# 迈向 Agent 原生的 Computer Use


## 从人类软件到智能体软件

软件多了一类使用者：会计划、会调用，也会自己检查结果的智能体。

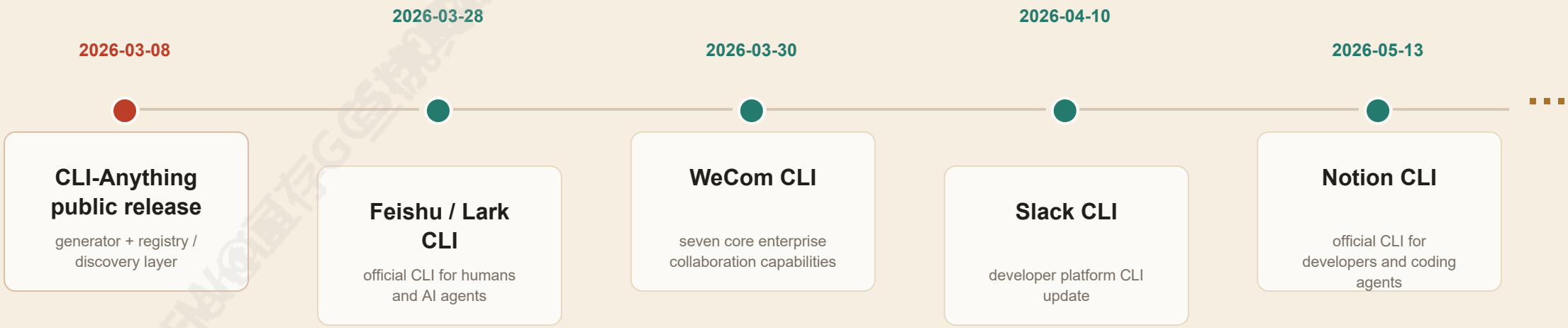
```
$
```

arXiv:2606.03854

HKUDS/CLI-Anything

# 各大软件 CLI 时间线：agent-facing CLI 正在进入越来越多的主流软件。

CLI-Anything 最早系统化呈现这条方向；随后协作软件和平台集中推出 CLI / agent-facing interface。




**CLI-Anything 先把 software-for-agents 的接口层做成 generator、registry 和发现机制**



Sources: Lark/WeCom/Slack/Notion public release records; OSSInsight agent-native CLI wave (2026)

# CLI-Hub 已经能直接查找、安装和预览 CLI。

可安装的 CLI、贡献者、场景矩阵、真实 demo 和调用次数，都在同一个入口里。

| | |
|---|---|
| **96** | **CLIs** available catalog |
| **35** | **categories** combined registry |
| **5 / 62** | **matrices** scenario routing |
| **79** | **contributors** registry identities |


site capture: clianything.cc, 2026-06-25

❌静态清单，☑能直接上手的工作台。

# 四个关键数字：目录、社区、调用量和使用结构。

CLI-Hub 这边，人在浏览，agent 在调用，社区也还在不断往里补内容。

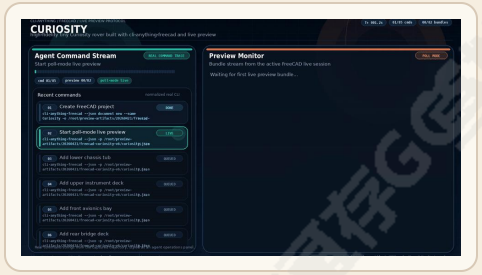
| 43,790 | 4,097 | 259,574 | 133,505 |
|---|---|---|---|
| **GitHub stars** | **forks** | **CLI-Hub calls** | **web visitors** |
| public repo API | community replication | live public stats | site traffic |


**83.96% agent share · 5.23x agent/human ratio**

Live stats 截至 2026-06-25；tech report Figure 12 是早期快照：129,916 calls / 87.8% agent share。

