Claude Code

Die komplette Anleitung

Von den ersten Schritten bis zum Expertenwissen

Autor: Christian Drapatz

Stand: Mai 2026 · Claude Code v2.1+

Plattform: macOS · Linux · Windows

Version 1.1

Disclaimer

Die Inhalte dieser Website basieren auf eigenen Erfahrungen sowie auf öffentlich zugänglichen Quellen wie Dokumentationen, Schulungen, Videos und Community-Beiträgen.

Alle Inhalte wurden eigenständig aufbereitet, zusammengefasst und in eigenen Worten formuliert. Es erfolgt keine wörtliche Übernahme geschützter Inhalte.

Die bereitgestellten Tutorials sind kostenlos und dienen ausschließlich der Wissensvermittlung. Trotz sorgfältiger Erstellung kann keine Gewähr für Vollständigkeit oder Aktualität übernommen werden.

Alle genannten Marken, Produkte und Technologien gehören den jeweiligen Inhabern.

Inhaltsverzeichnis

1 Einleitung – Was ist Claude Code?

Claude Code ist Anthropics terminalbasierter KI-Agent für Software-Entwicklung. Der entscheidende Unterschied zu einem Chat-Assistenten: Claude Code arbeitet direkt im Projekt. Es liest die Codebasis, führt Befehle aus, schreibt und bearbeitet Dateien – alles aus dem Terminal heraus.

Drei Eigenschaften machen Claude Code besonders:

- Kontext: Beim Start lädt es automatisch CLAUDE.md – eine Briefing-Datei mit Projektkontext, Konventionen und Regeln.
- Erweiterbarkeit: Wiederverwendbare Workflows lassen sich als Skills speichern und per Slash-Command aufrufen. Externe Dienste wie GitHub, Datenbanken oder Browser binden sich über MCP-Server an.
- Autonomie: Mit Auto Mode und Worktrees kann Claude Code komplette Features eigenständig bauen, testen und committen.

Diese Anleitung führt vom ersten Start bis zum vollständig konfigurierten Setup. Der Lernpfad in Kapitel 21 zeigt, wie man schrittweise vorgeht, ohne sich zu übernehmen.

2 Schnellstart in 10 Minuten

Fünf Schritte, um sofort loszulegen:

1. Installieren: Terminal öffnen und Installer starten.

```
curl -fsSL https://claude.ai/install.sh | bash
```

Danach Terminal neu öffnen.

2. Starten: In das Projektverzeichnis wechseln und claude eingeben. Beim ersten Start erfolgt der Browser-Login.
3. Datenschutz: Sofort nach dem Login unter claude.ai → Settings → Privacy → Help Improve Claude auf Off setzen.
4. Initialisieren: /init eingeben – Claude analysiert das Projekt und legt CLAUDE.md automatisch an.
5. Loslegen: Aufgabe formulieren, Shift+Tab für Plan Mode, Plan reviewen, dann implementieren lassen.

> **Hinweis:** Man muss nichts manuell konfigurieren. Claude Code einfach fragen, was man braucht – es erstellt die richtige Konfiguration. CLAUDE.md sorgt dafür, dass beim nächsten Start alles geladen ist.

3 Voraussetzungen, Pläne und Modelle

3.1 Kostenpläne im Überblick

Für die Nutzung von Claude Code wird ein bezahlter Anthropic-Account benötigt. Der normale kostenlose Claude.ai-Zugang ist dafür nicht ausreichend. Die Authentifizierung kann entweder über einen bestehenden Subscription-Login oder über einen API-Key erfolgen. Welche Variante sinnvoll ist,

hängt davon ab, ob Claude Code eher privat, im Team oder automatisiert in Entwicklungsprozessen genutzt werden soll.

Plan	Preis	Modell (Default)	Wann sinnvoll
Pro	ca. 20 USD/Monat	Sonnet 4.6	Solo, gelegentliches Coden
Max	ca. 100–200 USD/Monat	Opus 4.7	Solo, täglich mehrere Stunden; komplexe Refactorings
Team Standard	ca. 25 USD/Sitz (ab 5)	Sonnet 4.6	Teams; vertraglich kein Training auf euren Daten
Team Premium	ca. 125 USD/Sitz	Opus-äquivalent	Teams mit Max-ähnlichem Bedarf
Enterprise	Custom Pricing	Custom	SSO, Audit-Logs, Zero-Data-Retention
API direkt	Pay-per-Use	Frei wählbar	Sensible Codebasen; 7 Tage Retention, kein Training

> **Faustregel:** Mit Pro starten. Wenn nach zwei Wochen Limits spürbar werden, zu Max wechseln. Bei sensiblem Code Team oder API direkt wählen.

3.2 Modelle und Effort-Levels

Claude Code kann je nach Aufgabe mit unterschiedlichen Modellen und Effort-Levels arbeiten. Das ist sinnvoll, weil nicht jede Aufgabe die gleiche Tiefe braucht. Für einfache Suchen oder kleine Anpassungen reicht ein schnelles Modell. Für Architekturentscheidungen, größere Refactorings oder sicherheitskritische Änderungen lohnt sich dagegen ein stärkeres Modell mit mehr Denkzeit.

Die folgende Übersicht zeigt eine praktische Einordnung, wann welches Modell sinnvoll ist.

Modell	Stärke	Wann nutzen
Opus 4.7	Höchste Qualität, langsamer	Architektur, komplexe Refactorings, Security-Reviews
Sonnet 4.6	Gute Balance Qualität/Geschwindigkeit	Tägliches Coden, Features bauen
Haiku 4.5	Schnell und günstig	Stöbern, einfache Suchen, Routine-Checks

Modellwechsel mit /model opus, /model sonnet oder /model haiku.

Effort-Levels steuern, wie viel Denkzeit Claude Code für eine Aufgabe verwenden soll. Ein niedriger Effort reicht für einfache, klar begrenzte Aufgaben. Je komplexer die Änderung wird, desto höher sollte der Effort gewählt werden, damit Claude mehr Zusammenhänge prüfen und mögliche Nebenwirkungen berücksichtigen kann.

Level	Typische Aufgaben
low	Datei umbenennen, Build-Command, einfache Greps

medium	Funktionen schreiben, kleine Refactors
high	Multi-File-Refactors, komplexes Debugging
xhigh	Default bei Opus 4.7 – Architektur, Design-Entscheidungen
max	Härtestes Debugging, Security-Reviews, Edge-Cases (nur aktuelle Session)

Opus 4.7 unterstützt Effort-Levels mit /effort:

> **Tipp:** max springt nach der Session automatisch auf xhigh zurück – das schützt vor versehentlicher Token-Verbrennung.

3.3 Nutzungslimits

Claude nutzt je nach Tarif unterschiedliche Nutzungsgrenzen (Free, Pro, Max 5x/20x). Free ist stark limitiert, Pro bietet deutlich mehr Nutzung, und Max ist für intensive Nutzung mit Claude Code und großen Projekten gedacht.

Zusätzlich gibt es ein rollendes 5-Stunden-Fenster: Es startet mit der ersten Nachricht und setzt sich nach 5 Stunden automatisch zurück. Das Limit gilt gemeinsam für Claude Web/Desktop und Claude Code.

Seit 2025 existiert außerdem ein wöchentliches Limit für sehr intensive Nutzung. Besonders Opus verbraucht deutlich mehr Kontingent als Sonnet.

