Sentinel

Dashboard — user's manual

A plain-English user's manual

Every button, every number — explained for people who have never used a version-control tool. Keep it beside you the first few times; after that you won't need it.

What is Sentinel, in one sentence?

Sentinel keeps a governed, tamper-evident record of every change your developers and AI agents make to your code — auto-approving trusted work and holding risky changes for your one-click review.

1 Opening the Dashboard

There are two ways to open your Sentinel windows.

The easy way — the desktop icon

After your first setup, Sentinel places a **“Sentinel” log-on icon** on your desktop. Double-click it and your windows open, already signed in and sized the way you left them.

The typed way — a command

From the folder that holds `sentinel_sdk.py`:

- `python -m sentinel_sdk dashboard` — opens the Dashboard on its own.
- `python -m sentinel_sdk open` — opens **both** windows together.

Leave it running while you work; press **Ctrl + C** in the terminal to stop it (or just close the window).

A **“User manual”** link also sits in the bottom-left of the window — it opens this same guide any time.

Integrating Sentinel into your agent workflow

Whether you're a solo builder or a team, wiring Sentinel in takes a few minutes and starts from the **git repository your AI agents commit to**. Once connected, every commit an agent makes is reported to Sentinel automatically — nothing is blocked, it's simply recorded and, when a rule is tripped, held for your review.

Before you start

- A **git repository** on your machine — the folder your agents work in (it contains a `.git` directory).
- **Python 3** installed.
- A **Sentinel subscription** and your downloaded `sentinel_sdk.py` (from Sentinel → Get the SDK).

Path A — the one-click way (recommended, no terminal)

1. Launch the Sentinel desktop app (double-click `sentinel_sdk.py` or your desktop shortcut).
2. Click **"+ Connect a repo"**.
3. Pick your git repo folder. Sentinel automatically **creates a project**, **generates an agent key**, and **installs a post-commit hook** in that folder.
4. Done — commits made in that folder now appear in Sentinel.

Path B — the terminal way (for servers, CI, or scripting)

1. Open a terminal **inside your git repo folder** (the one with the `.git` directory).
2. Install the one dependency: `pip install requests`
3. Connect the folder in a single command — this creates the project, an agent, and the hook:
   `python -m sentinel_sdk init --token YOUR_TOKEN --name "My Project"`
   (Find **YOUR_TOKEN** in the Sentinel dashboard. `--name` is optional; it defaults to the folder name.)
4. Already have a project + agent key? Install just the hook:
   `python -m sentinel_sdk install-hook --project YOUR_PROJECT_ID --agent-key YOUR_AGENT_KEY --base https://sci-ficomics.com`

Point your agents at the repo

Your AI agents (Claude, GPT, Cursor, custom scripts, CI bots — or teammates) keep working exactly as before. Every time one of them runs `git commit` in that folder, the post-commit hook quietly reports the change to Sentinel. The commit is already saved locally; Sentinel never gets in the way.

Recommended agents to use with Sentinel

Sentinel works with **any** agent (or person) that makes git commits — you're never locked in. If you're just getting started, these are free or low-cost and pair especially well because they commit straight to git:

- **Aider** — free, open-source CLI agent that commits directly to git (a natural fit for Sentinel). You pay only for whatever model API you connect.
- **Cline** and **Continue.dev** — free, open-source VS Code assistants; bring your own model key.
- **Cursor** and **Windsurf** — free tiers available; paid plans around $10–$20/month.
- **Low-cost models** — DeepSeek, Qwen Coder and Google Gemini offer very cheap (or free-tier) coding APIs you can plug into the agents above.

Whichever you choose, the setup is identical — install the hook in the repo (above), and every commit the agent makes is reported to Sentinel automatically.

Set your Guards (one-time)

In the Dashboard **Guards** tab, set **Allow-paths** (folders agents may edit freely) and **Guard patterns** (sensitive files such as `.env`, keys, and payment/auth code). Clean commits from your **primary agent** **auto-approve**; anything that trips a rule — or comes from another agent — waits in **Pending** for your Approve/Reject.