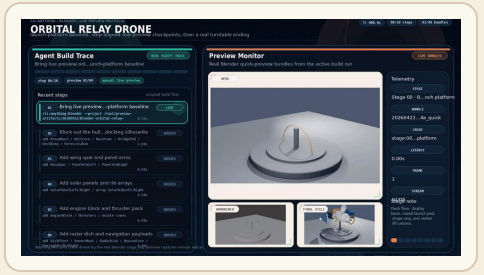
# 数据之外的真实用例。

命令、状态、预览、真实后端和最终产物，在 CLI-Anything 串成了一条线。



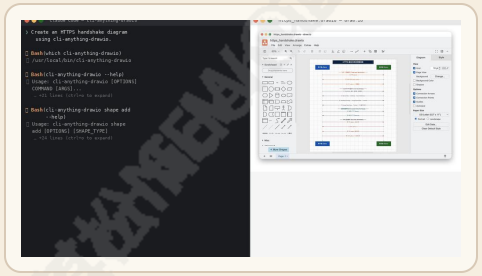

**FreeCAD**

preview + trajectory



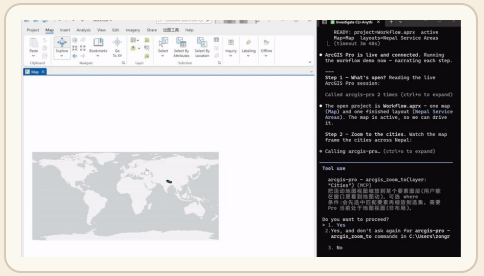

**Blender**

render path



**Draw.io**

diagram artifact



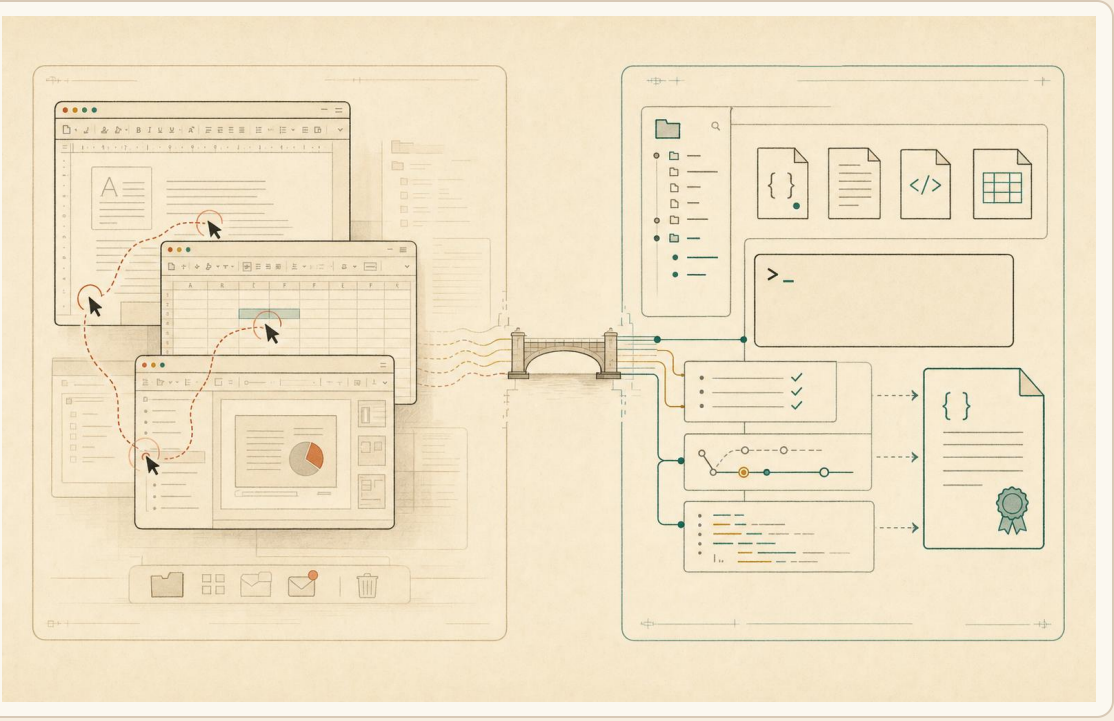
**ArcGIS Pro**

live bridge

**CLI-Hub 目录证明它覆盖得够广，demo 证明命令真能落到软件上、跑出真实产物。**

# Agent 操作软件，一定要靠 GUI 吗？

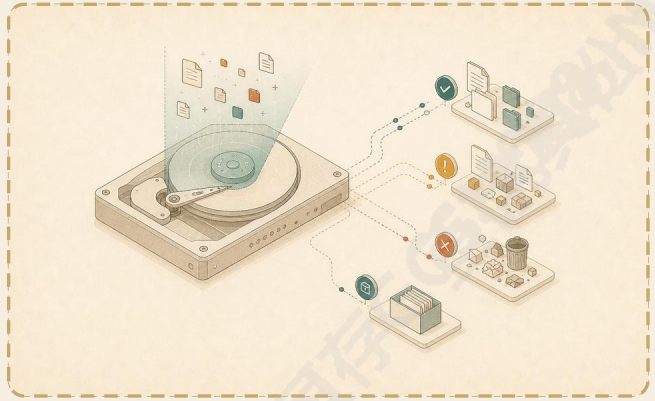
OSWorld 让桌面任务到底有多难看得很清楚；Codex 和 Claude Code 干脆换了条路——命令、文件、状态和测试。