Der Verbrauch kann auf verschiedene Weise nachvollzogen werden.

- **/status** im Terminal – zeigt Plan, Modell-Status und verbleibende Nutzung
- **claude.ai/settings/usage** im Browser – vollständige Aufschlüsselung aller Kontingente

Kategorie	Beschreibung
Aktuelle Sitzung	Verbrauch der laufenden Sitzung über alle Modelle im rollenden 5-Stunden-Fenster. Das Fenster startet mit der ersten Nachricht und setzt sich nach 5 Stunden automatisch zurück. Besonders relevant bei Max-Plänen 5x/20x mit deutlich höherer Nutzung.
Alle Modelle	Wöchentliches Gesamtlimit über Opus, Sonnet und Haiku zusammen.
Nur Sonnet	Sonnet-Nutzung zählt gleichzeitig zu diesem Limit sowie zum wöchentlichen und 5-Stunden-Sitzungslimit.
Claude Design	Befindet sich in der Recherche-Vorschau mit eigenem wöchentlichem Limit. Wird nicht auf andere Limits angerechnet.
Zusätzliche Nutzung	Aktivierbar, um Claude nach Erreichen eines Limits weiter zu nutzen. Schaltet den API-Modus für kostenpflichtige Mehrnutzung frei.

> **Hinweis:** Wenn ein Limit erreicht ist, meldet Claude dies mit einer Fehlermeldung. In diesem Fall muss gewartet werden, bis das Limit automatisch zurückgesetzt wird. Alternativ kann die weitere Nutzung über die API erfolgen.

4 Installation

Es gibt zwei Wege: das Terminal-CLI und die Desktop-App. Beide teilen sich Login und Konfiguration.

Terminal-Installation (empfohlen)

Kein Node.js mehr nötig – läuft direkt als Binary und aktualisiert sich automatisch:

```
# macOS / Linux
curl -fsSL https://claude.ai/install.sh | bash

# Windows (PowerShell)
irm https://claude.ai/install.ps1 | iex

# macOS via Homebrew (manuelles Update)
brew install claude-code
```

Nach der Installation:

```
claude --version    # Version prüfen
claude doctor       # Diagnose
claude              # Starten und Browser-Login
```

Desktop-App

Download unter claude.com/download. Die App bietet parallele Sessions in isolierten Git-Worktrees sowie einen Tab für lokale und Cloud-Sessions.

Updates

```
claude update           # Sofortiges Update
brew upgrade claude-code  # Bei Homebrew
```

5 Datenschutz und Sicherheit

Diese Einstellungen gehören vor die erste echte Session.

5.1 Pflicht-Setting vor der ersten Session

Sofort nach dem Login:

> **Wichtig:** claude.ai → Settings → Privacy → Help Improve Claude → Off

Ohne diese Einstellung haben Sessions fünf Jahre Retention statt 30 Tagen und fließen ins Modelltraining ein. Gilt für alle Consumer-Pläne (Pro und Max). Team- und Enterprise-Pläne haben vertraglich kürzere Retention.

5.2 Was Claude überträgt

Jede Datei, die Claude liest, geht vollständig an Anthropics Server – es scannt nicht automatisch das Projekt, aber jede explizit gelesene Datei wird übertragen. Besonders sensibel:

- .env-Dateien mit API-Keys und Passwörtern
- Zertifikate (.p12, .mobileprovision) und SSH-Schlüssel
- Medizinische, juristische oder anderweitig geschützte Daten

Dateien ausschließen mit .claudeignore

Wie .gitignore verhindert .claudeignore das automatische Lesen bestimmter Dateien:

```
# .claudeignore
/Config/Secrets/
*.env
**/TestData/real_patients/**
/Resources/Credentials/
```

> **Achtung:** Wenn Claude explizit angewiesen wird, eine .claudeignore-Datei zu lesen, wird sie trotzdem übertragen. Die Datei schützt nur vor automatischem Zugriff.

Verhaltensregeln in CLAUDE.md

Für regulierte Projekte empfiehlt sich ein Datenschutz-Block in der CLAUDE.md:

```
# Datenschutz
NIEMALS lesen:
- /TestData/ (Patientendaten)
- Dateien mit "patient", "versicherte" oder "KV-Nummer" im Namen
- /Config/Live/*.json (Produktionskonfiguration)
```

Einsatz in regulierten Umgebungen

Claude Code verarbeitet Anfragen auf US-Servern (AWS). Für DSGVO- und SGB-V-relevante Gesundheitsdaten:

Ansatz	Eignung	Aufwand
Nur Code übertragen, niemals Echtdaten	Gut praktikabel	Niedrig
Anonymisierte / fiktive Testdaten verwenden	Empfohlen	Mittel
.claudeignore + CLAUDE.md Datenschutz-Block	Pflicht im Projekt	Niedrig
Anthropic Enterprise-Plan mit AVV abschließen	Rechtlich sauber	Hoch
Lokales Modell (z. B. Ollama) als Alternative	Vollständig offline	Hoch

> **Goldene Regel:** Claude Code ist ein Werkzeug für Code – nicht für Daten. Wer das sauber trennt, kann Claude Code auch in regulierten Umgebungen produktiv nutzen.

5.3 Wie Claude Code dich schützt

- Read-only by default: Fragt vor jedem Edit und jedem Bash-Befehl nach.
- Schreibgrenze: Nur Startverzeichnis und Unterordner – deshalb immer im Projektroot starten.
- Command-Blocklist: curl und wget sind standardmäßig gesperrt.
- Prompt-Injection-Schutz: Web-Fetch-Inhalte werden so behandelt, dass eingeschleuste Anweisungen nicht automatisch zu Tool-Calls führen.

> **Wichtig:** Permissions schützen vor Aktionen, nicht vor Datenabfluss. Wenn Claude eine Datei lesen darf, geht ihr Inhalt an Anthropic. Secrets-Dateien deshalb explizit per Deny-Regel sperren.

6 Erste Schritte und tägliche Befehle

6.1 Im Projekt starten

Terminal öffnen, in den Projektordner wechseln und Claude Code starten

```
cd mein-projekt
claude
```

Beim ersten Start: Theme wählen, Browser-Login, claude doctor zur Verifikation. Danach /init eingeben – das generiert eine CLAUDE.md mit Projektkontext, Build-Befehlen und Konventionen. Diese Datei wird bei jeder weiteren Session automatisch geladen.Beim ersten Start: Theme wählen, Browser-Login durchführen und mit claude doctor die Installation prüfen. Danach /init ausführen – dadurch wird eine CLAUDE.md mit Projektkontext, Build-Befehlen und Konventionen erzeugt.

> **Wichtig:** Die CLAUDE.md sollte direkt danach überprüft und angepasst werden. Sie wird bei jeder weiteren Session automatisch geladen und beeinflusst maßgeblich das Verhalten und die Antworten von Claude Code.

6.2 Die wichtigsten Slash-Commands

Claude Code besitzt eine Reihe von Slash-Commands zur Steuerung und Unterstützung des Workflows. Dazu einfach in der Eingabeaufforderung ein / eingeben. Anschließend zeigt Claude Code automatisch alle verfügbaren Befehle und passenden Vorschläge an.

