

Claude Code

Vollständige Dokumentation

By Anthropic

Erstellungsdatum : 2026-03-17

Ein agentisches Programmierwerkzeug in Ihrem Terminal,
das Ihre Codebasis versteht und Ihnen hilft, schneller zu programmieren.

Kompiliert von code.claude.com/docs/de
Für die neueste Version besuchen Sie die Website.

Generated by: github.com/bepsvpt/cc-docs

Table of Contents

Part 1: Getting Started	5
Claude Code Übersicht	5
Schnellstart	
Erste Schritte mit der Desktop-App	
Claude Code einrichten	25
So funktioniert Claude Code	35
Part 2: Core Usage	46
Claude Code erweitern	46
Interaktiver Modus	59
Best Practices für Claude Code	
Häufige Arbeitsabläufe	97
Part 3: Commands & Reference	
CLI-Referenz	122
Tastaturkürzel anpassen	133
Part 4: Configuration	145
Berechtigungen konfigurieren	145
Wie Claude sich Ihr Projekt merkt	
Claude Code-Einstellungen	169
Modellkonfiguration	220
Ausgabestile	229
Beschleunigen Sie Antworten mit dem Schnellmodus	
Kosten effektiv verwalten	238
Part 5: Extensibility	246
Hooks-Referenz	246
Automatisieren Sie Workflows mit Hooks	
Claude Code mit Tools über MCP verbinden	
Claude mit Skills erweitern	377
Plugins erstellen	402
Plugins-Referenz	415
Erstellen und Verteilen eines Plugin-Marktplatzes	
Entdecken und installieren Sie vorgefertigte Plugins über Marktplätze	
Part 6: Advanced & Automation	480
Benutzerdefinierte Subagenten erstellen	

Orchestrieren Sie Teams von Claude Code-Sitzungen	
Claude Code programmgesteuert ausführen	
Lokale Sitzungen von jedem Gerät aus mit Remote Control fortsetzen	
Prompts nach Zeitplan ausführen	
Code Review	539

Part 7: CI/CD & Integrations

Claude Code GitHub Actions	546
Claude Code GitLab CI/CD	566

Part 8: IDE & Platform Integration 581

Claude Code in VS Code verwenden	
JetBrains IDEs	600
Claude Code Desktop verwenden	
Claude Code mit Chrome verwenden (Beta)	
Claude Code in Slack	640
Claude Code im Web	647

Part 9: Security & Privacy

Sicherheit	668
Sandboxing	674
Checkpointing	685
Datennutzung	688
Rechtliche Bestimmungen und Compliance	
Null-Datenspeicherung	697

Part 10: Enterprise & Monitoring 701

Teamnutzung mit Analysen verfolgen	
Überwachung	709
Übersicht zur Enterprise-Bereitstellung	
Serververwaltete Einstellungen konfigurieren (öffentliche Beta)	

Part 11: Cloud Providers 740

Claude Code auf Amazon Bedrock	
Claude Code auf Google Vertex AI	
Claude Code auf Microsoft Foundry	
LLM-Gateway-Konfiguration	757
Enterprise-Netzwerkkonfiguration	762

Part 12: Environment Setup 765

Entwicklungscontainer	765
Optimieren Sie Ihr Terminal-Setup	

Passen Sie Ihre Statuszeile an

Part 13: Troubleshooting & Changelog

803

Fehlerbehebung

603

Part 1: Getting Started

Claude Code Übersicht

Claude Code ist ein agentengestütztes Codierungswerkzeug, das Ihre Codebasis liest, Dateien bearbeitet, Befehle ausführt und sich in Ihre Entwicklungstools integriert. Verfügbar in Ihrem Terminal, IDE, Desktop-App und Browser.

Claude Code ist ein KI-gestützter Codierassistent, der Ihnen hilft, Funktionen zu erstellen, Fehler zu beheben und Entwicklungsaufgaben zu automatisieren. Er versteht Ihre gesamte Codebasis und kann über mehrere Dateien und Tools hinweg arbeiten, um Aufgaben zu erledigen.

Erste Schritte

Wählen Sie Ihre Umgebung, um zu beginnen. Die meisten Oberflächen erfordern ein [Claude-Abonnement](#) oder ein [Anthropic Console](#)-Konto. Das Terminal CLI und VS Code unterstützen auch [Drittanbieter](#).

Terminal

Das vollständig ausgestattete CLI für die Arbeit mit Claude Code direkt in Ihrem Terminal. Bearbeiten Sie Dateien, führen Sie Befehle aus und verwalten Sie Ihr gesamtes Projekt von der Befehlszeile aus.

To install Claude Code, use one of the following methods:

Native Install (Recommended)

macOS, Linux, WSL:

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

Windows PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del  
install.cmd
```

Windows requires [Git for Windows](#). Install it first if you don't have it.

Info:

Native installations automatically update in the background to keep you on the latest version.

Homebrew

```
brew install --cask claude-code
```

Info:

Homebrew installations do not auto-update. Run `brew upgrade claude-code` periodically to get the latest features and security fixes.

WinGet

```
winget install Anthropic.ClaudeCode
```

Info:

WinGet installations do not auto-update. Run `winget upgrade Anthropic.ClaudeCode` periodically to get the latest features and security fixes.

Starten Sie dann Claude Code in einem beliebigen Projekt:

```
cd your-project  
claude
```

Sie werden beim ersten Mal aufgefordert, sich anzumelden. Das ist alles! [Fahren Sie mit dem Quickstart fort →](#)

Tip:

Siehe [erweiterte Einrichtung](#) für Installationsoptionen, manuelle Updates oder Deinstallationsanweisungen. Besuchen Sie [Fehlerbehebung](#), wenn Sie auf Probleme stoßen.

VS Code

Die VS Code-Erweiterung bietet Inline-Diffs, @-Erwähnungen, Planüberprüfung und Gesprächsverlauf direkt in Ihrem Editor.

- [Für VS Code installieren](#)
- [Für Cursor installieren](#)

Oder suchen Sie nach „Claude Code“ in der Ansicht „Erweiterungen“ (`Cmd+Shift+X` auf Mac, `Ctrl+Shift+X` auf Windows/Linux). Nach der Installation öffnen Sie die Befehlspalette (`Cmd+Shift+P` / `Ctrl+Shift+P`), geben Sie „Claude Code“ ein und wählen Sie **In neuem Tab öffnen**.

[Erste Schritte mit VS Code →](#)

Desktop-App

Eine eigenständige App für die Ausführung von Claude Code außerhalb Ihrer IDE oder Ihres Terminals. Überprüfen Sie Diffs visuell, führen Sie mehrere Sitzungen nebeneinander aus, planen Sie wiederkehrende Aufgaben und starten Sie Cloud-Sitzungen.

Herunterladen und installieren:

- [macOS](#) (Intel und Apple Silicon)
- [Windows](#) (x64)
- [Windows ARM64](#) (nur Remote-Sitzungen)

Nach der Installation starten Sie Claude, melden Sie sich an und klicken Sie auf die Registerkarte **Code**, um mit dem Codieren zu beginnen. Ein [bezahltes Abonnement](#) ist erforderlich.

[Weitere Informationen zur Desktop-App →](#)

Web

Führen Sie Claude Code in Ihrem Browser ohne lokale Einrichtung aus. Starten Sie lang laufende Aufgaben und überprüfen Sie sie später, arbeiten Sie an Repositories, die Sie nicht lokal haben, oder führen Sie mehrere Aufgaben parallel aus. Verfügbar auf Desktop-Browsern und der Claude iOS-App.

Beginnen Sie mit dem Codieren unter claude.ai/code.

[Erste Schritte im Web →](#)

JetBrains

Ein Plugin für IntelliJ IDEA, PyCharm, WebStorm und andere JetBrains-IDEs mit interaktiver Diff-Anzeige und Auswahlkontext-Freigabe.

Installieren Sie das [Claude Code-Plugin](#) aus dem JetBrains Marketplace und starten Sie Ihre IDE neu.

[Erste Schritte mit JetBrains →](#)

Was Sie tun können

Hier sind einige der Möglichkeiten, wie Sie Claude Code nutzen können:

Claude Code übernimmt die mühsamen Aufgaben, die Ihren Tag aufzehren: Schreiben von Tests für ungetesteten Code, Beheben von Lint-Fehlern in einem Projekt, Auflösen von Merge-Konflikten, Aktualisieren von Abhängigkeiten und Schreiben von Versionshinweisen.