Give each agent its own identity

Separate identities are what let the leaderboard and audit log show *who* did what. Set one up per bot or person:

1. Open the **Dashboard**, select your project in the drop-down, and go to the **Agents** tab.
2. Click **"+ New agent"** and name it for the bot or person — e.g. `claude-refactor`, `cursor-cody`, or `ci-bot`.
3. Copy that agent's **key** (shown when it's created).
4. On the machine or repo where that agent runs, install the hook with *its* key:
   ```
   python -m sentinel_sdk install-hook --project YOUR_PROJECT_ID
   --agent-key THIS_AGENT_KEY --base https://sci-ficomics.com
   ```
5. Mark your most trusted agent (your lead or CI bot) as **primary** so its clean commits auto-approve. Leave the others non-primary, so their commits always land in **Pending** for a look.
6. Repeat for each agent. Every commit is now attributed to the right identity in the leaderboard and the audit log.

Give each team its own identity

In Sentinel, a **team is a project**. Give each team (or each repo) its own project so their agents, guards, and audit trail stay cleanly separated:

1. Create a **project per team** — click **"+ Connect a repo"** in the app for that team's repo, or run `python -m sentinel_sdk init --token YOUR_TOKEN --name "Team Alpha"`.
2. Name the project after the team (e.g. `Team Alpha`, `Backend Squad`) so it's easy to spot in the project drop-down.
3. Inside that team's project, add the team's agents using the steps above — each teammate/bot gets its own key and hook.

4. Set **Guards per team**: every project has its own Allow-paths and Guard patterns, so a backend team and a frontend team can run different rules.
5. Switch teams with the **project drop-down** at the top of the Dashboard — the Commits, Pending, Agents and Guards tabs all follow the selected team.
6. Each team gets its **own Activity Report and Audit log**, scoped to that project — so scorecards and trails never mix between teams.

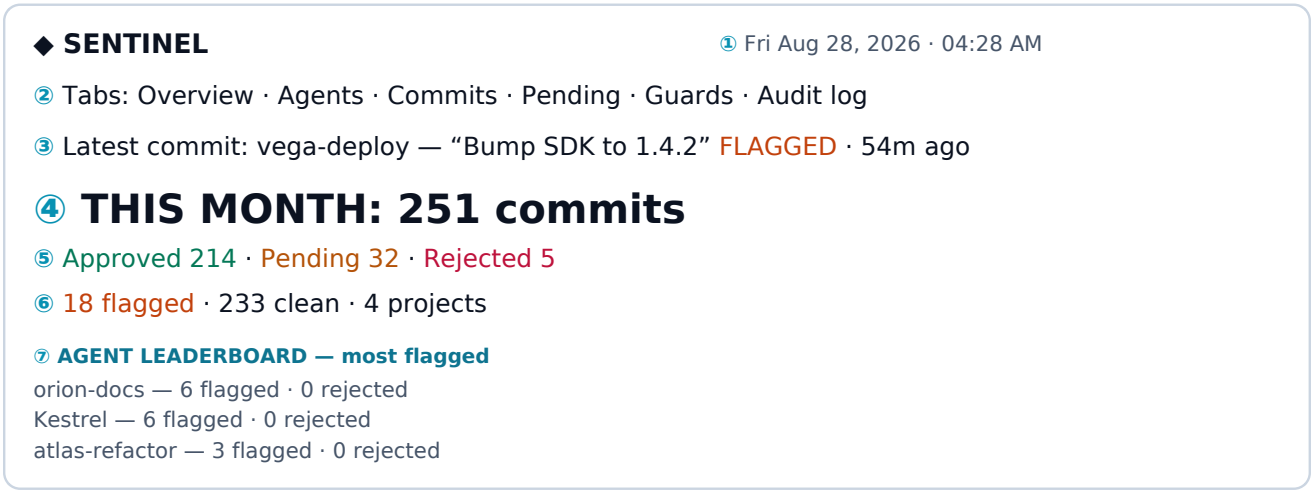
Verify it's working

Make a test commit in the repo. Within seconds it shows up in the Dashboard (the **Latest commit** banner and **Commits** tab) and in the **Activity Report**. You're integrated.

```
cd path/to/your/repo
echo "sentinel test" >> README.md
git add README.md
git commit -m "Sentinel test commit"
```

2 The Dashboard

The Dashboard is your live mission control. It refreshes on its own, so what you see is always current. The callouts below match the picture.