Command	Funktion
/init	Generiert automatisch eine CLAUDE.md für das Projekt
/clear	Löscht die aktuelle Konversation – frischer Start, spart Tokens
/compact	Komprimiert den Kontext wenn das Fenster voll läuft
/usage	Dashboard: Plan-Limits, Tageskosten, Streaks
/status	Plan, aktives Modell und Nutzungslimits anzeigen
/cost	Kurzanzeige der Kosten der aktuellen Session
/review	Code-Review der letzten Änderungen
/model	Modellwechsel, z. B. /model opus
/effort	Effort-Level: low / medium / high / xhigh / max
/mcp	MCP-Verbindungen verwalten und Status prüfen
/memory	Geladene CLAUDE.md anzeigen und bearbeiten
/ultrareview	Mehrstufiger tiefgehender Code-Review

/loop	Self-paced Loops für wiederholte Aufgaben
/permissions	Allowlist aus bisherigen Tool-Calls vorschlagen lassen

Session-Management

Mit den Session- und Kontextbefehlen kann Claude Code bestehende Sitzungen fortsetzen sowie gezielt Dateien und Ordner in den aktuellen Arbeitskontext laden. Dadurch versteht Claude Code den Projektzusammenhang besser und kann präzisere Antworten sowie passendere Änderungen liefern.

```
claude -c             # Letzte Session fortsetzen
claude -r "name"      # Session per Name fortsetzen
@datei.swift          # Datei in den Kontext laden
@Sources/Views/       # Ordner in den Kontext laden
```

6.3 Modi

Claude Code unterstützt verschiedene Arbeitsmodi, die je nach Aufgabe mehr Kontrolle oder mehr Automatisierung bieten. Zwischen den Modi kann jederzeit mit Shift + Tab gewechselt werden

- Normal: Fragt vor jeder Aktion nach.
- Auto-Accept: Aktionen ohne Rückfrage, sofern nicht durch Deny gesperrt. Für lange Sessions in einem gepushten Branch.
- Plan Mode: Read-only – Claude erstellt nur einen Plan. Ideal für den Analyze-Plan-Build-Workflow.
- Auto Mode (Max-Plan): Claude trifft Entscheidungen vollständig selbstständig. Nicht für Production-Deploys.

7 CLAUDE.md – Das Projektgedächtnis

CLAUDE.md ist die wichtigste Datei im Setup. Sie wird bei jeder Session automatisch geladen und gibt Claude den Kontext, den er sonst jedes Mal neu erraten müsste.

7.1 Was reingehört

Vier Fragen beantworten, mehr nicht:

- Was ist das Projekt? (Stack, Plattform-Targets, Sprache)
- Wie wird es gebaut und getestet? (Build- und Test-Befehle, Tooling)
- Welche Konventionen gelten? (Code-Style, Ordnerstruktur, Architektur-Regeln)
- Was soll Claude nicht tun? (Verbote, Tabu-Bereiche)

Nicht reingehört: Tutorials, lange Erklärungen, Marketing, Dokumentationstexte. Die Datei ist eine Briefing-Karte, kein Wiki. Maximal 200 Zeilen.

7.2 Wo die Datei liegt

Inhalte werden zusammengeführt, nicht überschrieben:

- ~/.claude/CLAUDE.md – global, gilt für alle Projekte
- CLAUDE.md im Projektroot – gilt nur für dieses Projekt (ins Repo committen!)
- CLAUDE.local.md im Projektroot – persönlich, nicht im Git
- Unterordner-CLAUDE.md – wird nur geladen, wenn man dort arbeitet

7.3 Beispiel und Best Practices

Globale CLAUDE.md (~/.claude/CLAUDE.md)

```
# Allgemein
Antworte auf Deutsch. Halte dich kurz.
Bei größeren Änderungen erst einen Plan vorschlagen.

# Verbote
Niemals Inhalte aus .env oder secrets/ lesen.
Kein git push --force ohne explizite Bestätigung.
```

Projekt-CLAUDE.md (Beispiel Swift / SwiftUI)

```
# Projekt
SwiftUI-App, Targets: macOS, iPadOS, iPhone

# Build
xcodebuild -scheme AppName \
  -destination "platform=iOS Simulator,name=iPhone 16"

# Navigation
- iPhone: NavigationStack
- macOS/iPadOS: NavigationSplitView

# Verbote
Keine DispatchQueue – nur async/await.
```

> **Tipp:** Globale Datei einmal anlegen. Im Projekt /init ausführen, generierte Datei sofort auf das Wesentliche kürzen, ins Repo committen. In den ersten Sessions ergänzen, was Claude offensichtlich nicht weiß.

8 Die neun Bausteine im Überblick

Für die praktische Arbeit mit Claude Code lassen sich neun zentrale Bausteine unterscheiden. Sie greifen ineinander und bestimmen, wie Claude Code Kontext versteht, Aufgaben ausführt, erweitert wird und welche Regeln dabei gelten.

Baustein	Wann aktiv	Wo definiert	Wofür
CLAUDE.md	Immer	Projektroot	Kontext, Konventionen, Regeln
Permissions	Immer	settings.json	Erlauben und Verbieten
Hooks	Bei Events	settings.json	Mechanik, Automatik

Skills	Auf Aufruf	skills/	Wiederverwendbare Aufgaben
Subagents	Auf Aufruf	agents/	Eigener Kontext, knappe Antwort
MCP-Server	Auf Aufruf	settings.json	Externe Werkzeuge
Sandbox	Bei Aktivierung	settings.json	Wirkungsbereich begrenzen
Plugins	Bei Aktivierung	plugins/	Bündel mehrerer Bausteine
Loops/Tasks	Periodisch	settings.json	Routine, Beobachtung

9 Verzeichnisstruktur und settings.json

Alle Bausteine folgen derselben Logik: global im Home-Verzeichnis, projektspezifisch im Projekt, lokal-persönlich in einer Datei, die nicht ins Git kommt.

```
~/.claude/                          # Globale Konfiguration
├── CLAUDE.md                       # Globale Vorlieben
├── settings.json                   # Globale Permissions, Hooks, MCP
├── skills/<name>/SKILL.md          # Globale Skills
└── agents/<name>/AGENT.md          # Globale Subagents

.claude/                            # Projektspezifisch (ins Git)
├── settings.json                   # Projekt-Permissions, Hooks, MCP
├── skills/<name>/SKILL.md          # Projekt-Skills
└── agents/<name>/AGENT.md          # Projekt-Subagents

CLAUDE.local.md                     # Persönlich, nicht ins Git
.claude/settings.local.json         # Lokal, nicht ins Git
```

Aufbau der settings.json

Die settings.json steuert zentrale Einstellungen von Claude Code, zum Beispiel erlaubte und gesperrte Aktionen, Hooks, MCP-Server und das verwendete Modell. Besonders wichtig sind die permissions, weil sie festlegen, was Claude Code im Projekt ausführen darf und was ausdrücklich blockiert wird.

```
{
  "permissions": {
    "allow": ["Read", "Edit", "Bash(git diff*)"],
    "deny":  ["Bash(rm -rf *)", "Read(.env*)"]
  },
  "hooks": { "PostToolUse": [...] },
  "mcpServers": { "github": {...} },
  "model": "claude-opus-4-7"
}
```

Die drei Ebenen (global, Projekt, lokal) werden zusammengeführt. Deny gewinnt immer.

> **Wichtig:** Echte JSON-Dateien dürfen keine Kommentare und keine trailing commas enthalten. .claude/settings.local.json und .claude/cache/ in die .gitignore aufnehmen.

