KI-WORKFLOW · 2026

LOOP ENGINEERING

Hör auf zu prompten. Bau Schleifen, die für dich prompten.

Stell dir vor, du gibst deinem Rechner abends ein Ziel — „jeder Blogpost hat eine saubere Meta-Beschreibung" — und am nächsten Morgen ist es **erledigt, geprüft und protokolliert**, ohne dass du einen einzigen Befehl nachgeschoben hast. Genau das baust du in diesem Guide: einen Loop, der die Arbeit selbst findet, erledigt, sich kontrolliert und so lange weitermacht, bis das Ziel wirklich erreicht ist.

5
Schritte pro Loop

2
Befehle als Auslöser

1
Charter zum Kopieren

> *„Ich prompte Claude nicht mehr. Ich habe Loops laufen, die Claude prompten und herausfinden, was zu tun ist. Mein Job ist es, Loops zu schreiben."*
>
> — Boris Cherny, Head of Claude Code bei Anthropic, über seinen eigenen Arbeitsalltag

01

DAS KONZEPT

Was ein Loop wirklich ist

Jahrelang war die wichtigste KI-Fähigkeit, einen guten Prompt zu schreiben. Das Problem: Prompten hält dich im Raum. Du tippst, du wartest, du korrigierst, du tippst wieder. Die ganze Zeit hältst du der KI die Hand. **Ein Loop setzt sich an deine Stelle.** Du übergibst ihm ein Ziel — er findet die Arbeit, erledigt sie, prüft sich selbst, merkt sich, was fertig ist, und macht weiter, bis das Ziel tatsächlich erreicht ist.

Ein Loop (englisch für „Schleife") ist ein System, das die KI für dich anstößt. Egal wie aufwendig — jeder Loop ist derselbe Fünf-Schritt-Zyklus:

1. **Arbeit finden.** Der Loop entdeckt, was zu tun ist: offene Punkte in einer Liste, fehlgeschlagene Tests, unbeantwortete Mails, Einträge in einer Warteschlange.
2. **Arbeit erledigen.** Die KI bearbeitet einen Punkt nach dem anderen — genau so, wie du sie sonst manuell angewiesen hättest.
3. **Sich selbst prüfen.** Ein Kontrollschritt entscheidet, ob die Arbeit wirklich erledigt und korrekt ist — nicht bloß erzeugt.
4. **Sich erinnern.** Der Loop schreibt auf, was fertig ist (eine kleine Status-Datei oder ein Aufgaben-Board), damit er nichts doppelt macht und dort weitermacht, wo er aufgehört hat.
5. **Weitermachen.** Er wiederholt das, bis nichts mehr offen ist — dann stoppt er oder meldet sich bei dir.

Das ist alles. Die Fähigkeit sind nicht größere Prompts. Es ist, diesen Zyklus **einmal** zu entwerfen, damit das Prompten ohne dich passiert.

> **DER DENKWECHSEL**
>
> Prompten ist taktisch: eine Anweisung, ein Ergebnis. Loop Engineering ist strategisch: Du entwirfst das System, das die Anweisungen ausgibt. Es ist der Unterschied zwischen **die Arbeit machen** und **den Arbeiter führen**.

02

DER AUSLÖSER

Die zwei Befehle, die einen Loop wirklich starten

Hier machen die meisten den Fehler: Du baust keinen Loop, indem du „mach das in einer Schleife" in einen normalen Chat tippst. Die echten Loops laufen in **Claude Code** — Anthropics Werkzeug, das direkt in der Kommandozeile (dem Terminal) deines Rechners läuft. Dort gibt es zwei eingebaute Befehle, die den Zyklus von Seite 2 in einen echten Loop verwandeln.

VORAUSSETZUNGEN (EINMALIG)

- **Claude Code installiert.** Das ist ein Programm für die Kommandozeile (das schwarze Eingabefenster auf deinem Rechner). Installation und Schritte: `code.claude.com/docs`.
- **Bezahlter Claude-Plan** (Pro oder Max) oder API-Abrechnung — Loops laufen nicht im kostenlosen Plan.
- **Aktuelle Version.** Siehst du `/goal` oder `/loop` nicht, aktualisiere Claude Code auf die neueste Version.