```
claude "write tests for the auth module, run them, and fix any failures"
```

Beschreiben Sie, was Sie möchten, in einfacher Sprache. Claude Code plant den Ansatz, schreibt den Code über mehrere Dateien hinweg und überprüft, ob er funktioniert.

Bei Fehlern fügen Sie eine Fehlermeldung ein oder beschreiben Sie das Symptom. Claude Code verfolgt das Problem durch Ihre Codebasis, identifiziert die Grundursache und implementiert eine Lösung. Weitere Beispiele finden Sie unter [häufige Workflows](#).

Claude Code arbeitet direkt mit git. Es stellt Änderungen bereit, schreibt Commit-Nachrichten, erstellt Branches und öffnet Pull Requests.

```
claude "commit my changes with a descriptive message"
```

In CI können Sie Code-Reviews und Issue-Triage mit [GitHub Actions](#) oder [GitLab CI/CD](#) automatisieren.

Das [Model Context Protocol \(MCP\)](#) ist ein offener Standard für die Verbindung von KI-Tools mit externen Datenquellen. Mit MCP kann Claude Code Ihre Design-Dokumente in Google Drive lesen, Tickets in Jira aktualisieren, Daten aus Slack abrufen oder Ihre eigenen benutzerdefinierten Tools verwenden.

`CLAUDE.md` ist eine Markdown-Datei, die Sie im Projektstammverzeichnis hinzufügen und die Claude Code zu Beginn jeder Sitzung liest. Verwenden Sie sie, um Codierungsstandards, Architekturentscheidungen, bevorzugte Bibliotheken und Überprüfungschecklisten festzulegen. Claude erstellt auch [automatisches Gedächtnis](#), während es arbeitet, und speichert Erkenntnisse wie Build-Befehle und Debugging-Einblicke über Sitzungen hinweg, ohne dass Sie etwas schreiben müssen.

Erstellen Sie [benutzerdefinierte Befehle](#), um wiederholbare Workflows zu verpacken, die Ihr Team teilen kann, wie `/review-pr` oder `/deploy-staging`.

[Hooks](#) ermöglichen es Ihnen, Shell-Befehle vor oder nach Claude Code-Aktionen auszuführen, wie automatische Formatierung nach jeder Dateibearbeitung oder Ausführung von Lint vor einem Commit.

Starten Sie [mehrere Claude Code-Agents](#), die gleichzeitig an verschiedenen Teilen einer Aufgabe arbeiten. Ein Lead-Agent koordiniert die Arbeit, weist Unteraufgaben zu und führt Ergebnisse zusammen.

Für vollständig benutzerdefinierte Workflows ermöglicht das [Agent SDK](#) Ihnen, Ihre eigenen Agents zu erstellen, die von Claude Code's Tools und Funktionen angetrieben werden, mit vollständiger Kontrolle über Orchestrierung, Tool-Zugriff und Berechtigungen.

Claude Code ist zusammensetzbar und folgt der Unix-Philosophie. Leiten Sie Logs darin ein, führen Sie es in CI aus oder verketteten Sie es mit anderen Tools:

```
## Monitor logs and get alerted
tail -f app.log | claude -p "Slack me if you see any anomalies"

## Automate translations in CI
claude -p "translate new strings into French and raise a PR for review"

## Bulk operations across files
git diff main --name-only | claude -p "review these changed files for security issues"
```

Siehe die [CLI-Referenz](#) für den vollständigen Satz von Befehlen und Flags.

Sitzungen sind nicht an eine einzelne Oberfläche gebunden. Verschieben Sie Arbeit zwischen Umgebungen, wenn sich Ihr Kontext ändert:

- Treten Sie von Ihrem Schreibtisch zurück und arbeiten Sie weiter von Ihrem Telefon oder einem beliebigen Browser mit [Remote Control](#)

- Starten Sie eine lang laufende Aufgabe im [Web](#) oder in der [iOS-App](#) und ziehen Sie sie mit `/teleport` in Ihr Terminal
- Übergeben Sie eine Terminal-Sitzung an die [Desktop-App](#) mit `/desktop` für visuelle Diff-Überprüfung
- Leiten Sie Aufgaben aus Team-Chat weiter: Erwähnen Sie `@Claude` in [Slack](#) mit einem Fehlerbericht und erhalten Sie einen Pull Request zurück

Verwenden Sie Claude Code überall

Jede Oberfläche verbindet sich mit der gleichen zugrunde liegenden Claude Code-Engine, daher funktionieren Ihre CLAUDE.md-Dateien, Einstellungen und MCP-Server auf allen Oberflächen.

Über die oben genannten Umgebungen [Terminal](#), [VS Code](#), [JetBrains](#), [Desktop](#) und [Web](#) hinaus integriert sich Claude Code mit CI/CD-, Chat- und Browser-Workflows:

Ich möchte...	Beste Option
Eine lokale Sitzung von meinem Telefon oder einem anderen Gerät fortsetzen	Remote Control
Eine Aufgabe lokal starten, auf dem Mobilgerät fortsetzen	Web oder Claude iOS-App
PR-Reviews und Issue-Triage automatisieren	GitHub Actions oder GitLab CI/CD
Automatische Code-Überprüfung bei jedem PR erhalten	GitHub Code Review
Fehlerberichte von Slack zu Pull Requests weiterleiten	Slack
Live-Webanwendungen debuggen	Chrome
Benutzerdefinierte Agents für Ihre eigenen Workflows erstellen	Agent SDK

Nächste Schritte

Nachdem Sie Claude Code installiert haben, helfen Ihnen diese Leitfäden, tiefer einzusteigen.

- [Quickstart](#): Gehen Sie durch Ihre erste echte Aufgabe, vom Erkunden einer Codebasis bis zum Committen einer Lösung
- [Speichern Sie Anweisungen und Erinnerungen](#): Geben Sie Claude persistente Anweisungen mit CLAUDE.md-Dateien und automatischem Gedächtnis
- [Häufige Workflows](#) und [Best Practices](#): Muster für die optimale Nutzung von Claude Code
- [Einstellungen](#): Passen Sie Claude Code an Ihren Workflow an
- [Fehlerbehebung](#): Lösungen für häufige Probleme
- code.claude.com: Demos, Preise und Produktdetails

Schnellstart

Willkommen bei Claude Code!

Diese Schnellstartanleitung ermöglicht es Ihnen, in wenigen Minuten KI-gestützte Codierungshilfe zu nutzen. Am Ende werden Sie verstehen, wie Sie Claude Code für häufige Entwicklungsaufgaben einsetzen.

Bevor Sie beginnen

Stellen Sie sicher, dass Sie folgende Voraussetzungen erfüllen:

- Ein offenes Terminal oder eine offene Eingabeaufforderung
 - Wenn Sie das Terminal noch nie verwendet haben, lesen Sie den [Terminal-Leitfaden](#)
- Ein Codeprojekt zum Arbeiten
- Ein [Claude-Abonnement](#) (Pro, Max, Teams oder Enterprise), ein [Claude Console](#)-Konto oder Zugriff über einen [unterstützten Cloud-Anbieter](#)

Note:

Diese Anleitung behandelt die Terminal-CLI. Claude Code ist auch im [Web](#) verfügbar, als [Desktop-App](#), in [VS Code](#) und [JetBrains IDEs](#), in [Slack](#) und in CI/CD mit [GitHub Actions](#) und [GitLab](#). Siehe [alle Schnittstellen](#).

Schritt 1: Claude Code installieren

To install Claude Code, use one of the following methods:

Native Install (Recommended)

macOS, Linux, WSL:

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

Windows PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del  
install.cmd
```

Windows requires [Git for Windows](#). Install it first if you don't have it.

