A RESOURCE FOR TEAMS WORKING WITH CODING AGENTS

Task verification checklist.

Done means a command exited 0, and the report says which.

Define it. Run it. Enforce it. Quote it.

28	04	02
checks, from task to report	gates: typecheck, lint, test, build	hooks that enforce the evidence

Claude stops when the work looks done.

Without a check it can run, "looks done" is the only signal an agent has, and you become the verification loop [S01]. The essay behind these guides puts it as Task, Verify, Commit: every task carries its done condition, and a task whose checks fail is not complete. This guide makes that loop concrete for a design-system repository and then takes it out of the prompt: a script produces the evidence, a Stop hook refuses to end the turn without it, and the final message quotes it.

The kit's demo project shows both outcomes. Its passing configuration runs three steps and exits 0; the failing one exits 1 with the failure quoted in `.verify/report.md`, and the Stop hook blocks until a passing report is newer than the last edit.

Version 1.0 / Sources checked 24 September 2026
Field Guide 30 of the Design × AI series, with 28 (CLAUDE.md), 29 (skills) and 31 (context). Hook fields checked against the Claude Code hooks reference (behavior up to v2.1.281); scripts tested on Node.js 22.22 with no dependencies.

Practical guidance, not a standard. Hook events, exit codes and JSON fields are from the Claude Code docs; the gate list, the evidence format and the suppression patterns are this guide's recommendations. Prepared with AI assistance and edited by hand.

START HERE

Pick how hard the gate should be.

The same check can be a request in a prompt or a rule the session cannot get past. Choose by how much you will watch the run [S01].

Gate	How it works	Use it when
In the prompt	"Run the tests and fix failures" in the task	You are watching; any task, today
`/goal`	A separate model checks a condition after every turn	A session-long target, such as all auth tests passing
Stop hook	Your script blocks the end of a turn until it passes	Unattended runs; the same rule in every session
Reviewer subagent	A fresh context tries to refute the result	Large or long-running work, before you count it done

From the Claude Code best practices [S01] and the /goal docs [S04]. They combine: this kit uses the prompt, the Stop hook and a reviewer.

Adding verification to a repo

Copy `verify.config.json` and `scripts/verify.mjs`, point the steps at your scripts, and add the Verification section to CLAUDE.md (Field Guide 28).

Enforcing it

Copy the two hook scripts to `.claude/hooks/` and merge `settings.example.json`. Run `validate-hooks.mjs` on the result.

Reviewing agent work

Section 06: check the final message against the report template before reading the diff.

Hooks that never fire

Items V21 and V23. A schema-valid config can still be ignored; the validator shows why.

Label	Meaning
`DOCS`	Behavior stated in the Claude Code documentation.
`KIT CHECK`	Produced or checked by a kit script: `verify.mjs`, `check-suppressions.mjs`, `validate-hooks.mjs`.
`HOOK`	Enforced by a hook, whatever the agent decides.
`PRACTICE`	A working method; the evidence line says what to review.

SECTION 01

Define done before the task starts.

A done condition written after the work is a description of the work. Write it first.

- [ ] **The task names the check that proves it**

  V01 DOCS

  "Write validateEmail; these three cases pass; run the tests after implementing." A check is anything that returns a pass or fail Claude can read: a test, an exit code, a screenshot to compare.

  Evidence: The task text contains a command or a comparison target.

  Give Claude a way to verify its work. S01.

- [ ] **The project's definition of done is written down**

  V02 KIT CHECK

  CLAUDE.md has a `## Verification` section listing the commands and what to report, so every task inherits it.

  Evidence: Field Guide 28's linter errors when the section is missing.

  Field Guide 28, item C07.

- [ ] **Checks are commands, not adjectives**

  V03 PRACTICE

  "`npm test -- tooltip` exits 0", not "tests pass". A command can be rerun by a hook, a reviewer or you.

  Evidence: Every line in the task's Verify list starts with a command or a file to open.

  The essay's task format: Action, Verify, Done.

- [ ] **Fast checks while working, the full gate at the end**

  V04 DOCS PRACTICE

  Run the affected test file after each change and the whole gate before finishing. Single tests keep the loop short; the gate catches what they miss.

  Evidence: The session shows focused runs, then one full `verify.mjs` run.

  Prefer running single tests (the docs' CLAUDE.md example). S01.

- [ ] **Out of scope is stated**

  V05 DOCS

  Name the files, APIs or behavior that must not change. A reviewer can then check that nothing outside the task moved.

  Evidence: The task or spec has an out-of-scope line.

  Self-contained specs state what is out of scope. S01.

If you cannot say how you would check it, the task is not ready.

SECTION 02

Run the four gates, every time.

Typecheck, lint, test and build are cheap, deterministic and already in the repo. `verify.mjs` runs them from one config and writes the evidence.

