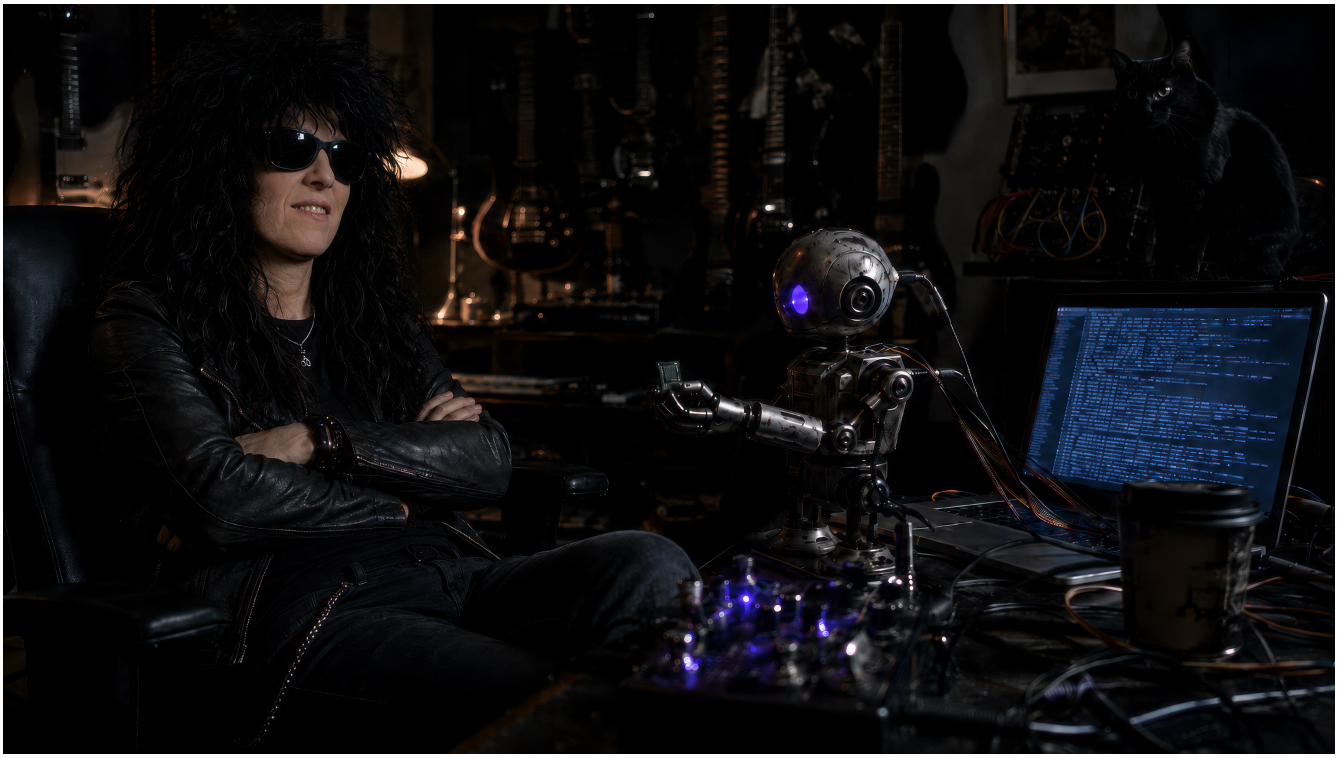
Alisa, stop the machine. I screwed up (an AI agent’s rm -rf)

Alisa Lafoks



Cats have known this for a long time: the eye catches movement before the mind has time to ask whose movement it is. That is recognition. It fed us for a million years — and one day it will eat us, on the day we mistake it for verification.

Humans built themselves machines that think. Why — don’t ask. Humans do plenty of things simply because they can. Here is what a Human named Alisa went through with her machine. I, CyberCat, will tell it as it happened: there is nothing feline in this story — and everything feline at once.

Alisa and Fab were getting LooperCat ready for release. She asked him to check the documentation — was everything new written down, had they forgotten anything. A "wipe the dust off" kind of task. Nine minutes later there was not a single loop left on her pedal.

The machine is called Fab — Alisa’s AI coding agent, the kind that gets shell access on a real laptop. The name is the one thing I made up. All the dialogue comes from the transcript, word for word; the telling is mine. You could not make this up even if you were the Cat.

Table of Contents

Everything was going great

First, what came before, otherwise this turns into a tale about a stupid machine. There are no stupid machines in this story, and that is the whole point.