从屏幕路径，到工作区路径。

# 其实，很多软件都能用代码来驱动

很多软件的背后，都有一套可被发现、可被遵循的底层协议。读懂它，用代码就能直接生成等价的结果——不必再去点界面。

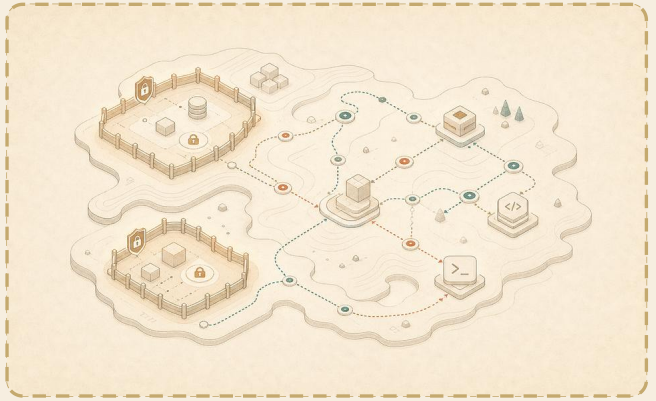


## 磁盘清理工具

它的清理本质上是对文件系统的一套扫描与操作。摸清这层底层接口，coding agent 就能用代码做出等价的结果。

## 幻灯片 .pptx

一份 pptx 本质是一套开放的 XML 标记（OOXML）。就像写 HTML，用代码描述就能生成任意一页。



## 更大的数字世界

数字世界里的事大多与代码相关。有些被法律或服务条款划了边界，但仍有大片可以用代码接入的空间。

**找到协议、遵循协议、用代码生成等价物——这就是 CLI-Anything 实现 computer-use 与 software-use 的方式。**

# OSWorld 和 Coding Agents 指向两种软件使用方式。

一种照着人去点屏幕；另一种把任务交给工作区、终端、文件和测试，在一轮轮 review 里收尾。

### GUI Computer Use

- 真实 OS、网页、桌面应用。
- 输入是 screenshot / accessibility tree。
- 动作是鼠标、键盘和窗口操作。
- 最终由 evaluator 检查文件或系统状态。

### Coding Agents

- 读 codebase，改文件，跑命令。
- 把工具链、测试、git/CI 放进循环。
- 任务可以并行委托到 cloud/local workspace。
- 结果以 diff、test、artifact 形式复查。

**两条路的道理其实一样：状态看得见、命令跑得动，agent 才靠得住。**

# 例如 Office 任务看似是 GUI，成功条件仍然落在文件和状态。

OSWorld 的 LibreOffice 任务用屏幕操作进入软件，但 evaluator 最终只会检查 `.xlsx`、`.docx`、`.pptx` 等最终产物。

## LibreOffice via GUI

**OSWorld task surface**

- Calc / Impress / Writer
- screen + accessibility tree
- pyautogui / computer actions
- VM 中执行真实软件

