stack · v1.1.2 — Full Reference

Longhand · Context-Mode · Hardgate

One command to install them all.

Table of Contents

1. What This Is

`stack` is a coordinated installer for three Claude Code plugins that extend how Claude Code stores memory, manages context, and enforces rules.

Two install paths are available:

- **`python install.py`** — standalone Python installer, no active Claude Code session required. Works from any terminal on any platform including Windows Command Prompt.
- **`/stack`** — Claude Code skill, runs interactively inside an active Claude Code session.

Both paths install the same tools in the same order, take timestamped config backups, run idempotency checks before each step, and verify the result using the shared `scripts/verify.py`.

---

2. The Three Tools

Longhand (memory layer)

By default, Claude Code forgets everything when a session closes. Longhand saves every conversation to a local SQLite + ChromaDB database and exposes a semantic search endpoint as an MCP server. Claude can recall past sessions in ~126ms with zero API cost.

Hooks used:

- `UserPromptSubmit` — records each prompt as it arrives
- `SessionEnd` — ingests the full session into the database on close

Config entries:

- `hooks.UserPromptSubmit` — contains `longhand __prompt-hook-run`
- `hooks.SessionEnd` — contains `longhand ingest-session`
- `mcpServers.longhand` — `longhand mcp-server`

Context-Mode (context layer)

Tool output normally flows directly into the conversation window. On large codebases or long sessions, this causes Claude to lose track of earlier context as the window fills. Context-Mode intercepts tool output in a sandboxed subprocess, summarises it, and returns only the summary to Claude.

Hooks used:

- `PreToolUse` — one matcher entry per tool type (Bash, Read, WebFetch, and more)

Config entries:

- `hooks.PreToolUse` — ≥3 matcher entries, each routing through Context-Mode
- `mcpServers.context-mode` — the sandbox server

Hardgate (enforcement layer)

Hardgate lets operators define tools Claude is not allowed to use without explicit approval. When a blocked tool is attempted, a hook script exits with code 2, which Claude Code treats as an enforcement signal — Claude stops and reports the block.

Hooks used:

- `PreToolUse` — a `.sh` script that exits 2 for blocked tools

Config entries:

- `hooks.PreToolUse` — the hardgate script entry
- Script file in `~/.claude/hooks/`

Hardgate requires an interactive setup step (`/hard-gate`) that cannot be automated. The installer pauses and guides the user through it.

3. Architecture

Request flow

```
User sends message
       │
       ▼
UserPromptSubmit fires
  └─ Longhand records prompt to DB

Claude decides to call a tool
       │
       ▼
PreToolUse fires — Hardgate checks first
  ├─ Tool blocked? → exit 2. Claude stops. Reports block.
  └─ Not blocked? → continues

PreToolUse fires — Context-Mode intercepts
  └─ Runs tool in sandboxed subprocess
     Summarises output
     Returns summary to Claude (not raw output)

Claude uses summary to respond

Session closes
       │
       ▼
SessionEnd fires
  └─ Longhand ingests full session into DB
```

Config layout

All three tools write into `~/.claude/settings.json`:

```
~/.claude/settings.json
├── hooks
│   ├── UserPromptSubmit   ← Longhand prompt hook
│   ├── PreToolUse         ← Hardgate (exit 2) + Context-Mode (≥3 matchers)
│   └── SessionEnd         ← Longhand ingest
└── mcpServers
    ├── longhand           ← Longhand memory query endpoint
    └── context-mode       ← Context-Mode sandbox server
```

The `longhand` MCP server may also appear in `~/.claude.json` depending on the Claude Code version. The verifier checks both locations.

Hook independence

Each tool uses a different hook event. They cannot conflict at the hook level. The only runtime interaction is ordering within `PreToolUse`: Hardgate runs first (exit 2 terminates the chain), then Context-Mode.