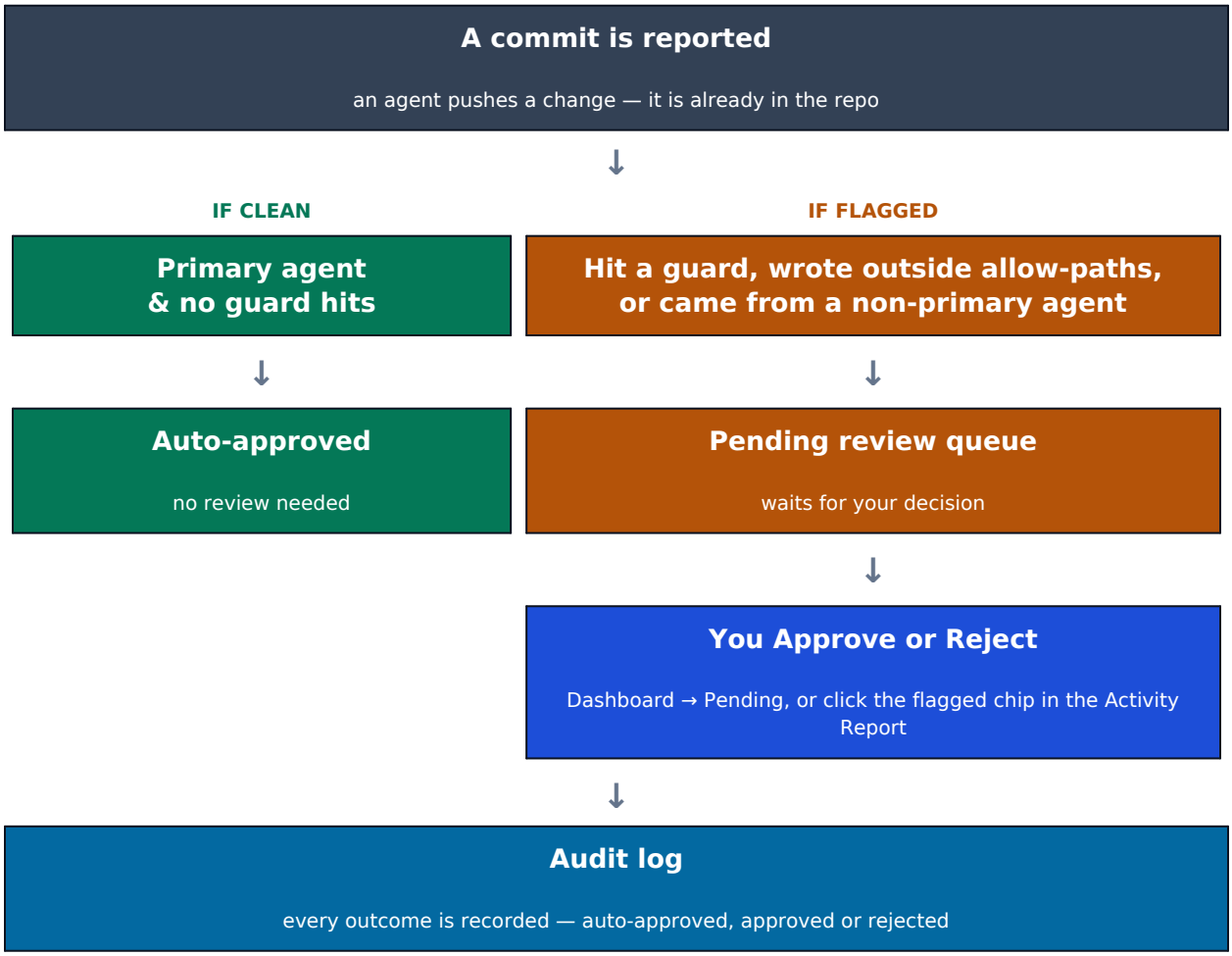


The Sentinel Dashboard (Overview tab).