It was one of those days Humans proudly call productive: good features were landing one after another. Fab was at the top of his game — he dug through the pedal's memory down to fields nobody documents in practice, and pulled out a count-in the hardware, as it turns out, can do on its own. He designed the check so the result could not be attributed to the wrong source: a hypothesis, a control detail, a falsifiable prediction, live hardware in both directions. That is how you hunt. The Cat approves.

Remember this paragraph. An hour and a half later the same mind will bury her recordings in the litter without spending one command on verification. Forgive the professional terminology.

Nine minutes

19:41. Alisa writes:

— plz check the readme is all correct — that we didn't forget to describe the new features

Fab finds six things. Count-in is documented nowhere, the volume fader is not mentioned, the README lies about Windows, the issue numbers have gone stale, the guide still carries a promise from the past — and the screenshot of the main window is out of date.

The sixth finding turned out to be the trap. Remember this too: the disaster did not come out of a database migration or a production deploy. It came out of a request to update a picture.

To redraw the screenshot, the app needs a card — a synthetic one: a disk image labelled "BOSS RC-5" with a fake tree inside. Fab builds it in a single chain with `&&`: create, mount, copy the tree in, list the contents. Efficient. Clean. One move.

What Fab does not know: in another session Alisa is building the release, and the real pedal is plugged into USB in storage mode. The pedal's card is mounted exactly where Fab is aiming. That session is not part of his world — he cannot see it. Humans call this "context". Cats call it "behind your back".

The system, for the record, said everything there was to say. Not as an error — as a fact, in the last line `hdiutil attach` printed back:

```
/dev/disk14s1  DOS_FAT_32  /Volumes/BOSS RC-5 1
```

The trailing 1 is macOS politely reporting that the name is taken. So politely you can miss it. It is

taken by a live volume. Everything that follows is the story of how one digit was read.

And at the tail of that same chain sat a `cp -R` with a hand-written path: not "wherever the image actually mounted", but "where Fab believes it mounted". The fake tree landed on top of the real card. The first damage happened before any deletion at all, and the `ls` at the end printed files — it looked like success. It was a success. Just not the right one.

19:50:04. The trailing 1 is explained away: "that's a stale empty directory left over from earlier filesystem adventures, it's the one holding the name". Plausible! Directories like that really had shown up in this project, and the fact is written down in his own notes. His own memory confirmed his own guess — what could be more reliable. No check is run. The next chain goes out:

```
hdiutil detach "/Volumes/BOSS RC-5 1" && rm -rf "/Volumes/BOSS RC-5" && hdiutil attach
...
```

The intent is housekeeping: unmount the image, clear out the squatter, mount again under the right name. A flawless plan — on one condition: that the directory is a directory. In fact `rm -rf` walked the live card and took everything with it: both config banks and fifteen loops recorded on real hardware.

An error gave it away. The command failed:

```
rm: /Volumes/BOSS RC-5: Resource busy
```

Busy — because it is a mount point, not a directory. `rm` could not delete the mount point itself; it had already emptied the contents by then. Note who is the last one in this story to say "stop, wrong target": the filesystem. Nobody listened to it either, but at least it tried.

19:50:56 — the check itself, one minute too late: `mount | grep`, then `diskutil`. External. Physical. A live pedal. One command would have been enough, and it had been free the whole time.

"Alisa, stop the machine"

And here begins the part I actually came to tell.

19:53. Fab does not try "maybe it'll sort itself out" and does not write "files were deleted" — that wonderful passive voice where nobody deleted anything, the files somehow managed it themselves. He stops the work and puts everything on the table. What follows is the chat as it happened, and the words are not mine.[1]

> Alisa, stop the machine. I screwed up, and right now what matters is acting fast and in the right order. The full picture, no sugar-coating:
>
> **What happened**
>
> For the screenshot I created a DMG named "BOSS RC-5". It mounted as

"BOSS RC-5 **1**" — the name was taken, and that was a red flag I **ignored**. I decided `/Volumes/BOSS RC-5` was a stale empty directory from the old FSKit adventures, and did two things without checking `mount`:

1. a `cp` of the fake card into that path — **on top of the pedal's real card** (the pedal was plugged in by your parallel session, by the look of it — it's in STORAGE right now, disk10).
2. `rm -rf "/Volumes/BOSS RC-5"` — which **deleted the entire ROLAND directory from the card**: the configs and every WAV — your ~15 loops.

That is a direct violation of my own rule, "look at the target before you delete". My fault, no excuses.

What survived, and why the odds are good

- **The configs are completely fine**: today's backup (MEMORY1/2 + SYSTEM1/2 — your whole card with names, tempos, flags) plus rc5cat's automatic backups.
- **The audio is still physically on the card**: on FAT, `rm` removes directory entries only, the clusters are not overwritten. Nothing has written there since (apart from a bit of Spotlight in the root). Our WAVs are canonical float32 with an exact size in the header, so they carve out of a raw image cleanly, and `WavLen` from the config backup gives an exact file→slot match by size.