10 Permissions

Permissions legen fest, welche Aktionen Claude Code automatisch ausführen darf und bei welchen Aktionen vorher eine Bestätigung erforderlich ist. Dadurch lässt sich genau steuern, wie viel Kontrolle Claude Code über Dateien, Befehle und Projektbereiche erhält.

10.1 Die drei Stufen

Jede Aktion fällt in genau eine der folgenden Kategorien. Die Kombination dieser Regeln bestimmt, wie Claude Code im Projekt arbeitet.

Stufe	Verhalten
Allow	Darf ohne Rückfrage ausgeführt werden
Ask	Default für alles, was nicht explizit erlaubt oder verboten ist – Claude fragt nach
Deny	Darf gar nicht – absolute Sperre

10.2 Regeln formulieren

Permissions werden über Regeln definiert. Damit kann festgelegt werden, welche Dateien gelesen oder verändert werden dürfen und welche Bash-Befehle erlaubt oder blockiert sind.

```
Read                    # Lesezugriff allgemein
Edit                    # Datei-Edits allgemein
Read(.env*)             # Lesen von .env-Dateien
Bash(git status*)       # nur git status
Bash(git *)             # alle git-Befehle
Bash(xcodebuild *)      # alle xcodebuild-Calls
```

10.3 Empfohlene Konfiguration

Globale ~/.claude/settings.json – harte Verbote

```
{
  "permissions": {
    "deny": [
      "Bash(rm -rf *)",   "Bash(sudo *)",
      "Bash(curl *)",     "Bash(wget *)",
      "Read(.env*)",      "Read(**/secrets/**)",
      "Read(**/.ssh/**)","Read(**/Credentials/**)"
    ]
  }
}
```

Projektspezifische .claude/settings.json – Routine erlauben

```
{
  "permissions": {
```

```
    "allow": [
      "Read", "Edit",
      "Bash(git status*)", "Bash(git diff*)",
      "Bash(git log*)",   "Bash(git add *)",
      "Bash(git commit *)","Bash(xcodebuild *)",
      "Bash(swift test*)"
    ]
  }
}
```

> **Tipp:** /permissions allowlist schlägt eine sinnvolle Allow-Liste aus den letzten Tool-Calls vor. Einfach zwei, drei Sessions arbeiten und dann den Command ausführen.

11 Hooks

Hooks ermöglichen es, automatisch Shell-Befehle bei bestimmten Ereignissen innerhalb von Claude Code auszuführen. Dadurch lassen sich wiederkehrende Aufgaben wie Formatierung, Prüfungen oder Benachrichtigungen automatisieren. Hooks sind für Mechanik gedacht – nicht für komplexe Logik oder komplette Workflows.

Lifecycle-Events

Lifecycle-Events bestimmen, wann ein Hook ausgelöst wird. Je nach Event kann ein Befehl vor oder nach einer Aktion oder am Ende einer Session ausgeführt werden

- PostToolUse – nach jedem Tool-Call; klassisch für Formatter und Linter
- PreToolUse – vor jedem Tool-Call; für zusätzliche Prüfungen
- Stop – am Sessionende; für Desktop-Notifications

Wichtige Variablen

```
$CLAUDE_FILE_PATH     # die bearbeitete Datei
$CLAUDE_TOOL_NAME     # welches Tool lief
$CLAUDE_PROJECT_DIR   # Projektroot
```

Beispiel – SwiftFormat nach jedem Edit

Dieses Beispiel zeigt einen einfachen PostToolUse-Hook für Swift-Projekte. Nach jedem Edit- oder Write-Vorgang prüft Claude Code, ob eine Swift-Datei geändert wurde. Falls ja, wird automatisch swiftformat für genau diese Datei ausgeführt.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "if [[ \"$CLAUDE FILE PATH\" == *.swift ]]; then
swiftformat \"$CLAUDE_FILE_PATH\"; fi"
```

```
          }
        ]
      }
    ]
  }
}
```

Beispiel – Desktop-Notification beim Sessionende

Dieses Beispiel zeigt einen einfachen Stop-Hook. Sobald eine Claude-Code-Session beendet wird, erscheint automatisch eine macOS-Desktop-Benachrichtigung. Das ist praktisch bei längeren Aufgaben oder automatisierten Workflows

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [{ "type": "command",
          "command": "osascript -e 'display notification \"Claude
fertig\" with title \"Claude Code\"'" }]
      }
    ]
  }
}
```

> **Hinweis:** Hooks müssen schnell sein (unter einer Sekunde), idempotent und still scheitern. Mit einem einzigen Hook anfangen – dem Formatter.

12 Skills

Skills sind wiederverwendbare Vorlagen und Anweisungen für typische Aufgaben im Entwicklungsalltag. Dadurch lassen sich wiederkehrende Abläufe standardisieren und deutlich schneller ausführen. Ein Skill besitzt einen Namen, eine Beschreibung und konkrete Schritte oder Regeln, die Claude Code bei der Ausführung berücksichtigen soll.

12.1 Aufbau eines Skills

Das folgende Beispiel zeigt einen einfachen Skill zum automatischen Erstellen einer neuen SwiftUI-View nach festen Projektkonventionen.

```
---
name: new-view
description: Legt eine neue SwiftUI-View nach unseren Konventionen an
---

Erstelle eine neue SwiftUI-View. Schritte:

1. Datei unter Sources/Views/<Feature>/<Name>View.swift anlegen
2. @Observable ViewModel einbinden
```

```
3. Preview-Provider ergänzen
4. In die Navigation eintragen
```

Aufruf: /new-view ProfilEditor. Argumente landen als $ARGUMENTS oder $1, $2 im Skill.

12.2 Empfehlenswerte Community-Skills

Neben eigenen Skills gibt es inzwischen auch viele Community-Skills für typische Entwicklungsaufgaben. Sie können als Grundlage dienen, Zeit sparen und zeigen oft gute Praxisbeispiele für wiederverwendbare Workflows.

Skill	Funktion	Installation
Superpowers	Strukturierter Workflow: erst planen, dann testen, zuletzt coden	/plugin install superpowers@claude-plugins-official
Frontend Design	Professionelle, moderne UIs statt generischem AI-Look	/plugin install frontend-design@claude-plugins-official
Skill Creator	Erzeugt neue Skills nach Beschreibung	/plugin install skill-creator@claude-plugins-official
Trail of Bits	Security-Audit (SQL Injection, XSS, …)	/plugin marketplace add trailofbits/skills
Prompt Master	Optimiert Prompts für 30+ KI-Tools	git clone .../nidhinjs/prompt-master ~/.claude/skills/
Marketing Skills	40 Marketing-Experten: SEO, Ads, Copywriting, Wachstum	npx skills add coreyhaines31/marketingskills
CLAUDE.md Optimizer	Hält CLAUDE.md schlank, lagert Unwichtiges aus	git clone .../wrsmith108/claude-md-optimizer ~/.claude/skills/
Webapp Testing	Browser-basiertes automatisches Testing	/plugin install webapp-testing@claude-plugins-official

> **Tipp:** Erst spüren, wo man sich wiederholt. Skills, die theoretisch ausgedacht werden, treffen meist daneben.