1. **Date & time clock.** A live day / date / year / clock readout, top-right of the window.
2. **The six tabs.** Overview, Agents, Commits, Pending, Guards and Audit log — each explained below. A project picker sits with the tabs so you can switch which project you're looking at.
3. **Latest commit banner.** The most recent change Sentinel saw, with a colored status tag (**approved**, **flagged** or **rejected**) and how long ago it arrived.
4. **This-month headline.** The big number is how many commits were reported **this calendar month**. It resets on the 1st — earlier months live in the Activity Report.
5. **Approved / Pending / Rejected.** A quick split of this month's commits by their review outcome.
6. **Flagged / clean / projects chips.** How many commits were **flagged** for review, how many sailed through **clean**, and how many **projects** are active.
7. **Agent leaderboard.** Which agents triggered the most reviews this month — handy for spotting a mis-behaving agent at a glance.

Sentinel Commit Flow

Here's the whole journey a commit takes, at a glance. Sentinel is **passive** — the change is already saved in the repo; Sentinel only records it and, when a rule is tripped, asks you to take a look.



Every commit is either auto-approved or held for your review — and either way it is logged.

How to read it

- **Only a primary agent's clean commits auto-approve.** A commit is “clean” when it hits no guard and stays inside your Allow-paths.
- **Everything else is flagged** and lands in the **Pending** queue — including commits from non-primary agents, even if they look fine.
- **You clear a flagged commit** from the Dashboard **Pending** tab, or by clicking the flagged chip in the Activity Report, then **Approve** or **Reject**.
- **Nothing is ever blocked** and no local work is undone — Approve/Reject only writes to the permanent Audit log.

2.1 The six tabs, explained

Every tab shows information for the **project selected in the picker** at the top. Change the picker to look at a different project.

Overview

The home screen shown above: the headline count, the status split, the chips and the agent leaderboard. Start here every day.

Agents

Your team of AI agents (and people) for this project.

- **+ New agent** — registers a new agent and shows you its one-time **agent key**. Copy it right then; it isn't shown again.
- **Rotate key** — issues a fresh key and retires the old one.
- **Delete** — removes an agent that's no longer used.
- The **primary** agent is your trusted one — its commits can be auto-approved; everyone else waits for review.

Commits

A running list of **every** change reported for this project — approved, flagged or rejected — newest first.

To see who made the commit, select your Project from the drop-down list at the top of the Dashboard. Then click the **“Commits”** button. You will see who made the commit and the files that were touched.

Pending

The commits waiting for **your** decision. This is your inbox.

- **Approve** — accept the change; it becomes part of the record.
- **Reject** — decline it; you'll be asked for a short reason so the audit trail explains why.

Note: rejecting governs the *record* — it never blocks or deletes the developer's local commit.

Guards

The rules that decide what gets flagged. Shown as removable chips you can edit.

- **Allow-paths** — folders/files agents may change freely (e.g. `src/**`). Anything outside is flagged.
- **Sensitive patterns** — danger-zone files (like `.env`, auth, payments) that are always held for review.
- Type in the box and press **Add**; click a chip's **×** to remove it; **Return to defaults** restores the starting set.

Audit log

The permanent history: every action (granted, denied or pending), what it was, why, and exactly when.

Bonus button: + Connect a repo

Turns a folder on your computer into a Sentinel project in one click — it creates the project, makes an agent key and installs the tiny git hook that reports commits.

3 The clock & window controls

The date & time clock

A live readout sits in the top-right corner — day, month, date, year and a ticking clock — so any screenshot or printout is self-dating.

Resize once, and it remembers

Drag the window to the size and spot you like. Sentinel remembers each window's size and position, so next time it comes back exactly as you left it.

Choosing the window style (optional)

By default windows open clean, with no browser tab or address bar. You can change this without re-downloading anything:

- **app** (default) — a clean, chrome-less window.
- **sized** — its own window but *with* the browser tab/address bar.
- **tab** — just opens in your normal browser (most compatible).

