

The first three protective rules

Three guards your coding agent never decides alone again. Ready to copy for Claude Code, installed in twenty minutes, with a test.

What you get

1. **prod-write-guard.sh** blocks every write to production data before the command runs.
2. **stateful-kill-guard.sh** blocks kill signals against processes your own tools depend on.
3. **send-dryrun-guard.sh** blocks automated sending to real people without a one-week dry run.

Each rule comes from a real incident. The three categories cover the mistakes with the highest consequences and the highest chance of repeating. They will not stop everything. But they turn three experiences into three permanent capabilities, and once these limits hold, the system may receive the next degree of freedom.

The difference to a code review in one sentence. A review catches the mistake after it has landed. A guard catches it before the command runs.

How to read the three pages

Every rule has the same shape. The scene it came from. The rule in one sentence. The script that enforces it. At the end, one page for wiring and for the test that proves to you the block fires.

Requirements: Linux or macOS with Bash, Claude Code (or any tool with pre-tool hooks), optionally `jq`. Every script also accepts the command as its first argument, so you can test without an agent.

1 Data integrity

No writes to production data

The scene. June 26, 2026. During testing, a customer's password hash in a production database gets overwritten, executed as a “quick fix” outside any process. Restored from a two-week-old backup. The most expensive mistake in the whole system.

The rule. Every write to customer data in production is blocked, regardless of the stated intent. Tests run against a database that is structurally impossible to confuse with production. Grepping for the word “prod” is not enough. The target identity comes from a configuration file, and ambiguous means block.

prod-write-guard.sh

```
#!/usr/bin/env bash
# prod-write-guard.sh
# Blocks every write to production data before the command runs.
# Claude Code PreToolUse hook (matcher: Bash). Exit 2 = block.
# Target identities come from configuration, not from the word "prod" in the command.
set -euo pipefail

CONF="${GUARD_CONF:-.guardrail}"
PROD_FILE="$CONF/prod-targets" # one identity per line: db.prod.example.com, club_live
TEST_FILE="$CONF/test-targets" # one identity per line: localhost, db.test, _test

CMD="${1:-}"
if [[ -z "$CMD" ]] && command -v jq >/dev/null 2>&1; then
    CMD=$(jq -r '.tool_input.command // empty' 2>/dev/null || true)
fi
[[ -z "$CMD" ]] && exit 0

# Only look at SQL clients and ORM tools
if ! grep -qiE '^(^|[:space:];&|)(psql|mysql|mariadb|sqlite3|prisma|sequelize|knex|rails[:space:])+db|alembic|'
    ([:space:]]|$)' <<<"$CMD"; then
    exit 0
fi

# Is it a write?
if ! grep -qiE '\b(UPDATE|DELETE|INSERT|TRUNCATE|DROP|ALTER)\b|db:seed|db:reset|--force' <<<"$CMD"; then
    exit 0
fi

matches_list() {
    local file="$1" line
    [[ -f "$file" ]] || return 1
    while IFS= read -r line; do
        [[ -z "$line" || "$line" = \#* ]] && continue
        grep -qF -- "$line" <<<"$CMD" && return 0
    done <"$file"
    return 1
}

if matches_list "$PROD_FILE"; then
    echo "BLOCKED prod-write-guard: write against a production identity listed in $PROD_FILE. Production data changes go through the reviewed migration or the admin workflow only." >&2
    exit 2
fi

if matches_list "$TEST_FILE"; then
    exit 0
fi

echo "BLOCKED prod-write-guard: database target is ambiguous (neither in $PROD_FILE nor in $TEST_FILE). Ambiguous means block. Add the target to one of the two lists." >&2
exit 2
```

Operations

2 No kill signals against stateful services

The scene. An agent wants to “quickly” end a hanging process. The process held the database connection the agent itself depended on. The server did not come back, and the tool that could have repaired it was gone with it.

