仍然生效;FTU 交互消耗 credits。Warp Docs · Full Terminal Use

02 Warp Docs · Warp Agents overview —— 本地 agent 嵌在终端里,能写代码、debug、运行命令、自动化开发任务,并使用代码库、Warp Drive 与连接工具上下文;本地 agent 交互运行,云端 agent 后台运行。Warp Docs · Agents overview

03 Warp Docs · Workflows —— 把一条参数化命令命名保存,带描述与参数默认值,从命令面板、^⇧R 或 Warp Drive 调用,免得重复手敲;Block Actions 和 agent 结果里都有 Save as Workflow。Warp Docs · Workflows

---

它不在乎是不是代码,
它在乎是不是命令.

沉淀 · 把重复三次的事存成一键.

跑通一次是运气,跑通三次就该沉淀。这章把一件本周重复了三次的操作从头固化:参数化成 Workflow、长流程写成 Notebook、给 agent 存一条 Prompt —— 再谈存了反而是负担的那类,以及 Drive 的月度体检。

跑通一次是运气,跑通三次是信号。这周你第三次敲同一条看日志的命令时,该意识到的不是「我记性真好」,是「这活该固化了」。Warp Drive 是跑通过程的归宿:命令存成 Workflow、长流程写成 Notebook、给 agent 的任务存成 Prompt、环境差异抽成变量。这章用一条主线走完整个固化动作,再谈什么存了反而是负担,以及 Drive 的月度体检。

— I

主线:把第三次看日志固化成一键.

场景:staging 的 api 服务这周第三次出问题,你第三次敲 `docker compose -f docker-compose.staging.yml logs --tail 200 api | grep -Ei "error|timeout"`。命令你快背下来了 —— 这正是最该沉淀的时刻:足够常用,值得存;还没熟到烦,存的动作成本最低。[1]

01

认出信号:第三次,停手,别敲第四次

一次是探索,两次是巧合,三次是模式。Explore 阶段的脏命令不值得存;但当一条命令第三次出现,它的形状已经稳定 —— 变的只有环境名、服务名、行数。变什么,待会儿就参数化什么。

02

块右键 Save as Workflow,参数化

在刚跑通的那个 block 上右键,选 `Save as Workflow`(agent 跑出来的结果同样有这入口)。把 `staging` 改成 `{{env}}`、`api` 改成 `{{service}}`、`200` 改成 `{{lines}}`,各给默认值;名称和描述可以让 AI Autofill 起草(每次消耗 1 credit),自己再过一遍。存完它就在 Drive 里,实时同步。[2]

03

验证:^⇧R 调出,SHIFT-TAB 填参

按 `^⇧R` 搜 `logs`,回车,参数依次高亮 —— `SHIFT-TAB` 在参数间循环,同名参数改一处全改(多光标同步)。填 `production` / `web` / `500`,跑通 —— 同一个 workflow 跨环境复用,这才是参数化的意义。写死成一条具体命令,它只够你这台机器用一次。

04

配套:长流程写 Notebook,agent 任务存 Prompt

一条命令存 Workflow;「看日志 → 查进程 → 看最近错误 → 人工判断」这种多步排障,写成 Notebook(markdown + 可执行命令块,还能嵌入刚存的 Workflow);「读 staging 日志找 5xx 原因」这种反复交给 agent 的活,存成带参数的 Prompt。三样各就各位,这套排障就再也不靠记忆了。

```
# 服务健康检查 Runbook

## 适用范围
- 环境:{{env}}
- 服务:{{service}}
- 不做:重启生产服务、删除数据、修改权限

## 1. 只读确认
```shell
pwd
git status --short
```

2. 进程与端口
```shell
ps aux | grep {{service}} | grep -v grep
lsof -i :{{port}}
```

3. 最近错误(嵌入刚存的 Workflow,免复制)
```shell
docker compose -f docker-compose.{{env}}.yml logs --tail {{lines}} {{service}} |
  grep -Ei "error|timeout"
```

4. 人工判断
- 只读检查失败:记录输出,停止。
- 需要写入 / 重启:另开变更流程,不在本 Notebook 自动执行。
```

配套的排障 NOTEBOOK(示意,MARKDOWN + 可执行命令块)

提示词

让 agent 帮你做沉淀(它跑通的,它最会参数化)

把刚才跑通的命令沉淀成 Warp Workflow。
请给我:
1. workflow 名称;
2. 一句 description;
3. tags;
4. 哪些部分应该变成 {{argument}};
5. 每个 argument 的默认值和说明;
6. 哪些命令仍然需要人工确认,不该直接放进 workflow。

```
把刚才跑通的命令沉淀成 Warp Workflow:
docker compose -f docker-compose.staging.yml logs --tail 200 api | grep -Ei
"error|timeout"
请给我:
1. workflow 名称;
2. 一句 description;
3. tags;
4. 哪些部分应该变成参数(我看是 compose 文件名、服务名、tail 行数);
5. 每个参数的默认值和说明;
6. 哪些命令仍然需要人工确认,不该直接放进 workflow。
```

– 注意

一个存量知识要更新:以前教程教你手写 YAML 放在 ~/.warp/workflows/ 下 —— 那是 **legacy**。老文件还能从 Command Search / 命令面板调出来跑,但不再出现在 Drive 里,也不再是新 workflow 的保存方式。新沉淀一律走 Drive(Save as Workflow / 面板新建),团队共享和 agent 上下文都只对 Drive 里的对象生效。[1]

• • •

– II

什么值得存,什么存了是负担.

Drive 存四样,关键的一层是它们能回头喂 agent:agent 默认自动取用你存的 workflow、notebook、规则作上下文,取用了会在对话里标 citation。你沉淀的不只是给自己看的手册,也是喂给发动机的燃料 —— 所以 Drive 的干不干净,直接影响 agent 靠不靠谱。[4]

你手里是什么	沉淀成	不要沉淀成
一条会重复的命令,只有路径/环境会变	**Workflow**	长 Notebook 或新脚本
多步操作手册、部署、排障、上手流程	**Notebook**	散落在群聊里的命令串
反复交给 agent 的自然语言任务	**Prompt**	每次重新描述
环境名、项目路径、非敏感开关	**Environment Variables**	共享明文密钥
一次性探索、临时脏命令	什么都不存	Drive 噪音

– 实践提示

密钥不进 Drive。静态环境变量是明文存的,官方明说它不是密钥管理器;真要用密钥,用**动态**环境变量 —— 接 1Password / LastPass CLI 或自定义取回命令,Warp 只存那条取回命令,不存秘密本身。[4]

– 要点

可截图判断:**一次性命令只有复盘后才值得沉淀;未经复盘的 Drive 只是第二个过期 README。**

— III

Drive 月度体检.

存太多和存太少一样糟。一堆没人用的 workflow 比没有还糟:它让人不信任整个 Drive,连同 agent 从里面取的上下文一起不信。一条朴素的节奏 —— 每月第一个周一,五分钟,按下面这张单过一遍:

- [ ] 一个月没人调用的对象:要么删,要么补上 description 让别人(和 agent)能用对。
- [ ] 重复检查:同一类命令只留一个参数化版本,不留三份近亲。
- [ ] 危险命令检查:删除、deploy、生产写库不能无说明地躺在 Workflow 里。
- [ ] 密钥检查:Environment Variables 里没有明文 secret;需要的都走动态变量。
- [ ] 范围检查:个人对象别误搬进 Team Drive —— 搬进去默认全团队可见可用。
- [ ] 调用验证:随机抽三个 workflow,从 ^⇧R 调出跑一遍,跑不通的当场修或删。

验收这一章 —— 主线那个「第三次」你自己也要抓到一次:

- [ ] 我认出了一条本周重复三次的命令,并用 Save as Workflow 存进了 Drive。
- [ ] 会变的部分都抽成了带默认值的参数,我能用 ^⇧R 调出、SHIFT-TAB 填参跑通。
- [ ] 一个多步流程写成了 Notebook,里面嵌了那个 Workflow。
- [ ] 一条反复交给 agent 的任务存成了 Prompt。
- [ ] 我知道 YAML 文件 workflows 是 legacy,新沉淀都走 Drive。
- [ ] 我给 Drive 体检定了一个月度提醒。

参考.

01 Warp Docs · Warp Drive 概览 —— 截至 2026-08-06:Warp Drive 存 Workflows、Notebooks、Prompts、Environment Variables,实时同步,个人或团队共享;老的 YAML 文件 workflows 是 legacy —— 通过 Command Search / 命令面板无限期支持,但不再进 Drive。Warp Docs · Warp Drive

02 Warp Docs · Workflows —— 截至 2026-08-06:参数化命令,从命令面板、^⇧R 或 Drive 调用;入口包括 Block Actions 与 agent 结果里的 Save as Workflow;参数写作 {{arg}}(字母数字/-/_、不以数字开头),有 text 与 enum 两种类型;SHIFT-TAB 在参数间循环,同名参数多光标同步;AI Autofill 自动生成名称/描述/参数(每次消耗 1 credit);个人 alias 可配。Warp Docs · Workflows

03 Warp Docs · Notebooks —— markdown 文字加可执行命令块组成可运行文档;命令块用 {{param}} 取参,CMD-ENTER / CTRL-ENTER 插入终端;可嵌入已有 Workflow(Embedded Workflow)免重复;团队编辑是单编辑者制(其余人 Viewing)。Warp Docs · Notebooks

04 Warp Docs · Drive as Agent Mode Context + Environment Variables —— agent 自动从 Drive 取用 Workflows、Notebooks、环境变量、Rules 与 MCP Servers 作上下文并以 citation 标注(Settings > Agents > Knowledge,默认开);环境变量分静态(存 Drive,不是密钥管理器)与动态(1Password / LastPass CLI 或自定义命令,Warp 只存取回命令、不存秘密)。Warp Docs · Drive as agent context

---

跑通是一次性的,
存下来才是复利.