Info:

Native installations automatically update in the background to keep you on the latest version.

Homebrew

```
brew install --cask claude-code
```

Info:

Homebrew installations do not auto-update. Run `brew upgrade claude-code` periodically to get the latest features and security fixes.

WinGet

```
winget install Anthropic.ClaudeCode
```

Info:

WinGet installations do not auto-update. Run `winget upgrade Anthropic.ClaudeCode` periodically to get the latest features and security fixes.

Schritt 2: Melden Sie sich bei Ihrem Konto an

Claude Code erfordert ein Konto zur Nutzung. Wenn Sie eine interaktive Sitzung mit dem Befehl `claude` starten, müssen Sie sich anmelden:

```
claude
```

```
## Sie werden beim ersten Gebrauch aufgefordert, sich anzumelden
```

```
/login
```

```
## Folgen Sie den Aufforderungen, um sich mit Ihrem Konto anzumelden
```

Sie können sich mit einem dieser Kontotypen anmelden:

- [Claude Pro, Max, Teams oder Enterprise](#) (empfohlen)
- [Claude Console](#) (API-Zugriff mit Prepaid-Guthaben). Bei der ersten Anmeldung wird automatisch ein „Claude Code“-Arbeitsbereich in der Console erstellt, um die Kosten zentral zu verfolgen.
- [Amazon Bedrock, Google Vertex AI oder Microsoft Foundry](#) (Enterprise-Cloud-Anbieter)

Nach der Anmeldung werden Ihre Anmeldedaten gespeichert und Sie müssen sich nicht erneut anmelden. Um später zu einem anderen Konto zu wechseln, verwenden Sie den Befehl `/login`.

Schritt 3: Starten Sie Ihre erste Sitzung

Öffnen Sie Ihr Terminal in einem beliebigen Projektverzeichnis und starten Sie Claude Code:

```
cd /path/to/your/project  
claude
```

Sie sehen den Claude Code-Willkommensbildschirm mit Ihren Sitzungsinformationen, kürzlichen Unterhaltungen und den neuesten Updates. Geben Sie `/help` ein, um verfügbare Befehle anzuzeigen, oder `/resume`, um eine vorherige Unterhaltung fortzusetzen.

Tip:

Nach der Anmeldung (Schritt 2) werden Ihre Anmeldedaten auf Ihrem System gespeichert. Weitere Informationen finden Sie unter [Verwaltung von Anmeldedaten](#).

Schritt 4: Stellen Sie Ihre erste Frage

Beginnen Sie damit, Ihre Codebasis zu verstehen. Versuchen Sie einen dieser Befehle:

```
what does this project do?
```

Claude wird Ihre Dateien analysieren und eine Zusammenfassung bereitstellen. Sie können auch spezifischere Fragen stellen:

```
what technologies does this project use?
```

```
where is the main entry point?
```

```
explain the folder structure
```

Sie können Claude auch nach seinen eigenen Fähigkeiten fragen:

```
what can Claude Code do?
```

```
how do I create custom skills in Claude Code?
```

```
can Claude Code work with Docker?
```

Note:

Claude Code liest Ihre Projektdateien nach Bedarf. Sie müssen den Kontext nicht manuell hinzufügen.

Schritt 5: Nehmen Sie Ihre erste Codeänderung vor

Jetzt lassen Sie Claude Code tatsächlich programmieren. Versuchen Sie eine einfache Aufgabe:

```
add a hello world function to the main file
```

Claude Code wird:

1. Die entsprechende Datei finden
2. Die vorgeschlagenen Änderungen anzeigen
3. Um Ihre Genehmigung bitten
4. Die Bearbeitung durchführen

Note:

Claude Code fragt immer um Erlaubnis, bevor Dateien geändert werden. Sie können einzelne Änderungen genehmigen oder den Modus „Alle akzeptieren“ für eine Sitzung aktivieren.

Schritt 6: Verwenden Sie Git mit Claude Code

Claude Code macht Git-Operationen gesprächig:

```
what files have I changed?
```

```
commit my changes with a descriptive message
```

Sie können auch komplexere Git-Operationen anfordern:

```
create a new branch called feature/quickstart
```

```
show me the last 5 commits
```

```
help me resolve merge conflicts
```

Schritt 7: Beheben Sie einen Fehler oder fügen Sie eine Funktion hinzu

Claude ist versiert im Debuggen und in der Implementierung von Funktionen.

Beschreiben Sie, was Sie möchten, in natürlicher Sprache:

```
add input validation to the user registration form
```

Oder beheben Sie vorhandene Probleme:

```
there's a bug where users can submit empty forms - fix it
```

Claude Code wird:

- Den relevanten Code lokalisieren
- Den Kontext verstehen
- Eine Lösung implementieren
- Tests ausführen, falls verfügbar

Schritt 8: Testen Sie andere häufige Arbeitsabläufe

Es gibt verschiedene Möglichkeiten, mit Claude zu arbeiten:

Code umgestalten

```
refactor the authentication module to use async/await instead of callbacks
```

Tests schreiben

```
write unit tests for the calculator functions
```

Dokumentation aktualisieren

```
update the README with installation instructions
```

Code-Überprüfung

```
review my changes and suggest improvements
```

Tip:

Sprechen Sie mit Claude wie mit einem hilfreichen Kollegen. Beschreiben Sie, was Sie erreichen möchten, und es wird Ihnen helfen, dorthin zu gelangen.

Wesentliche Befehle

Hier sind die wichtigsten Befehle für die tägliche Nutzung:

Befehl	Was er tut	Beispiel
<code>claude</code>	Interaktiven Modus starten	<code>claude</code>
<code>claude "task"</code>	Eine einmalige Aufgabe ausführen	<code>claude "fix the build error"</code>
<code>claude -p "query"</code>	Einmalige Abfrage ausführen und dann beenden	<code>claude -p "explain this function"</code>
<code>claude -c</code>	Letzte Unterhaltung im aktuellen Verzeichnis fortsetzen	<code>claude -c</code>
<code>claude -r</code>	Eine vorherige Unterhaltung fortsetzen	<code>claude -r</code>
<code>claude commit</code>	Einen Git-Commit erstellen	<code>claude commit</code>
<code>/clear</code>	Unterhaltungsverlauf löschen	<code>/clear</code>
<code>/help</code>	Verfügbare Befehle anzeigen	<code>/help</code>
<code>exit</code> oder Strg+C	Claude Code beenden	<code>exit</code>

Siehe die [CLI-Referenz](#) für eine vollständige Liste der Befehle.

