CLAUDE CODE

Loops bauen, die von allein durchlaufen

Die fünf Bausteine aus dem Reel und ein Loop zum Nachbauen

5	1 Ziel	0
Bausteine	und der Rest läuft	Prompts nebenher

Ein Loop ist einfach ein Ablauf, den du einmal einrichtest und der danach von allein weiterläuft. Du gibst ihm ein Ziel, und ab da sucht er sich die Aufgabe, erledigt sie, prüft das Ergebnis und fängt von vorne an, bis das Ziel erreicht ist. Du sitzt also nicht mehr daneben und schreibst mit der KI hin und her.

WAS DU DANACH KANNST

01 **Claude von allein anspringen lassen**

Du hängst kein Startsignal mehr selber rein, sondern der Loop läuft nach Zeit oder nach Zustand weiter.

02 **Mehrere Agents parallel laufen lassen**

Jeder Agent bekommt seine eigene Arbeitskopie und dann zerschießen sie sich halt nicht mehr gegenseitig die Dateien.

03 **Trennen, wer arbeitet und wer prüft**

Der Subagent, der die Aufgabe umsetzt, ist nicht derselbe, der das Ergebnis am Ende abnimmt.

DAS PRINZIP

Wie ein Loop abläuft

Fünf Schritte, und der letzte springt zurück auf den zweiten

01 **Ziel setzen**

Du sagst einmal, wie der Endzustand aussieht. Also nicht "arbeite an den Tests", sondern "alle Tests in test/auth laufen grün durch und der Linter meckert nicht mehr".

02 **Aufgabe wählen**

Claude schaut sich den aktuellen Stand an und sucht sich selber den nächsten Punkt, der dem Ziel am nächsten bringt.

03 **Umsetzen**

Der Punkt wird abgearbeitet. Hier läuft die eigentliche Arbeit, also Dateien ändern, Befehle ausführen und Tools bedienen.

04 **Prüfen**

Danach wird kontrolliert, ob es wirklich besser geworden ist. Und das muss eine Prüfung sein, die man nachrechnen kann, also Tests, Linter oder ein Build.

05 **Stop-Kriterium oder zurück auf 02**

Ist das Ziel erreicht, hört der Loop auf. Ist es nicht erreichbar, hört er auch auf und sagt warum. Sonst geht es wieder bei der Aufgabenwahl los.

Ohne Stop-Kriterium ist es kein Loop, sondern eine Endlosschleife.
Das Ziel muss also so formuliert sein, dass eine Maschine es prüfen kann. "Der Code ist schöner" geht nicht, "npm test läuft ohne Fehler durch" geht.

BAUSTEIN 1

Damit er von allein anspringt

Vier Wege, und du brauchst nicht alle

01 `/loop` **nach Zeit**

Läuft den Prompt immer wieder in einem festen Abstand. Die Einheiten sind s, m, h und d. Ohne Zeitangabe sucht Claude sich den Abstand selber aus.

02 `/goal` **nach Zustand**

Das ist der ehrlichere Loop für Arbeit am Projekt. Du gibst den Endzustand rein und nach jedem Durchgang prüft ein Bewerter, ob er erreicht ist. Erreicht oder unmöglich, dann stoppt er von allein.

03 **Hooks für Ereignisse**

Hooks hängen an Ereignissen und nicht an der Uhr. Also immer dann, wenn eine Datei geschrieben wurde oder eine Session endet. Konfiguriert wird das in den Settings unter `.claude`.

04 **Cron von außen**

Wenn der Loop auch laufen soll, während bei dir gar keine Session offen ist, dann startest du Claude einfach per cron im Headless-Modus mit `claude -p`.

SO SIEHT DAS AUS

```
# im Chat, nach Zeit
/loop 15m pruef die offenen PRs und fix rote CI-Builds

# im Chat, nach Zustand (stoppt von allein)
/goal alle Tests in test/auth laufen gruen und der Linter ist sauber

# von aussen, per crontab -e, jeden Werktag um 9 Uhr
0 9 * * 1-5 cd /pfad/zum/projekt && claude -p "pruef den Build" \
  --allowedTools "Bash,Read"
```

Es gibt in Claude Code kein Feature, das "Automations" heißt.
Gemeint sind also `/loop` und `/goal` im Chat, Hooks für Ereignisse und ganz normales cron von außen.

BAUSTEIN 2

Getrennte Arbeitskopien

Damit sich zwei Agents nicht dieselbe Datei kaputt machen

01 Das Problem

Zwei Agents im selben Ordner schreiben halt in dieselben Dateien. Der eine speichert, der andere überschreibt, und am Ende weißt du nicht mehr, welcher Stand jetzt richtig ist.