接外部 · 接一个真数据源,完成一次查询.

agent 默认只看得见你的终端。这章真的接上一个外部数据源 —— 一个只读的 GitHub MCP —— 用它回答一个光看终端答不了的问题,然后把权限收成只读。能连是接口给的能力,该连是你给的判断,红线本章划清。

到这里,agent 看得见的还只是你的终端和仓库。但你大半上下文不在那儿 —— 在 issue 跟踪、数据库、监控面板、内部 API 里。MCP(Model Context Protocol)是那根接线。这章不空谈接线图:真的接一个只读的 GitHub MCP,用它回答一个光看终端答不了的问题,然后把权限收成只读。接完这一次,你就有了判断「下一个该不该接」的全部手感。

— I

主线:接一次只读 GitHub MCP,答一题,收权限.

为什么选 GitHub:几乎人人都有,只读 token 好申请,答错了代价为零。问题也挑一个终端答不了的:「我们 repo 里,open 超过 30 天、最近一周还有人评论的 issue 有哪些?」—— 答案不在你的文件系统里,在 GitHub 的 API 后面。[1]

01

接:Settings > Agents > MCP servers 加一个 GitHub server

入口有三个:Settings > Agents > MCP servers、Drive 的 Personal > MCP Servers、命令面板搜 `Open MCP Servers`。用 GitHub 官方 MCP,token 只给只读 scope;OAuth 类的服务器首次启动会弹授权窗,授权完才上线。加完不用重启 Warp —— 下一条消息新工具就进上下文。[1]

02

问:把那句话交给它

⌘↩ 开会话,贴上下面 PromptBox 示例 tab 那段。注意它要求 agent 说出用了哪个 tool、数据来自哪 —— 这是防「凭空编」的验收钩子。

03

验:看它真读了外部数据

它调用 MCP 工具时,确认面板会显示正在跑的是哪个 tool、来自哪个 server —— 你看得见它碰的是 GitHub 而不是在猜。答案回来,抽查两个 issue 号去 GitHub 上核对:真实存在、日期对得上,才算这次接线成功。[1]

04

收:权限锁成只读

回到 MCP 设置,把这个 server 的写类工具(create_issue、add_comment 之类)放进 denylist —— denylist 优先于一切,命中必须人工批准。以后它只能读,想写得先过你。这一步就是「能连」和「该连」之间的那道闸。[3]

提示词

主线同款:一个光看终端答不了的问题

```
用 GitHub MCP 回答:<repo> 里 open 超过 <天数> 天、
最近 <天数> 天内还有人评论的 issue 有哪些?
要求:
1. 说明你调用了哪个 tool、数据来自哪里;
2. 列出 issue 号、标题、open 天数、最近评论时间;
3. 只读,不要创建或修改任何 issue / comment。


用 GitHub MCP 回答:我们 repo 里 open 超过 30 天、
最近 7 天内还有人评论的 issue 有哪些?
要求:
1. 说明你调用了哪个 tool、数据来自哪里;
2. 列出 issue 号、标题、open 天数、最近评论时间;
3. 只读,不要创建或修改任何 issue / comment。
```

没有 MCP 的同一题

开浏览器,进 Issues 页,按 updated 排序,一页页翻;open 天数要心算,最近评论要逐个 issue 点进去看;二十分钟后你有一份「大概齐」的清单,漏没漏你没把握。

刚才这三分钟

一句话,一张带 issue 号和日期的表,工具调用过程全程可见,你抽查了两个号都对。剩下的问题是判断:这些僵尸 issue 哪些该关 —— 那是你的活,agent 把找答案的活干完了。

• • •

— II

它能读你已有的配置.

你不必重新配一遍。Warp 认这些位置的 MCP 配置:自己的 `~/.warp/.mcp.json`(全局)和项目里的 `.warp/.mcp.json`,Claude Code 的 `~/.claude.json` / `.mcp.json`,Codex 的 `~/.codex/config.toml`,以及其他 agent 的 `~/.agents/.mcp.json`。auto-spawn 规则分三层:Warp 自己的全局 file-based servers 默认自动启动;第三方全局 servers 要你显式打开「Auto-spawn servers from third-party agents」;项目级 servers 永不自动启动,会话级生效,重启后要重新批准 —— 从陌生 repo clone 下来的 `.warp/.mcp.json` 不能自己启动本地命令,这条规则是你的保险。[2]

```
{
  "docs-ro": {
    "command": "npx",
    "args": ["-y", "mcp-remote", "https://example.com/mcp"],
    "env": {
      "DOCS_TOKEN": "<read_only_token>"
    },
    "working_directory": "/absolute/path/to/project"
  }
}
```

一个只读 MCP 服务器条目(CLI 型,示意;包名与参数以对应 SERVER 文档为准)

提示词

MCP 接入审查模板(接下一个之前先跑这个)

```
我要给 Warp Agent 接一个 MCP server: <server-name> 。
先不要安装或启动。
请帮我审查:
1. 它能读什么;
2. 它能写什么;
3. 凭据放在哪里,能否撤销;
4. 日志在哪里,是否可能含 token;
5. 该设为 allowlist、Agent decides 还是 denylist;
6. 第一次只读验证问题应该问什么。


我要给 Warp Agent 接一个 MCP server:linear-mcp。
先不要安装或启动。
请帮我审查:
1. 它能读什么;
2. 它能写什么;
3. 凭据放在哪里,能否撤销;
4. 日志在哪里,是否可能含 token;
5. 该设为 allowlist、Agent decides 还是 denylist;
6. 第一次只读验证问题应该问什么。
```

— 实践提示

两个省时间的:文件型服务器可以让 agent 自己配 —— 内置 skill /agent-add-mcp 能写全局或项目配置;新服务器先用 npx 在本地跑通再贴进 Warp,省得在 app 里反复试。相对路径的命令一定显式设 working_directory。[2]

• • •

— III

能连不等于该连.

每多接一个 MCP 服务器,agent 能做的事变多,能闯的祸也变多。三笔代价要算:多一个外部依赖就多一份攻击面,多一组工具就多烧 token,多一处写权限就多一处它能改坏的地方。所以默认连只读,写权限单独想清楚 —— denylist 优先于 allowlist 与「Agent decides」,命中必须人工批准,生产

库的写口就该进 denylist。[3] 连之前先问三句:这服务器要的权限是只读还是能写?它的工具会不会把敏感数据带出去?这件事我多久用一次、值不值得让它常驻?「能连」是接口给的能力,「该连」是你给的判断。

风险级别	例子	默认策略
只读	文档、只读数据库、GitHub issue 读取、日志查询	可先接,但仍记录权限和日志位置
草拟	起草 issue 回复、生成 PR 描述、创建草稿工单	允许生成,不自动发布
写入	改 issue 状态、写数据库、改云资源	默认 denylist,每次批准
高风险	支付、客户数据导出、密码管理器、生产控制台、邮件发送	不要接,或只在隔离环境临时接

— 要点

可截图判断:MCP 省的是接外部工具的成本,但每接一个 server,都是在扩大 agent 的权限边界。

你要做的事	优先选	原因
一次性调用一个公开 API	API CLI / curl	轻,不扩大长期权限边界
重复查内部资料或 issue	**MCP**	让 agent 直接读上下文
固定的数据清洗流程	shell script / Workflow	可审计、可复现、少 token
需要网页登录、点按钮、看页面状态	浏览器 / Computer Use 类工具	MCP 不是 GUI 自动化

验收这一章 —— 主线那一次接线,你自己也要做成:

- [ ] 我接上了一个只读 GitHub MCP,token 只给了只读 scope。
- [ ] 我用它回答了一个光看终端答不了的问题,并抽查了两个 issue 号核对。
- [ ] 我看到了它调用的 tool 名和来源 server,确认不是凭空编。
- [ ] 写类工具进了 denylist,这个 server 现在只能读。
- [ ] 我知道项目级 .warp/.mcp.json 不会自动启动,重启后要重新批准。
- [ ] 下一个想接的 server,我会先跑一遍接入审查模板。

参考.

01 Warp Docs · Model Context Protocol (MCP) —— 截至 2026-08-06:MCP servers 用标准接口给 Warp 本地 agent 暴露工具和数据源;支持 CLI command server 与 Streamable HTTP / SSE URL server、自定义 headers、环境变量;入口:Settings > Agents > MCP servers、Drive Personal > MCP Servers、命令面板「Open MCP Servers」;OAuth 服务器首次启动弹授权窗;2026-07-31 起,工具确认面板会显示正在运行的是哪个 tool、来自哪个 server。Warp Docs · MCP

02 Warp Docs · MCP security behavior —— 全局 Warp file-based servers 默认 auto-spawn;第三方全局 servers 需显式开启「Auto-spawn servers from third-party agents」;项目级 servers 永不自动启动、会话级生效(重启后重新批准);MCP 配置编辑需要批准;共享时 env 值会被 scrub,队友各自重填;日志可能含 token,分享前先清理;相对路径命令要显式设 working_directory。Warp Docs · MCP security

03 Warp Docs · Agent Profiles & Permissions —— 可针对每个 MCP 服务器选择「Agent decides」、allowlist 或 denylist;denylist 优先于 allowlist 与「Agent decides」,命中必须人工批准(详见第八章「管住它」)。Warp Docs · Profiles & Permissions

---

能连的是接口,
该连的是判断.

管住它 · 配到「敢放手」再自动跑.