- [ ] **Typecheck exits 0**

  V06 KIT CHECK

  The whole project, not only the files touched. Type errors surface in files the agent never opened.

  Evidence: `typecheck` row in `.verify/report.md`, exit 0.

  Kit script.

- [ ] **Lint exits 0 with no new warnings**

  V07 KIT CHECK

  Including the design-system gates from Field Guide 03, which reject raw colors and core tokens. `--max-warnings 0` on touched files.

  Evidence: `lint` row, exit 0; the PostToolUse hook runs ESLint per edited file.

  Kit script. Field Guide 03.

- [ ] **Tests exit 0, honestly**

  V08 KIT CHECK

  No new `.only` or `.skip`, and no expectation rewritten to match wrong output. The kit's failing demo is exactly that case: a test expecting "0.00" where the code returns "€0.00".

  Evidence: `test` row, exit 0; `check-suppressions.mjs` finds no new focused or skipped tests.

  Address root causes, not symptoms. S01.

- [ ] **Build exits 0**

  V09 KIT CHECK

  The production build, including token generation. A green test run on a package that does not build is not done.

  Evidence: `build` row, exit 0.

  Kit script.

- [ ] **Every gate runs, even after a failure**

  V10 KIT CHECK

  `verify.mjs` runs all steps and reports all failures, so one pass fixes everything. Optional steps (`"required": false`) are reported but do not fail the run.

  Evidence: The failing demo reports 3 of 4 steps passed, then exits 1.

  Kit script.

The gate is only as good as the fact that it always runs.

SECTION 03

Check what the gates cannot see.

A passing gate proves the checks ran, not that the change is right. Close the gaps the four commands leave open.

- [ ] **No new suppressions**
  V11 KIT CHECK
  `@ts-ignore`, `eslint-disable`, `as any`, `.skip` and `.only` turn a failing gate green. Each one needs a reason in the line (`verify-allow: <reason>`) or a fix.
  Evidence: `check-suppressions.mjs` on the kit's diff fixture: 5 findings, 1 allowed line ignored.
  Kit script.

- [ ] **UI changes are looked at**
  V12 DOCS
  Open the story or page, take a screenshot and compare it with the design or the previous state; list the differences.
  Evidence: The report names the stories checked and what was compared.
  Verify UI changes visually. S01.

- [ ] **Accessibility runs as a failure, not a warning**
  V13 PRACTICE
  Axe on the changed stories with violations failing the run, as Field Guide 02 requires for stable components.
  Evidence: The optional `stories` step, or your Storybook test run, in the report.
  Field Guide 02, item 05.10.

- [ ] **The app is run, not only the tests**
  V14 DOCS
  Claude Code's bundled `/verify` builds and runs the app to confirm a change, without falling back to tests. `/run-skill-generator` records the launch recipe once.
  Evidence: A `/verify` run or a recorded manual check in the report.
  Run and verify your app. S05.

- [ ] **A fresh context reviews large work**
  V15 DOCS
  A reviewer subagent sees only the diff and the plan. Tell it to report correctness gaps only; a reviewer asked for gaps always finds some.
  Evidence: Review findings in the pull request, with the ones acted on marked.
  Add an adversarial review step. S01, S08.

Green checks are evidence the checks ran. Look at the thing itself.

SECTION 04

Enforce it with hooks.

CLAUDE.md is context, not configuration; a hook runs at its lifecycle event whatever Claude decides [S09]. Two hooks cover most of the loop.

- [ ] **A Stop hook requires fresh evidence**

  V16 HOOK KIT CHECK

  `stop-require-evidence.mjs` lets a turn end only when the working tree is clean or a passing report is newer than every change. It reads the report; it never runs the suite, so it stays fast.

  Evidence: In the demo: blocks with no report, allows after a passing run, blocks again after one edit.

  Stop hook as a deterministic gate. S01, S02.

- [ ] **The hook blocks with a reason Claude can act on**

  V17 HOOK DOCS

  It prints `{"decision": "block", "reason": ...}` and exits 0. The reason names the failing steps and the command to rerun.

  Evidence: Hook output on the demo; the reason names `node scripts/verify.mjs`.

  Stop decision control: reason is required with block. S02.

- [ ] **A check that cannot pass does not trap the session**

  V18 DOCS

  Claude Code ends the turn after 8 consecutive Stop blocks, and the input's `stop_hook_active` shows when a turn is already a continuation. `VERIFY_HOOK_OFF=1` disables the kit hook for one session.

  Evidence: The 8-block cap is documented behavior; the kit does not test it.

  Stop input. S01, S02.

- [ ] **A PostToolUse hook checks each edit**

  V19 HOOK KIT CHECK

  `post-edit-check.mjs` runs one fast check on the edited file. On failure it exits 2: for PostToolUse, exit 2 shows stderr to Claude, while exit 1 is a non-blocking error Claude never sees.

