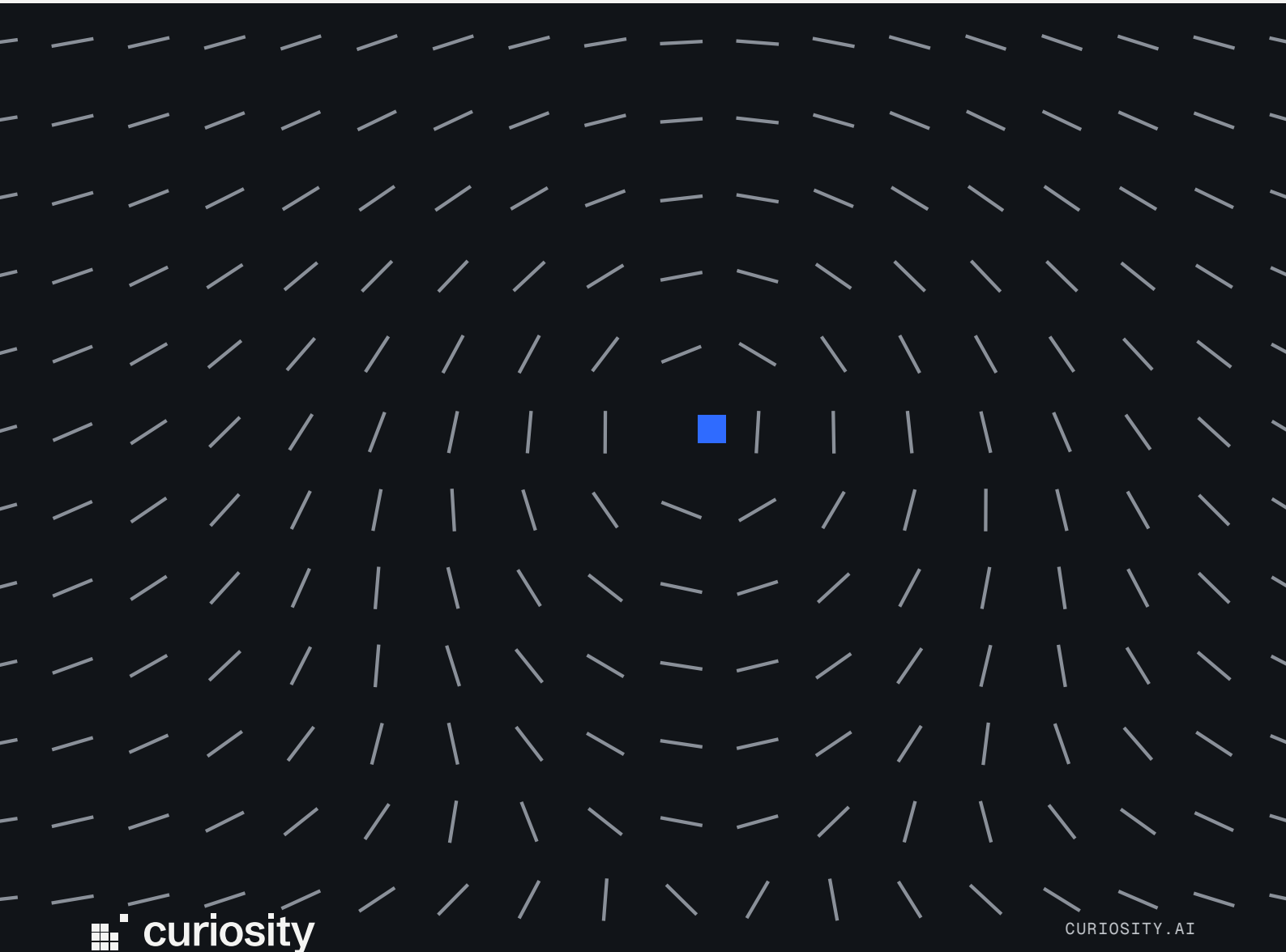


Inside Sudo: a sandbox, a diff, a person.