一个会自己跑命令的 agent,信任是要管的。这章从零配出一个敢放手的 profile:选模型(官方 credits / BYOK / custom endpoint 三条账本)、定权限(四档加 allowlist 与 denylist)、审命令(跑前你看见,跑完你看 diff)—— 配完用一次真实改动验收。

一个会自己跑命令、还能接外部工具的 agent,信任是要管的 —— 不能全凭它自觉。这章带你从零配出一个「敢放手」的 profile:建出来,权限定好,allowlist 与 denylist 写死,然后用一次真实改动当场验收 —— 看它该自动的自动、该拦的拦住。配完这三节细讲背后的三本账:选模型、定权限、审命令。

— I

主线:配一个敢放手的 local-dev profile.

01

建:Settings > Agents > Profiles 新建 local-dev

新建一个 profile,命名 `local-dev`,base model 先选你常用的那档(Auto 也行,模型账第二节再细算)。Profile 的意义是把「这台机器上干什么活、放到什么程度」固定成一个可以复用的组合,而不是每次会话现调。[1]

02

定:四档权限 + 两张清单

执行命令设 Agent decides(确信就做、不确定才问);Apply code diffs 注意:`Agent decides` 目前等同 Always ask,只有 Always allow 才跳过 diff 审查 —— 想看 diff 就不用动它。allowlist 放三条只读正则(`git (status|diff|log)` 这类),denylist 放 `rm`、`curl`、`wget`、`eval` 这类红线 —— denylist 优先于一切。[1]

03

验:一次真实改动当场验收

派一个真任务(第四章那类修 bug 就行),盯两件事:allowlist 里的 git status 它不问就跑 —— 这是省事;denylist 里的命令它必须停下来问你 —— 这是保命。两个都发生,profile 才算配成;拦了不该拦的、放了不该放的,回去改正则。

04

调:信任按类别逐步松手

同一类命令审过几次都没问题,把它的正则挪进 allowlist;没底的那类一直留在「每次问」。别一上来全放,也别永远每步都拦 —— 敢放手不是一次决定,是一条曲线。单个会话想临时放开,有 /fast-forward(GUI 和 TUI 都有);长期组合则从 2026-07-31 起全员可以从设置文件配置,不用在 UI 里逐台机器点。

— II

选模型：官方 credits、BYOK、custom endpoint 三本账.

模型这件事有三条路：用 Warp 官方模型和 credits，BYOK 接自己的 provider key，或者 custom inference endpoint 接自己的网关。官方 Model Choice 里有四档 Auto 和 OpenAI、Anthropic、Google、xAI、Fireworks-hosted 开源模型（完整名单和价格见第十一章）；BYOK 支持 OpenAI、Anthropic、Google，Free 就能用，选中带 key 图标的具体模型时推理走你的 provider 账单、不耗 Warp credits；[4] [2] custom endpoint 接任何 OpenAI-compatible 目标，2026-07-31 起还能选 schema（OpenAI Chat Completions / OpenAI Responses / Anthropic Messages），localhost 和私网地址不行。[3] 模型旋钮上还有一层：**custom model router**。它不是第四本账，是把「每次手挑模型」变成写好一次的策略：Settings > AI > Custom Routers 里建，或写 YAML 放进 `~/.warp/custom_model_routers/`，按任务复杂度分档或自然语言规则匹配 —— 「改数据库迁移」走便宜模型，「排查生产事故」走 frontier 模型。它在模型选择器里像模型一样被选，每次请求解析成一个具体模型，你始终能看到实际跑的是哪个。两处更新：router 现在能和 BYOK 组合（解析出模型后走你的 key）；但 router 永远不能用 custom endpoint。[5]

模型路径	适合	你承担什么
官方模型 / official credits	省心、Auto、custom routers、Cloud Agents、团队默认使用	credits、模型可用性、计划限制
BYOK	已有 OpenAI / Anthropic / Google 账单，希望控制 provider 成本	API key 安全、provider 成本、rate limit、provider 数据策略
custom inference endpoint	OpenAI-compatible router / gateway / self-hosted public endpoint	兼容性、公开可达性、延迟、稳定性、日志与隐私
local model	仅当通过公开 HTTPS 的 OpenAI-compatible gateway 暴露给 Warp	它不是本机 localhost 直连

— 注意

三个边界要写死:**你本机配的 BYOK 和 custom endpoint 都不适用于 Cloud Agents** —— key 存在你设备上,云端 run 拿不到,cloud run 照常消耗 Warp credits;**BYOK 失败默认不 fallback 到 credits**,要兜底去开 opt-in 的 credit fallback;**ChatGPT / Claude 的消费级订阅接不进来**,消费订阅路径只有 SuperGrok。另外,集中管理的 admin BYOK 还没上线,Enterprise 的托管推理走 BYOLLM(AWS Bedrock 起步)。别把 BYOK 理解成「所有 Warp AI 都不花 Warp 钱」。[2] [3]

提示词

BYOK 设置前 checklist

```
准备在 Warp 里配置 BYOK / custom endpoint。
先确认:
- 我用的是具体 provider model,不是 Auto;
- 这个 key 的权限最小、可撤销、有预算上限;
- provider 侧 retention / training 设置符合项目要求;
- 我知道请求会经 Warp backend in-flight;
- 这不会覆盖 Cloud Agents;
- 失败默认不 fallback,要不要开 opt-in credit fallback。


准备在 Warp 里配置 Anthropic BYOK。
先确认:
- 我选的是 Claude Sonnet 这个具体模型,不是 Auto;
- 这个 Anthropic key 只开了 inference 权限、能在控制台一键撤销、设了每月 50 美元上限;
- Anthropic 侧 retention 设为 30 天、已关掉 training,符合项目要求;
- 我知道请求会经 Warp backend in-flight;
- 这不会覆盖 Cloud Agents;
- 开 opt-in credit fallback 兜底,免得限流时断活。
```

• • •

— III

定权限:四档、两清单、一个绝不能开错的开关.

自主度不是一个开关,是一条从「每步都问」到「全放行」的滑杆 —— 你该停在中间。四档(Agent Decides / Always ask / Always allow / Never)分别管 diff 落地、读文件、做计划、执行命令、Full Terminal Use、问澄清问题;常用的中间档是「确信就做、不确定才问」。两侧用清单兜底:allowlist 在 Settings > Agents > Profiles,接正则,把信得过的只读命令放进去自动跑;denylist 放绝不能自动跑的,优先于 allowlist 和「Agent decides」,命中必须人工批准。[1]

```
# allowlist — 只读、低风险,自动免确认跑
which .*
ls(\s.*)?
grep(\s.*)?
git (status|diff|log)(\s.*)?

# denylist — 优先级最高,命中永远要你点头
wget(\s.*)?
curl(\s.*)?
rm(\s.*)?
eval(\s.*)?
```

起步 ALLOWLIST / DENYLIST(正则;DENYLIST 四条即文档默认例)

提示词

Agent 权限边界模板

Profile 名称: <safe-ops / local-dev / sandbox-yolo>
Base model: <model>
默认自主度: <Agent decides / Always ask / Always allow / Never>
允许自动读: <目录/文件类型>
允许自动执行: <只读命令正则>
必须询问:写入、删除、网络下载、sudo、MCP 写工具
禁止:生产数据库写入、deploy、secret 输出、home 目录批量改动
验收:每次结束必须给命令列表、变更 diff、验证命令和结果。

```
Profile 名称:local-dev
Base model:Claude Sonnet
默认自主度:Agent decides
允许自动读:./src 下的 .ts / .tsx 和 package.json
允许自动执行:^(ls|cat|git (status|diff|log)|npm run lint)\b
必须询问:写入、删除、网络下载、sudo、MCP 写工具
禁止:生产数据库写入、deploy、secret 输出、home 目录批量改动
验收:每次结束必须给命令列表、变更 diff、验证命令和结果。
```

— 注意

一档要单独警告:「Run until completion」(`⌘⇧I` / `Ctrl+Shift+I`)**连 denylist 都不看** —— 它是最纯的 YOLO,agent 一路跑到底、什么都不问,你写在 denylist 里的 rm、drop 也拦不住。只在沙箱或一次性环境里开,别在能碰到生产或重要数据的机器上用。单个会话的临时放开用 `/fast-forward`,别动全局档。[1]

• • •

— IV

审命令:它要跑什么、改了什么.

放手不等于不看 —— 它跑前给你命令,跑后给你 diff,这两眼你得花。默认让它「先给命令、确认再跑」,把要执行的那行摆到你眼前;改动则进 Code Review 面板,diff 框起来看,而不是淹在终端输出里。Apply code diffs 那档别贪快设成 Always allow —— 那是把 review 这一眼也省掉。

- [ ] 命令审查:目录对不对、账号对不对、环境变量对不对。
- [ ] 命令审查:有没有删除、覆盖、权限变更、网络下载、sudo、生产 host。
- [ ] 命令审查:能不能先 dry-run、小样本、只读 SELECT。
- [ ] 成本控制:Auto 消耗 Warp credits;router 可以走 BYOK,但 endpoint 不行;provider 账单也要设预算。

- [ ] 成本控制:Cloud Agents 不走 BYOK;上云前先看 credit 余额和验收方式。

> **— 实践提示**
>
> 什么时候不要用 BYOK:你不想管理两套账单;provider 侧没有你需要的 ZDR/retention 设置;团队需要中央模型治理而不是每个人本地加 key;任务主要跑 Cloud Agents;或者你只是偶尔问几句。BYOK 是控制权,不是免费额度。

验收这一章 —— 你的 local-dev profile 要当场配完、当场验过:

- [ ] 建好 `local-dev` profile,选好了模型路径:官方 credits、BYOK,还是 custom inference endpoint。
- [ ] allowlist 里有几条只读命令正则,denylist 里有 `rm`、`curl`、`wget`、`eval` 和指向生产的命令。
- [ ] 用一次真实改动验收过:该自动的自动跑了,该拦的拦住问我了。
- [ ] 自主度停在「确信才做、不确定就问」,没误开 Run until completion;临时放开用 `/fast-forward`。
- [ ] 记住:BYOK / custom endpoint 不覆盖 Cloud Agents;BYOK 失败默认不 fallback;消费级订阅接不进来。

参考.