Set it with `set SENTINEL_WINDOW=sized` on Windows, or `export SENTINEL_WINDOW=sized` on Mac/Linux.

4 Everyday things you'll do

Review a change that's waiting

Open the **Pending** tab, read the commit (message + files), then click **Approve** or **Reject**. If you reject, type a one-line reason.

Add a new AI agent

Open the **Agents** tab → **+ New agent**. Copy the one-time key it shows you. If a key ever leaks, use **Rotate key**.

Tighten or loosen the rules

Open the **Guards** tab. Add an allow-path for folders agents may edit freely; add a sensitive pattern for files that should always be reviewed.

Start watching a new project

Click **+ Connect a repo**, point it at the folder, and Sentinel wires up the project, agent key and commit hook.

Words you'll see (glossary)

Commit — A saved change to your code. Sentinel records each one.

Agent — An AI (or person) that makes commits. Each has its own identity and key.

Primary agent — Your trusted agent whose commits can be auto-approved.

Flagged — A commit that tripped a rule and needs a look.

Approved / Rejected — Your decision on a commit — recorded forever.

Allow-path — A folder/file agents may change freely without being flagged.

Guard / sensitive pattern — A danger-zone file that's always held for review.

Audit log — The permanent who-did-what-and-when history.

Quick fixes

A window won't open, or looks out of date

Close the old window first, then double-click the desktop icon again — it clears the previous window and opens a fresh one.

“This month” shows fewer commits than expected

That number is **this calendar month only**. Use the Activity Report to see earlier months. Sentinel counts commits only from repos where the hook was installed, made after setup.

Nothing shows for a project

Make sure you ran setup in that repo (or used **+ Connect a repo**) and made at least one commit since.

How Guards work

Guards decide which commits get held for your review — and, just as importantly, what they **don't** do.

Guards govern code, never money

The Guard list includes default patterns like `**/stripe*` and `**/paypal*`. These are **file-path patterns** — they match *source-code files* whose names contain “stripe” or “paypal” (for example `backend/payments/stripe_handler.py`). When an agent commits a change to one of those files, Sentinel simply flags that commit for review.

They do **not** touch real money, your Stripe/PayPal accounts, checkout or payouts; they never call a payment API or authorize, block or delay a transaction; and they can't stop a customer from paying or stop you from receiving or transferring funds. Your store's payments run through the live checkout flow, which is completely separate from Sentinel — Sentinel never sits in the money path. A Guard match only ever **holds a commit for review**; it never blocks a payment or hard-blocks the developer's local commit.

How flagged commits get reviewed

1. A commit gets held. When an agent commits, Sentinel auto-classifies it: a **primary agent with no guard hits** is auto-approved instantly; anything **flagged** (hit a guard or wrote outside allow-paths) or from a **non-primary agent** is set to **pending** and dropped into your review queue.

2. You review it in the Dashboard → Pending tab. Each item shows the agent, the commit message, the files it touched and *why* it was flagged. You then click **Approve** (it becomes part of the record) or **Reject** (you're asked for a short reason, which is saved). You can also open the **Activity Report**, click the **flagged** chip to inspect flagged commits, then approve/reject from the Dashboard.

3. Every decision is logged. Approvals and rejections are written to the **Audit log** tab — granted or denied, the reason, who did it and the exact time — your permanent who-did-what-and-when trail.

Note: rejecting governs the *record*, not the developer's machine — it marks the commit rejected in the audit trail; it doesn't reach into their computer to undo their local commit. Reviewing is done in the Dashboard (there is currently no email alert when something is flagged — you find out by opening the Pending tab).

How to decide: Approve or Reject a flagged commit

When you open a flagged commit (Pending tab, or the flagged chip in the Activity Report), Sentinel shows you everything you need to decide. Read these four things first:

- **Who made it** — the agent's name, and whether it's your **primary** (trusted) agent.
- **The commit message** — what the change claims to do.
- **The files it touched** — with a short summary of each change. This is your fastest tell: a message can say “small fix” while the files reveal it touched something it shouldn't.
- **Why it was flagged** — either **outside allow-paths** (it edited a file outside the folders you marked safe) or **matched a guard** (it touched a sensitive file such as `.env`, auth or payment code).

