

Mein Claude-Code-Setup

Damit dein Agent nie den Faden verliert — und nichts Kaputtes in deinem Code landet.

Die meisten nutzen Claude Code wie einen Chatbot: Frage rein, Antwort raus. Für echte Software reicht das nicht. Hier ist die **orchestrierte Werkzeugkette**, mit der ich arbeite — fünf Bausteine, die du dir einzeln anschauen kannst.

1 Hooks — die Umgebung feuert automatisch, nicht die KI

Der wichtigste Denkfehler: zu glauben, die KI müsse selbst dran denken. Tut sie nicht. **Hooks** hängen an Ereignissen und feuern automatisch — egal, was das Modell gerade „will“.

EVENT	WAS AUTOMATISCH PASSIERT
Vor jedem Command / Edit	Prüf-Gate: Darf das laufen? Ist es vorbereitet?
Nach Edit / Command	Nachbereitung: Formatierung, Muster lernen
Prompt abgeschickt	Task-Routing: der passende Subagent wird vorgeschlagen
Session-Start	Session wiederherstellen + Memory automatisch laden
Session-Ende	Auto-Snapshot ins Protokoll + Memory-Sync
Vor dem Komprimieren	Kontext sichern, bevor etwas verloren geht
Subagent Start/Stop	Status + Muster-Training pro Subagent

Der wertvollste eigene Hook: ein *Stop-Hook*, der bei jedem Session-Ende automatisch einen Snapshot an eine Tagesdatei anhängt (docs/protocol/YYYY-MM-DD.md) — idempotent, also kein Spam, wenn sich nichts geändert hat. Bei Tageswechsel: neue Datei, im Index verlinkt.

Zweiter Trick — das Code-Discovery-Gate: ein Hook auf Grep / Glob , der die KI zwingt, **erst den Codebase-Graph zu fragen** statt blinde Grep-Kaskaden zu fahren (siehe Punkt 4).

Zum Mitnehmen: Alles, woran du dich sonst „erinnern“ müsstest, gehört in einen Hook. Die Umgebung ist verlässlicher als jeder gute Vorsatz.

2 Dreifach abgesicherter Handoff — nichts geht über Sessions verloren

Drei Ebenen, klar getrennt nach Funktion:

1. **Memory** = *was dauerhaft gilt*. Persistente Fakten & Entscheidungen in einer `MEMORY.md`, die bei Session-Start automatisch geladen wird.
2. **Handoff-Doku** = *wo genau wir weitermachen*. Pro Arbeitsblock ein „Für die nächste Session“-Dokument: Stand, offene Punkte, Stolpersteine, Entscheidungen. Das wichtigste per Memory-Zeile als „**zuerst lesen**“ markiert.
3. **Protokoll** = *was ist wann passiert*. Fortlaufende History — teils manuell nach Template, teils automatisch vom Stop-Hook.

Merksatz: Memory = *was gilt*, Handoff = *wo weiter*, Protokoll = *was war*.

Und: ein sauberer **Handoff schlägt jeden** `/compact` — Compact schleppt den alten Müll mit, ein Handoff startet sauber.

3 Double Review — kein Merge ohne zwei unabhängige Reviewer

Auch bei trivial wirkenden Änderungen. Zwei Pässe:

- **Pass 1:** ein hauseigener `reviewer`-Subagent (Claude).
- **Pass 2:** eine zweite Meinung über ein **zweites Modell** (z. B. OpenAI Codex), `codex review`.
- **Blast-Radius zuerst:** der Codebase-Graph liefert, welche Symbole der Diff berührt — beide Reviewer wissen, was betroffen ist.
- **Beide haben Veto:** ein KRITISCH/HIGH-Fund von einem = kein Merge. Nur LOW darf ins Backlog.
- **Getrennt dokumentiert:** beide Ergebnisse als separate PR-Kommentare.

Warum so streng? Selbst reine Bugfix-Sprints haben schon echte Lücken offenbart (Session-Verwechslung, offene Buckets, fehlende Ownership-Prüfung). Zwei unabhängige Modelle fangen mehr als eines.

4 Codebase-Graph statt Grep

Die Codebase ist als **Knowledge Graph** indexiert. Vor jeder Suche wird der Graph gefragt, nicht blind gegreppt:

- `search_graph` — Symbole/Beziehungen finden statt Textstellen
- `trace_path` — Call-Chains über Dateigrenzen sichtbar machen
- `get_architecture` — Struktur auf einen Blick
- `detect_changes` — Blast-Radius eines Diffs (füttert die Reviewer aus Punkt 3)

Effekt: massiv weniger Kontext-Verschwendung, keine übersehenen Abhängigkeiten. Der Agent rät nicht mehr, er *sieht*.

5 Spezialisierte Subagenten + menschliches Gate

- **Subagenten statt Alleskönner** — eigene Agenten für Review, Recherche, Planung; ein Router-Hook schlägt den passenden vor.
 - **Zwei-Modell-Prinzip** — Claude und ein zweites Modell reviewen gegeneinander.
 - **Persistentes Lernen** — Memory-Files + Muster-Training über Sessions.
 - **Der Mensch bleibt das Gate** — Plan **vor** Code, Bestätigung vor riskanten Aktionen. Prod-Datenbank, geschützter Haupt-Branch und Merges nur mit OK. Kein Force-Push, nie.
-

✓ So fängst du an

- ☐ Eine `MEMORY.md`, die bei Session-Start automatisch geladen wird
- ☐ Ein Stop-Hook, der bei Session-Ende ein Protokoll anhängt (idempotent)
- ☐ Ein Handoff-Muster: Stand · offene Punkte · Stolpersteine · Entscheidungen
- ☐ Ein zweites Modell als Gegen-Reviewer (beide mit Veto)
- ☐ Codebase-Graph-Tooling statt Grep-Reflex
- ☐ Klare menschliche Gates: Prod-DB & Haupt-Branch nur mit OK

Wenn dir das geholfen hat: Ich zeige diese Workflows regelmäßig — folg mir für Claude-Code-Setups, die auch unter echter Last halten.