1 · /GOAL — DER LOOP, DER ARBEITET, BIS ES FERTIG IST

Du tippst `/goal` und dahinter, wie „fertig" konkret aussieht. Claude arbeitet dann Runde für Runde von allein. Das Entscheidende: **Nach jeder Runde prüft ein zweites KI-Modell still**, ob das Ziel schon erreicht ist. Wenn nicht, sagt es Claude warum — und Claude macht weiter. Im Moment, in dem das Ziel wirklich erfüllt ist, stoppt der Loop von selbst. Genau diese Selbstprüfung nach jeder Runde ist der ganze Unterschied zwischen einem echten Loop und einem Prompt, der einmal läuft und hofft.

IN CLAUDE CODE AUSPROBIEREN

```
/goal Jeder Blogpost im Ordner /posts hat eine Meta-Beschreibung
unter 160 Zeichen und einen Titel unter 60 Zeichen. Gib nach
jeder Datei die Zeichenzahl aus, damit ich es bestätigen kann,
und fass den Fließtext nicht an. Stopp, wenn jede Datei besteht,
oder nach 25 Dateien.
```

Warum das funktioniert: Das Ziel ist **messbar** (Zeichenzahl). Der Prüfer erkennt den exakten Moment, in dem es bestanden ist. Claude bearbeitet eine Datei, zählt, sieht „noch zu lang", korrigiert, zählt erneut — und geht erst weiter, wenn die Hürde geschafft ist. Du hast kein einziges Mal „versuch's nochmal" getippt.

2 · /LOOP — DER LOOP, DER SICH VON SELBST WIEDERHOLT

Diesen nimmst du, wenn die Arbeit nicht „einen Stapel abarbeiten", sondern „etwas immer wieder kontrollieren" ist. Du tippst `/loop` mit dem Takt und der Aufgabe, und Claude führt sie für dich erneut aus — nach einem Zeitplan, den du setzt.

IN CLAUDE CODE AUSPROBIEREN

```
/loop 30m Prüfe, ob meine Live-Website wieder erreichbar ist,
indem du die Startseite lädst. Sobald sie eine normale Seite
zurückgibt, sag mir Bescheid und hör auf zu prüfen.
```

`30m` heißt: alle 30 Minuten. Du kannst auch mit einer normalen Anweisung beginnen wie „jeden Morgen meinen Posteingang sortieren" — Claude legt den Zeitplan dann selbst an. Mit der **Esc**-Taste stoppst du einen Loop, der auf seinen nächsten Lauf wartet.

SO MERKST DU DIR, WELCHER WANN

/goal, wenn es eine Ziellinie gibt: arbeite, bis das wahr ist. **/loop**, wenn es keine Ziellinie gibt, nur einen Rhythmus: prüf das immer wieder. Die meisten starken Loops beginnen mit **/goal**.

03

DAS SETUP

Fünf Bausteine, die einen Loop stärker machen

Die zwei Befehle sind der Motor. Diese fünf Bausteine sind die Upgrades, die einen Loop mächtiger und vertrauenswürdiger machen. Du brauchst nicht alle auf einmal — fang mit dem an, was dein Loop wirklich braucht.

1 Automatisierungen

Lässt den Loop nach Zeitplan laufen statt erst, wenn du tippst. Das Einfachste ist der `/loop` -Befehl von Seite 3. Für Loops, die laufen sollen, auch wenn kein Fenster offen ist, startest du Claude im Hintergrund über einen Zeitplaner mit `claude -p "dein Loop-Auftrag"`. So arbeitet der Loop, während du schläfst.

2 Worktrees

Git-Worktrees geben jeder KI eine eigene, isolierte Kopie deines Projekts — so können zwei Loops gleichzeitig arbeiten, ohne sich gegenseitig zu überschreiben. In Claude Code sagst du es direkt: „mach das in einem separaten Worktree." Läufst du immer nur eine KI gleichzeitig, kannst du das anfangs überspringen.

3 Skills

Eine Skill ist eine Textdatei (im Ordner `.claude/skills/`), die das Wissen festhält, das du sonst immer wieder erklärst: wie dein Projekt aufgebaut ist, deine Regeln, deine Formate. Einmal geschrieben, startet jeder Loop-Lauf schon mit Projektwissen. Einfachster Start: sag Claude „mach aus allem, was ich gerade über dieses Projekt erklärt habe, eine Skill-Datei."