01 Warp Docs · Agent Profiles & Permissions —— 截至 2026-08-06：Profile 可配置 base model（也用于 Planning）、autonomy 与各类权限（Apply code diffs / 读文件 / 做计划 / 执行命令 / Full Terminal Use / 问澄清问题），每项分 Agent Decides / Always ask / Always allow / Never 四档；Apply code diffs 的 Agent decides 目前等同 Always ask，只有 Always allow 跳过 diff 审查；问问题权限三档（Never ask / Ask unless auto-approve / Always ask）；allowlist 接正则，denylist 优先于 allowlist 与 Agent Decides；Run until completion（⌘⇧I）连 denylist 都绕过。2026-07-23 起 /fast-forward 可切换单会话 autoexecute（GUI+TUI）；2026-07-31 起 execution profiles 全员可从设置文件配置。Warp Docs · Profiles & Permissions

02 Warp Docs · Bring Your Own API Key —— 截至 2026-08-06：BYOK 支持 OpenAI、Anthropic、Google；Free 与符合条件的付费档可用（个人及 10 人以内组织，大组织需 Business / Enterprise）；key 存本地 OS keychain、请求经 Warp backend in-flight、不存服务端；选中具体 BYOK 模型时优先于 credits；失败 / 限流默认不 fallback 到 Warp credits，有 opt-in 的 credit fallback；ChatGPT / Claude 消费级订阅不能接入（SuperGrok 是唯一消费订阅路径）；BYOK 不适用于 Cloud Agents；中心化 admin-managed BYOK 尚未上线，Enterprise 的托管推理走 BYOLLM（当前 AWS Bedrock，Azure Foundry 与 Google Vertex coming soon）。配置入口：Settings 搜 API keys。Warp Docs · BYOK

03 Warp Docs · Custom inference endpoint + changelog 2026-07-31 —— endpoint 接 OpenAI-compatible 目标（OpenRouter、LiteLLM、z.ai、公开 HTTPS 的内部网关）；localhost / 私网被拒（本机模型经 ngrok 类隧道暴露）；key 存本地；不适用于 Cloud Agents；Business / Enterprise 本地经 endpoint 仍消耗 platform credits；2026-07-31 起 endpoint 可选 OpenAI Chat Completions / OpenAI Responses / Anthropic Messages 三种 schema（文档页仍只写 Chat Completions，以 changelog 为准）。router 现在可以与 BYOK 组合（解析出模型后套用你的 BYOK key），但永远不能用 custom endpoint。Warp Docs · Custom endpoint

04 Warp Docs · Agent model choice —— 截至 2026-08-06：Auto 模式四档（Responsive / Cost-efficient / Genius / Open-weights），模型名单覆盖 OpenAI、Anthropic、Google、xAI 与 Fireworks-hosted 开源模型（完整名单与档位价格见第十一章）；存在 model fallback chain；每个 Profile 可设独立 base model；Grok 可走

连接的 SuperGrok 订阅(xAI 账号额度,不耗 Warp credits);ZDR 可用性按模型变化(Fable 5 要求 Anthropic 数据保留,无 ZDR)。Warp Docs · Model choice

05 Warp Docs · Custom routers —— 截至 2026-08-06:在 Settings > AI > Custom Routers 建,或写 YAML 放 ~/.warp/custom_model_routers/;按任务复杂度(easy / medium / hard 加必填 default)或自然语言规则(自上而下首条命中)把每次请求解析到一个具体模型;出现在模型选择器里;可与 BYOK 组合;Team-synced routers 是 Enterprise 功能。Warp Docs · Custom routers

---

你能放手的程度,
取决于你设界的程度.

放出去 · 派一批活上云,验收回来.

有些活不用你守着。这章把一批活真的派出去:两个并行的 cloud agent 在后台跑,你去喝咖啡,回来照着 transcript 逐条验收;跑顺的那个改成每周定时。也讲清对一个人,多 agent 编排什么时候是过度。

到这里 agent 都在你眼前跑 —— 你看着它一步步动。但有些活不值得你守着:夜里的维护、一开 PR 就该触发的检查、需要并行跑的一批杂活。这章把一批活真的派出去:两个 cloud agent 后台并行,你去喝咖啡,回来照着 transcript 和 diff 逐条验收;跑顺的那个,改成每周定时。然后讲清:对一个人,多 agent 编排什么时候是过度。

— I

主线:派两个 cloud agent 出去,验收回来.

挑两件安全又能并行验收的活:一件出报告(依赖审计),一件出 PR(清一组 lint 警告)。两个都是明确、可验证、不需要你中途判断的活 —— 这是上云的入场条件。跑 cloud agents 需要账户里 ≥20 credits(新用户有试用 credits);Slack / Linear 这类 integrations 要 Build / Max / Business 的团队,本章只用定时和手动,不要团队。[2]

01 备环境:让 /cloud-agent 带你建一个

在 Warp 里输入 `/cloud-agent`,没有 environment 它会引导你建:一个 environment 就是一个 Docker image + 你的 repo + setup 命令(装依赖、配 GitHub auth)。两条硬要求:镜像必须是 glibc 的(Alpine / musl 不行),不想写 Dockerfile 就用预置的 dev image。[2]

02 派:两条任务并行发出去

用下面 PromptBox 的模板填两条任务,各起一个 cloud run。命令行派用 `oz agent run-cloud`,app 里派就用 `/cloud-agent` 开 Cloud Mode。发完关上笔记本 —— 它们不占你的终端。

03 离开:run 的状态随处可看

每个 run 留下持久记录:状态、元数据、transcript、outputs。在 conversations 面板、session 链接、或 oz.warp.dev 的 Runs tab 都能看 —— 手机上也能盯。这一步是真的去喝咖啡,不是换个窗口焦虑。[1]

04 验收:照 transcript 逐条过

回来别只看结论。报告型任务:开 artifact,抽两条漏洞去 npm advisory 核对。PR 型任务:看 diff 全貌,跑一遍 CI,再问一句「它为什么这样改」—— transcript 里有它的推理。不合格就拒,这就是验收。

05 固化:跑顺的那个改成每周定时

依赖审计跑顺了,就别再手动派:`oz schedule create --cron "0 10 * * 1"` 每周一上午自动跑;或者 `/create-skill` 把这次 run 变成可复用、可调度的 skill。lint 那个如果只发生一次,就让它停留在一次性 —— 不是什么都值得定时。[2]

提示词 — Cloud Agent 任务模板(主线两件活都照这个填)

任务:`<后台要完成什么>`
环境:`<repo / image / setup commands>`
边界:只允许 `<读/写范围>`;禁止 `<生产写入/删除/deploy/secret 输出>`
验收:`<artifact / PR / 报告>`,附上你跑过的验证命令和结果
失败:失败时停止,保留 logs,不自动重试破坏性命令。

任务:跑一次依赖审计,找出 npm 里有已知漏洞或落后两个大版本的包
环境:repo github.com/acme/web,node:22 image,setup 跑 npm ci
边界:只允许读 package.json / package-lock.json 和跑 npm audit;禁止生产写入、删除、deploy、secret 输出
验收:一份按严重度排序的报告 artifact,附升级建议命令,说明每条结论来自哪条 advisory
失败:失败时停止,保留 logs,不自动重试破坏性命令。

```
# 后台起一个 cloud run,不占你的终端
oz agent run-cloud --prompt "清理 7 天前的临时文件,把释放的空间写进报告"

# 跑顺了之后,改成每周一 10:00 定时
oz schedule create --name "weekly-dep-audit" \
  --cron "0 10 * * 1" \
  --environment <ENV_ID> \
  --prompt "跑依赖审计,按严重度出报告 artifact"
```

命令行派活与定时(OZ CLI;APP 里对应 /CLOUD-AGENT 与 SCHEDULE UI)

— 实践提示

分清调度和事件触发:固定节奏的活(每周审计、夜里清理)用定时;要对外部发生的事做反应的(PR 一开、issue 一来)用事件触发 —— 但 Slack / Linear / GitHub 这类 integrations 需要团队档,一个人先把定时用顺。一次性的长活,手动起一个 run-cloud 就走,比守着本地会话省事。[2]

• • •

— II

本地起,云端接;云端完,本地审.

你不必一开始就决定本地还是云端 —— 一段对话可以跑着跑着交接。三个方向规则不同:本地 → 云端只有 Warp Agent 能走,fork 会话并把你未提交的改动快照带上去;云端 → 云端(Warp Agent、Claude Code、Codex harness 之间)同一会话继续,workspace 状态恢复;云端 → 本地用 Continue

locally / `/continue-locally`,但要记住:**workspace 补丁不会应用到本地** —— 回来的东西去 branch / PR 里审,别指望工作区自动变。附件全程带得过去,补丁应用是 best-effort,部分失败会明说。[3]

选择	适合	不要用在
本地 Agent	你要实时审命令、看输出、随时改方向	长时间等待、并行跑很多任务
Cloud Agent	明确、可验收、能后台跑、触发清楚	需要频繁人工判断、敏感本地凭据
Oz orchestration	多 agent 并行、团队监督、run 可观察性、事件/计划自动化	一个人临时跑一条命令

> — 要点
>
> 可截图判断:**云端 agent 省的是等待和并行,不省验收;Oz 省的是编排和可观察性,不替你承担结果责任。**

• • •

— III

单人何时过度,账单怎么算.