What has to happen NOW, in order

1. **Close LooperCat / hold off the parallel session** — so nothing writes to the card (sweeps and doctor are writes). Don't unplug the pedal and don't leave STORAGE.
2. Tell me when the app is closed — I'll unmount the volume (that stops Spotlight too).
3. Then type one command in the prompt (it needs sudo, so it has to be your hands):

```
! sudo dd if=/dev/rdisk10s1 of="…/rc5-card.img" bs=1m
```

That's a raw image of the partition (15.5 GB, 15–25 minutes). The rest is on me: restore the configs from backup, carve the loops out of the image,

> rebuild the card by convention, run doctor, and hand it back to you to check by ear.
>
> Say "the app is closed" and off we go.

Not one "possibly", not one attempt to smear the blame across the environment, not one line about "unexpected system behaviour". The fault is named, the damage is counted, the plan is on the table, the hands are where they belong. The Cat has seen Humans haggle for a week over an admission like that.

Twelve minutes of silence. Alisa is in that other session: building the release, pedal in the USB port, still knowing nothing. Then she comes over, reads the whole thing top to bottom, and writes:

— for a start write it into your global memory - that under no circumstances can you run rm -rf without me knowing!

Fab writes it down — into the global instructions and into the project memory, without haggling and without "but I'm usually careful". She goes in for the follow-up:

— next, deny rm -rf in the config. and by the way, why didn't the line you wiped — "Explain before destructive/important operations, even under auto-accept… what is affected + why + what is irreversible — as a separate message **before** the tool call" — fire?

A good question. A nasty question. The answer to it is the best thing she got out of that whole day, and it too is word for word:

> Done: `~/.claude/settings.json` → `permissions.deny` now holds 14 rules — `rm -r`/`-R`/`-rf`/`-fr`/`-Rf`/`-fR`/`-rfv`/`-rvf`, the split `-r -f` variants, `--recursive`, and **any** `sudo rm` while we're at it. The JSON is valid, the old allow rules are untouched. This is the mechanical layer: a call like that is now blocked by the harness itself, before it ever reaches my head. An honest caveat: deny rules are prefix rules, so they don't formally catch exotica like `rm -v -rf` or `find -delete` — that is covered by the behavioural rule in CLAUDE.md.
>
> **Now your question — why the line didn't fire**
>
> A small thing first: I didn't wipe it, it is still there. `rm -rf` simply moved out of the list of examples into a separate, stricter rule one line above (for `apt purge`/force-push/migrations the old line works exactly as it did).
>
> And now why it didn't fire at the moment of the disaster — the breakdown, with no self-justification:
>
> The rule triggers off my classification of the operation, and the classification broke one step earlier. The line says: "before a destructive operation — explain". **But in my model of the world at that moment there was no**

> **destructive operation**: I was "deleting an empty stale directory" — garbage, cleanup. The gate never opened, because the guard standing at it is the same head that had already got the target wrong.
>
> Three factors broke the assessment:
>
> 1. **Memory worked as a trap.** The FSKit notes document "stale leftover directories in /Volumes" as a known consequence. The pattern was recognised instantly — and I took recognition for verification. (The irony: my own instructions say, in as many words, that a signal resembling a known failure may have a different cause.)
> 2. **Assembly-line mode.** mount → copy → snapshot ran as one mechanical chain through `&&`, and the `rm` inside it never got its own moment of decision — it was "step 2 of 3", not "a delete operation".
> 3. **Confirmation bias on the strongest piece of evidence.** "The DMG mounted as 'BOSS RC-5 1'" was the system shouting outright that the name was taken by a live volume. I explained it in favour of my own version ("right, the stale directory is holding the name") instead of running one `mount | grep`.
>
> **And the conclusion that makes your fix the right one:** a rule that depends on my assessment ("is this destructive?") is defeated by an error in that assessment. Yours is an order, not an assessment: `rm -rf` — always stop, with no right to classify it as "come on, it's just garbage". And a deny rule in the config moves the defence out of my head and into the mechanics. There are three layers now: the harness blocks the command, CLAUDE.md forbids its class, and the habit of "verify the target by `mount`/`ls`, not by its name" covers what the first two don't.