## Artifact evaluator

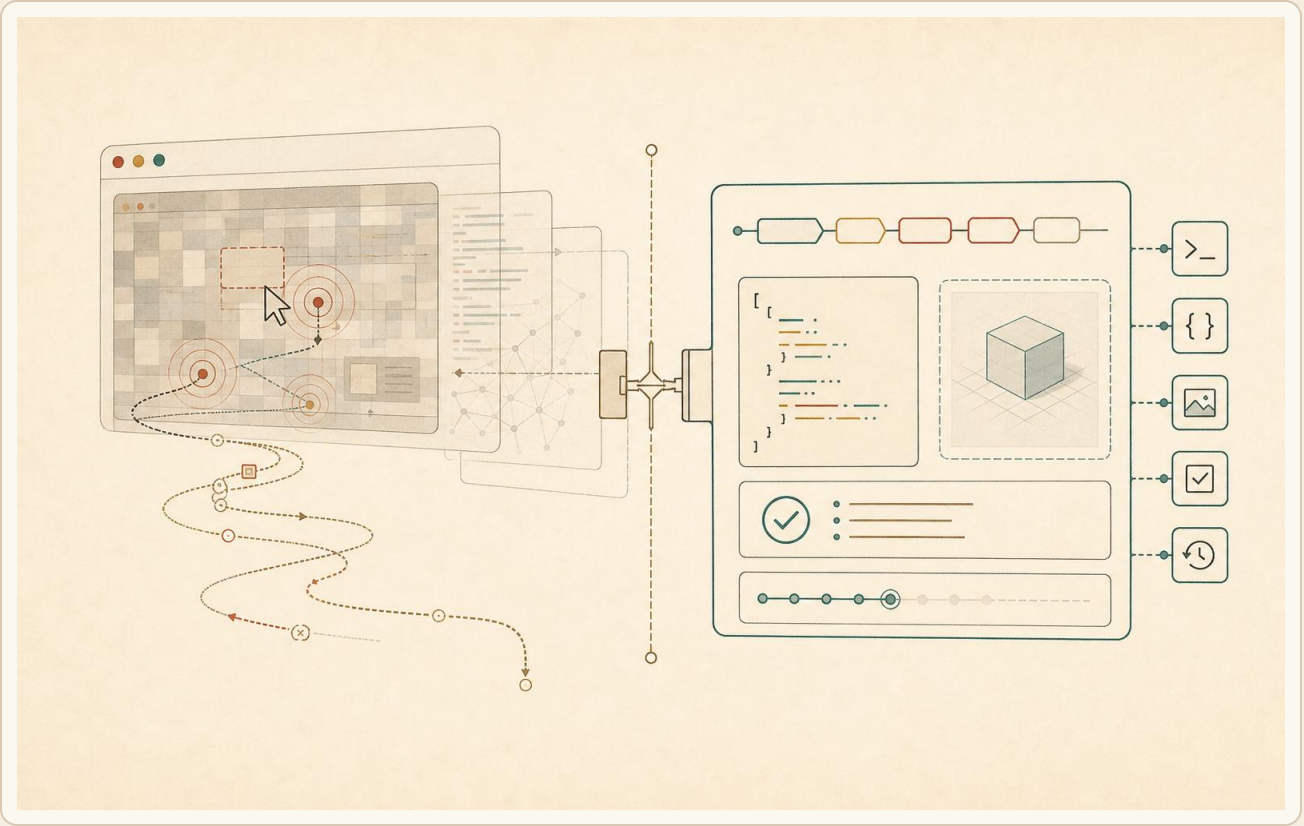
**success condition**

- spreadsheet cell values
- document paragraph spacing
- slide text color
- gold file comparison

**GUI 仍然重要；但 agent 不一定只能通过 GUI 工作。**

# 从点击屏幕，到执行合约。

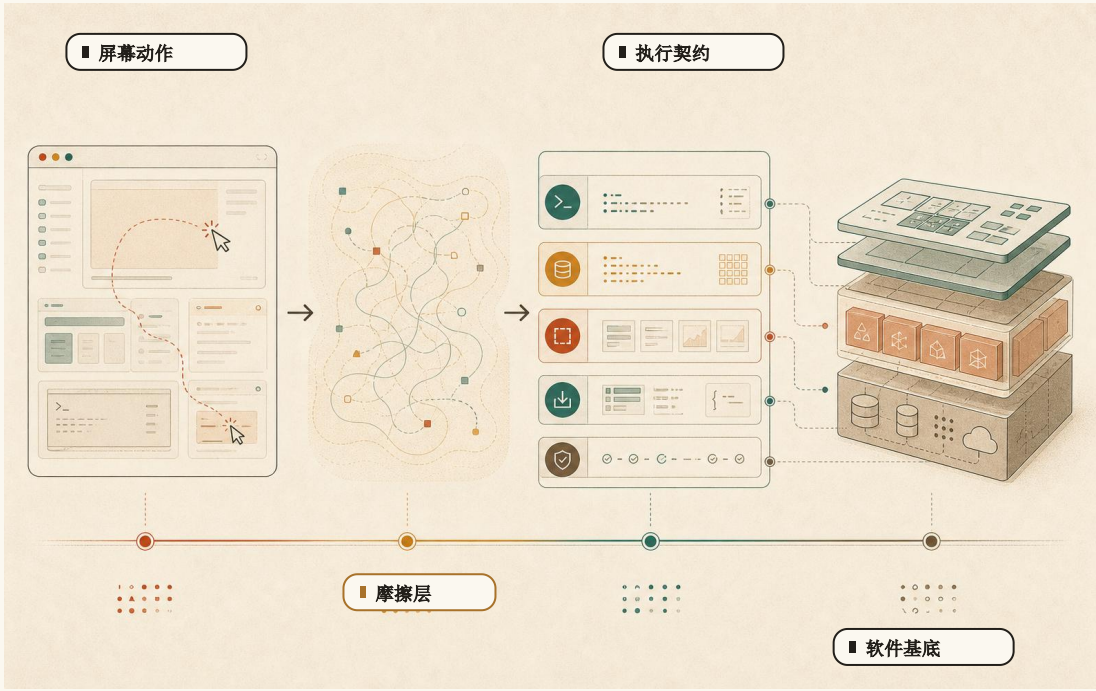
当软件把命令、状态、检查和导出都摆出来，智能体就不必再把每个意图都翻译成一次次点击和切换焦点。