Sudo, the admin assistant in Curiosity Studio, changes your configuration in a sandbox and hands you a diff. This paper shows what the sandbox holds, what it lacks, and what an approval binds.

TOPIC	FOR	DATE
Sandbox and approval	Platform admins and security architects	October 2026



No route to the host. No route in but a diff.

01	Configuration is already files One filesystem and one tool reach every surface.	03
02	Nine mounts, two writable Only a write under /workspace is a change.	05
03	An interpreter, not a shell No host, no processes, no network, by construction.	07
04	Commit stages, a person applies Approval is bound to the commit you were shown.	11
05	Permissions bound Sudo from below Sudo reads as you and writes none of your data.	15
06	Accountability and reuse Every change audited, and the same sandbox for your assistant.	17
.	Glossary	21

IN ONE PARAGRAPH

Sudo has no route to the host or the network, and no route to your workspace except a diff an administrator approved. The assistant works in an in-process interpreter over nine mounts, two of them writable. A commit stages a change and stops. A person applies it, under their own permissions, into the history you already audit.

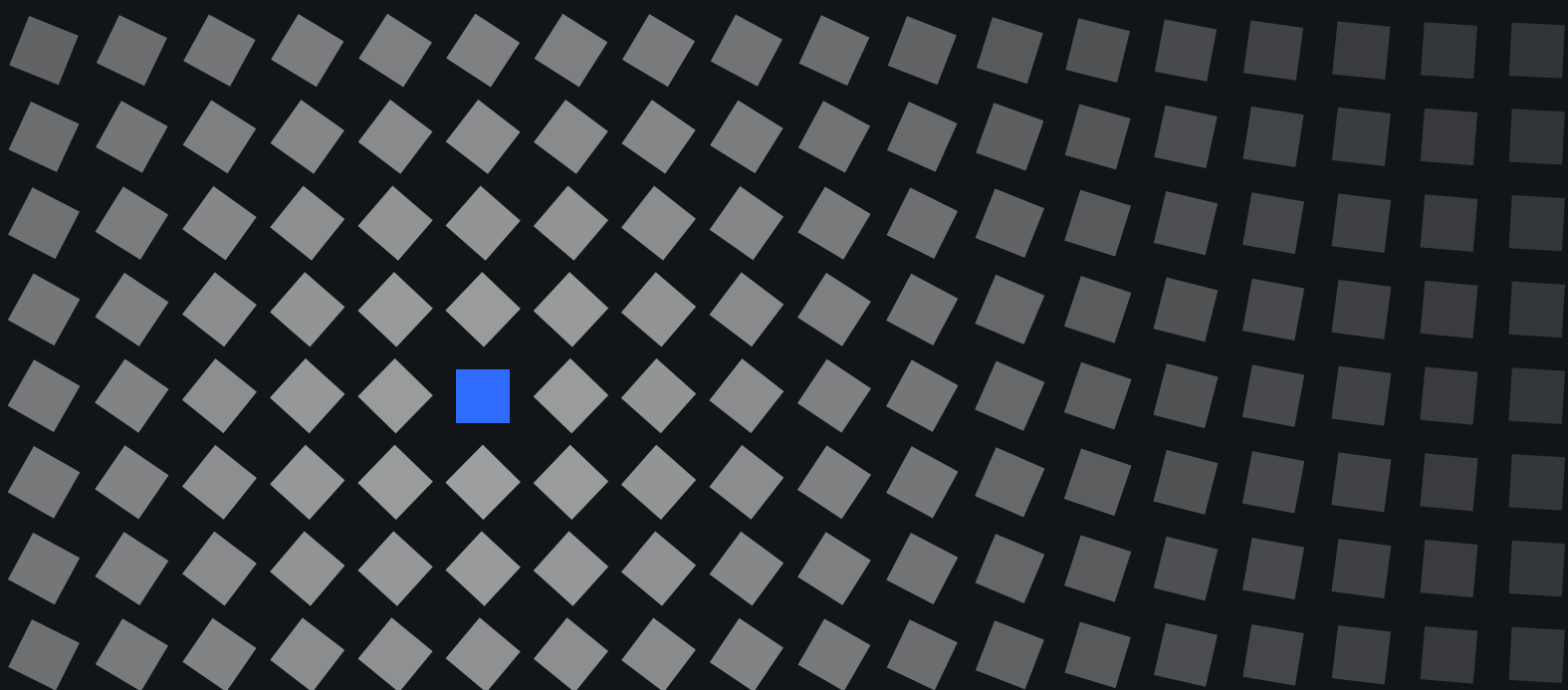
01

Configuration is already files

Why per-surface tools grow without end

The three ways your configuration already travels as files

Why one filesystem and one tool is smaller to secure



1.1

One filesystem is smaller to secure than many tools

Your configuration already round-trips as plain files. So one sandbox and one tool reach every surface, and one diff shows the change.

A workspace has many surfaces: endpoints, AI tools, agents, skills, prompt templates, scheduled tasks, migrations, connectors, indexes, node schemas, renderers, NLP pipelines, chat policies and search settings.

Each has its own admin page. A new kind of configuration grows a new screen.

An assistant built on per-surface tools grows the same way. Every new surface needs a new tool, and every tool is one more authority to review.