多 agent 编排是给团队规模设计的 —— 一个人,一两个定时或手动的云端 agent 就够,再往上的 swarm 大多是过度,还多烧 credits。账单要会算:credits 分三桶 —— AI(推理)、compute(Warp 托管沙箱)、platform(run 生命周期与观测),同一池扣;每个 cloud run 至少扣 platform,用 Warp 托管算力加 compute,用 Warp 模型再加 AI。你本机配的 BYOK 云端拿不到 —— cloud run 照常消耗 Warp credits;Enterprise 要在云端用自己的推理,走 BYOLLM,不是本地那套 key。Cloud

Runners(算力配置)2026-07-23 起全员可用,配 `oz runner`;但 self-hosted execution 仍是 Enterprise。Oz 还能把 Claude Code、Codex 当云端 harness 跑(pricing 页标 beta):推理记你的 provider 账,Warp 仍计 compute 与 platform —— 你已付费的 CLI 订阅上云接着用,但不是免费。

4 6 5 验收这一章 —— 主线那两个 run,你要真的派出去、真的验收:

- [ ] 我用 `/cloud-agent` 或 `oz agent run-cloud` 派出了两个并行的 cloud run。
- [ ] 报告型任务:artifact 里的结论我抽了两条去源头核对。
- [ ] PR 型任务:我读完了 diff,让它过了 CI,不合格就拒。
- [ ] 跑顺的那个改成了 `oz schedule create` 定时或 `/create-skill`;一次性的那个停留一次性。
- [ ] 我知道无人值守的禁止清单:生产写入、删除、付款/发信、deploy、secret 输出。
- [ ] 上云前我确认了 credit 余额 ≥20,知道 BYOK 不覆盖云端。

参考.

01 Warp Docs · Oz Platform overview —— 截至 2026-08-06:Cloud Agents 运行在 Oz Platform 上 —— trigger(schedule、Slack / Linear / GitHub、CI、webhook、API、手动)创建任务,编排层(Warp cloud 控制面)跟踪生命周期,任务在 host(默认 Warp 托管;self-hosted 是 Enterprise)上执行,可选 environment;每个 run 留下 status、metadata、transcript、outputs 的持久记录。Oz CLI 是 headless 接口;另有 Agent API 与 Python / TypeScript SDK。Warp Docs · Oz Platform overview

02 Warp Docs · Cloud agents quickstart —— 截至 2026-08-06:入口是 /cloud-agent slash 命令(没有 environment 会引导你建;/create-environment 显式建);environment = Docker image + repos + setup commands(要求 glibc,不支持 Alpine / musl;有预置 dev images;GitHub auth 可按用户或按团队配);跑 cloud agents 需要 ≥20 credits(新用户有试用 credits);run 在 conversations 面板、session 链接、或 Oz web app(oz.warp.dev)的 Runs tab 看,手机也行;/create-skill 把一次 run 变成可复用、可调度的 skill;oz schedule create 建定时;Slack / Linear 等 integrations 需要 Build / Max / Business 的团队。Warp Docs · Cloud agents quickstart

03 Warp Docs · Handoff between local and cloud agents —— 截至 2026-08-06:本地 → 云端(仅 Warp Agent):fork 会话并应用未提交的 workspace 快照;云端 → 云端(Warp Agent 与 Claude Code / Codex harness):同一会话继续,workspace 状态恢复;云端 → 本地:Continue locally / /continue-locally,但 workspace 补丁不会应用到本地 —— 去审 branch / PR 这个 artifact;附件会带过去;补丁应用是 best-effort,部分失败会报告。Warp Docs · Handoff

04 Warp Docs · Credits —— 截至 2026-08-06:credits 分三桶:AI(推理)、compute(Warp 托管沙箱,仅 cloud run)、platform(run 生命周期 / integrations / 观测 —— 每个 cloud run 都扣,Business / Enterprise 用自带推理的本地 run 也扣);同一额度池(Settings > Billing and usage);每个 turn 有 credit chip 可 hover 看消耗;个人可以从自己的 credits 经 CLI / API 跑 cloud agents;Slack / Linear integrations 需要团队成员身份;普通 shell 命令不耗 credits。Warp Docs · Credits

05 Warp Docs · Harnesses in Oz —— Claude Code 与 Codex 可作云端 harness,共享 triggers / environments / secrets / skills / rules / 观测;推理由你提供的凭据记 provider 的账,Warp 计 compute 与 platform credits;在 Cloud Mode 的 harness 下

拉、Oz web app 或 API 的 harness 字段选;pricing 页把「Use any harness in the cloud」标为 beta(并列在 Free 下)。Warp Docs · Harnesses in Oz

06 Warp Docs · BYOK + changelog 2026-07-23 —— BYOK 不适用于 Cloud Agents:key 只存在你设备上,云端 run 拿不到,cloud runs 照常消耗 Warp credits;旧的「Enterprise team-managed keys」例外已取消,集中托管推理走 BYOLLM。2026-07-23 起 Cloud Runners 与 Cloud Agent Runners 全员开放,配 oz runner CLI;2026-08-06 本机轻测:Warp bundle 带 oz(v0.2026.07.01),oz --help 列出 agent / environment / mcp / schedule / secret 等命令。Warp Docs · BYOK

在场的活留本地,
离场的活放云端.

边界 · 画清哪些活不该交给它.

元工具不等于万能。命令行、文件、脚本、系统在线内;去点别的 GUI app 在线外。Warp 自己的 Computer Use 是云端沙箱里验证它写的界面,不是替你开本机应用。这章把线画清,也修正一条旧建议:远程与 SSH 场景,现在多了 Warp Agent CLI 这个选项。

前面几章都在说 Warp 能干多少活。这一章反过来:说它不该干的。元工具不等于万能 —— Warp 的 agent 强在命令行、文件、脚本、系统这一侧;一旦活儿要去点别的 GUI app,那是另一类工具的事。这章也把一条旧建议改掉:以前说「远程和 SSH 场景回普通终端」,现在多了 Warp Agent CLI 这个选项。线画清楚,比再多一个功能重要。

— I

元工具不等于万能.

命令、文件、代码库、交互式终端程序、MCP —— 这些都在线内,agent 在你真实的终端会话里动手。[4] 但要在你本机的设计软件里调图层、在某个网页后台里一页页点、在桌面邮件客户端里收发,那不是它的能力范围。把它当成「终端里的元工具」,而不是「替你操作电脑的万能助手」—— 这个定位差一格,期待就不会落空。判断很简单:这件事能不能用命令、文件、脚本表达?能,在线内;只能靠点一个非终端的图形界面,出界。

留在 Warp	不该留在 Warp 本地会话
shell 命令、脚本、文件批处理	网页表单、账号登录、付款、法律确认
日志分析、测试、构建、部署前检查	设计软件、Office 精排、图形化后台点选
MCP 只读资料、issue、docs、监控查询	密码管理器、生产控制台写权限、客户数据导出
云端沙箱里验证 UI 变更	在你本机替你点别的 app

– 要点

可截图判断:Warp 能做终端形状的事;Computer Use 能在云端沙箱里验证界面,不等于能安全控制你的整台电脑。

– II

它的 Computer Use 是验证,不是替你点.

Warp 确实有 Computer Use —— 但它是沙箱里给 agent 验自己的活,碰不到你本机。它只在 Warp 的沙箱云端环境跑,和你的本机、凭据隔离,用途是让 agent 在反馈闭环里验证自己改出来的界面 —— 它改了前端,自己开个浏览器看渲染对不对、表单校验过不过,完事留一段剪掉空闲时间的标注录像,可以直接附到 PR 上当验收证据。[1] 两处默认值 2026 年夏变了,别把老印象当事实:Warp 内置 harness 的云端 run,Computer Use 默认开(服务端默认);从 app 发起的 run 跟随 app 设置,默认关;第三方 harness 默认关。目前只支持 Anthropic Claude 模型,自动选,不用你挑。[1] 线没变:**沙箱里验证自己写的(Warp)** vs **在你机器上替你动手(Claude in Chrome、Operator 那类)**。要后者,就该开后者,别指望 Warp 这个沙箱伸到你的桌面上来。

任务	优先工具	最低边界
跑命令、改文件、看日志	**Warp Agent**	命令审查、dry-run、目录范围
测试本地网页 UI	Warp Cloud Computer Use / Playwright	沙箱、测试账号、无真实付款
登录真实网站替你提交	浏览器 agent / 手动	人类确认最终提交
桌面 GUI 软件操作	专用 RPA / Computer Use 类工具 / 手动	隔离账号、录屏/日志、可回滚

• • •

— III

何时普通终端或专用工具更好.