  Evidence: In the demo, a syntax error returns exit 2 with the `node --check` output on stderr.

  Exit code 2 behavior per event. S02.

- [ ] **Scripts run in exec form from the project root**

  V20 DOCS

  `"command": "node", "args": ["${CLAUDE_PROJECT_DIR}/.claude/hooks/..."]`. Exec form needs no quoting and works on Windows, where npm shims cannot be spawned without a shell.

  Evidence: `settings.example.json` uses exec form for both hooks.

  Exec form and shell form; reference scripts by path. S02.

Instructions ask. Hooks decide.

SECTION 05

Keep the gate honest.

A hook that silently does nothing is worse than none: everyone believes the gate exists.

- [ ] **The hook config is validated, not only parsed**

  V21 KIT CHECK

  A matcher on Stop is silently ignored, `if` on a non-tool event means the hook never runs, and `async` removes a gate's power to block. The SchemaStore schema and `claude plugin validate` both accept such a file; `validate-hooks.mjs` does not.

  Evidence: `settings.silent-failures.example.json`: schema-valid, passes `claude plugin validate`; the kit reports 2 errors and 2 warnings.

  Matcher, if and async rules. S02, S07.

- [ ] **Repository hooks are reviewed before scripted runs**

  V22 DOCS

  Interactive sessions hold hooks back until you trust the folder. `claude -p` treats the folder as trusted, so a cloned repo's hooks run. Review `.claude/settings.json` first.

  Evidence: A review note, or `--settings '{"disableAllHooks": true}'` for untrusted repos.

  Workspace trust. S02.

- [ ] **A broken hook path is caught on the first run**

  V23 DOCS PRACTICE

  A mistyped script path exits 127, which is a non-blocking error: the action proceeds and the gate is silently off. Trigger each hook once after setup and read the transcript.

  Evidence: `/hooks` lists both hooks, and one deliberate failure shows the block.

  Other exit codes; `/hooks` menu. S02.

Test the gate by making it fail once.

SECTION 06

Report with evidence.

Reviewing evidence is faster than rerunning the checks, and it works for sessions nobody watched [S01].

- [ ] **Every check is listed with its exit code**
  V24 KIT CHECK
  Command, exit code, pass or fail, in a table. `REPORT.template.md` gives the shape; `.verify/report.md` fills it.
  Evidence: The final message contains the table.
  Show evidence rather than asserting success. S01.

- [ ] **Failures are quoted, not summarized**
  V25 KIT CHECK
  The failing output itself, trimmed to the relevant lines. "One test is flaky" is a claim; the assertion diff is evidence.
  Evidence: `report.md` includes the last 40 lines of each failing step.
  Kit script. S01.

- [ ] **What was not verified is named**
  V26 PRACTICE
  No fixture, needs production data, needs a device. A gap stated is a gap someone can close; a gap omitted is a surprise.
  Evidence: The "Not verified" line in the report is filled or says "nothing".
  Report template.

- [ ] **The evidence is a file others can open**
  V27 KIT CHECK
  `.verify/report.json` for hooks and tools, `.verify/report.md` for people, linked from the pull request.
  Evidence: Both files written on every run.
  Kit script.

- [ ] **One task, one commit, after the checks pass**
  V28 PRACTICE
  Commit when the gate is green, one task per commit. The history becomes a log of verified steps, and a bisect lands on a single task.
  Evidence: Commit messages reference the task; no commit without a passing report.
  The essay's Task, Verify, Commit loop.

"Done" is a claim. The report is the proof.

APPENDIX A

The loop, run on the demo project.

`examples/demo-app` is a tiny package with a check, tests and a build. Output below is from the kit.

terminal (examples/demo-app)

```
$ node ../../scripts/verify.mjs --config verify.fail.config.json
pass  check        exit 0     npm run check
pass  test         exit 0     npm test
FAIL  regression   exit 1     node --test broken/*.test.mjs
pass  build        exit 0     npm run build
3 of 4 step(s) passed. Failed: regression. Evidence: .verify/report.md

$ node ../../scripts/verify.mjs
pass  check        exit 0     npm run check
pass  test         exit 0     npm test
pass  build        exit 0     npm run build
3 of 3 step(s) passed. Verification passed. Evidence: .verify/report.md

$ touch src/format-price.mjs && echo '{"cwd":"."}' | node ../../hooks/stop-require-evidence.mjs
{"decision":"block","reason":"1 file(s) changed after the last passing verification
(src/format-price.mjs). Re-run `node scripts/verify.mjs` before you finish."}
```

The hook input is abbreviated to `cwd`; Claude Code sends the full Stop payload (see `examples/hook-input/stop.json`).