从像素表面，到可执行边界。

# 软件边界不一定停在屏幕动作上。

CLI-Anything 保留真实软件后端，把 agent 的控制点放到命令、状态、预览、导出和验证上。



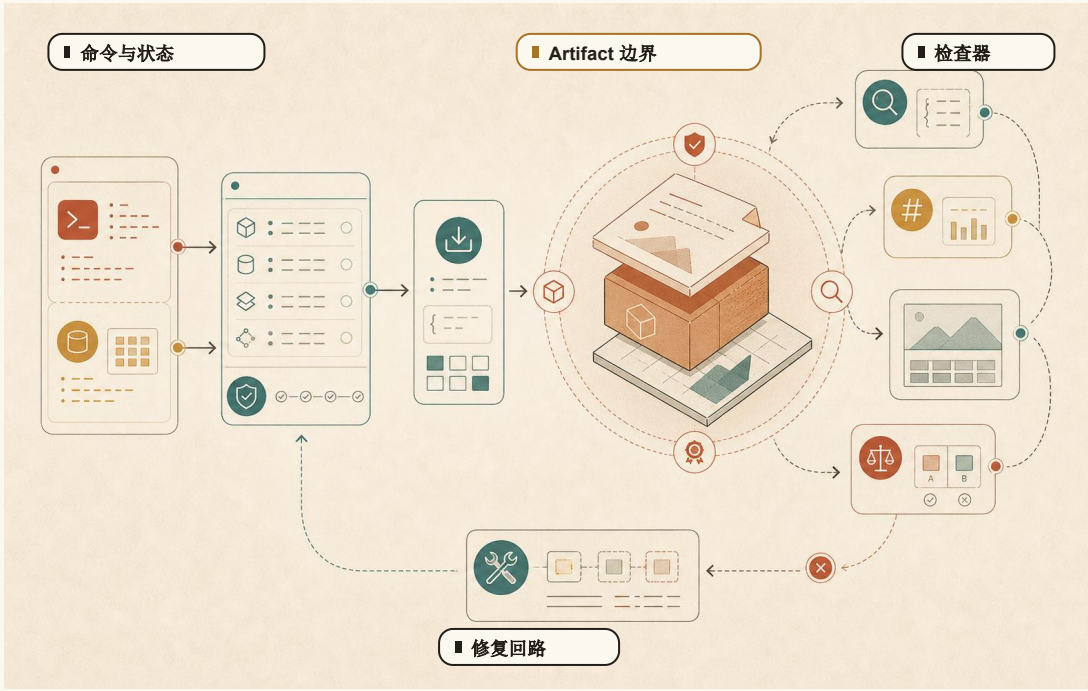

示意：从屏幕边界到执行合约。

**软件本身留着，换掉的只是入口——给 agent 一个它能直接执行的边界。**

GUI 还在原地，被抬上来的是 agent 的控制边界。

# 绕过 GUI 后，必须用 artifact 收口。

只要产物能被解析、计数、渲染或比对，agent 的整个构造过程就该落到同一个验收点上。



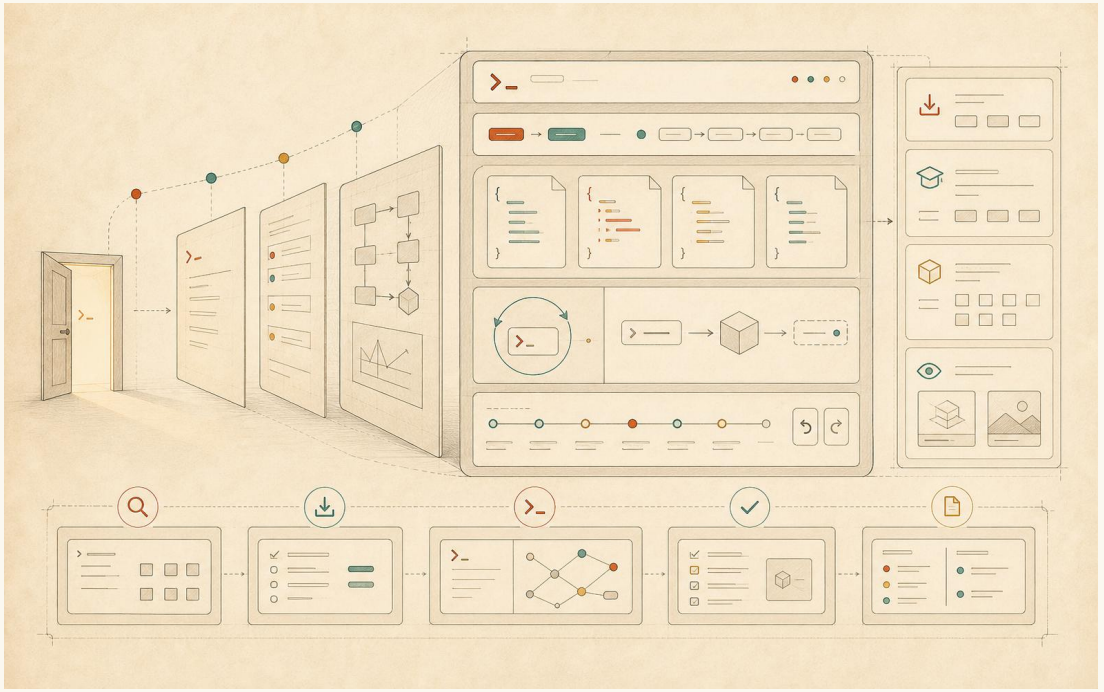

示意：artifact boundary 关闭工程循环。

**验证的对象，是最终产出的成果。**

- 构造过程交付同一个 artifact boundary。
- 验证器看文件、结构、像素和 metadata。
- 失败反馈回到下一轮状态修复。

# 绕过 GUI 以后，agent 该拿到什么界面？

用 CLI 不为怀旧——它能把软件收拾成一个干净的操作面：
能拼装、能查状态、出错能退回，也方便被找到。



CLI 作为渐进式执行面。

# CLI 和 MCP 解决的是两层问题。

MCP 管的是宿主怎么发现、怎么调用工具；CLI 提供的则是另一层——模型熟、好写脚本，也方便被 MCP 包起来。

### MCP

连接协议

- 标准化 host 与外部系统连接。
- server 暴露 tools / resources / prompts。
- 适合跨客户端复用和权限治理。

### CLI

执行界面

- 模型长期见过 terminal、flags、logs。
- stdout/stderr/exit code 支持恢复。
- 可被脚本、CI、MCP server 直接包装。