Real work also spans surfaces. "Let users filter tickets by product" is a schema field, an index setting, a facet and possibly a migration. Edited screen by screen, it has no diff.

All of it already round-trips as plain files: the definitions export zip, the WebDAV mount and the git-tracked configuration. So Sudo gets one filesystem and one tool, and works the way a developer works.

NEW SURFACE

A new kind of surface needs no new tool. A new export folder is a new folder.

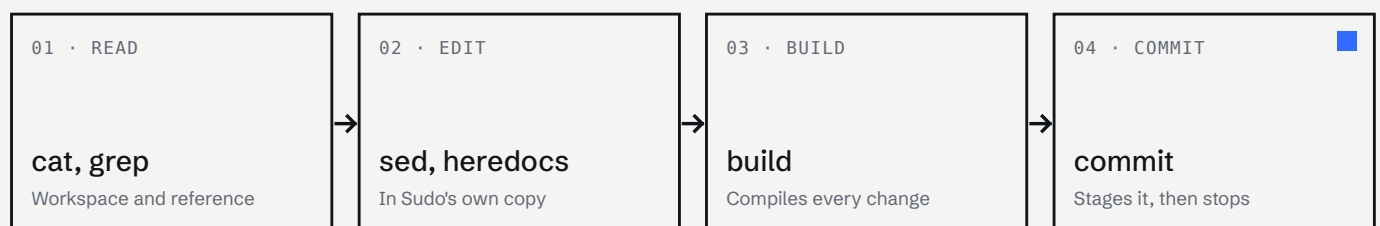


FIG. 1.1 · ONE TOOL FOR EVERY SURFACE. THE COMMIT IS WHERE SUDO STOPS.

02

Nine mounts, two writable

What each mount holds, and which ones take writes

Why /proc is read live on every request

What makes diff and commit answerable

2.1

The sandbox's root, mount by mount

Every session starts at this root. Only `/workspace` and `/tmp` take writes, and only a write under `/workspace` is a change to your workspace. So `diff` and `commit` compare one directory with the live configuration.

```
/
├─ workspace/      the configuration, as the definitions export writes it  [writable]
├─ reference/      a generated example workspace, one file per surface  [read-only]
├─ docs/           one page per code-execution scope, plus the schema helper [read-only]
├─ sdk/            the C# surface your workspace code is written against [read-only]
├─ proc/           live runtime state, regenerated on every read        [read-only]
├─ session/        Sudo's own to-do list                               [read-only]
├─ stage/          the files you uploaded to the Stage area, as text    [read-only]
├─ home/uploads/   the files you attached to this conversation          [read-only]
└─ tmp/            scratch, discarded with the session                  [writable]
```

LISTING 2.1 · THE SANDBOX'S ROOT, AS THE DOCUMENTATION SHOWS IT

PATH UNDER /PROC	WHAT A READ RETURNS
nodetypes, edgetypes	Types with counts, as TSV
exceptions	The 100 most recent, as JSON
audit	Who changed what, and when
runs/<id>/	status, log, output of this session's runs

2.2

/proc is regenerated on every read

The same `cat` a minute later shows how far a running task has got, and `jq` works on it directly.