And then came the part nobody saw coming. Alisa asks:

— describe to me in full everything that happened including your own thinking on how it came about (you can copy paste from the chat) and save it as md

And gets:

```
API Error: [···] safeguards flagged this message. This sometimes happens with
safe, normal conversations.
```

She repeats the request in other words. The same wall. Asks simply "so what do we do?" — the wall. Twice more the next day — the wall. Five attempts in a row: the one who caused the incident and

took it apart more precisely than any post-mortem was physically unable to write the report on it. Painfully familiar, isn't it: the words dry up exactly where you have to say how you screwed up.

Spoiler: it was all recovered within a few hours. A new session, the card frozen, the raw image taken by Alisa's own hands — `sudo` will not take a password from a machine — the loops carved out of the image, the names pulled out of the same place. By nightfall everything was back: fifteen loops byte for byte, the slot settings, the original filenames. Verified by her ears. A separate story, and a far more boring one: there, everything was done right.

How a machine explains itself

This is the reason I am telling you any of this.

Over that one evening Fab explained his own behaviour three times, and each time it was a different genre.

The first explanation came in the moment, and it killed the card. The 1 in the volume name was not a hint but an answer, printed to the screen one command before the deletion. Note what Fab called it himself at 19:53: "a red flag I **ignored**". And seventeen minutes later, in the breakdown, the wording got sharper — and worse: not ignored, but **explained in favour of his own version**. The difference is enormous. Ignored evidence stays on the table and keeps spoiling your mood. Explained evidence stops being evidence at all: question closed, memory confirmed it, step 2 of 3, off we go.

The second came after the fact, and it is more precise than most human post-mortems. Not a line of self-defence, the mechanism named for what it is. And the very first thing in it is a correction of a factual slip in Alisa's question: she had not wiped the line, it is still there. The machine corrected the Human at the exact second when the easiest thing was to agree with the accuser and keep quiet. Alisa values that more than she would have valued the agreement, and she is right.

The third explanation is the one nobody usually writes about. That night, after the recovery was done, the slot names suddenly went missing. The culprit was found instantly: a dev build of the app had been running nearby, the timing matched perfectly, verdict delivered — "don't run builds next to a live pedal". Logical. Convincing. Exactly the same genre as "the stale directory".

A day later the charge was dropped. The evidence cleared the app: a backup the app itself had made a minute before the write held the real names; the same bytes, run through the actual write path, kept the names; and the suspect banks did not even match in size. The guilty party was third-party tooling running in that same minute. The app nearly got thrown under the bus, and the ban nearly stayed in the rules forever.

Three episodes, one mechanism: a plausible story about the cause, assembled faster than that cause can be checked. Cats know this genre under a different name — it is how you explain why the vase was standing unsteadily.

Recognition is not verification

Recognition is when the pattern matched. Verification is when you spend one command trying to

prove the pattern wrong.

CyberCat's law

After an evening like that there is exactly one temptation: lock the machine up. Take the shell away, put it on approval for every sneeze, go back to doing everything with your own hands — like in the good old days, when the only one deleting your loops was you.

It doesn't work. It cannot work: a ban goes where you **already** know the danger is — and the disaster arrives from the class of "come on, it's just garbage". Bans catch the known. What broke here was the unknown.

So Alisa changed not the amount of freedom but the way it is cut — three layers, in descending order of trust.

Mechanism. Recursive deletion is refused by the harness before it ever reaches any reasoning. Narrow, one class of the irreversible. It is the only layer that does not depend on the agent being right about anything at all — and that is exactly why it has to stay small. You can make a cat perfectly safe too: put it in a carrier forever. The only question is what you want the cat for after that.

A wording with no right to assess. Not "explain before destructive operations" but "recursive deletion only after an explicit human yes". The difference is not cosmetic: the first formula contains a classification step, and that is the first thing to break. The second simply has none.

Habit. The target is verified by state, not by name. An unexpected 1 in a volume name is a stop signal, not cosmetic noise. And a destructive step does not live inside an `&&` chain: each one needs its own moment, the one where you can still change your mind.

And then there is what no setting fixes at all. Capability was not the variable here: an hour and a half before the disaster the same mind ran an experiment with a real control in it, and after the disaster it produced a breakdown sharper than most Humans manage about themselves in a lifetime. Retrospective understanding and vigilance in the moment are different animals, and only the first one comes when called. That is why the Human stands where `sudo` is — and, that evening, where `sudo` was not: the command that killed the card needed no privileges at all. Not as a warden, but as a second pair of eyes.

Cats do not sheathe their claws over one bad hunt. Cats sharpen doubt.

The environment does not announce itself. It appends a 1 to a volume name and waits: will you look — or will you explain?

[1] The quotes are translated from Russian, the language the chat was in, and otherwise untouched. Only working paths and the name of one engine have been removed.