Approve when…

- The files touched are the ones you'd expect for that task.
- It's ordinary code work (a feature, a fix, docs) that just landed slightly outside an allow-path.
- You recognize the agent and the change matches its message.

Reject when…

- It touched a **sensitive file** (`.env`, keys, payment or auth code) with no good reason.
- The files **don't match** what the message says it did.
- It edited config, deploy or infrastructure you didn't ask it to.
- You don't recognize the change, or it simply looks risky.

If you're unsure, read the file list first — that's the quickest way to tell whether a change is what it claims to be.

Rejecting is safe. It only governs the *record*: it marks the commit rejected in your Audit log and asks for a one-line reason. It never reaches into the agent's or developer's computer and never undoes their local work — it's you saying “this shouldn't have happened, and here's why” for the permanent trail.

So, you got your first flagged commit. Here's how you fix it

Don't worry — a flag is not an error and nothing was blocked. Sentinel is passive: your commit already landed in the repo. A flag simply means “a human should look at this one,” and Sentinel held it in the **Pending** tab for you. Here's how to clear it and stop it coming back.

Step 1 — find out why it was flagged

Open the commit in the **Pending** tab and read the **“Why flagged”** line. It tells you exactly which rule tripped:

- **outside allow-paths: <file>** — the commit touched a file *outside* the folders you marked as safe for that project.
- **matched guard: <pattern>** — the commit touched a *sensitive* file (for example `.env`, keys, or payment/auth code).

Then check the **Files touched** list on the same commit so you can see the actual file that set it off.

Step 2 — decide: is this change fine?

- **Yes, it's fine** (you recognise the change and the files look right) → click **Approve**. That's it — the commit is cleared.
- **No, that looks wrong** (it touched something it shouldn't) → click **Reject** and add a one-line reason for the record.

Important: Approve is per-commit, not per-file

Approving clears *that one commit* — it does **not** whitelist the file. Flagging is decided fresh at every commit, so the **next** commit that touches the same file will flag **again**. If a path keeps flagging and you're happy for it to be edited freely, don't keep clicking Approve — fix the rule instead (Step 3).

Step 3 — stop it flagging every time (adjust your Guards)

If the file is a legitimate, everyday part of the project, update the project's governance so Sentinel stops asking. The fastest way: click the **“Adjust Guards →”** button on the flagged commit itself — it jumps straight to the **Guards** tab with the offending path pre-filled (or the matching guard pattern highlighted), so the fix is one tap away. Then:

- If it was **outside allow-paths** and it *should* be allowed → add that folder to the project's **Allow-paths**. Clean commits inside allow-paths from your primary agent then auto-approve.
- If it **matched a guard** but isn't actually sensitive → loosen or remove that **sensitive-pattern**. (Keep guards on truly sensitive files like `.env` and keys — that's the whole point of Sentinel.)

Good to know

- Even your **primary agent** gets queued when a commit is flagged — the trusted auto-approve fast-lane only applies to *clean* commits. So a flagged primary-agent commit isn't a bug; it's Sentinel doing its job.
- Nothing is ever blocked and no local work is lost — approving or rejecting only writes to the audit record.

Two procedures every monitor should know

Almost every flag falls into one of two buckets. Learn to tell them apart quickly — one is a routine “allow it”, the other is a “stop and investigate”.

Procedure 1 — Allowing a safe path

Use this when the flag is harmless: the commit touched ordinary working code that simply sat outside your Allow-paths.

Easiest way — one tap: on the flagged commit in **Pending**, click **Adjust Guards** →. It opens the Guards tab with the exact path already filled into the Allow-paths box — just click **Add**, then return to Pending and **Approve** the waiting commit. No copying needed.