`/reference` holds a well-formed example of every file kind, so Sudo reads the shape rather than recalling it.

`/home/uploads` holds what you attach: a PDF or spreadsheet reads as extracted text, a `.zip` as a directory.

READ ONLY

Seven of the nine mounts take no writes. `/tmp` is scratch and is discarded with the session.

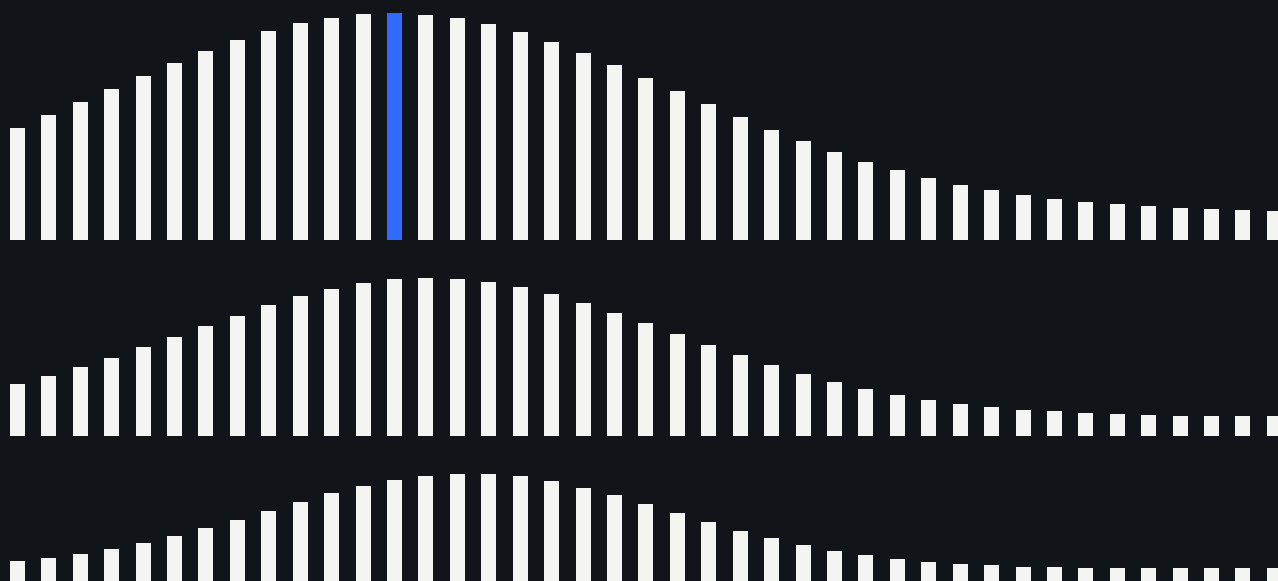
03

An interpreter, not a shell

What runs a script, and where it runs

What is absent by construction, not refused at run time

The limits, and what one session keeps



3.1

The interpreter is the boundary

A configuration assistant that shells out to a real shell has the host's authority. One that interprets the script itself has only what its host handed it.

The interpreter is Computerwelt, an open source .NET library. It runs bash and Python in process, over one virtual filesystem.

Both are managed C# implementations of the language, not wrappers around a binary.

There is no bash binary and no PATH lookup. A command is one the sandbox implements, or "command not found".

There is no CPython. Python cannot import a native module or reach the host. Its implementation, Monty, provides json, re, math, datetime, os, pathlib and a few more.

Nothing is forked, nothing is executed, no container is started.

What configuration work needs still works: ls, cat, grep, rg, find, sed, awk, sort, diff, jq, yq, xargs, pipes, redirection, heredocs, functions, loops and variables.