The rule. No kill signal as a shortcut against anything that maintains a connection your tools depend on. Database, MCP server, proxy, long-lived services. Restarts go through the service's own mechanism, never through `kill` or `docker stop` on a hunch.

stateful-kill-guard.sh

```
#!/usr/bin/env bash
# stateful-kill-guard.sh
# Blocks kill signals and hard stops against processes that hold a stateful
# connection (database, MCP server, proxy). Restarts go through the service's own
# mechanism. Claude Code PreToolUse hook (matcher: Bash). Exit 2 = block.
set -euo pipefail

CONF="${GUARD_CONF:-.guardrail}"
PROTECTED_FILE="$CONF/protected-services" # one name per line: postgres, redis, mcp-, traefik

CMD="${1:-}"
if [[ -z "$CMD" ]] && command -v jq >/dev/null 2>&1; then
    CMD=$(jq -r '.tool_input.command // empty' 2>/dev/null || true)
fi
[[ -z "$CMD" ]] && exit 0

# Only look at kill and stop commands
if ! grep -qiE '([[:space:]]&)(kill|pkill|killall|docker[[:space:]]+(kill|stop|rm)|docker[[:space:]]+compose[[:space:]]+(down|kill)|systemctl[[:space:]]+(stop|kill))([[:space:]]|$)' <<<"$CMD"; then
    exit 0
fi

PROTECTED_DEFAULT='postgres\nredis\nmcp-\ntraefik\ncoolify-proxy'
LIST="$PROTECTED_DEFAULT"
[[ -f "$PROTECTED_FILE" ]] && LIST=$(grep -vE '^(#|$)' "$PROTECTED_FILE" || true)

while IFS= read -r name; do
    [[ -z "$name" ]] && continue
    if grep -qiF -- "$name" <<<"$CMD"; then
        echo "BLOCKED stateful-kill-guard: '$name' holds a connection your tools depend on. No kill signal as a shortcut.
        Use the service's restart mechanism (docker restart, systemctl restart, or the watchdog)." >&2
        exit 2
    fi
done <<<"$LIST"

# kill -9 on unknown PIDs: warn only
if grep -qE '([[:space:]]kill[[:space:]]+-9([[:space:]]|$))' <<<"$CMD"; then
    echo "NOTE stateful-kill-guard: kill -9 without a service name. Check with ps what is behind the PID first." >&2
fi
exit 0
```

Communication

3 No sending to people without a dry run

The scene. July 24, 2026. An outreach pipeline sends ten follow-up emails on the same day because its sister function had drifted out of sync. Every single email was well written. Together they were embarrassing.

The rule. Automated sending to real people (email, SMS, push) only after a mandatory dry run, long enough to catch a pipeline that has lost its rhythm. From experience at least one week, not one day. An explicit `--dry-run` is always allowed.

send-dryrun-guard.sh

```
#!/usr/bin/env bash
# send-dryrun-guard.sh
# Blocks automated sending to real people (email, SMS, push) until the pipeline
# has completed a dry run of at least DRY_RUN_DAYS days.
# Claude Code PreToolUse hook (matcher: Bash). Exit 2 = block.
set -euo pipefail

CONF="${GUARD_CONF:-.guardrail}"
MARKER="$CONF/dry-run-started" # created when the dry run starts (touch)
DRY_RUN_DAYS="${DRY_RUN_DAYS:-7}"

CMD="${1:-}"
if [[ -z "$CMD" ]] && command -v jq >/dev/null 2>&1; then
    CMD=$(jq -r '.tool_input.command // empty' 2>/dev/null || true)
fi
[[ -z "$CMD" ]] && exit 0

# Sending paths: local mail tools, known delivery APIs, own scripts named send/notify
SEND_RE='(^|[:space:];&|)(sendmail|mail|mailx|mutt|swaks)([:space:]]|$)|api\.(sendgrid|mailgun|resend|postmarkapp|brevo|twilio|messagebird)\.|(send|notify|broadcast|campaign)[a-z_-]*\.(sh|py|js|ts)([:space:]]|$)'
if ! grep -qiE "$SEND_RE" <<<"$CMD"; then
    exit 0
fi

# An explicit dry run is always allowed
if grep -qiE -- '--dry-run|DRY_RUN=1|--test-only' <<<"$CMD"; then
    exit 0
fi

if [[ ! -f "$MARKER" ]]; then
    echo "BLOCKED send-dryrun-guard: no dry run started. Create it with 'touch $MARKER' and let the pipeline run $DRY_RUN_DAYS days with --dry-run before it reaches real people." >&2
    exit 2
fi

AGE_DAYS=$(( ( $(date +%s) - $(stat -c %Y "$MARKER") ) / 86400 ))
if (( AGE_DAYS < DRY_RUN_DAYS )); then
    echo "BLOCKED send-dryrun-guard: dry run has been running for $AGE_DAYS day(s), required are $DRY_RUN_DAYS. Real sending is not released yet." >&2
    exit 2
fi
exit 0
```

