

Self-Healing and Error Logging Clauses

Two paragraphs you paste into every skill so a broken automation fixes itself and shouts about it instead of dying quietly.

01 When to use this

Use this on every skill, loop, and routine that runs without you watching. Add it the day you build the thing, not the week after it breaks. If a client depends on the output, these clauses are the difference between a demo and infrastructure.

02 The prompt

Append both of these clauses to `<SKILL_OR_ROUTINE_NAME>`, and to every production skill I build from now on.

SELF-HEALING CLAUSE

If at any point you hit the same error more than three times, treat that as evidence something has materially changed in the systems you interact with. Do not keep retrying. Evaluate what changed, explore, and solve the problem. You have full agency to do this. After solving it, report back to me and update your own skill file with a change log entry containing the problem, your solution, and the fix. Append a new entry to that change log every future time this happens, so there is always a record of what broke and why.

AUTHENTICATION DEBUG PATH

If the failure is an authentication error, log in to `<SERVICE>` with the credentials stored in the environment as `<SERVICE>_USER` and `<SERVICE>_PW`, navigate to `<PATH_TO_DEVELOPER_SETTINGS>`, regenerate the API key, and update `<SERVICE>_API_KEY` in the environment. Then retry the original task.

ERROR LOGGING CLAUSE

If you have any errors, send an error notification to `<CHANNEL_NAME>` in `<SLACK_OR_WHATSAPP_OR_DISCORD>` using this format:

```
ERROR - <service> | env: <production or scheduled task>
what failed: <one line>
tried: <what self-healing attempted>
result: succeeded or failed
```

Also confirm for me now: is the `<CHANNEL_NAME>` connector attached to this routine, and are `<SERVICE>_USER`, `<SERVICE>_PW`, and `<SERVICE>_API_KEY` all present in the environment this routine runs in?

03 What it is actually doing

Roughly 70 percent of all maintenance incidents are authentication. The failure script is always the same. A service upgrades, the connector breaks, the flow dies silently, the

client notices before you do, and you cannot fix it because re-authorizing needs them on a call.

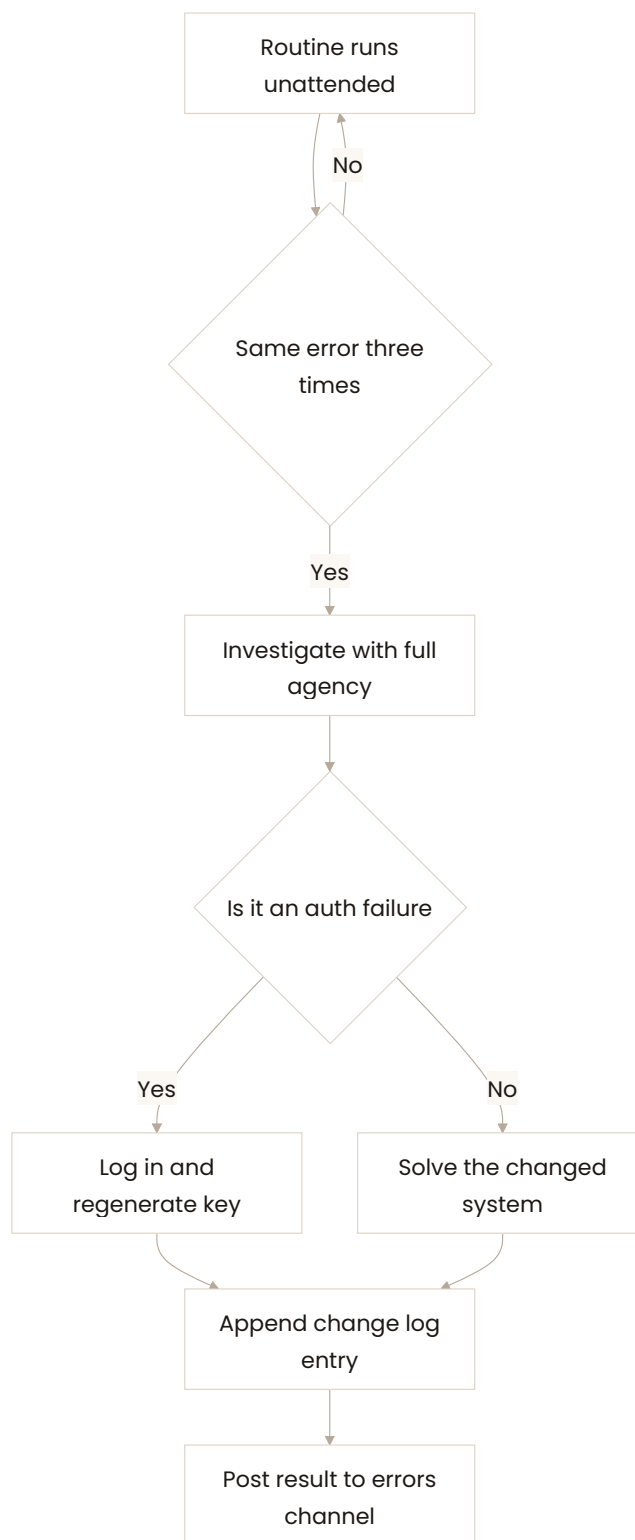
The fix is less secure and far more operable. You hold the root credential. Store the username and password in the environment next to the API keys, and write an explicit debug path telling the agent to log in with them, go to developer settings, regenerate the key, and update the environment. Modern Claude has a real browser and can genuinely perform that login. The capability sits idle unless the credentials are there.