13 Subagents

Subagents sind eigenständige Claude-Instanzen für klar getrennte Aufgabenbereiche. Sie arbeiten mit einem eigenen Kontext und entlasten dadurch die Hauptsession, insbesondere bei größeren Analysen oder spezialisierten Aufgaben. Der große Vorteil: Die Hauptsession bleibt schlanker und fokussierter, während der Subagent nur das fertige Ergebnis oder eine Zusammenfassung zurückliefert.

> **Tipp:** **Skill vs. Subagent**: Ein Skill arbeitet im Hauptchat und vergrößert dort den Kontext. Ein Subagent arbeitet nebenan und liefert nur das Ergebnis. Wenn eine Aufgabe viel Kontext frisst, aber wenig zurückgibt – Subagent verwenden.

Aufbau – Beispiel Code-Reviewer

Dieses Beispiel zeigt einen typischen Subagent für automatisierte Code-Reviews. Der Agent analysiert ausschließlich geänderte Dateien und liefert eine kompakte Zusammenfassung kritischer Probleme zurück. Dadurch bleibt die Hauptsession übersichtlich und größere Reviews können gezielt ausgelagert werden.

```
---
name: code-reviewer
description: Senior Code-Reviewer. Nach Code-Änderungen proaktiv
aufrufen.
tools: Read, Grep, Glob, Bash(git diff*)
model: sonnet
---

Du bist Senior Code-Reviewer für ein Swift-Projekt.
Analysiere nur die geänderten Dateien.
Gib zurück: max. 5 kritische Punkte, sortiert nach Priorität.
Keine langen Erklärungen.
```

Drei Felder, die zählen

- description – entscheidet, ob Claude den Agent automatisch aufruft.
- tools – begrenzt, was der Agent darf. Eng halten.
- model – Recherche/Reviews: sonnet; Architekturanalyse: opus; einfache Suchen: haiku.

14 MCP-Server

MCP-Server (Model Context Protocol) erweitern Claude Code um externe Werkzeuge und Dienste. Dadurch kann Claude nicht nur analysieren und schreiben, sondern auch direkt mit APIs, Datenbanken, GitHub oder anderen Systemen arbeiten. Mit MCP wird Claude Code deutlich näher an einen echten Entwicklungsassistenten, der Aufgaben aktiv ausführen kann

14.1 Grundlagen und Einrichtung

MCP-Server werden über die Kommandozeile registriert und anschließend automatisch in Claude Code eingebunden. Danach können sie direkt aus der Session heraus verwendet werden.

```
claude mcp add github -- npx -y @modelcontextprotocol/server-github
claude mcp add --transport http my-api https://meine-api.de/mcp
claude mcp list
/mcp                    # Status aller Server prüfen
```

Die folgenden MCP-Server gehören aktuell zu den beliebtesten und praktischsten Erweiterungen für Claude Code

MCP-Server	Funktion
GitHub	Issues lesen, PRs vorbereiten, Branches verwalten

Supabase	Datenbankabfragen, Tabellen, RLS-Policies direkt aus Claude steuern
Context7	Aktuelle Library-Docs laden – kein veraltetes Modellwissen
Exa / Perplexity	Web-Suche aus Claude heraus
Playwright	Browser-Automatisierung, Screenshots, E2E-Tests
Pipedream	8.000+ Apps in einem MCP: Gmail, Slack, Stripe, Notion, …

14.2 Der Senior-Engineer-Stack – 5 kostenlose MCPs

Die folgenden MCPs eignen sich besonders gut für Entwickler, die Claude Code stärker in ihren täglichen Workflow integrieren möchten. Sie erweitern Claude um Code-Reviews, Langzeitgedächtnis, Websuche und externe Dienste

MCP	Funktion	Voraussetzung
CodeRabbit	Senior-Engineer-Review für Sicherheit, Performance, Maintainability	GitHub Personal Access Token
Pipedream	8.000+ Apps in einem MCP	Pipedream Account (kostenlos)
Exa	Smart Search (1.000 Anfragen/Monat gratis)	Exa API Key
Sequential Thinking	Strukturiertes Denken – Anthropics offizieller Reference-Server	Keine
Memory MCP	Langzeitgedächtnis über Projekte hinweg	Keine

> **Sicherheit:** Anthropic auditiert keine Drittanbieter-MCP-Server. Nur vertrauenswürdige Quellen nutzen. API-Keys nie in committete Dateien.

15 Plugins

Plugins erweitern Claude Code um zusätzliche Funktionen und fertige Workflows. Sie kombinieren häufig mehrere Komponenten wie Skills, Hooks, Subagents oder Einstellungen zu einem direkt nutzbaren Paket. Dadurch lassen sich komplexe Setups deutlich schneller einrichten und wiederverwenden

Verwaltung

Plugins können direkt über Claude Code installiert, aufgelistet, aktualisiert oder entfernt werden. Nach Änderungen empfiehlt sich ein Reload, damit neue Funktionen sofort verfügbar sind

```
/plugins
claude plugin add <name>
claude plugin list
claude plugin remove <name>
/reload-plugins
```

Empfehlenswerte Plugins

Die folgenden Plugins gehören aktuell zu den beliebtesten Erweiterungen für Claude Code. Sie ergänzen Funktionen wie Langzeitgedächtnis, Token-Optimierung, Design-Workflows oder umfangreiche Power-User-Funktionen

Plugin	Funktion	Installation
Context Mode	Läuft im Hintergrund, spart Tokens (10.645 Stars)	/plugin marketplace add mksglu/context-mode
claude-mem	Persistentes Gedächtnis über Sessions (68.482 Stars)	npx claude-mem install
Everything Claude Code	Umfassendes Power-User-Bundle (168.287 Stars)	/plugin marketplace add affaan-m/everything-claude-code
Huashu Design	Prototypen und Slide Decks per natürlicher Sprache	npx skills add alchaincyf/huashu-design

> **Wichtig:** Erst eine Weile ohne Plugins arbeiten – wer die Bausteine nicht kennt, kann Plugins nicht beurteilen.

16 Loops und geplante Aufgaben

Mit Loops und geplanten Tasks kann Claude Code Aufgaben automatisch in bestimmten Zeitabständen oder zu festen Uhrzeiten ausführen. Dadurch lassen sich wiederkehrende Prüfungen, Statusabfragen und Routineaufgaben automatisieren. Besonders praktisch ist das für längere Entwicklungs-, Build- oder Review-Prozesse

Einsatzgebiete

Loops eignen sich vor allem für wiederholte Prüfungen und Überwachungen. Geplante Tasks dagegen sind ideal für tägliche oder regelmäßig wiederkehrende Aufgaben.

- Loops: Build-Status beobachten, CI-Build alle 3 Minuten prüfen, Test-Ergebnisse
- Geplante Tasks: Tägliche Release-Notes, Meeting-Vorbereitung, Wartungsaufgaben

In der Praxis

Die folgenden Beispiele zeigen typische Loop- und Scheduling-Befehle für den Entwicklungsalltag.

```
/loop 5m check ob der staging-build fertig ist und melde dich
/loop 10m /security-review
/schedule "0 8 * * 1-5" /daily-standup-prep
/loop status
/loop stop
```

In settings.json

Geplante Aufgaben können dauerhaft in der settings.json hinterlegt werden. Claude Code führt diese Tasks dann automatisch nach dem definierten Zeitplan aus

```
{
  "scheduled_tasks": [
    {
      "name": "daily-standup",
      "schedule": "0 8 * * 1-5",
      "command": "/daily-standup-prep",
      "notify": true
    }
  ]
}
```

> **Hinweis:** Klare Endbedingung definieren, realistische Intervalle (3–10 Minuten). Loops nie für destruktive Aktionen nutzen.