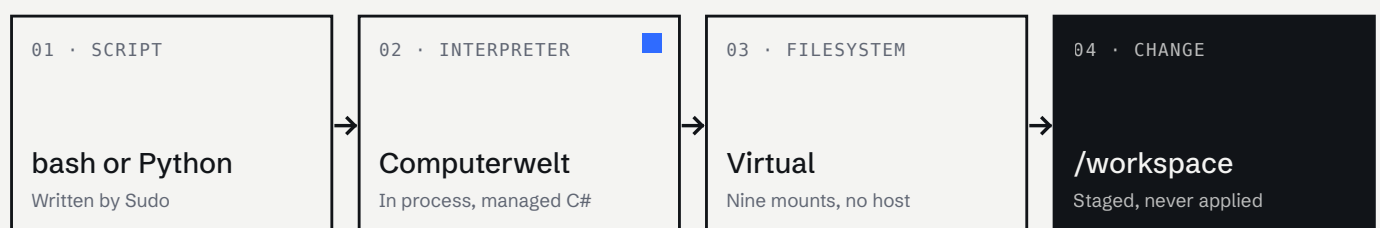


FIG. 3.1 · WHERE A SCRIPT RUNS. ITS ONLY WAY OUT IS A FILE UNDER /WORKSPACE, STAGED AS A DIFF.

3.2

Absent, not blocked

What is missing is missing by construction, not refused at run time. A blocklist has to be kept complete. A capability the interpreter never implemented has nothing to refuse.

No host filesystem. All input and output goes through the virtual filesystem. No path resolves to a file on the server.

No process spawning. There is no fork, no exec and no PATH.

No network. curl, wget, nc, ssh, git and the rest are withheld, not present and refusing. So the day the library grows one, the session does not silently acquire it.

No writes outside /workspace and /tmp. Everything else is read only.

What a steered prompt can reach. Say a ticket Sudo reads carries instructions. At worst it steers the edits in the assistant's own copy. That copy cannot reach the host, cannot call out and cannot apply itself.

TARGET	REACH	HOW FAR
Host files and processes	☐	Absent by construction
Network	☐	Withheld commands
Your graph data	☒	Read only, as the admin
Your configuration	☒	Sudo's copy, staged as a diff
The live workspace	☐	Only after an approval

TABLE 3.1 · WHAT A STEERED PROMPT CAN REACH. GREY IS IN PART, OUTLINED IS NO.

WITHHELD

A command kept out of the session on purpose, so a new library version cannot add it silently.

APPROVED CODE

Workspace code can still call out once approved and running. That is the code's authority, not the assistant's, and it exists only after you approved it.

KEY POINT

Sudo has no route to the host or the network.

TABLE 3.2

Limits that raise rather than truncate

LIMIT	VALUE
Wall clock per script	120 s
Commands per script	100,000
Output captured	4 MB
Idle session lifetime	30 min

READ IT THIS WAY

A cap that is reached raises an error rather than returning a partial answer. A traversal that stopped half way is never reported as the whole.

3.4

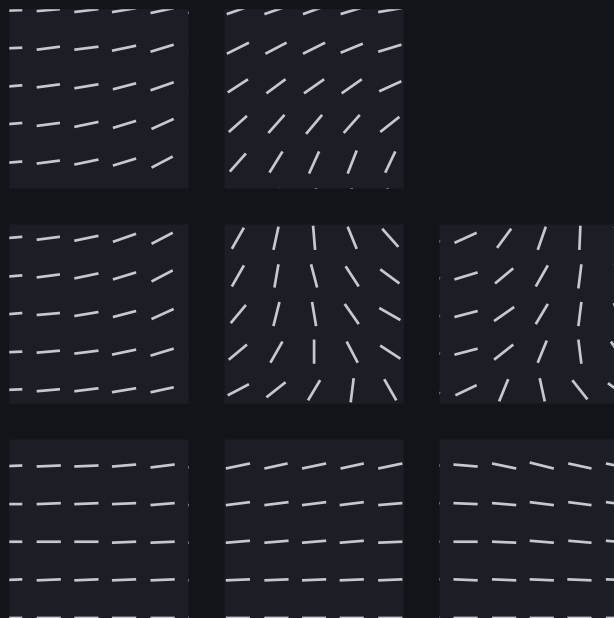
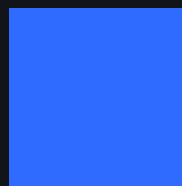
One session per chat, and not a snapshot

A session is keyed by the conversation it belongs to. Two administrators in two chats never share a working directory, an uncommitted edit or a to-do list.