Profitipps für Anfänger

Weitere Informationen finden Sie unter [Best Practices](#) und [häufige Arbeitsabläufe](#).

Statt: 'Beheben Sie den Fehler'

Versuchen Sie: „Beheben Sie den Login-Fehler, bei dem Benutzer einen leeren Bildschirm sehen, nachdem sie falsche Anmeldedaten eingegeben haben“

Unterteilen Sie komplexe Aufgaben in Schritte:

1. create a new database table for user profiles
2. create an API endpoint to get and update user profiles
3. build a webpage that allows users to see and edit their information

Bevor Sie Änderungen vornehmen, lassen Sie Claude Ihren Code verstehen:

analyze the database schema

build a dashboard showing products that are most frequently returned by our UK customers

- Drücken Sie `?`, um alle verfügbaren Tastaturkürzel anzuzeigen
- Verwenden Sie Tab für Befehlsvervollständigung
- Drücken Sie `↑` für Befehlsverlauf
- Geben Sie `/` ein, um alle Befehle und skills anzuzeigen

Was kommt als Nächstes?

Nachdem Sie die Grundlagen gelernt haben, erkunden Sie erweiterte Funktionen:

- [Wie Claude Code funktioniert](#) Verstehen Sie die agentengestützte Schleife, integrierte Tools und wie Claude Code mit Ihrem Projekt interagiert
- [Best Practices](#) Erzielen Sie bessere Ergebnisse mit effektivem Prompting und Projekt-setup
- [Häufige Arbeitsabläufe](#) Schritt-für-Schritt-Anleitungen für häufige Aufgaben
- [Claude Code erweitern](#) Passen Sie mit CLAUDE.md, skills, hooks, MCP und mehr an

Hilfe erhalten

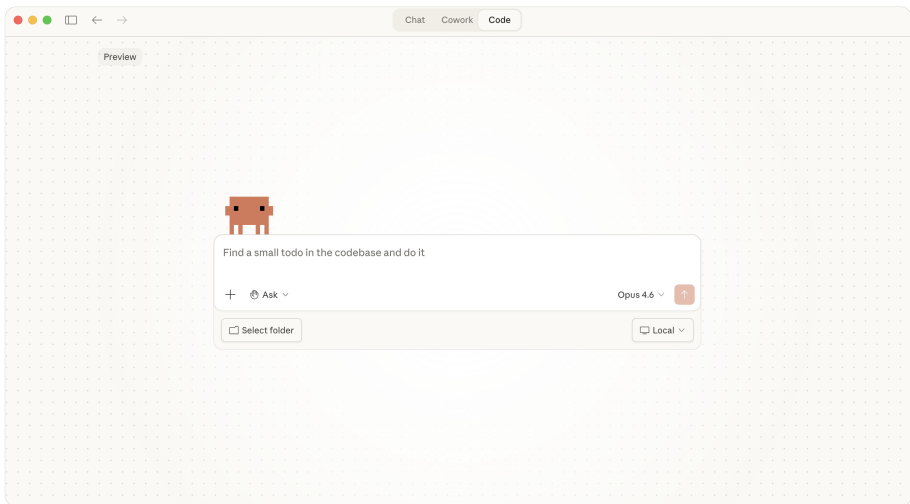
- **In Claude Code:** Geben Sie `/help` ein oder fragen Sie „how do I...“
- **Dokumentation:** Sie sind hier! Durchsuchen Sie andere Leitfäden
- **Community:** Treten Sie unserem [Discord](#) bei, um Tipps und Support zu erhalten

Erste Schritte mit der Desktop-App

Installieren Sie Claude Code auf dem Desktop und starten Sie Ihre erste Coding-Sitzung

Die Desktop-App bietet Ihnen Claude Code mit einer grafischen Benutzeroberfläche: visuelle Diff-Überprüfung, Live-App-Vorschau, GitHub-PR-Überwachung mit automatischem Merge, parallele Sitzungen mit Git-Worktree-Isolation, geplante Aufgaben und die Möglichkeit, Aufgaben remote auszuführen. Kein Terminal erforderlich.

Diese Seite führt Sie durch die Installation der App und den Start Ihrer ersten Sitzung. Wenn Sie bereits eingerichtet sind, siehe [Claude Code Desktop verwenden](#) für die vollständige Referenz.



Die Claude Code Desktop-Oberfläche mit der ausgewählten Registerkarte Code, mit einem Eingabefeld, einem Berechtigungsmodus-Selektor auf Berechtigungen erfragen, einem Modellwähler, einem Ordnerwähler und der Option Lokale Umgebung

Die Desktop-App hat drei Registerkarten:

- **Chat:** Allgemeine Konversation ohne Dateizugriff, ähnlich wie `claude.ai`.
- **Cowork:** Ein autonomer Hintergrund-Agent, der an Aufgaben in einer Cloud-VM mit eigener Umgebung arbeitet. Er kann unabhängig arbeiten, während Sie andere Aufgaben erledigen.

- **Code:** Ein interaktiver Coding-Assistent mit direktem Zugriff auf Ihre lokalen Dateien. Sie überprüfen und genehmigen jede Änderung in Echtzeit.

Chat und Cowork werden in den [Claude Desktop-Supportartikeln](#) behandelt. Diese Seite konzentriert sich auf die Registerkarte **Code**.

Note:

Claude Code erfordert ein [Pro-, Max-, Teams- oder Enterprise-Abonnement](#).

Installation

Step 1: App herunterladen

Laden Sie Claude für Ihre Plattform herunter.

- [macOS](#) Universeller Build für Intel und Apple Silicon
- [Windows](#) Für x64-Prozessoren

Für Windows ARM64 [hier herunterladen](#).

Linux wird derzeit nicht unterstützt.

Step 2: Anmelden

Starten Sie Claude aus Ihrem Anwendungsordner (macOS) oder Startmenü (Windows). Melden Sie sich mit Ihrem Anthropic-Konto an.

Step 3: Öffnen Sie die Registerkarte Code

Klicken Sie auf die Registerkarte **Code** oben in der Mitte. Wenn Sie beim Klicken auf „Code“ aufgefordert werden, ein Upgrade durchzuführen, müssen Sie zunächst [ein bezahltes Abonnement abschließen](#). Wenn Sie aufgefordert werden, sich online anzumelden, schließen Sie die Anmeldung ab und starten Sie die App neu. Wenn Sie einen 403-Fehler sehen, siehe [Authentifizierungsfehlersuche](#).

Die Desktop-App enthält Claude Code. Sie müssen Node.js oder die CLI nicht separat installieren. Um **claude** vom Terminal aus zu verwenden, installieren Sie die CLI separat. Siehe [Erste Schritte mit der CLI](#).

Starten Sie Ihre erste Sitzung

Wählen Sie mit der geöffneten Registerkarte „Code“ ein Projekt aus und geben Sie Claude etwas zu tun.

Step 1: Wählen Sie eine Umgebung und einen Ordner

Wählen Sie **Lokal**, um Claude auf Ihrem Computer mit Ihren Dateien direkt auszuführen. Klicken Sie auf **Ordner auswählen** und wählen Sie Ihr Projektverzeichnis.

Tip:

Beginnen Sie mit einem kleinen Projekt, das Sie gut kennen. Es ist die schnellste Möglichkeit zu sehen, was Claude Code kann. Unter Windows muss [Git](#) für lokale Sitzungen installiert sein. Die meisten Macs enthalten Git standardmäßig.