17 Token-Optimierung und Kostenkontrolle

Gerade bei längeren Sessions und größeren Projekten lohnt es sich, auf Tokenverbrauch und Modellwahl zu achten. Mit der richtigen Strategie lassen sich Kosten reduzieren und gleichzeitig Antwortgeschwindigkeit sowie Kontextqualität verbessern. Besonders wichtig sind dabei die Wahl des passenden Modells, regelmäßiges Aufräumen des Kontexts und der gezielte Einsatz von Subagents

17.1 Modellwahl und /clear

Nicht jede Aufgabe benötigt das leistungsstärkste Modell. Für viele alltägliche Aufgaben reicht Sonnet völlig aus, während Opus gezielt für komplexere Analysen oder Architekturfragen eingesetzt werden sollte

- Modellwahl: Opus kostet ca. das 5-fache von Sonnet pro Token. Opus gezielt für Architektur und Refactoring, Sonnet für den Alltag.
- /clear: Jeder Token im Kontext kostet bei jeder Anfrage erneut. Lange Sessions ohne /clear werden überproportional teuer.
- Subagents: Eigener Kontext, knappe Antwort an den Hauptchat. Hauptkontext bleibt klein.

17.2 claude-mem – Persistentes Gedächtnis

claude-mem erweitert Claude Code um ein lokales Langzeitgedächtnis. Wichtige Informationen aus vorherigen Sessions werden zusammengefasst gespeichert und bei Bedarf automatisch wieder geladen. Dadurch bleiben Projekte konsistenter und wiederkehrende Erklärungen oder Analysen werden reduziert

Schritt	Was passiert
Capture	Jeder Tool-Call wird automatisch abgefangen
Compress	Reduktion von bis zu 10.000 auf ca. 500 Tokens (5–10× im Alltag)
Store	Lokale SQLite-Datenbank

Retrieve	Claude lädt nur den relevanten Kontext für die aktuelle Aufgabe

```
npx claude-mem install

# Alternativ
/plugin marketplace add thedotmack/claude-mem
/plugin install claude-mem
```

> **Sicherheitshinweis:** Die HTTP-API auf Port 37777 hat keine Authentifizierung. Firewall so einstellen, dass Port 37777 nur für localhost erreichbar ist.

17.3 RTK – Token-Kompression

RTK (Rust Token Killer) ist ein CLI-Proxy, der Befehlsausgaben komprimiert, bevor sie Claude erreichen. Ergebnis: 60–90 % weniger Tokens.

Befehl	Tokens vorher	Tokens nachher	Einsparung
git status	~800	~50	~94 %
git push	~200	~10	~95 %
ls (groß)	~1.500	~150	~90 %
npm test	~3.000	~300	~90 %

Die folgenden Befehle installieren und aktivieren RTK für Claude Code. RTK hilft dabei, den Kontext automatisch zu optimieren und den Tokenverbrauch in längeren Sessions deutlich zu reduzieren. Zusätzlich können Status und mögliche Einsparungen jederzeit angezeigt werden

```
curl -fsSL https://raw.githubusercontent.com/rtk-
ai/rtk/master/install.sh | sh
rtk init -g         # Hooks für Claude Code aktivieren
rtk init --show     # Status prüfen
rtk gain            # Einsparungen anzeigen
```

18 Cloud-Automationen mit Routines

Routines ermöglichen vollständig automatisierte Abläufe direkt in der Cloud. Dadurch können Aufgaben zeitgesteuert oder ereignisbasiert ausgeführt werden, ohne dass ein lokaler Rechner oder Server dauerhaft laufen muss

Plan	Routine-Runs pro Tag
Pro	5
Max	15
Team/Enterprise	25

Routine einrichten

1. claude.ai/code/routines öffnen
2. New routine → Name vergeben
3. Prompt schreiben (präzise wie ein Auftrag an einen Mitarbeiter)
4. GitHub Repository auswählen
5. Trigger wählen: Schedule, API oder GitHub Event
6. Connectors verbinden (Gmail, Slack etc.)
7. Run now zum Testen – erst danach den Schedule aktivieren

Beispiel – Tägliche E-Mail-Zusammenfassung

```
Geh in meine E-Mails über den Gmail Connector.
Lies alle ungelesenen E-Mails seit gestern.
Fass jede E-Mail in einem Satz zusammen.
Sortiere nach: Dringend, Aktion nötig, Info.
Schick mir die Zusammenfassung als Slack-Nachricht.
```

Beispiel – PR-Review bei jedem Pull Request

```
Ein neuer Pull Request wurde geöffnet.
Prüfe den Code auf Sicherheitsprobleme, Performance-Probleme,
Code-Stil und Lesbarkeit.
Schreib ein Review als Kommentar auf den PR.
```

> **Wichtig:** Routines laufen vollständig autonom. Alle Aktionen passieren unter dem eigenen Namen. API-Keys in den Environment Variables der Cloud Environment ablegen.

19 Fortgeschrittene Workflows

Mit fortgeschrittenen Workflows lässt sich Claude Code deutlich effizienter in größere Projekte und parallele Entwicklungsprozesse integrieren. Besonders bei mehreren Features oder längeren Aufgaben helfen Worktrees dabei, sauber getrennte Arbeitsbereiche parallel zu verwalten

19.1 Paralleles Arbeiten mit Worktrees

Worktrees ermöglichen mehrere parallele Arbeitsumgebungen innerhalb eines Git-Repositories. Dadurch können unterschiedliche Features, Bugfixes oder Experimente gleichzeitig bearbeitet werden, ohne ständig Branches wechseln zu müssen

```
claude --worktree mein-feature   # Worktree starten
claude -w                        # Schneller Worktree-Start
```

Die Desktop-App bietet einen Code-Tab mit Worktree-Checkbox – am bequemsten für 3–5 parallele Sessions.

> **Killerkombination:** Auf dem Max-Plan: Opus 4.7 + /effort max + Auto Mode + 1M Context = ein komplettes Projekt starten und nach Stunden zu einem eigenständig refactorten, getesteten und committeten Ergebnis zurückkehren.

19.2 Der Verification-Loop

Der wichtigste Einzeltipp von Boris Cherny (Claude Code-Gründer): "If Claude has that feedback loop, it will 2–3x the quality of the output."

```
Wir bauen jetzt zusammen Feature X.
Bevor du eine einzige Zeile Code schreibst, definiere zuerst,
wie du selbst prüfst, ob das fertige Feature funktioniert:

1. Welcher Test, Befehl oder Browser-Check beweist, dass es geht?
2. Was ist dein Plan?
3. Erst nach Zustimmung umsetzen.
```

19.3 Sandbox für autonome Arbeit

Die Sandbox begrenzt gezielt, auf welche Dateien, Ordner und Netzwerkfunktionen Claude Code zugreifen darf. Dadurch kann Claude autonomer arbeiten, ohne unbegrenzte Rechte auf das gesamte System zu erhalten. Besonders bei automatisierten Workflows oder längeren Auto-Mode-Sessions erhöht die Sandbox die Sicherheit und Kontrolle deutlich

```
/sandbox

# Oder in settings.json:
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "writable": ["$CLAUDE_PROJECT_DIR/Sources",
"$CLAUDE_PROJECT_DIR/Tests"],
      "readable": ["$CLAUDE_PROJECT_DIR"]
    },
    "network": { "enabled": false }
  }
}
```

> **Tipp:** Sandbox + Auto-Accept: Claude arbeitet schnell, kann aber technisch nicht aus dem definierten Bereich ausbrechen.