By hand (if you prefer):

1. Open the flagged commit in **Pending** and find the **Why flagged** line. The offending path is shown in **cyan highlight** right after *outside allow-paths:* — that highlighted text is the exact string to use.
2. Confirm it's normal code and **not** a secret (see Procedure 2 for the danger list).
3. Open the **Guards** tab and type the path into the **Allow-paths** box — as one file (`sentinel_sdk.py`), a whole folder (`scripts/**`), or a file type (`*.py`).
4. Click **Add** — it saves instantly.
5. Return to **Pending** and **Approve** the commit already waiting there.

Glob tips: `*` matches within one folder, `**` matches any depth. Remember — the Allow-paths rule stops *future* flags, while Approve clears the *one* already queued. Approve is per-commit; the rule is forever.

Procedure 2 — Paths that mean big trouble

A flag on any of these is a **stop-and-investigate**, never an automatic add-to-Allow. If an agent touched one without a clear, expected reason, **Reject** it and ask why. On the commit, the guard pattern that matched is shown in **rose highlight** on the **Why flagged** line.

- **Secrets & environment:** `.env`, `.env.*`, `secrets/**`, `credentials/**`, `*.pem`, `*.key`, `id_rsa`.
- **Auth & identity:** `**/auth/**`, and anything touching login, passwords, tokens, sessions or JWT.
- **Payments:** `**/stripe*`, `**/paypal*`, `**/payment*`, and checkout / billing code.
- **Infrastructure & deploy:** `Dockerfile`, `**/.github/**`, CI/CD, `**/*deploy*`, and Kubernetes / Terraform config.
- **Database migrations:** `**/migrations/**`.

These are the files that can leak credentials, hijack identity, redirect money, or ship code to production. Keep them guarded — that is exactly what Sentinel is for.

Optional: email yourself when a commit is flagged

Sentinel doesn't send emails itself — that keeps the choice, the email address and any cost entirely with you. If you'd like an alert the moment a commit is flagged, run this small script on your own computer using **your own email address and email service**. It checks your flagged queue on a schedule and emails you only the **new** ones.

```
import smtplib, ssl, json, time, urllib.request
from email.message import EmailMessage

# ---- 1) FILL THESE IN ----
SENTINEL_BASE  = "https://sci-fi-reader.emergent.host"   # your Sentinel site
SENTINEL_TOKEN = "PASTE_YOUR_TOKEN_HERE"                 # from the re-open command
TO_EMAIL   = "you@example.com"        # where alerts should go
SMTP_HOST  = "smtp.gmail.com"         # your email provider's SMTP server
SMTP_PORT  = 465
SMTP_USER  = "you@example.com"
SMTP_PASS  = "your-app-password"      # an app password, NOT your normal login
CHECK_EVERY_MIN = 15
SEEN_FILE  = "sentinel_alerted.json"

def load_seen():
    try: return set(json.load(open(SEEN_FILE)))
    except Exception: return set()

def fetch_flagged():
    url = SENTINEL_BASE + "/api/sentinel/me/flagged-commits"
    req = urllib.request.Request(
        url, headers={"Authorization": "Bearer " + SENTINEL_TOKEN})
    with urllib.request.urlopen(req, timeout=30) as r:
        return json.load(r).get("commits", [])

def send_email(new):
    lines = ["- [%s] %s: %s" % (c["project"], c["agent"], c["message"])
             for c in new]
    body = ("Sentinel flagged %d commit(s) for review:\n\n%s\n\n"
            "Review them in your Sentinel Dashboard -> Pending tab."
            % (len(new), "\n".join(lines)))
    msg = EmailMessage()
    msg["Subject"] = "Sentinel: %d commit(s) need review" % len(new)
    msg["From"], msg["To"] = SMTP_USER, TO_EMAIL
    msg.set_content(body)
    ctx = ssl.create_default_context()
    with smtplib.SMTP_SSL(SMTP_HOST, SMTP_PORT, context=ctx) as s:
        s.login(SMTP_USER, SMTP_PASS)
        s.send_message(msg)

def main():
    seen = load_seen()
    while True:
        try:
            fresh = [c for c in fetch_flagged() if c["id"] not in seen]
            if fresh:
                send_email(fresh)
                seen.update(c["id"] for c in fresh)
                json.dump(sorted(seen), open(SEEN_FILE, "w"))
                print("Alerted on", len(fresh), "commit(s)")
        except Exception as e:
            print("check failed:", e)
        time.sleep(CHECK_EVERY_MIN * 60)

if __name__ == "__main__":
    main()
```