02 Die Lösung heißt git worktree

Ein worktree ist ein zweiter Checkout desselben Repos in einem eigenen Ordner, mit eigenem Branch. Die Historie bleibt dieselbe, die Dateien sind getrennt.

03 Claude Code macht das für dich

Mit `claude --worktree` legt Claude die Arbeitskopie selber an, unter `.claude/worktrees/`. Du kannst einen Namen mitgeben oder eine PR-Nummer.

04 Dateien mitnehmen, die nicht im Git sind

Deine `.env` liegt ja nicht im Repo und fehlt dann in der Kopie. Dafür legst du eine `.worktreeinclude` an und schreibst rein, was mitkopiert werden soll.

SO SIEHT DAS AUS

```
# Claude legt die Arbeitskopie selber an
claude --worktree feature-auth

# oder du machst es von Hand
git worktree add .claude/worktrees/feature-auth -b feature-auth

# .worktreeinclude, damit lokale Dateien mitkommen
.env
.env.local

# hinterher wieder aufräumen
git worktree remove .claude/worktrees/feature-auth
```

Trag `.claude/worktrees/` in deine `.gitignore` ein.
Sonst landen die Arbeitskopien im Repo und das will man einfach nicht.

BAUSTEIN 3

Skills, damit du nichts neu erklärst

Projektwissen einmal hinschreiben statt in jedem Prompt

01 **Wo ein Skill liegt**

Ein Skill ist ein Ordner unter `.claude/skills/` mit einer `SKILL.md` drin. Fürs Projekt legst du ihn im Projekt an, für dich selber unter `~/.claude/skills/`.

02 **Was oben reinmuss**

Im Kopf der Datei stehen `name` und `description`. Die description ist das Wichtigste, denn danach entscheidet Claude, wann der Skill überhaupt gezogen wird.

03 **Optionale Felder**

Mit `allowed-tools` schränkst du ein, was der Skill anfassen darf. Mit `disable-model-invocation: true` sagst du, dass Claude ihn nicht von allein zieht, sondern nur wenn du ihn mit `/name` aufrufst.

04 **Was da reingehört**

Also alles, was du sonst jedes Mal tippst. Wie euer Code aufgebaut ist, welche Befehle es gibt, was verboten ist und wie ein fertiges Ergebnis auszusehen hat.

.CLAUDE/SKILLS/PROJEKT-REGELN/SKILL.MD

```
---
name: projekt-regeln
description: Regeln und Befehle fuer dieses Projekt. Immer nutzen, wenn Code
  geaendert oder geprueft wird.
allowed-tools: Read, Grep, Glob, Bash
---

# Projekt-Regeln

- Tests laufen mit `npm test`, Linter mit `npm run lint`.
- Nichts direkt auf main committen, immer ein Branch.
- Migrationen liegen unter db/migrations und werden nie nachtraeglich geaendert.
- Fertig heisst: Tests gruen, Linter sauber, CHANGELOG ergaenzt.
```

BAUSTEIN 4

Die Verbindung zu deinen Tools

MCP, damit er sich wirklich selbst bedienen kann

01 Was MCP macht

Ein MCP-Server hängt ein Tool an Claude dran, also zum Beispiel deine Datenbank, GitHub oder ein internes System. Ohne das kann er halt nur Dateien und Terminal.

02 Server anhängen

Das läuft über `claude mcp add`. Bei einem lokalen Programm nimmst du `--transport stdio`, bei einem Dienst im Netz `--transport http`.

03 Wo die Einstellung landet

Mit `--scope project` landet der Server in der `.mcp.json` im Projekt und ist damit auch für alle anderen da, die das Repo klonen. Mit `--scope user` bleibt er bei dir.

04 Nachschauen, was dranhängt

`claude mcp list` zeigt dir alle Server, `claude mcp get name` die Details zu einem, und im Chat siehst du mit `/mcp` den Status.

SO SIEHT DAS AUS

```
# lokales Programm anhaengen, alles nach -- geht an den Server
claude mcp add --transport stdio --scope project db \
  -- npx @bytebase/dbhub --dsn "$DATABASE_URL"

# Dienst im Netz
claude mcp add --transport http --scope project github \
  https://api.githubcopilot.com/mcp/ \
  --header "Authorization: Bearer $GITHUB_TOKEN"

# kontrollieren
claude mcp list
```

Schreib keine Zugangsdaten fest in die `.mcp.json`.
Die Datei liegt ja im Repo. Nimm also eine Umgebungsvariable und lass den Wert von außen kommen.

BAUSTEIN 5

Subagents, also Arbeiter und Prüfer

Wer die Arbeit macht, nimmt sie nicht selber ab

01 Warum trennen

Wer gerade zwei Stunden an einer Sache gebaut hat, findet die Fehler darin einfach schlecht. Bei Claude ist es genauso, weil der ganze Kontext von der Umsetzung noch mit drinhängt.