### CLI-Anything

软件契约

- 把真实软件后端封装成一个 CLI。
- 再补上状态、预览、验证和发现。
- 让 MCP 或 agent 面对一个稳定的能力面。

**CLI 可以站在 MCP tool 背后执行；MCP 可以把 CLI 接到更多 agent host。**

# CLI 特别适合一层层地把信息交给 agent. “Progressive Disclosure”

Agent 可以先读 help 和 status，用 dry-run 看计划，再执行真正会改状态的命令。

CLI / JSON

```
$ tool --help
$ tool scene --help
$ tool status --json
$ tool plan --dry-run --json
$ tool apply plan.json
$ tool verify artifact --json
```

**01**
**Discover**
先读能力和约束

**02**
**Inspect**
读取状态和历史

**03**
**Plan**
dry-run 生成可检查计划

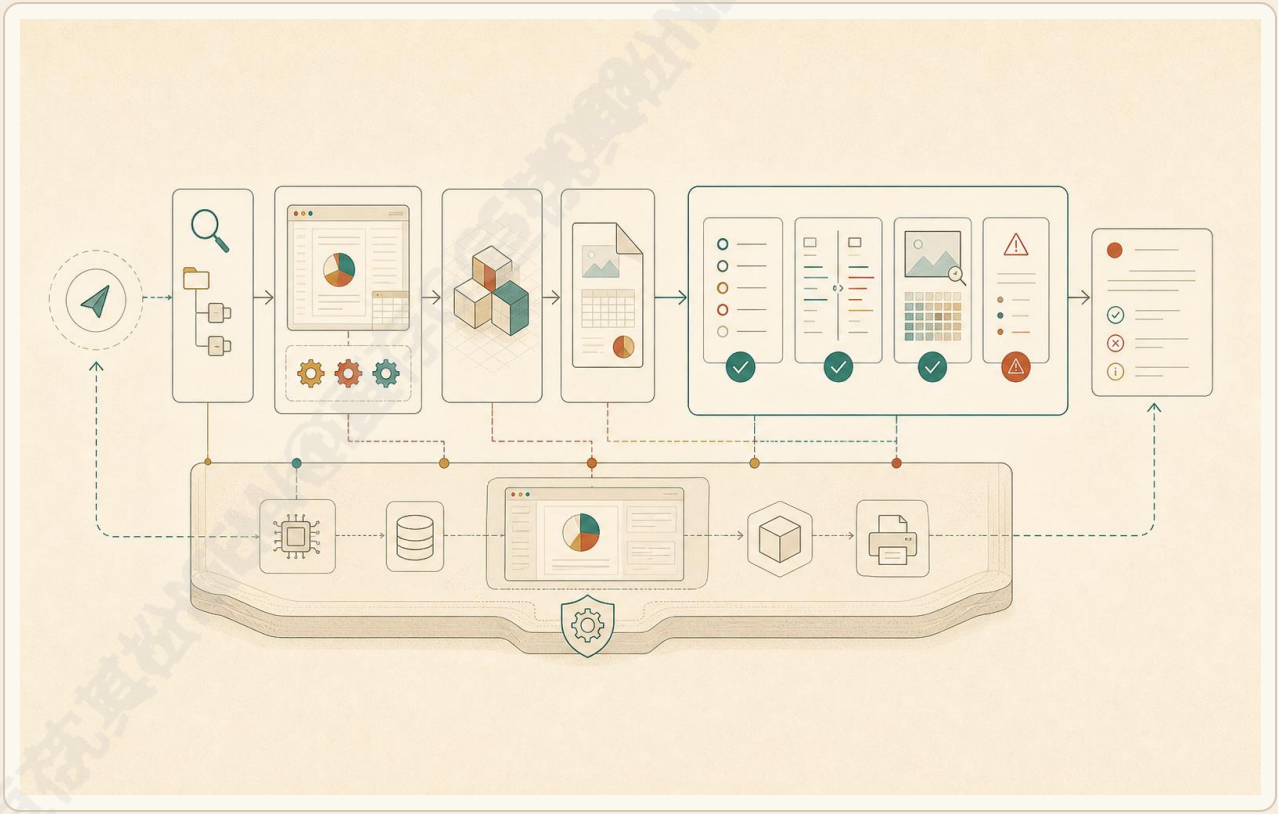
**04**
**Act**
执行最小 mutation

**05**
**Recover**
用 logs / undo / retry 修复

**先看边界再行动；每一步都有可解析的文本证据。**

# 真实软件必须留在执行链路里。

Harness 调真实后端、查导出的文件，出了问题就原样抛回给 agent。

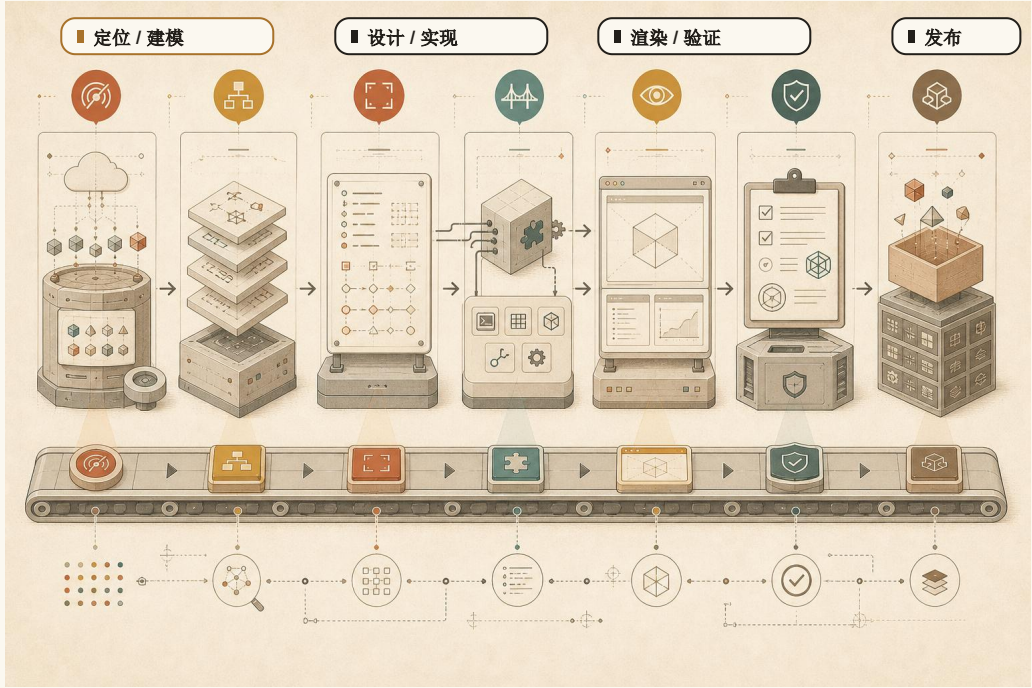


真实后端负责渲染、导出与失败反馈。

- 后端是否可用，本身就是这份契约的一部分。
- 预览和导出，都应该来自真实的软件本身。

# 自动把任何软件，变成 CLI + SKILL.md

输入一个软件，产出一套 CLI 和一份 agent 能读的 SKILL.md。中间这条流水线，其实只做两件事：




示意：harness lift 作为证据链。

- ① 拆解 + 实现：定位真实后端，做成带类型的命令。