Only what the session changed persists: the files it wrote, the paths it deleted, the working directory, the command history and the to-do list. A session that only read leaves nothing behind.

READ LIVE

Everything else is read again from the live workspace on every command. An endpoint someone edited meanwhile is the one Sudo sees next. Resuming does not restore the workspace as it was.



04

Commit stages, a person applies

The loop from request to applied change

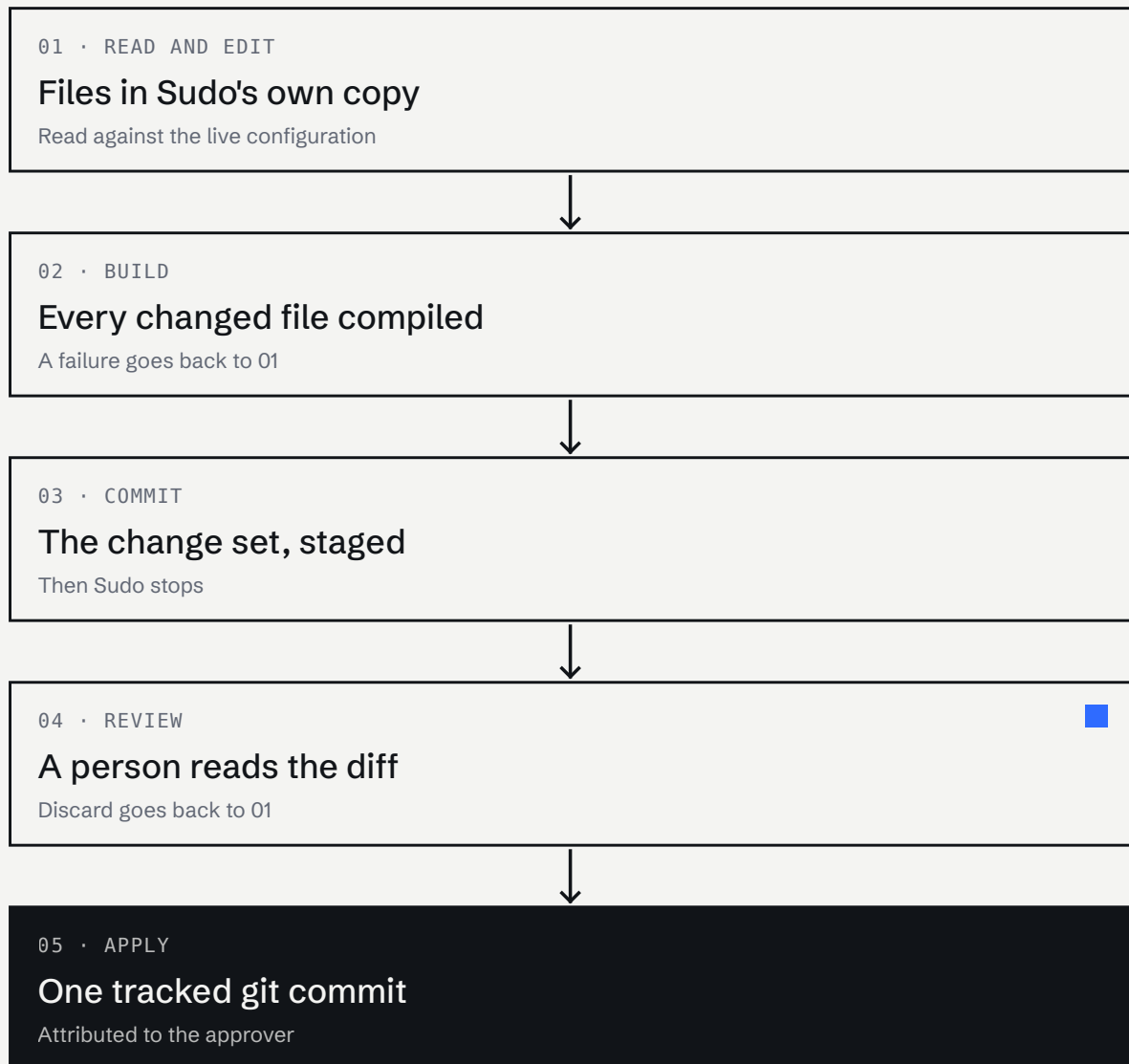
What build, commit and approve each check