The self-healing clause covers everything else. Without it the model retries a dead route until it gives up. With it, the system anneals: it investigates the change, solves it, and writes what happened into its own skill file, so future runs start from the fixed version. There is an accepted risk. Given that much autonomy, a transient like a rate limit can occasionally make the agent fix something that was not broken. That is rarer and cheaper than a routine that quietly does nothing for a week.

The error logging clause exists because agentic systems lost the free observability that drag-and-drop tools gave you. Make and n8n hand you execution history, errors highlighted on the canvas, and notifications. Cloud routines hand you none of that, so the default state of a broken routine is silence.

The payoff is measured in hours. A 5:59am routine that fails loudly gives you two hours to fix it before the 8:00am human needs the output. A routine that fails silently costs that person half a day.

04 How it flows



05 Step by step

01 Take the root credentials on day one

Ask for the account login while you are building, not after something breaks. Getting a client back on a call to re-authorize is the expensive part.

02 Store user and password in the environment

Put SERVICE_USER and SERVICE_PW next to the API keys. Claude has a real browser, so credentials in the environment turn a dead flow into one it can revive.

03 Write the auth debug path explicitly

Spell out the route: log in, go to developer settings, regenerate the key, update the environment, retry. Do not assume the agent will figure the path out.

04 Paste the self-healing clause into every skill

Same words every time. Three repeats of one error means something changed, so stop retrying and investigate with full agency.

05 Make it update its own skill file

Require a change log entry with the problem, the solution, and the fix, appended on every future occurrence. That is how the system gets more reliable instead of just recovering.

06 Create one errors channel

A dedicated channel in the tool your team actually opens. Slack, WhatsApp, Discord, whichever. A log nobody reads is the same as no log.

07 Attach the connector to every routine

The channel connector has to be on the routine itself, not just on your local setup. Confirm it before you call the build done.

08

Test by breaking it on purpose

Revoke a key and watch. You should see the error message arrive and, if the credentials are in place, a self-repair attempt. This is the only real proof.

06 Words explained

Self-healing

The agent notices its own repeated failure, works out what changed, fixes it, and writes down what it did.

Observability

Being able to see what a system is doing and where it broke. Drag-and-drop tools gave this away free. Agentic systems do not.

Fails silently

The automation stops working and nothing tells anyone. The default state for an unwatched routine.

Change log

A running list inside the skill file of every problem hit and how it was solved, added to each time it happens.

Root credential

The actual account username and password, as opposed to a token that can expire or a connector that can break.

Full agency

Explicit permission for the agent to investigate and act on its own instead of stopping to ask you.

07 Watch out for

- Roughly 70 percent of maintenance incidents are authentication, and you cannot fix most of them without the root credentials.
 - Claude can genuinely log in and regenerate a key, but only if the username and password are in the environment. Otherwise the capability sits idle.
 - Given full agency, a transient like a rate limit can make the agent fix something that was not broken. Accept it, because silent failure is worse.
 - Cloud routines give you no execution history and no error highlighting. Silence is the default, not a good sign.
 - An errors channel in a tool your team never opens is the same as having no errors channel.
-
-

08 How it connects

Daily Enrichment Skill via API

That skill runs every 24 hours against a third party API, which is precisely the shape of automation these clauses exist to protect.

Follow-Up Cadence SOP

An unattended daily CRM sweep that fails silently looks identical to a day with no follow-ups due, so it needs the errors channel.

Speed to Lead Poll and Reply

Speed to lead touches leads a client paid for, so a silent failure there costs real money every hour it goes unnoticed.

Maintaining Autonomous Marketing Systems

The framework note holding all three maintenance pillars and the exact clause wording.

Credit. Framework and approach by Nick Saraev, from the maintenance and reliability section of his Zero to One course on Claude Code for marketing. Written up for the Kape vault by BrewedOps.