有些活留在 Warp 里反而更费 —— 该出手时就切出去。出界规则更新到四条半:离线、或锁死到装不了任何东西的环境,普通终端仍最稳;**远程与 SSH 的旧建议作废** —— 以前说「回普通终端」,现在有 Warp Agent CLI:`warp` 二进制把同一个 Warp Agent 带进任何终端,包括 SSH 会话,同账户、同 rules、同 skills,会话还能同步回账户、顺手 handoff 上云;服务器上跑一次 `/tui-migrate-setup` 还能把 app 里兼容的设置带过去(凭据不拷)。[3] 稳定重复的流水线,写成脚本挂 cron 比每次叫 agent 强;要驱动 GUI,用 Computer Use 那类工具;大段、跨多文件、靠编辑器来回改的活,切到 Cursor —— 终端侧和编辑器侧本就是一对元工具,下一章讲分工。[2] 知道什么时候不用它,和知道怎么用它一样,是判断力。

> **提示词**
>
> 留在 Warp 还是切走

> 判断这个任务该留在 Warp 还是切走：
> 任务：`<描述>`
> 它能否用命令/文件/脚本/日志表达：`<是/否>`
> 是否需要 GUI 点击、真实账号、付款、发送、权限变更：`<是/否>`
> 是否可 dry-run、小样本、回滚：`<是/否>`
> 涉及数据：`<公开/内部/客户/生产/密钥>`
> 请输出：留在 Warp / Warp Agent CLI / 普通终端 / Cursor / 浏览器 agent / 手动，并说明边界。
>
> 判断这个任务该留在 Warp 还是切走：
> 任务：SSH 到生产只读备库，查慢查询 top 10 并给出索引建议
> 它能否用命令/文件/脚本/日志表达：是
> 是否需要 GUI 点击、真实账号、付款、发送、权限变更：否
> 是否可 dry-run、小样本、回滚：是（全程只读 SELECT）
> 涉及数据：生产（只读）
> 请输出：留在 Warp / Warp Agent CLI / 普通终端 / Cursor / 浏览器 agent / 手动，并说明边界。

验收这一章 —— 把边界落到你自己的清单上：

- [ ] 需要切走的信号：任务核心是看屏幕、点按钮、跨账号状态、不可逆提交。
- [ ] 如果必须做 GUI 自动化：只用测试账号、测试数据、测试环境；最后一步提交、付款、发送必须人工确认。
- [ ] 写下 3 件你曾想（或差点）让 Warp agent 做、但它不在主场的活，并归类：普通终端 / 写脚本 / 切 Cursor / Computer Use 那类工具。
- [ ] 知道远程/SSH 的新答案：装一个 `warp` CLI，同一个 agent 跟着你进服务器；装不了的地方才回普通终端。
- [ ] 知道 Computer Use 的默认值：云端内置 harness 默认开，app 发起的默认关；它在沙箱里验证，不替你点本机。

参考.

01 Warp Docs · Computer Use —— 截至 2026-08-06:仍只在 Warp 沙箱云端环境,与本机和凭据隔离;但默认值变了:Warp 内置 harness 的云端 run 默认开启(服务端默认),从 Warp app 发起的 run 跟随 app 设置(Settings > Agents > Warp Agent > Experimental > Computer use in Cloud Agents,app 里默认关),第三方 harness 默认关;oz 有 --computer-use / --no-computer-use,API 有 config.computer_use_enabled;带 Chromium + 脚本化 Playwright;产物是标注过的录像 artifact(空闲时间剪掉,可附到 PR);目前只支持 Anthropic Claude 模型(自动选)。Warp Docs · Computer Use

02 Warp Docs · Full Terminal Use —— agent 在 PTY 里与交互式终端程序协作(psql、vim、gdb、REPL、dev server);这是 Warp agent 的主场,与「驱动桌面 GUI」是两回事。Warp Docs · Full Terminal Use

03 Warp Docs · Warp Agent CLI —— 截至 2026-08-06:warp 二进制(2026-07-23 由 warp-tui 改名)把 Warp Agent 带进任何终端,包括 SSH 会话;与 app 同一个 harness、账户、计划、rules、skills;自带 PTY(长跑与全屏程序可用);流式 transcript 带 diff / 工具调用 / 权限卡;首次跑 device-authorization 登录,CI 用 WARP_API_KEY / --api-key;会话持久并同步到账户;内置云端 handoff;认共享路径的 AGENTS.md / skills / MCP;Drive 里存的 Prompts 可用,其他 Drive 对象不行;/tui-migrate-setup 能拷兼容的 app 设置(凭据不拷);macOS(Apple Silicon + Intel)、Linux / Windows(x64 + Arm64)。Warp Docs · Warp Agent CLI

04 Warp Docs · Agents in Warp —— Warp agent 的能力围绕命令、文件、代码库、终端程序与 MCP;本地 agent 在你的真实终端会话里动手。Warp Docs · Agents in Warp

---

它能验证它写的界面,
碰不到你开的应用.

选型 · 按 2026-08 价目表选档与分工.

选档先分清你缺的是额度还是管控:Free 试水(现在带 Warp Agent CLI 与按量充值),Build 是多数个人的答案,Max 是重度并发的容量档,Business 管 SSO 与团队可见,Enterprise 才是 BYOLLM 与自托管。再划清 Warp、Cursor、CLI agent 与普通终端的分工。数字截至 2026-08-06。

这一章不问「Warp 值不值」,而问三个更实际的问题:你该付哪一档,模型钱该走 Warp credits 还是自己的 key,这台终端形状的元工具又该和 Cursor、CLI agents、普通终端怎么分工。数字都写死日期 —— 本章是 2026-08-06 的价目表,因为 Warp 的计费和模型表改得很勤。

– I

先看撞的是额度墙,还是管控墙.

截至 2026-08-06,Warp pricing 页显示 Free / Build / Max / Business / Enterprise 五档。[1] 不要按「越贵越强」理解它们,按「我现在缺什么」来读:Free 和 Build 答额度,Max 答容量,Business 答管控,Enterprise 答采购。

档位	截至 2026-08-06 的官方价目	适合谁	别误会成什么
Free	$0;无自带 agent 用量 —— 接 BYOK / custom endpoint / SuperGrok,或按 PAYG 价格充值 credits;含 Warp Agent CLI、有限 cloud access、cloud harness(beta)。	先验证 Warp 是否真能替你省终端工作 —— 手里有 provider key 的人最顺;只有 SSH 场景的人,光 Agent CLI 就值回安装。	长期重度 agent 档,或「免费送模型额度」。
Build	月付 $20 / 年付折算 $18;1,500 credits(约 $20 的 API 价 agent 用量);frontier + 开源模型;加购有量折扣、auto-reload、团队 spend cap;无限 Drive objects。	多数个人用户。你缺的是稳定额度,不是组织治理。	无限用量 —— 月度 credits 用不完不结转。
Max	月付 $200 / 年付折算 $180;18,000 credits(Build 的 12 倍)。	全天重度、多任务并发跑 agent 的个人。	团队管控档;它是容量,不是治理。
Business	月付 $50 / 年付折算 $45 每座;pricing 页写至多 25 座席(FAQ 页仍写 50,以 pricing 页为准);1,500 credits/座;团队指标、管理员数据控制、SAML SSO。	需要统一身份、数据控制、团队可见性的组织。	比 Build 更大的个人额度档。
Enterprise	定制;共享额度池、BYOLLM、自托管 cloud agents、Analytics API、custom indexing、cross-harness agent memory(Research Preview)。	AI 开发环境已进入采购、合规、平台工程的公司。	一个人想省钱的捷径。

— 实践提示

月度 credits 烧完了不用硬升档:加购 400/$10、1,000/$20、3,000/$50、6,500/$100,越买越便宜;auto-reload 默认关,开了也在余额 100 时才触发,默认月支出上限 $200 兜底。加购的 credits 随有效订阅结转 12 个月 —— 比月度配额的「不用就没」厚道。[2]

• • •

— II

模型名单与三条账本.

截至 2026-08-06,Model Choice 的名单:OpenAI 到 GPT-5.6(Sol / Terra / Luna)与 Codex 系;Anthropic 到 Claude Fable 5、Sonnet 5、Opus 4.8;Google 到 Gemini 3.1 Pro;xAI 到 Grok 4.5;Fireworks 托管的开源模型到 GLM 5.2、Kimi K2.7 Code、DeepSeek V4 Pro 等。不想挑就用 Auto 四档(Responsive / Cost-efficient / Genius / Open-weights)。两个心眼:pricing 页的营销名单跑在文档前面(Claude Opus 5、Kimi K3 之类),以文档名单为准;Fable 5 要求 Anthropic 数据保留、无 ZDR,在意数据策略的先看这条。[3] 同一件 agent 任务,钱可以走三条路:用 Warp 自带 credits,用你的模型商 key,或走一个 OpenAI-compatible endpoint。别把它们混在一起。

账本	省什么	硬边界
Warp credits	省配置:Auto 模型、custom routers、fallback、cloud/local agents 都走同一个账。	好模型更费:文档给的相对胃口 Opus 4.7 > Sonnet 4.6 > GPT-5.5 > Gemini 3.1 Pro;月度配额不结转。
BYOK	省 Warp AI credits:OpenAI / Anthropic / Google 由模型商直接计费,key 存本机,选中时优先于 credits。	不适用于 Cloud Agents;失败默认不 fallback(opt-in);ChatGPT / Claude 消费级订阅接不进;Warp 不能替你的模型商 enforce ZDR。
Custom endpoint	省供应商锁定:OpenRouter、LiteLLM、z.ai、公司内网关,三种 schema 可选(Chat Completions / Responses / Anthropic Messages)。	必须公网 HTTPS;localhost、私网、VPN-only 不行;不适用于 Cloud Agents;router 用不上 endpoint。
BYOLLM	省企业治理碎片:托管推理(当前 AWS Bedrock,Azure Foundry / Google Vertex 在路上)、编排、治理、观测一体接入。	Enterprise only,不是个人用户的本地模型开关。

最容易误判的还是 Cloud Agents:你自己配的 BYOK 与 custom endpoint 都是本地配置,云端拿不到,cloud run 照常消耗 Warp credits;用 Claude Code / Codex 当云端 harness 时,推理记你的 provider 账,但 compute 与 platform credits 照计。[4] [6]「Enterprise team-managed keys」那条旧例外已经取消 —— 集中托管推理的路是 BYOLLM,别按旧材料配。

提示词

Warp tier self-audit

请根据我过去 30 天的真实使用,给我一个 Warp 选档建议,不要按「越贵越好」推荐。

我的任务:
- 每周大概运行 agent: <次数>
- 其中多步改代码 / 排障: <比例>
- 其中文件批处理 / 数据清理 / 运维: <比例>
- 是否需要 Cloud Agents / Slack / GitHub 触发: <是/否>
- 是否需要 SSH / 远程服务器上的 agent: <是/否>
- 是否需要 SSO、管理员数据控制、团队 usage metrics: <是/否>
- 是否已有 OpenAI / Anthropic / Google key 或 OpenAI-compatible gateway: <有/无>

请输出:
1. 推荐档位:Free / Build / Max / Business / Enterprise
2. 推荐账本:Warp credits / BYOK / custom endpoint / 混合
3. 升档前的 7 天验证动作
4. 哪些任务不该放进 Warp

请根据我过去 30 天的真实使用,给我一个 Warp 选档建议,不要按「越贵越好」推荐。

我的任务:
- 每周大概运行 agent:约 40 次
- 其中多步改代码 / 排障:60%
- 其中文件批处理 / 数据清理 / 运维:30%
- 是否需要 Cloud Agents / Slack / GitHub 触发:否
- 是否需要 SSH / 远程服务器上的 agent:是,两台只读排查
- 是否需要 SSO、管理员数据控制、团队 usage metrics:否
- 是否已有 OpenAI / Anthropic / Google key 或 OpenAI-compatible gateway:有 Anthropic key

请输出:
1. 推荐档位:Free / Build / Max / Business / Enterprise
2. 推荐账本:Warp credits / BYOK / custom endpoint / 混合
3. 升档前的 7 天验证动作
4. 哪些任务不该放进 Warp

• • •

— III

Warp、Cursor、CLI agent、普通终端怎么分工.