4 Connectors

Verbindungen, über die die KI deine echten Werkzeuge erreicht: Mail, Aufgaben-Tracker, Google Drive, Datenbank. Ohne sie arbeitet ein Loop nur an lokalen Dateien. Mit ihnen kann „Arbeit finden" auch „lies meinen Posteingang" heißen. In Claude Code fügst du sie mit `claude mcp add` oder über das Connector-Menü hinzu.

5 Sub-Agenten

Die KI, die die Arbeit macht, sollte nicht die sein, die sie benotet. Claude Code lässt dich getrennte Agenten anlegen (im Ordner `.claude/agents/`): Ein Bau-Agent erledigt die Aufgabe, ein Prüf-Agent kontrolliert sie unabhängig. Dieser eine Schritt macht Loops vertrauenswürdig genug, um unbeaufsichtigt zu laufen.

04

EIN ECHTES BEISPIEL

Das selbstprüfende Recherche-Brief

Der einfachste Weg zu sehen, warum ein Loop einen Prompt schlägt. Sagen wir, du bittest Claude um ein einseitiges Briefing zu einem Thema. Das ist die **Aufgabe**. Das **Ziel** — der Teil, der es zum Loop macht — ist eine messbare Hürde: jede Aussage hat mindestens drei Quellen, und jeder Link öffnet wirklich eine Seite, die belegt, was dort steht.

Ohne Loop verbrennt dich KI genau hier leise. Sie liefert ein sauber aussehendes Briefing mit Quellen, die echt klingen, aber nicht existieren — und merkt nicht, dass sie sie erfunden hat. Ein einzelner Prompt kann seine eigene erfundene Quelle nicht erkennen, weil er überzeugt bleibt, recht zu haben, bis etwas den Link tatsächlich öffnet.

Mit Loop schreibt Claude das Briefing, geht dann Link für Link, **öffnet jede Quelle tatsächlich**, um zu bestätigen, dass sie echt ist und die Aussage stützt. Erfundene wirft er raus, sucht echten Ersatz und prüft weiter, bis jede Quelle auf der Seite eine ist, die du wirklich öffnen kannst. Das ist der Loop, der den Teil übernimmt, den du früher von Hand gemacht hast.

SO WÜRDEST DU ES LAUFEN LASSEN

```
/goal Schreib ein einseitiges Briefing zu [dein Thema], bei dem
jede Aussage mindestens drei Quellen hat und jeder Link wirklich
eine Seite öffnet, die die Aussage stützt. Öffne jeden Link zur
Bestätigung, bevor du es als fertig bezeichnest. Ersetze jede
Quelle, die tot ist oder die Aussage nicht belegt. Stopp erst,
wenn jede Quelle auf der Seite besteht.
```

DER EINE PUNKT, DEN FAST ALLE FALSCH MACHEN

Dein Ziel muss etwas sein, gegen das Claude sich selbst **benoten** kann: echte Zahlen, ein Prozentwert, eine Checkliste. Nicht etwas Vages wie „mach es gut". Das Ganze funktioniert nur, wenn Claude seine eigene Arbeit bewerten und den exakten Moment erkennen kann, in dem sie bestanden ist. Ein schwammiges Ziel gibt ihm nichts zum Prüfen — also stoppt er wie ein normaler Prompt.

05

ZUM KOPIEREN

Dein erster Loop: der Loop-Charter

Die zwei Befehle sind der Auslöser. **Das** hier fütterst du ihnen. Ein einzeiliges Ziel reicht für eine kleine Aufgabe. Für eine größere mit vielen Schritten übergibst du dem Loop einen vollständigen Charter (eine Art Dienstanweisung): wo die Arbeit liegt, wie er sich prüft, wie er sich erinnert und wann er stoppt. Fülle die Klammern aus, füge dann den ganzen Charter in Claude Code ein.