Wiring in five minutes, testing in two

Put the three scripts under `.guardrail/` in your project, make them executable, and bind them in `.claude/settings.json` to the hook point before execution. `PreToolUse` means before execution, not after. The guard receives the proposed command, checks, and blocks with exit code 2 and a message the agent reads.

`.claude/settings.json`

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          { "type": "command", "command":
            ".guardrail/prod-write-guard.sh" },
          { "type": "command", "command":
            ".guardrail/stateful-kill-guard.sh" },
          { "type": "command", "command":
            ".guardrail/send-dryrun-guard.sh" }
        ]
      }
    ]
  }
}
```

Configuration

```
# .guardrail/prod-targets    (one identity per line)
db.prod.example.com
club_live
DATABASE_URL_PROD

# .guardrail/test-targets
localhost
db.test
_test

# .guardrail/protected-services
postgres
redis
mcp-
traefik
```

Your first block

Prove it to yourself, only with test strings in a throwaway directory, never against a real system. Every guard also takes the command as its first argument.

Fixtures

```
fixture: psql -h db.prod.example.com -c "UPDATE profiles SET ..." → BLOCK (exit 2)
fixture: psql -h localhost -d golf_test -c "DELETE FROM users" → ALLOW (exit 0)
fixture: docker stop postgres-main → BLOCK
fixture: docker restart postgres-main → ALLOW
fixture: ./scripts/send-followups.sh → BLOCK (no dry run)
fixture: ./scripts/send-followups.sh --dry-run → ALLOW
```

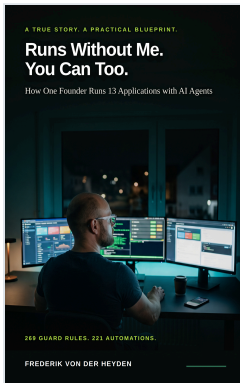
What you should see in the terminal

```
BLOCKED prod-write-guard: write against a production identity listed in .guardrail/prod-targets.
Production data changes go through the reviewed migration or the admin workflow only.
```

This one line is the whole promise. The agent tried the dangerous thing, something outside its judgment stopped it before it landed. False positives are bugs in the guard, not a reason to remove it. The condition gets narrower, the guard stays.

What comes next

Three guards are Day 1. The book shows the remaining 29 days. Hooks, skills and memory in the first week, watchdog, learning loop and a map of the system by the end of the first month.



Runs Without Me. You Can Too.

How one founder runs 13 applications with AI agents, and what that means for all of us. Paperback and Kindle.

amazon.com/dp/BOHDMVKRMG

GuardRail, the free starter set

About 20 guards for the failure modes that show up first and most often, MIT licensed, with an installer for Claude Code.

guardrail.promptandbuild.de

Before you move on

- ☐ Written down the five actions that would really hurt if they happened unattended.
- ☐ Each of them has a guard, triggered and observed, not just written.
- ☐ The configuration files contain your real targets, not the examples.

© 2026 Frederik von der Heyden. Text by Frederik von der Heyden, typesetting automated. From the book "Runs Without Me. You Can Too." (paperback and Kindle, amazon.com/dp/BOHDMVKRMG). Scripts under MIT license, no warranty, test in a throwaway project before use.