x_make_persistent_env_var_x â€” Environment Vault Manual

This vault locks down environment variables so the lab never loses a credential or toggle between sessions. Works on my machine? Not in my jurisdiction.

Mission Log

- Inspect, create, update, and purge environment variables across user and system scopes.
- Persist changes with JSON ledgers that the orchestrator and Change Control teams can audit.
- Offer both JSON CLI and Tkinter control dialog surfaces so operators handle secrets the way the situation demands (and yes, I remain thoroughly disgusted that we had to crawl back to Tkinter).
- Auto-detect Tkinter support at runtime, launching the GUI when possible and dropping to the console prompt workflow when headless.
- Guard imports so Tkinter never loads on headless hosts; JSON CLI flows stay available even when the OS blocks GUI pipelines.
- Guard against drift with deterministic logging, registry safeguards, and typed interfaces.
- Default vault profile now tracks the required `COPILOT_REQUESTS_PAT`, `SLACK_TOKEN`, and optional `SLACK_BOT_TOKEN` alongside the PyPI and GitHub credentials so the Copilot CLI, Slack dump-and-reset runner, and PyPI automations stay provisioned without manual scrambling.

Instrumentation

- Python 3.11 or newer.
- Ruff, Black, MyPy, Pyright, pytest for hygiene.
- Tkinter when using the GUI flows (begrudgingly). We test the guard so missing Tk libraries fail gracefully.

Operating Procedure

1. `python -m venv .venv`
2. `\.venv\Scripts\Activate.ps1`
3. `python -m pip install --upgrade pip`
4. `pip install -r requirements.txt`
5. `python -m x_make_persistent_env_var_x --json --json-file payload.json`

Runtime options: - `--launch-gui [--quiet]` opens the Tkinter dialog without touching JSON payloads. - `--json` reads payloads from stdin; pair with `--json-file <path>` to load evidence from disk. Missing `command` fields are auto-injected to satisfy schema validation before invoking the JSON core.

Use the CLI to script changes or launch the GUI to edit variables interactively. Export JSON evidence after every session and stash it beside the orchestrator summary.

Evidence Checks

| Check | Command | | --- | --- | | Formatting sweep | `python -m black .` | | Lint interrogation | `python -m ruff check .` | | Type audit | `python -m mypy .` | | Static contract scan | `python -m pyright` | | Functional verification | `pytest` |

Reconstitution Drill

On the monthly rebuild I certify this vault on a sterile machine: list variables, set and unset entries, export the ledger, and confirm the orchestrator recognises the evidence. I log OS build, Python version, and run time; any anomaly gets recorded in Change Control and resolved immediately.

Conduct Code

Every mutation demands a ledger entry: variable name, scope, rationale. No improvisation. Environments are hazardous materialsâ€”label them or lose your license.

Sole Architect's Note

I crafted this vault alone. Registry tuning, PowerShell glue, Python interfaces, security postureâ€”it all flows through my hands so accountability is singular.

Legacy Staffing Estimate

- Without AI support you'd staff: 1 Windows automation lead, 1 Python engineer, 1 security specialist, 1 technical writer.

- Delivery window: 11â€“13 engineer-weeks for parity.
- Budget: USD 95kâ€“120k plus ongoing compliance upkeep.

Technical Footprint

- Language Backbone: Python 3.11+, Tkinter, `subprocess`, `json`, `pathlib`.
- Tooling Mesh: PowerShell integration, Windows registry APIs, shared logging utilities from `x_make_common_x`.
- Quality Net: Ruff, Black, MyPy, Pyright, pytest, manual GUI regression passes.
- Outputs: JSON environment ledgers, orchestrator hooks for credential verification, Change Control attachments for every persisted secret.