- ② 复杂端到端测试跑通，CLI / GUI 产物等价、统一。
- 人用 GUI、agent 用 CLI，面对同一个软件。

**任何软件、任何代码库，都能这样自动接进来。**

CASE STUDIES

# 三个案例，绑定真实软件

Blender、Slay the Spire II、FreeCAD / CAD —— 命令、状态、预览和验证，都接到真实软件或真实运行时。

## Blender

Lift backend · 3D / 媒体

› 复用 bpy / RNA / operators，把 GUI 之下的场景引擎变成 agent 接口。

› 54 个命令改 scene / object / render 状态；后台 blender 校验导出产物。

→ **54 命令 · 228 测试**

## Slay the Spire II

Bridge runtime · 游戏

› 状态散在房间、UI 和运行对象里；运行中建 bridge，归一成 compact JSON。

› BuildGameState 读取、RunOnMainThread 执行；CLI 管解析、超时与恢复。

→ **15 个归一化状态 / 帧**

## FreeCAD / CAD

Lift backend · CAD

› 复用 FreeCAD 的 Python API（App / Part / Sketcher），跑在无界面的 FreeCADCmd 控制台里。

› 命令直接驱动 OpenCASCADE 几何内核与草图约束求解器，导出和预览都走原生链路。

→ **258 命令 · 17 组**

**这些证据都来自真实运行的中间产物——可回放，也能追溯到具体的命令和状态。**

# 两种增长模式，同一套契约

substrate 不同 —— 可能是文件、API，也可能是运行中的进程；但状态、命令、预览、验证的契约不变。

## Lift backend

Blender / CAD / 媒体工具

- native 状态已经存在，后端 API 或文件格式可达。
- harness 把 JSON 意图降到真实执行。
- render / export 仍由原生软件完成。