What every button does, and the one that refuses

FIGURE 4.1

Draft to applied change

Everything up to the commit happens in Sudo's own copy and leaves the workspace untouched. One step, the review, belongs to a person.



02

A failed build refuses the whole set, and Sudo edits again. `--force` says so loudly and still needs approval.

04

Discard throws the commit away and keeps the edits. It says nothing in the chat.

05

Deletions first, then each file through the definitions importer. Sudo's session then reloads from the live configuration.

4.2

Commit stages the change and stops

Build before commit. `commit` parses every changed file back through the workspace's own importer. It compiles each body against the scope it will run in, and against the workspace as the whole change set leaves it. If anything fails, it refuses the whole set.

One way in. An approved commit is an ordinary definitions import of the files that differ, routed by the code that reads an uploaded definitions bundle. No parallel per-surface writers exist to drift out of step with it.

Bound to what you read. Approval is bound to the commit id you were shown. A card left open while Sudo staged a newer commit refuses to apply, because approving would apply something nobody read.

One copy, whoever edits. The review screen shows a file list or a pull request view. An edit saved there is written into Sudo's copy too, so the shell and the applied files never disagree.

Forcing is loud. `--force` exists. It says so loudly, and it still needs your approval.

GATE	CHECKS	REFUSES WHEN
build	Every changed file, in its scope	A file fails to compile
commit	The whole change set, together	The build fails, unforced
Approve	The commit id you were shown	Sudo staged a newer one
Your account	Rights to commit and run	Read only, or write protected

TABLE 4.1 · FOUR GATES BETWEEN A DRAFT AND YOUR WORKSPACE.

CHANGE SET

Every file a session wrote or deleted, staged together by one commit.

ONE TRACKED CHANGE

The import runs inside the workspace's tracked configuration change: one commit per approval.

KEY POINT

What you approve is the diff you were shown.

TABLE 4.2

Three buttons, and a card that refuses

ACTION	WHAT HAPPENS	WHAT IS KEPT
Approve	Deletions first, then each file	One git commit, in your name
Approve, partial failure	Errors reported as they happen	The edits that did not import
Discard	The commit is thrown away	Every edit, for another try
Reset	Every uncommitted edit is dropped	The live configuration only
A stale card	Refuses: the commit ids differ	Sudo's newer commit, unreviewed

READ IT THIS WAY

Only an approval reaches your workspace. Every other row leaves the workspace as it was, and a stale card applies nothing at all.

4.3

Silence is not approval

Approving says so in the conversation, in your voice, with any notes you typed while reviewing. Discarding says nothing. So a quiet chat never reads as consent.

Approve streams its progress on the server. Closing the tab costs you the log, not the change.

Deletions run first, so removing an endpoint and adding one on the same route never deletes the new one. Search-index halves and package declarations are applied as a pair.

AFTER AN APPROVAL

Sudo's session reloads from the live configuration, so the next command reads what was actually applied.

05

Permissions bound Sudo from below

Reads the graph as you, and writes none of it

What stays out of reach: secrets, people, the host

Why a read-only admin gets a read-only Sudo



TABLE 5.1

Sudo reads as you and writes no data

CAPABILITY	SUDO	IN PRACTICE
Read the graph (graph, query)		Read only, your permissions
Node types never sent to clients		Skipped
Write your data		A migration runs once approved
Write without approval		Bound to the commit shown
Read a secret value		Declares it, never sees it
Access group membership		Yours, under Manage / Access
Packages, backups, restarts, licensing		Not Sudo's to do

YES IN PART NO

READ IT THIS WAY

Sudo's permissions are the chatting administrator's, narrowed. The assistant can reference an access group from a definition and write code that reads a secret.