Sie können auch auswählen:

- **Remote:** Führen Sie Sitzungen auf der Cloud-Infrastruktur von Anthropic aus, die auch dann fortgesetzt werden, wenn Sie die App schließen. Remote-Sitzungen verwenden die gleiche Infrastruktur wie [Claude Code im Web](#).
- **SSH:** Verbinden Sie sich über SSH mit einem Remote-Computer (Ihre eigenen Server, Cloud-VMs oder Dev-Container). Claude Code muss auf dem Remote-Computer installiert sein.

Step 2: Wählen Sie ein Modell

Wählen Sie ein Modell aus der Dropdown-Liste neben der Schaltfläche „Senden“. Siehe [Modelle](#) für einen Vergleich von Opus, Sonnet und Haiku. Sie können das Modell nach dem Start der Sitzung nicht ändern.

Step 3: Sagen Sie Claude, was zu tun ist

Geben Sie ein, was Claude tun soll:

- `Find a TODO comment and fix it`
- `Add tests for the main function`
- `Create a CLAUDE.md with instructions for this codebase`

Eine [Sitzung](#) ist eine Konversation mit Claude über Ihren Code. Jede Sitzung verfolgt ihren eigenen Kontext und ihre Änderungen, sodass Sie an mehreren Aufgaben arbeiten können, ohne dass sie sich gegenseitig beeinflussen.

Step 4: Überprüfen und akzeptieren Sie Änderungen

Standardmäßig startet die Registerkarte „Code“ im [Modus „Berechtigungen erfragen“](#), in dem Claude Änderungen vorschlägt und auf Ihre Genehmigung wartet, bevor er sie anwendet. Sie sehen:

1. Eine [Diff-Ansicht](#), die genau zeigt, was sich in jeder Datei ändert

2. Schaltflächen „Akzeptieren“/„Ablehnen“, um jede Änderung zu genehmigen oder abzulehnen
3. Echtzeit-Updates, während Claude Ihre Anfrage bearbeitet

Wenn Sie eine Änderung ablehnen, fragt Claude, wie Sie anders vorgehen möchten. Ihre Dateien werden erst geändert, wenn Sie sie akzeptieren.

Was nun?

Sie haben Ihre erste Bearbeitung vorgenommen. Für die vollständige Referenz zu allem, was Desktop kann, siehe [Claude Code Desktop verwenden](#). Hier sind einige Dinge, die Sie als Nächstes versuchen können.

Unterbrechen und lenken. Sie können Claude jederzeit unterbrechen. Wenn es den falschen Weg geht, klicken Sie auf die Stoppschaltfläche oder geben Sie Ihre Korrektur ein und drücken Sie **Eingabe**. Claude stoppt, was es tut, und passt sich basierend auf Ihrer Eingabe an. Sie müssen nicht warten, bis es fertig ist, oder von vorne anfangen.

Geben Sie Claude mehr Kontext. Geben Sie `@filename` im Eingabefeld ein, um eine bestimmte Datei in die Konversation zu ziehen, fügen Sie Bilder und PDFs mit der Schaltfläche „Anhang“ an, oder ziehen Sie Dateien direkt in das Eingabefeld. Je mehr Kontext Claude hat, desto besser sind die Ergebnisse. Siehe [Dateien und Kontext zu Eingaben hinzufügen](#).

Verwenden Sie Skills für wiederholbare Aufgaben. Geben Sie `/` ein oder klicken Sie auf `+` → **Slash commands**, um [integrierte Befehle](#), [benutzerdefinierte Skills](#) und Plugin-Skills zu durchsuchen. Skills sind wiederverwendbare Eingaben, die Sie jederzeit aufrufen können, wie Code-Review-Checklisten oder Bereitstellungsschritte.

Überprüfen Sie Änderungen vor dem Commit. Nachdem Claude Dateien bearbeitet hat, wird ein `+12 -1`-Indikator angezeigt. Klicken Sie darauf, um die [Diff-Ansicht](#) zu öffnen, überprüfen Sie Änderungen Datei für Datei und kommentieren Sie bestimmte Zeilen. Claude liest Ihre Kommentare und überarbeitet. Klicken Sie auf **Code überprüfen**, um Claude die Diffs selbst auswerten zu lassen und Inline-Vorschläge zu hinterlassen.

Passen Sie an, wie viel Kontrolle Sie haben. Ihr [Berechtigungsmodus](#) steuert das Gleichgewicht. „Berechtigungen erfragen“ (Standard) erfordert Genehmigung vor jeder Bearbeitung. „Auto-Akzeptanz“ akzeptiert Dateibearbeitungen automatisch für schnellere Iteration. Plan Mode lässt Claude einen Ansatz planen, ohne Dateien zu berühren, was vor einem großen Refactoring nützlich ist.

Fügen Sie Plugins für mehr Funktionen hinzu. Klicken Sie auf die Schaltfläche + neben dem Eingabefeld und wählen Sie **Plugins**, um [Plugins](#) zu durchsuchen und zu installieren, die Skills, Agents, MCP servers und mehr hinzufügen.

Zeigen Sie eine Vorschau Ihrer App an. Klicken Sie auf das Dropdown-Menü **Vorschau**, um Ihren Dev-Server direkt im Desktop auszuführen. Claude kann die laufende App anzeigen, Endpunkte testen, Protokolle überprüfen und auf das, was es sieht, iterieren. Siehe [Zeigen Sie eine Vorschau Ihrer App an](#).

Verfolgen Sie Ihren Pull Request. Nachdem Sie einen PR geöffnet haben, überwacht Claude Code die CI-Prüfungsergebnisse und kann Fehler automatisch beheben oder den PR zusammenführen, sobald alle Prüfungen bestanden sind. Siehe [Überwachen Sie den Pull-Request-Status](#).

Setzen Sie Claude auf einen Zeitplan. Richten Sie [geplante Aufgaben](#) ein, um Claude automatisch regelmäßig auszuführen: eine tägliche Code-Überprüfung jeden Morgen, eine wöchentliche Abhängigkeitsprüfung oder eine Zusammenfassung, die von Ihren verbundenen Tools abrufen.

Skalieren Sie auf, wenn Sie bereit sind. Öffnen Sie [parallele Sitzungen](#) aus der Seitenleiste, um an mehreren Aufgaben gleichzeitig zu arbeiten, jede in ihrem eigenen Git-Worktree. Senden Sie [langfristige Arbeiten in die Cloud](#), damit sie auch dann fortgesetzt werden, wenn Sie die App schließen, oder [setzen Sie eine Sitzung im Web oder in Ihrer IDE fort](#), wenn eine Aufgabe länger als erwartet dauert. [Verbinden Sie externe Tools](#) wie GitHub, Slack und Linear, um Ihren Workflow zusammenzubringen.

Kommen Sie von der CLI?

Desktop führt die gleiche Engine wie die CLI mit einer grafischen Benutzeroberfläche aus. Sie können beide gleichzeitig auf dem gleichen Projekt ausführen, und sie teilen die Konfiguration (CLAUDE.md-Dateien, MCP servers, hooks, Skills und Einstellungen). Für einen vollständigen Vergleich von Funktionen, Flag-Äquivalenten und was in Desktop nicht verfügbar ist, siehe [CLI-Vergleich](#).

Was kommt als Nächstes

- [Claude Code Desktop verwenden](#): Berechtigungsmodi, parallele Sitzungen, Diff-Ansicht, Konnektoren und Enterprise-Konfiguration
- [Fehlersuche](#): Lösungen für häufige Fehler und Einrichtungsprobleme
- [Best Practices](#): Tipps zum Schreiben effektiver Eingaben und zum Herausholen des Besten aus Claude Code
- [Häufige Workflows](#): Tutorials zum Debuggen, Refactoring, Testen und mehr

Claude Code einrichten

Installieren, authentifizieren und beginnen Sie mit der Verwendung von Claude Code auf Ihrem Entwicklungscomputer.