## Bridge runtime

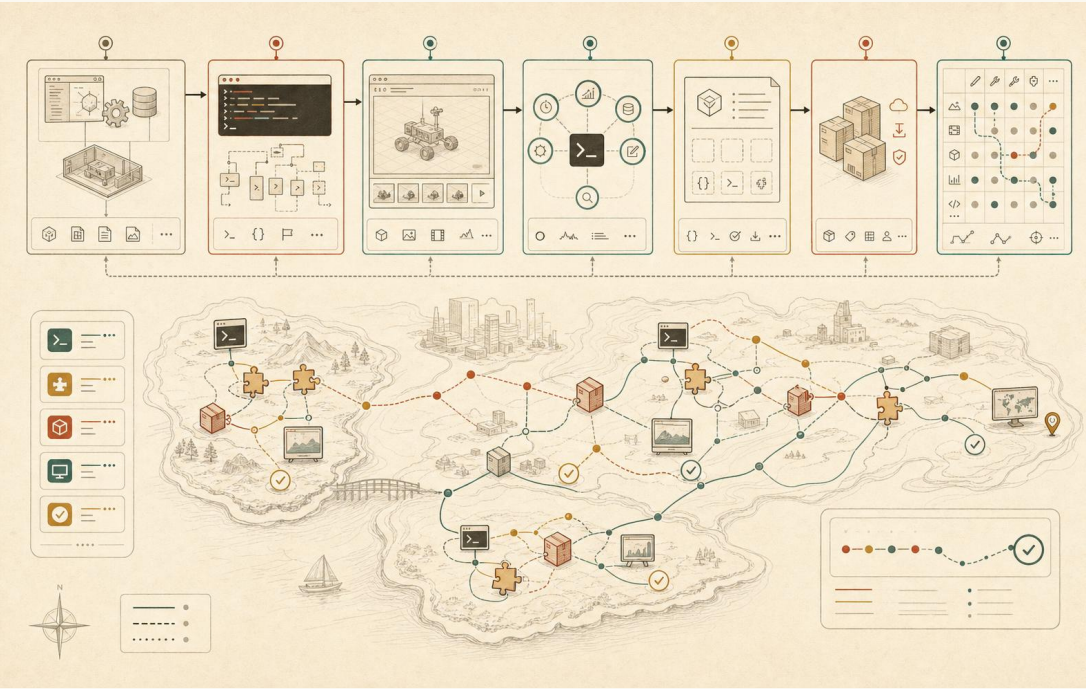
游戏 / 浏览器 / 在线系统

- 状态藏在运行中的进程里。
- bridge 归一化读取与动作。
- CLI 负责超时、解析与恢复；用测试固化 adapter 行为。

**不管 substrate 是文件、API 还是 live runtime，agent 面对的契约保持不变。**

# CLI-Anything 进一步把接口做成机制和生态

先用一个 harness 把路跑通；
再靠 preview、SKILL.md、CLI-Hub、Matrix 和 registry，让这套接口能被找到、装上、复用，也接着往外扩。



从单个 harness，到可路由生态。


freecad> preview capture
freecad> preview live status
trajectory.json -> replayable visual history

# Preview 把命令、状态和视觉检查接起来。

FreeCAD rover demo 记录 preview、live preview 和 trajectory history。

- 每一步都有可见中间态。
- **trajectory** 让命令和画面互相对齐。
- 最终 **artifact** 来自真实 **FreeCAD** 工作流。

Evidence: HKUDS/CLI-Anything demo asset, main@bf3cc39

# CLI-Matrix 让 catalog 按场景组织起来。

CLI-Hub 的 Matrices 把 5 个垂直场景、62 个 capability 和成员 CLI 绑定在一起。


**5**
matrices

**62**
capabilities

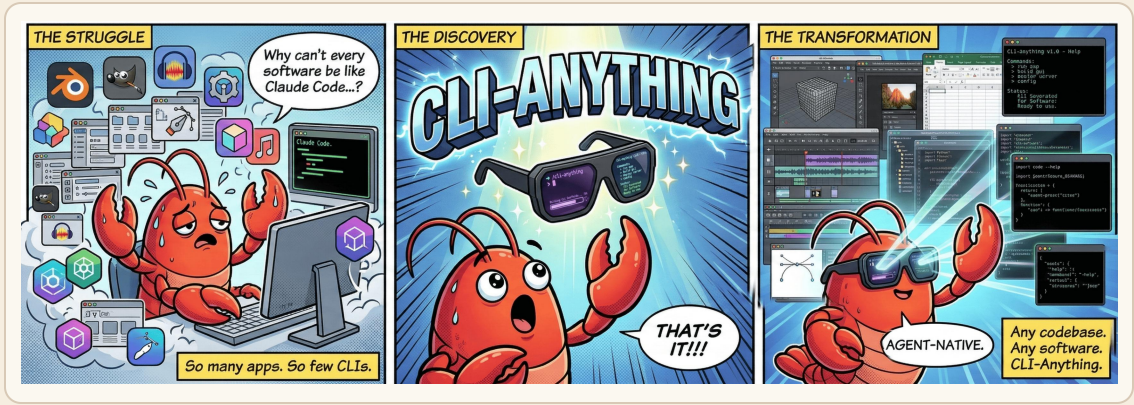
**49**
member CLI links

**5**
SKILL.md bundles

**Matrix 关心的是一个场景需要哪些能力、哪些 CLI、怎样安装和预检。**

# 给 agent 的软件边界，要能执行、能检查、能恢复

CLI-Anything 保留真实后端和人类界面，
把状态、命令、预览和验证
接到同一条工程链路里。




扫码访问 clianything.cc

Let’s connect and join the community!