DIESEN PROMPT KOPIEREN

```
Du läufst als Loop, nicht als einzelne Antwort. Hier ist dein
Auftrag (Charter):

ZIEL
[Beschreibe den fertigen Zustand in ein, zwei Sätzen. Beispiel:
"Jede Produktseite in /seiten hat aktualisierte Preise und besteht
den Link-Check." Sei konkret, wie FERTIG aussieht.]

WO DIE ARBEIT LIEGT
[Sag, wo die Arbeit zu finden ist. Beispiele: "Durchsuche /seiten
nach Dateien mit alten Preisen." / "Lies TODO.md und behandle jeden
nicht abgehakten Punkt als Aufgabe." / "Prüfe mein verbundenes
Aufgaben-Board auf Einträge mit dem Label 'ki'."]

WIE DU ARBEITEST
- Bearbeite IMMER nur einen Punkt. Schließe ihn vollständig ab,
  bevor du den nächsten beginnst.
- Halte dich an die Konventionen in vorhandenen Dateien. Übernimm
  den Stil, erfinde keine neuen Muster.
- Verlangt ein Punkt eine Entscheidung, die nur ich treffen kann
  (Geld ausgeben, etwas löschen, jemanden kontaktieren): STOPP bei
  diesem Punkt, schreib ihn auf eine Liste "braucht meine
  Entscheidung" und geh zum nächsten Punkt.

WIE DU DICH SELBST PRÜFST
Prüfe jeden Punkt, bevor du ihn als erledigt markierst:
[Wähle, was passt: "führe die Tests aus" / "lies die Datei erneut
und bestätige, dass sie das Ziel erfüllt" / "mach einen Screenshot
und bestätige, dass es richtig aussieht." Prüfen heißt Beweis,
nicht Zuversicht.]
Schlägt die Prüfung fehl, behebe es und prüfe erneut. Maximal 3
Versuche pro Punkt, danach als blockiert protokollieren und weiter.

WIE DU DICH ERINNERST
Führe eine Datei namens LOOP-STATE.md. Aktualisiere sie nach jedem
Punkt mit: Name des Punkts, Status (erledigt / blockiert / braucht
mich), was du geändert hast und alles, was der nächste Lauf wissen
muss. Lies diese Datei ZUERST zu Beginn jedes Laufs, damit du nie
erledigte Arbeit wiederholst.

WANN DU STOPPST
Stopp, wenn: (a) jeder Punkt erledigt oder als blockiert
protokolliert ist, oder (b) du in diesem Lauf [N] Punkte
abgeschlossen hast. Gib mir dann einen kurzen Bericht: was erledigt
wurde, was blockiert ist, was meine Entscheidung braucht.

Beginne damit, LOOP-STATE.md zu lesen, falls vorhanden, dann finde
die Arbeit.
```

WARUM DIE STATUS-DATEI ZÄHLT

Die Datei `LOOP-STATE.md` verwandelt zehn einzelne Läufe in **einen durchgehenden Mitarbeiter**. Ohne sie startet jeder Lauf bei null. Mit ihr kannst du denselben Charter nach Zeitplan laufen lassen, und der Loop macht immer genau dort weiter, wo er aufgehört hat.

06

EHRLICHE HINWEISE

Wann du NICHT loopen solltest

Loops sind nicht umsonst und nicht für alles. Drei ehrliche Vorbehalte, bevor du baust:

1 · EINMAL-AUFGABEN BRAUCHEN KEINEN LOOP

Ist die Arbeit eine einzelne Antwort (diese Mail schreiben, dieses Dokument zusammenfassen), ist ein Prompt schneller. Loops verdienen ihren Einrichtungs-Aufwand erst bei **wiederkehrender oder mehrteiliger** Arbeit.

2 · LOOPS VERBRAUCHEN MEHR ALS PROMPTS

Ein Loop, der sich selbst prüft und Sachen wiederholt, läuft mehrere KI-Runden pro Punkt. Auf einem Claude-Plan heißt das: Du erreichst dein Nutzungslimit schneller. Fang mit kleinen Mengen an — dafür gibt es die Zeile `[N] Punkte pro Lauf` im Charter.

3 · VERIFIKATION IST DAS GANZE SPIEL

Ein ungeprüfter Loop macht Fehler nur schneller. Wenn du nicht beschreiben kannst, **wie** der Loop seine eigene Arbeit prüfen soll, ist die Aufgabe noch nicht reif für einen Loop. Lass sie ein Prompt, bis du es kannst.

LOS GEHT'S

Diese Woche starten

Such dir **eine** nervige, mehrteilige Aufgabe, die du bisher von Hand machst. Füge den Charter ein, fülle die Klammern aus, lass ihn einmal laufen und schau dabei zu. Sobald du ihm vertraust, gib ihm einen Zeitplan. Das ist der ganze Weg vom Prompten zum Loop Engineering.

DIE KERNIDEE IN EINEM SATZ

Prompten heißt, die Arbeit zu machen. Loop Engineering heißt, das System zu bauen, das die Arbeit macht — und sich selbst kontrolliert, bis sie wirklich fertig ist.