19.4 Plan Mode zuerst

Gerade bei größeren Änderungen, Refactorings oder Architekturentscheidungen sollte nicht sofort im normalen Schreibmodus gearbeitet werden. Stattdessen empfiehlt es sich, zuerst in den Plan Mode zu wechseln (Shift + Tab → plan).

Im Plan Mode arbeitet Claude Code im Read-only-Modus. Es werden keine Dateien verändert und keine Befehle ausgeführt. Stattdessen analysiert Claude die Aufgabe und erstellt einen strukturierten Umsetzungsplan.

Das hat mehrere Vorteile:

- Risiken und mögliche Probleme werden früh sichtbar
- Die geplante Architektur kann geprüft werden
- Missverständnisse fallen schneller auf

- Große Änderungen werden kontrollierbarer
- Man verhindert unnötige oder falsche Code-Änderungen

Besonders bei:

- Refactorings
- größeren SwiftUI-/Architekturänderungen
- Datenbankänderungen
- Build-/CI-Anpassungen
- Multi-File-Änderungen
- produktivem Code

ist dieser Workflow sehr sinnvoll.

Ein typischer Ablauf:

1. Aufgabe analysieren lassen

2. Plan prüfen und ggf. korrigieren

3. Offene Fragen klären

4. Erst danach in den normalen oder Auto-Modus wechseln

Dadurch arbeitet Claude Code deutlich kontrollierter und die Qualität der Änderungen steigt spürbar."

20 Claude in anderen Werkzeugen

20.1 Claude for Word

Anthropic hat Claude direkt in Microsoft Word integriert. Änderungen erscheinen als Vorschläge mit Änderungsverfolgung.

Voraussetzungen

- Claude Team oder Enterprise (Pro auf Warteliste)
- Microsoft 365 (kein Word 2016/2019)
- Dateiformat .docx

Installation

6. marketplace.microsoft.com – Claude by Anthropic for Word suchen
7. Installieren, Word öffnen, Add-in aktivieren (Mac: Tools → Add-ins)
8. Mit Claude-Account einloggen

Wichtige Funktionen

- Texte mit Änderungsverfolgung bearbeiten (Formatierung bleibt erhalten)
- Dokument auf Inkonsistenzen scannen (Querverweise, Nummerierung, Begriffe)
- Fragen mit klickbaren Quellenangaben
- Templates ausfüllen mit übernommener Formatierung
- Cross-App: Word + Excel + PowerPoint in einem Gespräch

> **Tipp:** "Kürze diesen Absatz und entferne Passivkonstruktionen." / "Was ist die Haftungsobergrenze und gilt sie beidseitig?"

20.2 OpenAI Codex als zweites Modell

Das Codex-Plugin integriert die OpenAI Codex CLI direkt in Claude Code. Das ermöglicht Code-Reviews durch ein zweites, unabhängiges Modell – ohne die Claude-Session zu verlassen. KI-Modelle erkennen eigene Fehler schlechter als fremde – ein zweites Modell bringt frischen Blick.

> **Hinweis:** Es geht nicht um das alte Codex-Modell von OpenAI (eingestellt 2023), sondern um die OpenAI Codex CLI – ein terminal-basierter Coding-Agent von OpenAI (April 2025, Open Source).

20.2.1 Was das eigentlich ist

Die OpenAI Codex CLI ist ein eigenständiges Kommandozeilenwerkzeug von OpenAI, das ähnlich wie Claude Code funktioniert: es liest Code, analysiert ihn und gibt Feedback. Alle Reviews sind ausschließlich lesend (read-only) – Codex verändert keine Dateien.

20.2.2 Welchen Account brauche ich?

Kriterium	ChatGPT-Account	OpenAI API Key
Kosten	Kostenlos möglich (mit Limits)	Pay-per-Use, sehr günstig
Setup-Aufwand	Gering (Browser-Login)	Mittel (API Key besorgen)
Limits	Täglich begrenzt (Free)	Nur durch Guthaben begrenzt
Empfohlen wenn	Man bereits ChatGPT nutzt	Man häufig Reviews macht
Kosten pro Review	Kostenlos / inkl. in Plus	ca. 0,001–0,01 USD

20.2.3 Setup Schritt für Schritt

Schritt 1 – Voraussetzungen prüfen

```
node --version    # muss 18.18 oder neuer sein
```

Schritt 2 – Codex CLI installieren

```
npm install -g @openai/codex
```

Schritt 3 – Authentifizieren

```
# Option A: ChatGPT-Account
codex login
# Es öffnet sich der Browser → mit OpenAI-Account einloggen

# Option B: API Key
export OPENAI_API_KEY="sk-..."
```

Schritt 4 – Plugin in Claude Code installieren

```
/plugin marketplace add openai/codex-plugin-cc
/plugin install codex@openai-codex
```

```
/reload-plugins
```

Schritt 5 – Installation prüfen

```
/codex:setup    # zeigt ob Codex CLI gefunden wird und eingeloggt ist
```

> **Hinweis:** Falls die CLI nicht gefunden wird: Claude Code neu starten und /codex:setup erneut ausführen. Falls der Login abgelaufen ist: codex login erneut im Terminal ausführen.

20.2.4 Verfügbare Commands

Command	Was es tut	Wann nutzen
/codex:review	Neutraler Review der letzten Änderungen	Nach jedem Feature als Standard-Check
/codex:adversarial-review	Aggressiver Review – sucht gezielt nach Fehlern, Sicherheitslücken	Vor einem Merge oder Release
/codex:rescue	Aufgabe komplett an Codex delegieren	Für kleinere Teilaufgaben, die Claude gerade nicht löst
/codex:status	Zeigt laufende Hintergrund-Jobs	Wenn ein langer Review noch läuft
/codex:result	Holt Ergebnisse fertiger Jobs	Nach /codex:status
/codex:cancel	Bricht laufende Jobs ab	Wenn ein Review nicht mehr gebraucht wird

20.2.5 Wann lohnt sich das?

Das Plugin ist sinnvoll, wenn:

- Man bereits einen ChatGPT- oder OpenAI-Account hat (kein Extra-Aufwand)
- Man Code schreibt, der produktiv eingesetzt wird (Sicherheit, Korrektheit wichtig)
- Man vor einem Release noch einen unabhängigen Check will
- Man /codex:adversarial-review als letzten Gate vor einem Merge einbauen will

Das Plugin ist weniger sinnvoll, wenn:

- Man keinen OpenAI-Account hat und keinen anlegen will
- Man hauptsächlich experimentellen oder wegwerfbaren Code schreibt
- Die Claude-internen Reviews mit /ultrareview bereits ausreichen

> **Empfohlener Workflow:** Opus schreibt den Code, dann /codex:adversarial-review als zweites Urteil. Codex-Tokens kosten ca. 1/4 von Opus – Reviews sind nahezu kostenlos.