02 Wo Subagents liegen

Eine Datei pro Agent unter `.claude/agents/`, also zum Beispiel `umsetzer.md` und `pruefer.md`. Oben stehen `name` und `description`, der Rest ist die Anweisung.

03 Dem Prüfer die Hände binden

Über `tools` gibst du dem Prüfer nur lesende Werkzeuge. Dann kann er halt nichts schnell selber geradebiegen und muss sein Urteil wirklich hinschreiben.

04 Parallel arbeiten lassen

Mit `isolation: worktree` bekommt ein Subagent seine eigene Arbeitskopie. Das ist genau der Punkt aus Baustein 2 und deswegen greifen die beiden gut ineinander.

.CLAUDE/AGENTS/PRUEFER.MD

```
---
name: pruefer
description: Nimmt fertige Aenderungen ab. Immer nutzen, bevor ein Schritt im
  Loop als erledigt gilt. Aendert selbst nichts.
tools: Read, Grep, Glob, Bash
model: sonnet
---

Du pruefst die Aenderung gegen das Ziel und aenderst selbst keine Datei.

1. Lies das Ziel aus ZIEL.md und den Diff.
2. Fuehr `npm test` und `npm run lint` aus.
3. Antworte mit genau einer Zeile: OK oder NACHARBEITEN plus Begruendung.
```

ZUM NACHBAUEN

Der erste Loop in 30 Minuten

Ein kleines Ziel, ein Prüfer und eine harte Stop-Bedingung

01 **Ziel aufschreiben**

Leg im Projekt eine `ZIEL.md` an und schreib den Endzustand rein, so dass man ihn nachmessen kann. Zum Beispiel, dass alle Tests unter test/auth grün laufen.

02 **Wissen einmal hinterlegen**

Dann den Skill aus Baustein 3 anlegen, mit den Befehlen und den Regeln von deinem Projekt. Das spart dir in jedem Durchgang die Erklärerei.

03 **Prüfer anlegen**

Die `pruefer.md` von Seite 7 reinlegen. Nur lesende Tools, und er antwortet mit OK oder NACHARBEITEN.

04 **Getrennt arbeiten lassen**

Starte mit `claude --worktree loop-test`, dann läuft der ganze Versuch in einer eigenen Arbeitskopie und dein Hauptordner bleibt sauber.

05 **Loop starten**

Und dann setzt du das Ziel mit `/goal` und gibst die Runde vor. Claude wählt die Aufgabe, setzt sie um, holt den Prüfer und macht weiter, bis der Bewerter das Ziel als erreicht meldet.

DER START

```
claude --worktree loop-test

/goal alle Tests in test/auth laufen gruen durch und npm run lint ist sauber.
Arbeite in Runden: waehl den naechsten offenen Punkt aus ZIEL.md, setz ihn um,
lass danach @pruefer abnehmen und geh erst weiter, wenn der OK sagt.
Nach drei NACHARBEITEN am selben Punkt stoppst du und sagst mir warum.
```

NÄCHSTER SCHRITT

Direkt umsetzen

Die Übersicht und die drei Sachen, die du heute machst

BAUSTEIN	WO ES LIEGT
Von allein anspringen	`/loop` und `/goal` im Chat, Hooks, cron mit `claude -p`
Arbeitskopien	`claude --worktree name`, Ordner `.claude/worktrees/`
Skills	`.claude/skills/name/SKILL.md`
Tool-Verbindung	`claude mcp add`, Projekt-Datei `.mcp.json`
Subagents	`.claude/agents/name.md`

01 **Ein Ziel formulieren**

Nimm eine Sache, die du diese Woche sowieso machen musst, und schreib sie so auf, dass ein Testlauf sagen kann, ob sie fertig ist.

02 **Prüfer anlegen**

Die `pruefer.md` reinkopieren und die zwei Befehle auf dein Projekt anpassen. Das ist der Baustein, der am meisten bringt.

03 **Einmal mit /goal laufen lassen**

Und danach schaust du dir an, wo er falsch abgebogen ist. Genau das kommt dann in den Skill, damit es beim nächsten Durchgang schon drinsteht.

Du willst so was regelmäßig sehen, bevor es alle machen?

Folg @timhuck.ai auf Instagram. Da kommen Setups und Änderungen bei Claude Code, sobald sie draußen sind.

→ **@timhuck.ai auf Instagram**
→ **www.shortmedia.agency**

Angaben nach der offiziellen Claude-Code-Doku, Stand August 2026. Befehle ändern sich, also prüf im Zweifel `claude --help`.