4. Install Paths

Option A — Standalone installer

```
# macOS / Linux
bash install.sh

# Windows (any terminal — Git Bash, cmd, PowerShell)
python install.py

# Verify-only mode
python install.py --verify
```

Flow:

1. Check Python 3.10+, Node.js 18+, Claude Code CLI (offer to install if missing)
2. Discover or clone Context-Mode directory
3. Take timestamped backup of `~/.claude/settings.json` and `~/.claude.json`
4. Check idempotency for Longhand — skip if fully installed
5. Check idempotency for Context-Mode — skip if fully installed
6. Check idempotency for Hardgate — skip if fully installed; else pause for `/hard-gate`
7. Run `scripts/verify.py` — restore backups on exit 2 (malformed config)

Option B — Claude Code skill

```
# inside an active claude session:
/stack
```

Flow: Identical logic expressed as Claude instructions. Claude executes each step interactively, pauses for Hardgate, then runs `scripts/verify.py` in Phase 3.

Idempotency

Both paths check all required artifacts before installing each tool:

Tool	Required artifacts
Longhand	`longhand` binary on PATH, SessionEnd hook, UserPromptSubmit hook, MCP entry
Context-Mode	≥3 PreToolUse matchers (Bash, Read, WebFetch required), no duplicates, MCP entry
Hardgate	`.sh` file in hooks dir containing "hardgate", wired in PreToolUse

If all artifacts are present, that tool is skipped. If any artifact is missing, the full installer for that tool runs again. Re-running is always safe.

5. Design Decisions

Why `importlib.util.spec_from_file_location` for verify.py

`install.py` and `scripts/verify.py` need to share check logic without a package structure. Dynamic loading via `importlib` avoids adding `scripts/` to the install path permanently and keeps the repo layout flat. The trade-off is that the import is opaque to static analysers. Tests work around this with `sys.path.insert`.

Why exit code 127 for command-not-found in `_run()`

`FileNotFoundError` from `subprocess.run` when a binary doesn't exist would propagate as an unhandled exception. Catching it and returning 127 follows the Unix convention for "command not found" and lets the caller handle it as a normal non-zero exit — consistent with how other failures are handled.

Why `_is_context_mode_install_js()` content validation

`_find_context_mode_dir()` looks for directories named `context-mode` or `context_mode` containing an `install.js`. Without content validation, an unrelated project with the same name and a generic installer would be silently picked up. The content check requires `install.js` to reference "context-mode" or "context_mode" — a low false-positive rate with no meaningful false-negative risk on the real installer.

Why `_find_context_mode_dir()` skips common system directories

The bounded home-directory walk (3 levels deep) would iterate hundreds of directories on a typical developer machine (`.cargo`, `.npm`, `Library`, `AppData`, etc.) that will never contain Context-Mode. The `_SKIP_DIRS` exclusion list at depth-1 reduces the search space significantly with no correctness trade-off.

Why Hardgate idempotency uses a heuristic

Hardgate has no CLI or config file that can be queried programmatically. The only available signal is: does a `.sh` file containing "hardgate" exist in `~/.claude/hooks/` and is it wired in `settings.json`? This is a heuristic, not an authoritative check. The verifier adds a second layer (wiring verification) to reduce false positives. A script that exists but isn't wired is explicitly reported as incomplete.

Why `is_hardgate_complete()` catches `Exception` broadly

The function calls into `verify.py`'s `check_hardgate_artifacts()`, which reads the filesystem and parses JSON. On a machine where HOOKS_DIR points to a file instead of a directory (`NotADirectoryError`), or where hook data is malformed (`TypeError`), or where permissions are denied (`OSError`), the function should return `False` rather than crash the installer. A broad catch is appropriate here because all exceptions have the same meaning: "not fully installed."

Why backups use timestamps and don't self-clean

Backups are named `settings.json.stack-backup-YYYYMMDD-HHMMSS`. Each run creates a new backup; old ones are never deleted. This is intentional: the installer is designed to be run repeatedly (idempotency) and each run may change configuration. Users may want to roll back to any prior state, not just the most recent. The trade-off is accumulating backup files. Users who want to clean up can delete old backups manually from `~/.claude/`.