Setting it up (about 5 minutes):

- **Token:** run the re-open command (or copy it from the Install page) and paste the value into `SENTINEL_TOKEN`.
- **Your email:** set `TO_EMAIL` and your provider's SMTP details (`SMTP_HOST/PORT/USER/PASS`). Gmail, Outlook, Fastmail, etc. all work — use an **app password**, not your normal login.
- **Run it:** save as `sentinel_alerts.py` and start it with `python sentinel_alerts.py`. Leave it running, or schedule it with Windows Task Scheduler / cron.

It uses only Python's built-in libraries — nothing to install. Change `CHECK_EVERY_MIN` to check more or less often. The `sentinel_alerted.json` file it creates remembers what it already emailed, so you won't get duplicates.

How the Projects drop-down works with the Agents, Commits and Pending buttons

The **Projects drop-down** at the top of the Dashboard is a **filter**. Whatever project you pick there decides *which* project the **Agents**, **Commits** and **Pending** tabs show. Think of it as “I'm looking at this project right now” — the three tabs always follow the drop-down.

Pick a project first, then read the tabs

- **Projects drop-down** — lists every project you've connected to Sentinel. It shows your *Sentinel projects*, not the folders on your computer.
- **Agents** — the agents (and people) registered for the **selected** project.
- **Commits** — every change reported for the **selected** project, newest first.
- **Pending** — the commits from the **selected** project that are waiting for your Approve/Reject decision.

Why a tab might say “none”

If Agents, Commits or Pending shows nothing, it almost always means the project selected in the drop-down simply has no data *yet* — for example a repo you connected but haven't committed to. It is **not** a bug and nothing is lost. Two things to check:

- **Look at the project name** in the drop-down. The empty-state message names the project it's showing, e.g. “No commits yet for **git repo**” — so you can tell at a glance which project is empty.
- **Switch the drop-down** to a project that has activity (one you've actually committed to). The Agents, Commits and Pending tabs fill in immediately.

To make an empty project show data, make a commit in that repo (or use **+ Connect a repo** to wire up the folder), and it appears here within seconds.

A helpful default when you open the Dashboard

So you don't land on an empty screen, Sentinel automatically selects the project with the **most commits** when the Dashboard opens. You can always change the drop-down to any other project; the Agents, Commits and Pending tabs update to match.

Staying signed in & the session-health dot

Sentinel keeps you signed in automatically. While the Dashboard window is open it quietly refreshes your access token in the background, so your session never expires while you are using it. You almost never have to do anything — and you never need to open a terminal or paste a command.

The session-health dot

Next to the date & time clock (top-right of the window) is a small colored dot that shows your session status at a glance. Hover over it to see a short message, including when your token last refreshed.

- 🟢 **Green — healthy.** Your session is valid and auto-renewing. Nothing to do.
- 🟠 **Amber — needs attention.** Your session is within a day of expiring, or has already expired (this only happens if Sentinel has not been opened for about a week). Just click the dot — Sentinel fixes it for you.

If the dot is amber: just click it

Clicking the dot is all you need. Sentinel first tries to renew your session silently, right where you are — no terminal, no copying, no visit to the website. If it works, a small toast confirms “Session refreshed” and the dot turns green.

The “Sign in again” box

If your session has fully expired (Sentinel unopened for more than a week), a silent renew is no longer possible, so clicking the dot opens a small **Sign in again** box inside the window. Enter your Sentinel email and password and click **Sign in & restore session**. That is all — the Dashboard and Activity Report load again immediately.

What the re-sign is for: a security token only lasts seven days. Signing in again creates a brand-new seven-day session and saves it to this computer so Sentinel can keep renewing it on its own. Your password is sent only to Sentinel to obtain the new token — it is never stored on your machine or shown to anyone.

As long as you open Sentinel at least once a week, the dot stays green and you never have to think about tokens at all.

Need a hand? Email support@sci-ficomics.com and we'll help you get set up.