The sysadmin's AI cheat sheet: What to adopt, ignore, or automate

Treat AI like a fast junior sysadmin: useful, quick, and absolutely not to be trusted unsupervised. Every guardrail you add dulls the "wow"... but it also lowers the chance you ship the wrong change at the worst possible moment.

Quick rules

Default to read-only: If the output can't directly change state, the worst-case outcome is wasted time, not an outage. Drafts, summaries, queries, draft change plans — all fair game. But if you're going to execute something AI came up with, an experienced human should be involved.

Demand receipts: If it can't point to your runbooks, logs, CMDB, tickets, or pasted command output, treat the answer as a hypothesis. Useful ... but unproven.

Keep secrets out of tools you can't govern: If it's not an approved system with clear retention rules, audit logs, and scoped access, treat it like the public internet: no tokens, keys, customer data, sensitive internal URLs, or "here's the config from prod." Use placeholders.

ADOPT

High signal, low regret

- **Tickets & incident summaries**. Capture "what happened / what changed / what's next / who owns it." This move has a potentially big ROI but tiny blast radius.
- **Runbook cleanup**. Turn institutional knowledge into: prereqs → steps → validation → rollback → stop conditions without as much legwork.
- **Log & alert triage**. Generate ranked hypotheses & next checks. But do not trust any stated root causes without receipts.
- **Script drafts (PowerShell/Bash/Python)**. Include -WhatIf, idempotency, logging, and a rollback plan. These are helpful drafts, not run-ready.
- **User comms drafts**. Write status updates and maintenance notes that reduce "what's going on?" pings (and keep everyone calmer).

IGNORE

Time-wasters dressed as AI

- **Answers without assumptions, inputs, or verifiable steps**. If you can't see assumptions or inputs, you can't operate it responsibly.
- **Auto-root-cause confidence**. Identifying the root cause requires actual evidence.
- **"Connect to everything" on day one**. Broad access + unclear retention + no audit trail = breach fuel.
- **Auto-written policies you can't defend**. If you don't understand it, it's not a control — it's future-you in an audit, suffering.

AUTOMATE

... but only with gates

- **Software deployment (with rings)**. Deploy apps in canary → pilot → broad waves, then verify installs and roll back cleanly when something breaks.
- **Patch workflows**. Group endpoints, run prechecks, patch on schedule, run postchecks, and produce a compliance report you can drop into the change record.
- **Routine diagnostics**. Run read-only "grab me the facts" scripts (logs, services, disk, event slices) and attach outputs to the ticket instead of Slack archaeology.
- **Drift detection & change drafts**. Detect config drift, generate the diff, and open the PR/change request with receipts attached — reviewable, auditable, boring.
- **Reporting**. Collect inventory and status reports (apps, versions, patch levels) so audits and "what changed?" questions stop being a scramble.

Nonnegotiables for your SOP

- **Redact by default (prompts especially)**. Mask keys/tokens, internal URLs, sensitive hostnames, and customer data before pasting anything.
- **Require verification steps**. Include one check to confirm and one check to rule out with every recommendation.
- **Log + audit the workflow**. Attach a ticket/PR/change record to every run and capture the outputs.
- **Default to least privilege**. Start read-only, then grant scoped, time-bound access only when necessary.

PDQ Connect is a strong endpoint management backbone to deploy apps, run scripts, patch, and standardize configs. And with auditability built in, changes aren't just fast, they're trackable. Start with the boring stuff. Scale from there.