Event	Exit 2 does	JSON decision	Kit use
`Stop`	Prevents stopping; stderr becomes the reason	`decision: "block"` with `reason`	Require fresh evidence
`PostToolUse`	Shows stderr to Claude; the tool already ran	`decision: "block"` adds the reason	Check the edited file
`TaskCompleted`	Keeps the task open	Exit code, or `continue: false`	Alternative: verify per task
`SubagentStop`	Prevents the subagent from stopping	`decision: "block"` with `reason`	Alternative: agent-hook review

From the hooks reference [S02]. Exit 1 blocks nothing on these events; use exit 2 or JSON.

APPENDIX B

The hook config, and what the validator catches.

`settings.example.json` validates against the SchemaStore schema and the kit's validator with no errors or warnings. The silent-failures example validates against the schema too, which is the problem.

settings.example.json (hooks excerpt)

```
"PostToolUse": [{ "matcher": "Edit|Write", "hooks": [{
  "type": "command", "command": "node",
  "args": ["${CLAUDE_PROJECT_DIR}/.claude/hooks/post-edit-check.mjs"],
  "timeout": 60, "statusMessage": "Checking the edited file" }] }],
"Stop": [{ "hooks": [{
  "type": "command", "command": "node",
  "args": ["${CLAUDE_PROJECT_DIR}/.claude/hooks/stop-require-evidence.mjs"],
  "timeout": 30 }] }]
```

The full file also re-states the definition of done after compaction with a `SessionStart` hook on the `compact` matcher.

terminal

```
$ node scripts/validate-hooks.mjs settings.example.json
settings.example.json: 3 handler(s), 0 error(s), 0 warning(s)

$ node scripts/validate-hooks.mjs examples/settings.silent-failures.example.json
error    hooks.Stop[0].hooks[0]: `if` is evaluated only on tool events; on Stop a
         handler with `if` never runs.
error    hooks.SessionStart[0].hooks[0]: SessionStart does not support "prompt"
         handlers; it accepts command, mcp_tool.
warning  hooks.Stop[0]: Stop has no matcher support; "Bash" is silently ignored.
warning  hooks.Stop[0].hooks[0]: An async hook cannot block or decide anything.
2 handler(s), 2 error(s), 2 warning(s)
```

The same file passes the SchemaStore schema (Ajv 8) and `claude plugin validate` in Claude Code 2.1.271. The broken example, with shape errors, reports 7 errors and 4 warnings.

KEEP WITH THE PULL REQUEST

Leave a verification record.

One record per agent-completed task that is merged. It is what a reviewer reads before the diff.

Task / pull request	**Definition of done (link)**
Verify run: steps passed / total	**Report (.verify/report.md link)**
Suppressions added, with reasons	**Stories or pages checked**
Reviewer findings acted on	**Not verified, and why**
Hooks active (Stop, PostToolUse)	**Commit**

> **No report, no merge.**
> If the evidence is missing, the task is not finished, however good the diff looks.

SOURCES / MAINTENANCE

Keep the guide current.

Sources checked 24 September 2026. The settings docs note that the SchemaStore schema can lag the newest CLI releases; when the docs and the schema disagree, the docs win.

S01 / Claude Code docs, best practices
Give Claude a way to verify its work; gate strength; evidence over assertion; adversarial review.
https://code.claude.com/docs/en/best-practices

S02 / Claude Code docs, hooks reference
Events, matchers, handler fields, exit codes per event, Stop and PostToolUse decision control, 8-block cap, workspace trust.
https://code.claude.com/docs/en/hooks

S03 / Claude Code docs, hooks guide
Worked hook examples; re-inject context after compaction.
https://code.claude.com/docs/en/hooks-guide

S04 / Claude Code docs, /goal
A session-scoped completion condition checked after every turn.
https://code.claude.com/docs/en/goal

S05 / Claude Code docs, skills (bundled /run and /verify)
Build and run the app to confirm a change.
https://code.claude.com/docs/en/skills

S06 / Claude Code docs, settings
Where settings live; the $schema line and its lag.
https://code.claude.com/docs/en/settings

S07 / SchemaStore, claude-code-settings.json
The JSON Schema editors use for settings files; used to cross-check the kit's configs.
https://json.schemastore.org/claude-code-settings.json

S08 / Claude Code docs, subagents
Fresh-context workers for review and verification.
https://code.claude.com/docs/en/sub-agents

S09 / Claude Code docs, how Claude remembers your project
CLAUDE.md is context, not enforced configuration; use hooks for fixed points.
https://code.claude.com/docs/en/memory

Maintenance: recheck the hooks reference at each Claude Code minor release and update the event lists in `validate-hooks.mjs` (EVENTS, NO_MATCHER, TOOL_EVENTS). Update the PDF, HTML, Markdown and JSON together.