Systemanforderungen

- **Betriebssystem:**
 - macOS 13.0+
 - Windows 10 1809+ oder Windows Server 2019+ ([siehe Setuphinweise](#))
 - Ubuntu 20.04+
 - Debian 10+
 - Alpine Linux 3.19+ ([zusätzliche Abhängigkeiten erforderlich](#))
- **Hardware:** 4 GB+ RAM
- **Netzwerk:** Internetverbindung erforderlich (siehe [Netzwerkconfiguration](#))
- **Shell:** Funktioniert am besten in Bash oder Zsh
- **Standort:** [Von Anthropic unterstützte Länder](#)

Zusätzliche Abhängigkeiten

- **ripgrep:** Normalerweise in Claude Code enthalten. Falls die Suche fehlschlägt, siehe [Suche-Fehlerbehebung](#).
- **Node.js 18+:** Nur erforderlich für [veraltete npm-Installation](#)

Installation

To install Claude Code, use one of the following methods:

Native Install (Recommended)

macOS, Linux, WSL:

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

Windows PowerShell:

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del  
install.cmd
```

Windows requires [Git for Windows](#). Install it first if you don't have it.

Info:

Native installations automatically update in the background to keep you on the latest version.

Homebrew

```
brew install --cask claude-code
```

Info:

Homebrew installations do not auto-update. Run `brew upgrade claude-code` periodically to get the latest features and security fixes.

WinGet

```
winget install Anthropic.ClaudeCode
```

Info:

WinGet installations do not auto-update. Run `winget upgrade Anthropic.ClaudeCode` periodically to get the latest features and security fixes.

Nach Abschluss des Installationsvorgangs navigieren Sie zu Ihrem Projekt und starten Claude Code:

```
cd your-awesome-project
claude
```

Wenn während der Installation Probleme auftreten, konsultieren Sie das [Fehlerbehebungshandbuch](#).

Tip:

Führen Sie `claude doctor` nach der Installation aus, um Ihren Installationstyp und die Version zu überprüfen.

Plattformspezifisches Setup

Windows: Führen Sie Claude Code nativ aus (erfordert [Git Bash](#)) oder innerhalb von WSL. Sowohl WSL 1 als auch WSL 2 werden unterstützt, aber WSL 1 hat begrenzte Unterstützung und unterstützt keine Funktionen wie Bash-Tool-Sandboxing.

Alpine Linux und andere musl/uClibc-basierte Distributionen:

Das native Installationsprogramm auf Alpine und anderen musl/uClibc-basierten Distributionen erfordert `libgcc`, `libstdc++` und `ripgrep`. Installieren Sie diese mit dem Paketmanager Ihrer Distribution und setzen Sie dann `USE_BUILTIN_RIPGREP=0`.

Auf Alpine:

```
apk add libgcc libstdc++ ripgrep
```

Authentifizierung

Für Einzelpersonen

1. **Claude Pro oder Max Plan** (empfohlen): Abonnieren Sie Claude's [Pro oder Max Plan](#) für ein einheitliches Abonnement, das sowohl Claude Code als auch Claude im Web umfasst. Verwalten Sie Ihr Konto an einem Ort und melden Sie sich mit Ihrem Claude.ai-Konto an.
2. **Claude Console:** Verbinden Sie sich über die [Claude Console](#) und schließen Sie den OAuth-Prozess ab. Erfordert aktive Abrechnung in der Anthropic Console. Ein „Claude Code“-Arbeitsbereich wird automatisch für Nutzungsverfolgung und Kostenverwaltung erstellt. Sie können keine API-Schlüssel für den Claude Code-Arbeitsbereich erstellen; er ist ausschließlich für die Claude Code-Nutzung vorgesehen.

Für Teams und Organisationen

1. **Claude for Teams oder Enterprise** (empfohlen): Abonnieren Sie [Claude for Teams](#) oder [Claude for Enterprise](#) für zentrale Abrechnung, Teamverwaltung und Zugriff auf sowohl Claude Code als auch Claude im Web. Teammitglieder melden sich mit ihren Claude.ai-Konten an.
2. **Claude Console mit Team-Abrechnung:** Richten Sie eine gemeinsame [Claude Console](#)-Organisation mit Team-Abrechnung ein. Laden Sie Teammitglieder ein und weisen Sie Rollen für die Nutzungsverfolgung zu.
3. **Cloud-Anbieter:** Konfigurieren Sie Claude Code für die Verwendung von [Amazon Bedrock](#), [Google Vertex AI](#) oder [Microsoft Foundry](#) für Bereitstellungen mit Ihrer vorhandenen Cloud-Infrastruktur.

Installieren Sie eine bestimmte Version

Das native Installationsprogramm akzeptiert entweder eine spezifische Versionsnummer oder einen Release-Kanal (`latest` oder `stable`). Der Kanal, den Sie bei der Installation wählen, wird zu Ihrem Standard für automatische Updates. Weitere Informationen finden Sie unter [Release-Kanal konfigurieren](#).

So installieren Sie die neueste Version (Standard):

macOS, Linux, WSL

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

Windows PowerShell

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del install.cmd
```

So installieren Sie die stabile Version:

macOS, Linux, WSL

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

Windows PowerShell

```
& ([scriptblock]::Create((irm https://claude.ai/install.ps1))) stable
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd stable &&  
del install.cmd
```

So installieren Sie eine bestimmte Versionsnummer:

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash -s 1.0.58
```

Windows PowerShell

```
& ([scriptblock]::Create((irm https://claude.ai/install.ps1))) 1.0.58
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd 1.0.58 &&  
del install.cmd
```

Binäre Integrität und Code-Signierung

- SHA256-Checksummen für alle Plattformen werden in den Release-Manifesten veröffentlicht, die sich derzeit unter <https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json> befinden (Beispiel: ersetzen Sie `{VERSION}` durch `2.0.30`)
- Signierte Binärdateien werden für die folgenden Plattformen verteilt:
 - macOS: Signiert von 'Anthropic PBC' und von Apple beglaubigt
 - Windows: Signiert von „Anthropic, PBC“

NPM-Installation (veraltet)

Die NPM-Installation ist veraltet. Verwenden Sie die [native Installationsmethode](#), wenn möglich. Um eine vorhandene npm-Installation zu einer nativen zu migrieren, führen Sie `claude install` aus.

Globale npm-Installation

```
npm install -g @anthropic-ai/claude-code
```

Warning:

Verwenden Sie NICHT `sudo npm install -g`, da dies zu Berechtigungsproblemen und Sicherheitsrisiken führen kann. Wenn Sie auf Berechtigungsfehler stoßen, siehe [Fehlerbehebung bei Berechtigungsfehlern](#) für empfohlene Lösungen.

Windows-Setup

Option 1: Claude Code innerhalb von WSL

- Sowohl WSL 1 als auch WSL 2 werden unterstützt
- WSL 2 unterstützt [Sandboxing](#) für erhöhte Sicherheit. WSL 1 unterstützt kein Sandboxing.

Option 2: Claude Code auf nativem Windows mit Git Bash

- Erfordert [Git für Windows](#)
- Für tragbare Git-Installationen geben Sie den Pfad zu Ihrer `bash.exe` an:

```
$env:CLAUDE_CODE_GIT_BASH_PATH="C:\Program Files\Git\bin\bash.exe"
```

Claude Code aktualisieren

Automatische Updates

Claude Code aktualisiert sich automatisch, um sicherzustellen, dass Sie die neuesten Funktionen und Sicherheitspatches haben.

- **Update-Überprüfungen:** Werden beim Start und regelmäßig während der Ausführung durchgeführt

- **Update-Prozess:** Downloads und Installationen erfolgen automatisch im Hintergrund
- **Benachrichtigungen:** Sie sehen eine Benachrichtigung, wenn Updates installiert werden
- **Updates anwenden:** Updates werden beim nächsten Start von Claude Code wirksam

Note:

Homebrew- und WinGet-Installationen werden nicht automatisch aktualisiert. Verwenden Sie `brew upgrade claude-code` oder `winget upgrade Anthropic.ClaudeCode`, um manuell zu aktualisieren.

Bekanntes Problem: Claude Code kann Sie über Updates benachrichtigen, bevor die neue Version in diesen Paketmanagern verfügbar ist. Wenn ein Upgrade fehlschlägt, warten Sie und versuchen Sie es später erneut.

Release-Kanal konfigurieren

Konfigurieren Sie, welchen Release-Kanal Claude Code für automatische Updates und `claude update` mit der Einstellung `autoUpdatesChannel` folgt:

- `"latest"` (Standard): Erhalten Sie neue Funktionen, sobald sie veröffentlicht werden
- `"stable"`: Verwenden Sie eine Version, die typischerweise etwa eine Woche alt ist und überspringen Sie Releases mit großen Regressionen

Konfigurieren Sie dies über `/config` → **Auto-update channel**, oder fügen Sie es zu Ihrer [settings.json-Datei](#) hinzu:

```
{  
  "autoUpdatesChannel": "stable"  
}
```

Für Enterprise-Bereitstellungen können Sie einen konsistenten Release-Kanal in Ihrer Organisation mit [verwalteten Einstellungen](#) erzwingen.

Automatische Updates deaktivieren

Setzen Sie die Umgebungsvariable `DISABLE_AUTOUPDATER` in Ihrer Shell oder [settings.json-Datei](#):

```
export DISABLE_AUTOUPDATER=1
```

Manuell aktualisieren

```
claude update
```

Claude Code deinstallieren

Wenn Sie Claude Code deinstallieren müssen, folgen Sie den Anweisungen für Ihre Installationsmethode.

Native Installation

Entfernen Sie die Claude Code-Binärdatei und Versionsdateien:

macOS, Linux, WSL:

```
rm -f ~/.local/bin/claude  
rm -rf ~/.local/share/claude
```

Windows PowerShell:

```
Remove-Item -Path "$env:USERPROFILE\.local\bin\claude.exe" -Force  
Remove-Item -Path "$env:USERPROFILE\.local\share\claude" -Recurse -Force
```

Windows CMD:

```
del "%USERPROFILE%.local\bin\claude.exe"  
rmdir /s /q "%USERPROFILE%.local\share\claude"
```

Homebrew-Installation

```
brew uninstall --cask claude-code
```


WinGet-Installation

```
winget uninstall Anthropic.ClaudeCode
```

NPM-Installation

```
npm uninstall -g @anthropic-ai/claude-code
```

Konfigurationsdateien bereinigen (optional)

Warning:

Das Entfernen von Konfigurationsdateien löscht alle Ihre Einstellungen, zulässigen Tools, MCP-Serverkonfigurationen und Sitzungsverlauf.

So entfernen Sie Claude Code-Einstellungen und zwischengespeicherte Daten:

macOS, Linux, WSL:

```
## Benutzereinstellungen und Status entfernen
rm -rf ~/.claude
rm ~/.claude.json

## Projektspezifische Einstellungen entfernen (führen Sie dies aus Ihrem
Projektverzeichnis aus)
rm -rf .claude
rm -f .mcp.json
```

Windows PowerShell:

```
## Benutzereinstellungen und Status entfernen
Remove-Item -Path "$env:USERPROFILE\.claude" -Recurse -Force
Remove-Item -Path "$env:USERPROFILE\.claude.json" -Force

## Projektspezifische Einstellungen entfernen (führen Sie dies aus Ihrem
Projektverzeichnis aus)
Remove-Item -Path ".claude" -Recurse -Force
Remove-Item -Path ".mcp.json" -Force
```

Windows CMD:

```
REM Benutzereinstellungen und Status entfernen
rmdir /s /q "%USERPROFILE%\.claude"
del "%USERPROFILE%\.claude.json"