Warp 官方对比 Claude Code 时,核心差别不是「谁更聪明」,而是表面:Claude Code 是 CLI tool;Warp 的 agent 内建在终端里,有可视化 diff、Profiles、Rules、多模型与 context referencing。[6] 所以别问谁赢,问这件事在哪个表面上最自然。截至 2026-08-06,这道分工题有两个条件变了。其一,「用 Warp」和「用你已付费的 CLI agent」不互斥:Warp 自动识别跑在它里面的第三方 CLI agent —— 官方名单 10 个,Claude Code、Codex、OpenCode、Amp、Auggie、Copilot CLI、Cursor CLI、Gemini CLI、Droid、Pi —— 叠上 rich input、code review、上下文引用等 UI 层(注意:agent 通知只有 Claude Code / Codex / OpenCode 三家支持)。其二,Warp 自己也出了 CLI:`warp` 二进制把 Warp Agent 带进任何终端含 SSH,同账户同 harness,Free 档就能用 —— 「远程场景回普通终端」的旧建议已被它改写。[5] 但别读成「Warp 罩住一切」:这是 Warp 想成为 agent 之上那层的产品策略,你的 CLI agent 在普通终端里跑一样成立;值不值得为这层驾驶舱多养一个 GUI 终端,仍按下面这张表判断。

任务重心	优先工具	理由
跑命令、读日志、查进程、写一次性脚本、批处理文件、排系统问题。	Warp	输入、输出、agent、审查、Drive、MCP 都在同一个终端上下文里。
在一个大仓库里长时间读代码、跨很多文件改结构、靠编辑器导航和 diff 推进。	Cursor / 编辑器侧 agent	编辑器是它的主场;终端只负责跑测试、生成证据。
SSH 进服务器、远程环境、要可移植会话与云端 handoff。	Warp Agent CLI(warp 二进制)	同一个 agent、同账户、同 rules 跟着你进任何终端;Free 档可用。
离线、锁死装不了软件的环境、别人的终端、极简可移植。	普通终端 + 你已付费的 CLI agent	少一层产品依赖;装不了 warp 的地方,订阅或 key 仍能带着走。
团队事件触发、后台排队、PR / Slack / Linear 触发、需要 run logs 与可见性。	Warp Cloud Agents / Oz	这不是本地终端体验问题,是编排、身份、额度和观测问题。
点网页后台、操作 GUI app、验证浏览器 UI。	浏览器自动化 / 专门 Computer Use;Warp 的 cloud sandbox 只用在适配场景。	Warp 的 Computer Use 是云端沙箱能力,不是接管你本机桌面。

— 补充

最实用的组合不是只用一个工具,而是让每个工具待在它的自然表面:Warp 管终端侧,Cursor 管编辑器侧,Warp Agent CLI 管远程,你已付费的 CLI agent 管可移植兜底,Cloud Agents 管后台与团队事件。工具栈变多不是问题,边界模糊才是问题。

• • •

— IV

最后用五个问题做决定.

- [ ] 我每周是否至少有 3 件终端侧的手动活,能被 agent 接手或被 Drive 沉淀?没有,先 Free。
- [ ] 我撞到的是 credits 不够,还是组织控制不够?前者看 Build / Max / 加购 / BYOK / custom endpoint;后者看 Business / Enterprise。
- [ ] 月度配额烧完前,我是否先算了加购的单价(最低约 $0.015/credit)再决定升档?
- [ ] 我有没有把 SSH 场景错判成「只能普通终端」?先试试 Warp Agent CLI,它在 Free 档就有。
- [ ] 我准备上 Cloud Agent 之前,是否确认了触发、上下文、验收、额度、日志与人工回滚?没有,就先留本地。

参考.

01 Warp · Pricing —— 截至 2026-08-06(逐字重取):Free $0,含「Bring your own AI inference」「Limited cloud agents access」「Warp Agent CLI access」「Reload credits at pay-as-you-go rates」「Use any harness in the cloud (beta)」,Drive 与云端会话存储有限;Build 月付 $20 / 年付折算 $18,含 1,500 credits(按 API 价格折 $20 的 agent 用量)、frontier + 开源模型、加购(volume discounts + auto-reload + team spend cap)、扩展 cloud access、最高 codebase indexing、无限 Drive objects;Max 月付 $200 / 年付折算 $180,18,000 credits(Build 的 12 倍);Business 月付 $50 / 年付折算 $45,pricing 页写至多 25 座席(pricing FAQ 文档页仍写 50,本书以 pricing 页为准并标日期),1,500 credits/座、团队指标、管理员数据控制、SAML SSO;Enterprise 定制:共享额度池、BYOLLM、自托管 cloud agents、Analytics API、custom indexing、cross-harness agent memory(Research Preview)。未用完的月度 credits 不结转;Build 是 2025-10-30 发布、2025-12-01 起替换旧 Pro / Turbo / Lightspeed 的档。Warp · Pricing

02 Warp Docs · Add-on credits —— 截至 2026-08-06:加购档位 400 credits/$10(基准 $0.025/credit)、1,000/$20(8 折)、3,000/$50(约 65 折)、6,500/$100(约 6 折);auto-reload 默认关、余额 100 credits 触发、默认月支出上限 $200(可调);加购 credits 随有效订阅结转 12 个月(Free 能留着但不能用);Build / Max / Business 按用户计,Enterprise 团队池;团队有团队级 spend cap 与管理员管理的 auto-reload。Warp Docs · Add-on credits

03 Warp Docs · Agent model choice —— 截至 2026-08-06 的名单:OpenAI GPT-5.6 Sol / Terra / Luna、GPT-5.5 / 5.4、GPT-5.3 / 5.2 Codex、GPT-5.2;Anthropic Claude Fable 5、Sonnet 5、Opus 4.8 / 4.7 / 4.6 / 4.5、Sonnet 4.6 / 4.5、Haiku 4.5;Google Gemini 3.1 Pro、Gemini 3.5 Flash;xAI Grok 4.5 / 4.3、Grok Build 0.1;Fireworks-hosted GLM 5.2、Kimi K2.7 Code、Kimi K2.6、Minimax 3 / 2.7、Qwen 3.7 / 3.6 Plus、DeepSeek V4 Pro。Auto 四档:Responsive / Cost-efficient / Genius / Open-weights。Fable 5 要求 Anthropic 数据保留(无 ZDR,Enterprise 默认关)。注意:pricing 页的营销名单(Claude Opus 5、Kimi K3 等)跑在文档名单前面 —— 以文档名单为准,标日期。Warp Docs · Model choice

04 Warp Docs · Credits + BYOK —— 截至 2026-08-06:credits 分 AI / compute / platform 三桶同一池扣;相对胃口(文档标日期排序)Opus 4.7 > Sonnet 4.6 > GPT-5.5 > Gemini 3.1 Pro,单次 run 不确定;Free 无自带 agent 用量,需 BYOK / custom

endpoint / SuperGrok 或升级;BYOK 选中时优先于 credits、失败默认不 fallback(opt-in);消费级 ChatGPT / Claude 订阅接不进;cloud runs 不支持用户级 BYOK。Warp Docs · Credits

05 Warp Docs · Third-party CLI agents + Warp Agent CLI —— 截至 2026-08-06:universal agent support 官方名单 10 个 —— Claude Code、Codex、OpenCode、Amp、Auggie、Copilot CLI、Cursor CLI、Gemini CLI、Droid、Pi;agent 通知只有 Claude Code / Codex / OpenCode 支持(需插件/配置);叠加能力:rich input(Ctrl-G)、code review、attach code as context、vertical tabs、Tab Configs、Remote Control。Warp 自己的 Warp Agent CLI(warp 二进制)把 Warp Agent 带进任何终端含 SSH,同账户同 harness —— Free 档就可用。Warp Docs · Third-party CLI agents

06 Warp Docs · Warp vs Claude Code + Harnesses in Oz —— 官方对比:Claude Code 是 CLI tool,Warp agent 内建于 Warp terminal,提供可视化 diff、Profiles、Rules、多模型、context referencing;Oz 可把 Claude Code / Codex 当云端 harness 跑,推理记你的 provider 账,Warp 计 compute 与 platform credits,pricing 页标 beta。Warp Docs · Warp vs Claude Code

---

选档看缺口,
分工看表面.

收束 · 30 天升级路线与全书验收.