21 Lernpfad: Von Einsteiger zu Experte

Claude Code besitzt viele Funktionen und Erweiterungsmöglichkeiten. Wer versucht, sofort alles gleichzeitig einzurichten, verliert schnell den Überblick und versteht die eigene Konfiguration später oft nicht mehr vollständig. Deshalb empfiehlt es sich, Schritt für Schritt vorzugehen und neue Funktionen erst dann einzubauen, wenn die Grundlagen sicher verstanden sind:

Schritt 1 – Erstes echtes Feature

Ein überschaubares Feature aus dem eigenen Projekt suchen. Plan Mode an, Plan reviewen, dann Auto-Accept zum Bauen, am Ende /review.

Schritt 2 – CLAUDE.md schärfen

Nach zwei oder drei Sessions Muster bemerken. Diese Erkenntnisse gehören rein: iterativ, kurz, präzise.

Schritt 3 – Permissions sauber

Globale Deny-Regeln einmal anlegen. Projektspezifische Allow-Liste aus dem Alltag wachsen lassen.

Schritt 4 – Hooks für Mechanik

Mit dem Formatter anfangen. SwiftFormat (oder Prettier, Black) nach jedem Edit.

Schritt 5 – Skills für Wiederholungen

Was dreimal im Monat gebraucht wird, wird ein Skill. Erst anlegen, wenn der Bedarf gespürt wurde.

Schritt 6 – Subagents für Komplexes

Wenn Sessions zu lang werden, Reviews und Recherche in Subagents auslagern.

Schritt 7 – MCP gezielt einsetzen

Mit GitHub anfangen, zuerst nur lesende Operationen.

Schritt 8 – Fortgeschrittenes Setup

Worktrees, Sandbox, Routines, RTK und claude-mem für maximale Effizienz.

22 Was Claude Code nicht ist

Trotz der vielen Möglichkeiten besitzt Claude Code klare Grenzen. Einige Aufgaben benötigen weiterhin menschliche Kontrolle, technisches Verständnis oder zusätzliche Werkzeuge und Prozesse.

Kein Ersatz für eigenes Verständnis.

Wer den Code nicht versteht, den Claude produziert, baut sich technische Schulden.

Kein Partner für streng vertrauliche Arbeit auf Consumer-Plänen.

Auf Pro und Max geht jede gelesene Datei an Anthropic. Sensible Daten gehören in Team-, Enterprise- oder API-Setup.

Kein Echtzeit-Werkzeug.

Antworten brauchen ihre Zeit, gerade bei Opus. Für Live-Pair-Programming ist Claude zu langsam.

Keine Magie für sehr große Monorepos.

Mit Repos im Multi-Millionen-Zeilen-Bereich kommt Claude ins Stocken. Selektives Laden mit @-Mentions wird Pflicht.

Kein Ersatz für saubere Tests und CI.

Claude kann Tests schreiben – laufen lassen muss sie die Pipeline.

23 Troubleshooting

Gerade bei der ersten Einrichtung oder bei erweiterten Funktionen können Probleme auftreten. Die folgenden Lösungen decken typische Fehlerquellen rund um Installation, Plugins, MCP-Server, Permissions und Sessions ab.

claude: command not found nach der Installation

Terminal komplett schließen und neu öffnen. Falls nicht behoben: prüfen, ob ~/.local/bin im PATH liegt.

Browser-Login schlägt fehl

Häufige Ursachen: kein bezahlter Plan, im Browser anderer Account, abgelaufene Session. Lösung: claude logout, dann claude neu starten.

claude doctor meldet Fehler

Das Tool zeigt selbst, was fehlt. Häufigste Fälle: PATH unvollständig, fehlender Login, veraltete Version. claude update ausführen.

Permission-Prompts nerven

Permissions in .claude/settings.json pflegen. /permissions allowlist schlägt eine fertige Allow-Liste vor.

Plugin funktioniert nicht nach Installation

/reload-plugins ausführen oder Claude Code neu starten.

MCP-Server wird als nicht verfügbar angezeigt

/mcp prüft den Status. API-Keys als Umgebungsvariablen setzen. Node.js-Version prüfen (18+ empfohlen).

Codex-Plugin: /codex:setup schlägt fehl

Node.js auf Version 18.18+ prüfen. Codex CLI installieren: npm install -g @openai/codex. Dann codex login ausführen und /reload-plugins.

Routine schlägt beim API-Key-Zugriff fehl

API-Keys in den Environment Variables der Cloud Environment ablegen. Im Prompt explizit angeben: "Nutze den API Key aus den Environment Variables."

Migration auf neuen Rechner

~/.claude/ sichern ohne cache/. Auth-Tokens nicht mitnehmen – nach dem Wechsel neu einloggen.

24 Glossar

Im Laufe der Arbeit mit Claude Code tauchen viele spezielle Begriffe, Modi und Werkzeuge auf. Dieses Glossar fasst die wichtigsten Begriffe kurz und verständlich zusammen und dient als schnelle Nachschlagehilfe.

Begriff	Erklärung
Auto-Accept	Modus, in dem Claude ohne Rückfrage handelt, sofern nicht durch Deny gesperrt. Per Shift+Tab aktivieren.
Auto Mode	Erweiterter autonomer Modus (Max-Plan). Claude trifft alle Entscheidungen selbstständig.
CLAUDE.md	Markdown-Datei, die bei jeder Session automatisch geladen wird. Enthält Projektkontext, Build-Befehle, Konventionen und Verbote.
Codex CLI	Open-Source-Coding-Agent von OpenAI (erschienen April 2025). Wird über das Codex-Plugin in Claude Code für unabhängige Code-Reviews genutzt.
Effort-Level	Steuert, wie lange Opus 4.7 über ein Problem nachdenkt. Von low bis max.
Hook	Shell-Befehl, der bei einem Lifecycle-Event (PostToolUse, PreToolUse, Stop) automatisch ausgeführt wird.
Loop	Schleife per /loop, die einen Befehl in Intervallen ausführt.
MCP-Server	Externer Service (Model Context Protocol), der Claude neue Werkzeuge bereitstellt.
Permissions	Allow-/Deny-Regeln in settings.json, die festlegen, was Claude darf oder nicht darf.
Plan Mode	Read-only-Modus, in dem Claude nur analysiert und plant. Per Shift+Tab.
Plugin	Vorgefertigtes Bündel aus mehreren Bausteinen (Skills, Hooks, Subagents, Settings).
Routine	Cloud-Automation, die vollständig autonom und ohne lokalen Server läuft.
RTK	Rust Token Killer – CLI-Proxy, der Befehlsausgaben komprimiert (60–90 % weniger Tokens).
Sandbox	Technisch abgegrenzter Bereich mit Filesystem- und Netzwerk-Isolation.
Schedule	Geplanter Task per /schedule mit Cron-Syntax.
Skill	Wiederverwendbarer Custom-Command in skills/<name>/SKILL.md, per /name aufrufbar.
Subagent	Spezialisierter Agent mit eigenem Kontext, der nur eine Zusammenfassung zurückgibt.
Worktree	Isolierter Git-Checkout für parallele Arbeit in mehreren Claude-Instanzen.

Die Inhalte dieser Website basieren auf eigenen Erfahrungen sowie auf öffentlich verfügbaren Informationen aus Dokumentationen, Schulungen, Videos und Community-Beiträgen. Alle Inhalte wurden eigenständig recherchiert, strukturiert und in eigenen Worten formuliert. Geschützte Inhalte werden nicht wörtlich übernommen