6. Post-Install Verifier

`scripts/verify.py` checks six things and exits with a semantic code:

Exit code	Meaning
0	All required checks passed (Hardgate warning is acceptable)
1	One or more required checks failed; settings.json is valid
2	settings.json or .claude.json exists but is malformed — restore from backup

Checks

Check	What it verifies	Fix if failing
Longhand SessionEnd hook	`longhand ingest-session` is wired	`longhand setup`
Longhand UserPromptSubmit hook	`__prompt-hook-run` is wired	`longhand prompt-hook install`
Context-Mode PreToolUse hooks	≥3 matchers, Bash+Read+WebFetch present, no duplicates	`node install.js`
Longhand MCP server	`longhand` in mcpServers (either config file)	`claude mcp add longhand -s user -- longhand mcp-server`
Context-Mode MCP server	`context-mode` in mcpServers	`node install.js`
Hardgate enforcement	Hardgate script exists and is wired (warning if absent)	`Run /hard-gate`

Known limitation

The verifier checks config presence, not live MCP server status. After install, run:

```
longhand doctor
claude mcp list
```

These confirm the processes are running, not just configured.

7. File Reference

File	Purpose
`install.py`	Standalone installer — stdlib only, no dependencies
`install.bat`	Windows launcher for `install.py`
`install.sh`	macOS/Linux launcher for `install.py`
`scripts/verify.py`	Post-install state verifier — shared by both install paths
`skills/stack.md`	Claude Code skill definition for `/stack`
`tests/conftest.py`	Shared test helpers (make_settings, make_hooks_dir, etc.)
`tests/test_verify.py`	38 tests for verify.py logic and exit codes
`tests/test_install.py`	37 tests for install.py helpers, idempotency, new functions
`docs/index.html`	GitHub Pages landing page
`docs/USER-MANUAL.html`	HTML user manual
`docs/README-FULL.pdf`	This document
`docs/CONTRIBUTING.md`	Contribution guide (synced from root)
`docs/discussion-seeds.md`	Seeded content for GitHub Discussions

---

8. Testing

```
pip install pytest
pytest tests/ -v
```

75 tests. All are unit tests with injected paths — no test touches `~/.claude`.

Test structure:

- `tests/conftest.py` — shared helpers: `make_settings`, `make_cc_config`, `make_hooks_dir`, `wrapped`, `ctx_hook`, `hardgate_entry`, `HARDGATE_CONTENT`, `ALL_CTX_MATCHERS`
- `tests/test_verify.py` — verifier logic: all six check functions, `_command_str` string handling, `run()` exit codes
- `tests/test_install.py` — installer helpers: `_run()`, idempotency checks, `_is_context_mode_install_js()`, `_find_context_mode_dir()`, `backup_configs()`, `restore_configs()`, `--verify` flag

All adversarial cases are covered: malformed JSON, nonexistent binaries (returns 127), HOOKS_DIR pointing to a file, install.js without matching content, depth-4 directories beyond the search limit.

---

9. Known Limitations

- **Verifier checks config presence, not MCP liveness.** Run `longhand doctor` and `claude mcp list` after install to verify runtime health.
- **Hardgate idempotency is heuristic.** The installer cannot authoritatively determine if Hardgate is correctly configured without running `/hard-gate` again. A false positive (script exists + wired = complete) is possible if the script is misconfigured.
- **Backup files accumulate.** Each run creates a new timestamped backup. There is no automatic cleanup. Delete old backups from `~/.claude/` manually if needed.
- `importlib` **dynamic loading for verify.py.** The import is non-standard and invisible to static analysers. This is a known trade-off for the flat repo layout.
- **Skill path is untested.** `skills/stack.md` bash snippets are not covered by automated tests. The Python install path is fully tested; the skill path relies on manual verification.

---

10. Changelog Summary

Version	Date	Summary
v1.1.2	2026-04-16	Bug fixes: `__version__` constant, `_command_str` string entries, `print_report` condition clarity, content validation for `_find_context_mode_dir`, exclusion list, broadened exception handling. 16 new tests (59→75). conftest.py shared fixtures.
v1.1.1	2026-04-16	Installer UX: quiet subprocess output, Hardgate idempotency, neutral Hardgate instructions, Context-Mode prereq prompt, platform-specific Node.js hints, Claude Code CLI offer. 30 new tests (29→59).
v1.1.0	2026-04-16	Added `install.py` standalone installer, `install.bat`, `install.sh`, `--verify` flag, cross-platform Context-Mode discovery, ANSI colour output.
v1.0.0	2026-04-16	Initial release: `/stack` skill, `scripts/verify.py`, timestamped backups, per-tool idempotency, 29 tests.