把整本书压成一条 30 天路线:第一周首胜与底座习惯,第二周交办与沉淀,第三周接外部与权限,第四周云端与边界。三条 starter workflows 直接可抄,最后是一张全书验收单 —— 逐项勾完,你就是那个交办自如的人。

把整本书压成一句:你装的不是一个终端,是一台终端形状的元工具。它把命令行、agent、Drive、MCP、Cloud Agents / Oz 放在同一个工作台上;真正的变化不是多了一个输入框,而是你开始把命令行工作流当成可以派发、审查、沉淀、复用、放到后台的东西。这一章把十二章压成一条 30 天路线,最后给你一张全书验收单。<Footnote n={1} />

– I

30 天,跟着章序走.

别从「我要把 Warp 用满」开始,那会把工具变成负担。30 天路线就是这本书的章序 —— 每周两章的量,每周都有可勾的验收:

时间	章节	动作	验收
第 1 周	02 首胜 · 03 底座	装好、登录、设好三设置、完成第一次只读交办并存下第一个 Workflow;^R 找回昨天那条命令。	十分钟内拿到第一个有用答案;昨天的命令不靠翻 history 翻回来。
第 2 周	04 发动机 · 05 派活 · 06 沉淀	把一个真实报错修到好;跑一件带 dry-run 的批量活;把重复三次的操作固化成 Workflow。	修 bug 不用开浏览器搜;批量活有回滚;第三次重复不用再敲。
第 3 周	07 接外部 · 08 管住它	接一个只读 GitHub MCP 答一题,写类工具进 denylist;配出 local-dev profile 并当场验收。	多给了上下文,没多给写权限;该自动的自动、该拦的拦住。
第 4 周	09 放出去 · 10 边界 · 11 选型	派两个 cloud run 验收回来,一个改定时;写下绝不上云的那件;按缺口选定档位。	第一次上云是可检查的 run;边界清单落在纸上;档位有理由不是跟风。

· · ·

— II

从这三条 starter workflows 开始.

如果你不知道第一件该交给 Warp 的活是什么,就从下面三条里选 —— 它们分别对应第四、五、六章的主线,共同点是:终端侧、可审查、可复用,出错能回滚。

提示词

```
Starter workflow · local project debugging
```

请在这个本地项目里做一次只读排障。

目标：
- 找出 `<失败现象>` 的最可能原因
- 先不要改文件
- 可以运行只读命令、测试、lint、日志查看

请按这个顺序：
1. 先列你要看的文件和命令
2. 每次命令前解释风险
3. 总结证据，不要只给猜测
4. 给出最小修复方案
5. 等我确认后再进入改动阶段

请在这个本地项目里做一次只读排障。

目标：
- 找出 npm run dev 启动后访问首页就 500、终端报「Cannot read properties of undefined」的最可能原因
- 先不要改文件
- 可以运行只读命令、测试、lint、日志查看

请按这个顺序：
1. 先列你要看的文件和命令
2. 每次命令前解释风险
3. 总结证据，不要只给猜测
4. 给出最小修复方案
5. 等我确认后再进入改动阶段

提示词

Starter workflow · file batch dry-run

请帮我批处理这些文件，但先只做 dry-run。

范围：
- 目录：`<path>`
- 文件类型：`<ext>`
- 目标：`<rename/convert/extract/clean>`

安全要求：
1. 先生成 manifest，列出会碰到的文件
2. 先处理 3 个样本
3. 任何删除、覆盖、移动都必须等我确认
4. 输出完整命令和回滚办法
5. 通过样本后，再给 full-run 命令

请帮我批处理这些文件,但先只做 dry-run。

范围:
- 目录:./assets/photos
- 文件类型:.png
- 目标:convert,全部转成质量 80 的 .webp,原文件保留

安全要求:
1. 先生成 manifest,列出会碰到的文件
2. 先处理 3 个样本
3. 任何删除、覆盖、移动都必须等我确认
4. 输出完整命令和回滚办法
5. 通过样本后,再给 full-run 命令

提示词

Starter workflow · server health runbook

请把一次服务器健康检查整理成 Warp Drive Notebook。

目标主机 / 环境:`<host/env>`
检查项:
- disk
- memory
- process
- service status
- recent logs
- open ports

要求:
1. 每个检查项给命令、正常输出特征、异常解释
2. 默认只读,不要 restart / delete / deploy
3. 最后给「什么时候升级给人」的判断线
4. 把可参数化的部分标成 `<env>` / `<service>` / `<log_path>`

```
请把一次服务器健康检查整理成 Warp Drive Notebook。

目标主机 / 环境:web-prod-01 / production
检查项:
- disk
- memory
- process
- service status
- recent logs
- open ports

要求:
1. 每个检查项给命令、正常输出特征、异常解释
2. 默认只读,不要 restart / delete / deploy
3. 最后给「什么时候升级给人」的判断线
4. 把可参数化的部分标成 <env> / <service> / <log_path>
```

这三条跑通以后,才谈沉淀。Drive 不是收藏夹,它应该收纳已经被验证过、下次会复用的工作流。[2] 如果一个 Workflow 或 Notebook 三个月没人用,就删掉或重写;不更新的自动化,比没有自动化更危险。

• • •

— III

留下、沉淀、接外部、放出去、切走.

判断	用它处理	别越过的线
留在 Warp	命令、日志、文件、脚本、系统、数据清理;能通过输出与 diff 审查。	不要把不可回滚的生产操作交给自动批准。
沉淀到 Drive	重复出现、参数稳定、别人也会跑的过程。	不要保存没审过的危险命令。
接 MCP	外部系统的只读上下文、文档、issue、监控、内部 API。	第一条 MCP 不要给支付、生产数据库、客户数据导出、密钥库写权限。
放到 Cloud Agent / Oz	后台任务、事件触发、团队可见性、持续检查;每个 run 留下持久记录可复查。<Footnote n={3} />	触发、验收、额度、失败通知、人工回滚说不清,就不要上云。
带上 Warp Agent CLI	SSH 进服务器、远程环境、要可移植会话与云端 handoff 的活。<Footnote n={5} />	装不了软件的环境别硬装 —— 普通终端仍是兜底。
切给 Cursor / 编辑器	大段读代码、跨文件重构、架构迁移、靠 IDE 语义导航推进的活。	不要在终端里硬做编辑器的活。
切给浏览器自动化 / Computer Use	需要看页面、点击 UI、验证前端流程;Warp 的 Computer Use 是云端沙箱,不是本机控制。<Footnote n={4} />	不要把「能跑命令」误读成「能安全控制你的电脑」。

— IV

全书验收单.

十二章,每章一件可勾的事。逐项勾完,你不是「读过」这本书,是把它跑通过了一遍:

- [ ] 01 引子:我说得出 Warp 和普通终端的差别在哪 —— 不是输入框,是工作流可派发。
- [ ] 02 首胜:装好登录好,第一次只读交办拿到有用答案,第一个 Workflow 存进 Drive。
- [ ] 03 底座:^R 找回过一条昨天的命令,高频命令加了书签。
- [ ] 04 发动机:一个真实报错从交办修到验收,没开浏览器搜。
- [ ] 05 派活:一件批量活带 dry-run 和回滚跑完。
- [ ] 06 沉淀:一件重复三次的事固化成了参数化 Workflow。
- [ ] 07 接外部:只读 MCP 答了一题,写类工具在 denylist。
- [ ] 08 管住它:local-dev profile 配好并当场验收过 —— 该自动的自动,该拦的拦住。
- [ ] 09 放出去:两个 cloud run 派出去、照 transcript 验收回来,一个成了定时。
- [ ] 10 边界:写下三件不该交给它的活,和那条绝不上云的线。
- [ ] 11 选型:档位是按缺口选的,加购单价算过,分工表落在自己的工具栈上。
- [ ] 12 收束:30 天路线排进了日历,三条 starter workflows 至少跑通一条。

勾完这十二项,你就不会把 Warp 当玩具,也不会把它当神。它只是一台很强的工作台:命令在这里发生,agent 在这里动手,知识在这里沉淀,后台在这里接力,人仍然在这里负责。

参考.

01 Warp Docs home —— 截至 2026-08-06,Warp 官方把它定义为 open-source Agentic Development Environment,agents 由 Oz(orchestration platform,本地与云端)驱动:terminal、agents、Drive、MCP、Cloud Agents / Oz 共用一套本地与云端工作台。Warp Docs

02 Warp Docs · Warp Drive —— Drive 保存 Workflows、Notebooks、Prompts、Environment Variables,并在个人与团队空间中同步;适合把跑通过的一次性过程沉淀为可复用资产。YAML 文件式 workflows 已是 legacy(只在 Command Search / Palette,不进 Drive)。Warp Docs · Warp Drive

03 Warp Docs · Oz Platform overview —— 截至 2026-08-06,Cloud Agents 适合事件触发、后台运行、团队可见性与持续任务;每个 run 留下 status、metadata、transcript、outputs 的持久记录,可检查与分享。Warp Docs · Oz Platform overview

04 Warp Docs · Computer Use —— 截至 2026-08-06,Warp 的 Computer Use 运行在 Warp sandboxed cloud environments(内置 harness 的云端 run 默认开,app 发起的跟随 app 设置、默认关),不是本机桌面控制;产物是可附到 PR 的标注录像。Warp Docs · Computer Use

05 Warp Docs · Warp Agent CLI —— 截至 2026-08-06,warp 二进制把 Warp Agent 带进任何终端含 SSH,同账户同 harness,会话同步回账户,Free 档可用;装不了软件的环境才回普通终端。Warp Docs · Warp Agent CLI

一台终端形状的元工具真正的价值,
不是替你控制电脑,
而是让命令行工作流少丢一次。.

WARP

终端形状的元工具.

不只是更漂亮的终端 —— 一台带 agent、接 MCP 的元工具:10 分钟首胜开始,逐章跑通一件真事。

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
12	2.1 万字	2026.08	PDF 1.0
