

Die ersten drei Schutzregeln

Drei Guards, die dein Coding-Agent nie wieder allein entscheidet.
Kopierfertig für Claude Code, in zwanzig Minuten eingebaut, mit Test.

Was du hier bekommst

1. **prod-write-guard.sh** blockiert jeden Schreibzugriff auf Produktionsdaten, bevor der Befehl läuft.
2. **stateful-kill-guard.sh** blockiert Kill-Signale auf Prozesse, von denen deine Werkzeuge selbst abhängen.
3. **send-dryrun-guard.sh** blockiert automatisiertes Senden an echte Menschen ohne Trockenlauf von einer Woche.

Jede Regel geht auf einen echten Vorfall zurück. Die drei Kategorien decken die Fehler mit den höchsten Konsequenzen und der höchsten Wiederholungswahrscheinlichkeit ab. Sie stoppen nicht alles. Aber sie verwandeln drei Erfahrungen in drei dauerhafte Fähigkeiten, und sobald diese Grenzen halten, darf das System den nächsten Freiheitsgrad bekommen.

Der Unterschied zu einem Code Review in einem Satz. Ein Review fängt den Fehler, nachdem er gelandet ist. Ein Guard fängt ihn, bevor der Befehl läuft.

So liest du die drei Seiten

Jede Regel hat dieselbe Form. Die Szene, in der sie entstanden ist. Die Regel in einem Satz. Das Skript, das sie durchsetzt. Am Ende eine Seite für den Einbau und den Test, mit dem du dir selbst beweist, dass der Block feuert.

Voraussetzungen: Linux oder macOS mit Bash, Claude Code (oder ein Werkzeug mit Pre-Tool-Hooks), optional `jq`. Alle Skripte lassen sich auch mit einem Befehl als Argument testen, ohne Agent.

Datenintegrität

1

Kein Schreibzugriff auf Produktionsdaten

Die Szene. 26. Juni 2026. Beim Testen wird der Passwort-Hash eines Kunden in einer Produktionsdatenbank überschrieben, ausgeführt als „Quick Fix“ außerhalb jedes Prozesses. Wiederhergestellt aus einem zwei Wochen alten Backup. Der teuerste Fehler im ganzen System.

Die Regel. Jeder Write auf Kundendaten in Produktion wird blockiert, unabhängig von der behaupteten Absicht. Tests laufen gegen eine Datenbank, die strukturell nicht mit Produktion verwechselbar ist. Ein Suchmuster nach dem Wort „prod“ reicht nicht. Die Zielidentität kommt aus einer Konfigurationsdatei, und unklar heißt blockieren.

```
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
```

Betrieb

2 Kein Kill-Signal auf zustandsbehaftete Dienste

Die Szene. Ein Agent will einen hängenden Prozess „schnell“ beenden. Der Prozess hielt die Datenbankverbindung, von der der Agent selbst abhing. Der Server kam nicht zurück, und das Werkzeug, das ihn hätte reparieren können, war mit ihm weg.

Die Regel. Kein Kill-Signal als Abkürzung auf alles, das eine Verbindung aufrechterhält, von der deine Werkzeuge abhängen. Datenbank, MCP-Server, Proxy, langlebige Dienste. Neustarts laufen über den eigenen Mechanismus des Dienstes, niemals über `kill` oder `docker stop` auf gut Glück.

```
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 -qiE '^[[: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
```

Kommunikation

3 Kein Versand an Menschen ohne Trockenlauf

Die Szene. 24. Juli 2026. Eine Outreach-Pipeline verschickt zehn Follow-Up-E-Mails am selben Tag, weil ihre Schwesterfunktion aus der Synchronisation geraten war. Jede einzelne Mail war korrekt formuliert. Zusammen waren sie peinlich.

Die Regel. Automatisiertes Senden an echte Menschen (E-Mail, SMS, Push) nur nach einem obligatorischen Trockenlauf, lang genug, um eine Pipeline zu fangen, die aus dem Takt geraten ist. Aus Erfahrung mindestens eine Woche, nicht ein Tag. Ein ausdrücklicher `--dry-run` ist immer erlaubt.

```
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:]]|$)|api\.(sendgrid|mailgun|resend|postmarkapp|brevolt|twilio|messagebird)\.|(send|notify|broadcast|campaign)[a-z_-]*\.(sh|py|js|ts)([:space:]]|$)'
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
```

Einbau in fünf Minuten, Test in zwei

Lege die drei Skripte unter `.guardrail/` in dein Projekt, mach sie ausführbar, und binde sie in `.claude/settings.json` an den Hook-Punkt vor der Ausführung.

`PreToolUse` heißt vor der Ausführung, nicht danach. Der Guard bekommt den vorgeschlagenen Befehl, prüft, und blockiert mit Exit-Code 2 und einer Nachricht, die der Agent liest.

`.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" }
        ]
      }
    ]
  }
}
```

Konfiguration

```
# .guardrail/prod-targets  (eine Kennung pro Zeile)
db.prod.example.com
club_live
DATABASE_URL_PROD

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

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

Dein erster Block

Beweise es dir selbst, ausschließlich mit Test-Strings in einem Wegwerf-Verzeichnis, nie gegen ein echtes System. Jeder Guard nimmt den Befehl auch als erstes 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 (kein Trockenlauf)
fixture: ./scripts/send-followups.sh --dry-run → ALLOW
```

Was du im Terminal sehen solltest

```
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.
```

Diese eine Zeile ist das ganze Versprechen. Der Agent hat das Gefährliche versucht, etwas außerhalb seines Urteils hat es gestoppt, bevor es landete. Fehlalarme sind Fehler im Guard, kein Grund, ihn zu entfernen. Die Bedingung wird enger, der Guard bleibt.

Was als Nächstes kommt

Drei Guards sind Tag 1. Das Buch zeigt die restlichen 29 Tage. Hooks, Skills und Memory in der ersten Woche, Watchdog, Lernschleife und Systemkarte bis zum Ende des ersten Monats.



Läuft ohne mich. Du kannst das auch.

Wie ein Solo-Founder mit KI-Agenten 13 Anwendungen betreibt, und was das für uns alle bedeutet. Taschenbuch und Kindle.

amazon.de/dp/BOHDMT162J

GuardRail, das kostenlose Starter-Set

Etwa 20 Guards für die Fehlermodi, die zuerst und am häufigsten auftreten, MIT-lizenziert, mit Installer für Claude Code.

guardrail.promptandbuild.de

Bevor du weitermachst

- ☐ Die fünf Aktionen aufgeschrieben, die wirklich wehtun würden, wenn sie unbeaufsichtigt passieren.
- ☐ Jede davon hat einen Guard, ausgelöst und beobachtet, nicht nur geschrieben.
- ☐ Die Konfigurationsdateien enthalten deine echten Ziele, nicht die Beispiele.

© 2026 Frederik von der Heyden. Text von Frederik von der Heyden, Satz automatisiert. Aus dem Buch „Läuft ohne mich. Du kannst das auch.“ (Taschenbuch und Kindle, amazon.de/dp/BOHDMT162J). Skripte unter MIT-Lizenz, ohne Gewähr, vor dem Einsatz in einem Wegwerfprojekt testen.