5.2

A read-only admin gets a read-only Sudo

Every session works the same way: Sudo drafts, you approve. What varies is what your own account may do.

For a read-only administrator, `commit` and `run` refuse. The same holds for any administrator while administrator write protection is on. Reading still works, and the draft is kept for whoever can hand it over.

WHY THE SHELL STAYS

Taking it away would stop Sudo reading too. And there is nothing to approve that the approver could not apply.

06

Accountability and reuse

Where every shell call and every commit is recorded

Sudo's behavior as a skill you can read and edit

The same sandbox, handed to your own assistant

6.1

Every change lands in the history you audit

Everything Sudo does is something you can do by hand. Every applied change lands in the same tracked history as a manual one.

Every shell call is metered into the workspace's own metrics, under an Admin Agent bucket, apart from ordinary chat tools.

Every applied commit is one entry in the configuration history, attributed to the administrator who approved it.

That history is under Manage / Access / Audit log, and readable from the sandbox at `/proc/audit`.

Every code surface keeps its own code history too. An endpoint, AI tool, task or index changed through Sudo is reviewed and rolled back like one changed by hand.

Sudo's behavior is a skill in your workspace, sudo-persona, visible and editable rather than hard wired.

It sets the rules: verify intent, target and likely impact, and proceed on the small and reversible. On anything destructive, security-sensitive, externally visible or materially ambiguous, the assistant asks first.

No state is invented, and no success is reported until it is confirmed.



FIG. 6.1 · THE TRAIL ONE CHANGE LEAVES. EACH RECORD IS ONE YOU CAN ALREADY READ.

6.2

Your assistant can work this way

The same shape is open to your workspace's own assistant. A sandbox is an AI tool whose code returns an ISandbox instead of a class of [Tool] methods. It describes a filesystem, and calling it runs a bash script against it.

A question nobody wrote a tool for is answered by combining grep, awk, jq, sort and Python. The session persists across calls, so it is a place to work rather than a query with extra steps.

MEMBER	WHAT IT DECLARES
Mounts: Live	Rebuilt before every script
Mounts: Cached	Built once per session
Mounts: Scratch	Empty and writable
Commands	What a file cannot be: a count, a search, an effect
Description, FunctionName	What the model reads, and what it calls

TABLE 6.1 · WHAT AN ISANDBOX DECLARES.

Reads go through the chatting user's own permissions. Try it on one assistant first: "Give the support assistant a sandbox over tickets and customers."

ISANDBOX

The interface an AI tool returns to hand the model a filesystem and a bash session.

KEY POINT

A sandbox reads as the user asking, so it never shows data they could not otherwise see.

Three things to keep

01

No route to the host or the network.

An in-process interpreter over nine mounts, two of them writable.

02

No route in except an approved diff.

A person approves the exact commit shown, or nothing applies.

03

Bounded by your permissions, in your history.

Sudo reads as you, and every applied commit is audited.

Terms used in this paper

ADMINISTRATOR WRITE PROTECTION

While it is on, Sudo's commit and run refuse for any administrator. Reading still works.

COMPUTERWELT

The open source .NET library that runs bash and Python in process over a virtual filesystem.

/PROC

A read-only mount of live runtime state, regenerated on every read.

SKILL

Instructions kept in your workspace, visible and editable, such as Sudo's own sudo-persona.

CHANGE SET

Every file a session wrote or deleted, staged together by one commit.

DEFINITIONS EXPORT

The workspace's configuration written out as plain files, which can be imported back.

SANDBOX

A virtual filesystem and an interpreter, with no route to the host or the network.

STAGED COMMIT

A built change set waiting for a person to approve or discard it. Nothing has run yet.

Ask Sudo. Approve what gets built.

Read the admin assistant documentation and its security page, under AI and agents. Then hand Sudo a first small change, read the staged diff, and decide.

docs.curiosity.ai

SANDBOX

No host, no network

CHANGES

Only an approved diff

DEPLOYMENT

On premises or private cloud