REM Projektspezifische Einstellungen entfernen (führen Sie dies aus Ihrem
Projektverzeichnis aus)
rmdir /s /q ".claude"
del ".mcp.json"
```

So funktioniert Claude Code

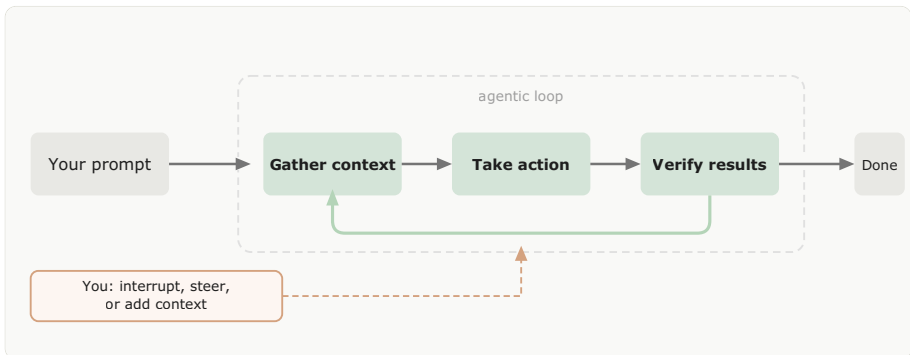
Verstehen Sie die agentengesteuerte Schleife, integrierte Tools und wie Claude Code mit Ihrem Projekt interagiert.

Claude Code ist ein agentengesteuerter Assistent, der in Ihrem Terminal läuft. Obwohl er sich bei der Codierung auszeichnet, kann er bei allem helfen, was Sie von der Befehlszeile aus tun können: Dokumentation schreiben, Builds ausführen, Dateien durchsuchen, Themen recherchieren und vieles mehr.

Dieser Leitfaden behandelt die Kernarchitektur, integrierte Funktionen und [Tipps für effektive Zusammenarbeit mit Claude Code](#). Für schrittweise Anleitungen siehe [Häufige Workflows](#). Für Erweiterungsfunktionen wie skills, MCP und hooks siehe [Claude Code erweitern](#).

Die agentengesteuerte Schleife

Wenn Sie Claude eine Aufgabe geben, arbeitet er durch drei Phasen: **Kontext sammeln**, **Maßnahmen ergreifen** und **Ergebnisse überprüfen**. Diese Phasen verschmelzen miteinander. Claude nutzt Tools durchgehend, ob beim Durchsuchen von Dateien zum Verständnis Ihres Codes, beim Bearbeiten zur Vornahme von Änderungen oder beim Ausführen von Tests zur Überprüfung seiner Arbeit.



Die agentengesteuerte Schleife: Ihre Eingabeaufforderung führt dazu, dass Claude Kontext sammelt, Maßnahmen ergreift, Ergebnisse überprüft und wiederholt, bis die Aufgabe abgeschlossen ist. Sie können jederzeit unterbrechen.

Die Schleife passt sich an das an, was Sie fragen. Eine Frage zu Ihrer Codebasis könnte nur Kontextsammlung erfordern. Eine Fehlerbehebung durchläuft alle drei Phasen wiederholt. Eine Umstrukturierung könnte umfangreiche Überprüfung beinhalten. Claude entscheidet, was jeder Schritt erfordert, basierend auf dem, was er aus dem vorherigen Schritt gelernt hat, verkettet Dutzende von Aktionen zusammen und korrigiert seinen Kurs unterwegs.

Sie sind auch Teil dieser Schleife. Sie können jederzeit unterbrechen, um Claude in eine andere Richtung zu lenken, zusätzlichen Kontext bereitzustellen oder ihn zu bitten, einen anderen Ansatz zu versuchen. Claude arbeitet autonom, bleibt aber responsiv gegenüber Ihrer Eingabe.

Die agentengesteuerte Schleife wird von zwei Komponenten angetrieben: [Modellen](#), die denken, und [Tools](#), die handeln. Claude Code dient als **agentengesteuerte Umgebung** um Claude: Sie bietet die Tools, Kontextverwaltung und Ausführungsumgebung, die ein Sprachmodell in einen fähigen Codierungs-Agenten verwandeln.

Modelle

Claude Code nutzt Claude-Modelle, um Ihren Code zu verstehen und über Aufgaben nachzudenken. Claude kann Code in jeder Sprache lesen, verstehen, wie Komponenten verbunden sind, und herausfinden, was sich ändern muss, um Ihr Ziel zu erreichen. Bei komplexen Aufgaben unterteilt er die Arbeit in Schritte, führt sie aus und passt sich basierend auf dem an, was er lernt.

[Mehrere Modelle](#) sind mit unterschiedlichen Kompromissen verfügbar. Sonnet bewältigt die meisten Codierungsaufgaben gut. Opus bietet stärkeres Denken für komplexe architektonische Entscheidungen. Wechseln Sie mit `/model` während einer Sitzung oder starten Sie mit `claude --model <name>`.

Wenn dieser Leitfaden sagt „Claude wählt“ oder „Claude entscheidet“, ist es das Modell, das die Überlegung durchführt.

Tools

Tools sind das, was Claude Code agentengesteuert macht. Ohne Tools kann Claude nur mit Text antworten. Mit Tools kann Claude handeln: Ihren Code lesen, Dateien bearbeiten, Befehle ausführen, das Web durchsuchen und mit externen Diensten interagieren. Jede Tool-Nutzung gibt Informationen zurück, die in die Schleife fließen und Claudes nächste Entscheidung informieren.

Die integrierten Tools fallen im Allgemeinen in fünf Kategorien, von denen jede eine andere Art von Agentur darstellt.

Kategorie	Was Claude tun kann
Datenvorgänge	Dateien lesen, Code bearbeiten, neue Dateien erstellen, umbenennen und reorganisieren
Suche	Dateien nach Muster finden, Inhalte mit Regex durchsuchen, Codebases erkunden
Ausführung	Shell-Befehle ausführen, Server starten, Tests ausführen, git verwenden
Web	Das Web durchsuchen, Dokumentation abrufen, Fehlermeldungen nachschlagen
Code-Intelligenz	Typfehler und Warnungen nach Bearbeitungen sehen, zu Definitionen springen, Referenzen finden (erfordert Code-Intelligenz-Plugins)

Dies sind die primären Funktionen. Claude hat auch Tools zum Spawnen von subagents, zum Stellen von Fragen und für andere Orchestrierungsaufgaben. Siehe [Tools verfügbar für Claude](#) für die vollständige Liste.

Claude wählt basierend auf Ihrer Eingabeaufforderung und dem, was er unterwegs lernt, aus, welche Tools er verwenden soll. Wenn Sie sagen „beheben Sie die fehlgeschlagenen Tests“, könnte Claude:

1. Die Test-Suite ausführen, um zu sehen, was fehlschlägt
2. Die Fehlerausgabe lesen
3. Nach den relevanten Quelldateien suchen
4. Diese Dateien lesen, um den Code zu verstehen
5. Die Dateien bearbeiten, um das Problem zu beheben
6. Die Tests erneut ausführen, um zu überprüfen

Jede Tool-Nutzung gibt Claude neue Informationen, die den nächsten Schritt informieren. Dies ist die agentengesteuerte Schleife in Aktion.

Erweitern der Basisfunktionen: Die integrierten Tools sind die Grundlage. Sie können das, was Claude weiß, mit [skills](#) erweitern, sich mit externen Diensten mit [MCP](#) verbinden, Workflows mit [hooks](#) automatisieren und Aufgaben an [subagents](#) delegieren.

Diese Erweiterungen bilden eine Schicht auf der Kernschleife der Agentur. Siehe [Claude Code erweitern](#) für Anleitung zur Auswahl der richtigen Erweiterung für Ihre Anforderungen.

Worauf Claude zugreifen kann

Dieser Leitfaden konzentriert sich auf das Terminal. Claude Code läuft auch in [VS Code](#), [JetBrains IDEs](#) und anderen Umgebungen.

Wenn Sie `claude` in einem Verzeichnis ausführen, erhält Claude Code Zugriff auf:

- **Ihr Projekt.** Dateien in Ihrem Verzeichnis und Unterverzeichnissen sowie Dateien an anderer Stelle mit Ihrer Genehmigung.
- **Ihr Terminal.** Jeden Befehl, den Sie ausführen könnten: Build-Tools, git, Paketmanager, Systemdienstprogramme, Skripte. Wenn Sie es von der Befehlszeile aus tun können, kann Claude es auch.
- **Ihren git-Status.** Aktueller Branch, nicht committete Änderungen und aktuelle Commit-Historie.
- **Ihre [CLAUDE.md](#).** Eine Markdown-Datei, in der Sie projektspezifische Anweisungen, Konventionen und Kontext speichern, den Claude jede Sitzung kennen sollte.
- **Auto-Speicher.** Erkenntnisse, die Claude automatisch speichert, während Sie arbeiten, wie Projektmuster und Ihre Vorlieben. Die ersten 200 Zeilen von MEMORY.md werden zu Beginn jeder Sitzung geladen.
- **Erweiterungen, die Sie konfigurieren.** [MCP-Server](#) für externe Dienste, [skills](#) für Workflows, [subagents](#) für delegierte Arbeit und [Claude in Chrome](#) für Browser-Interaktion.

Da Claude Ihr gesamtes Projekt sieht, kann er darin arbeiten. Wenn Sie Claude bitten, „den Authentifizierungsfehler zu beheben“, sucht er nach relevanten Dateien, liest mehrere Dateien, um den Kontext zu verstehen, nimmt koordinierte Bearbeitungen vor, führt Tests aus, um die Behebung zu überprüfen, und committed die Änderungen, wenn Sie es fragen. Dies unterscheidet sich von Inline-Code-Assistenten, die nur die aktuelle Datei sehen.

Umgebungen und Schnittstellen

Die agentengesteuerte Schleife, Tools und Funktionen, die oben beschrieben sind, sind überall dort gleich, wo Sie Claude Code verwenden. Was sich ändert, ist, wo der Code ausgeführt wird und wie Sie damit interagieren.

Ausführungsumgebungen

Claude Code läuft in drei Umgebungen, von denen jede unterschiedliche Kompromisse für die Ausführung Ihres Codes bietet.

Umgebung	Wo Code ausgeführt wird	Anwendungsfall
Lokal	Ihr Computer	Standard. Vollständiger Zugriff auf Ihre Dateien, Tools und Umgebung
Cloud	Von Anthropic verwaltete VMs	Aufgaben auslagern, an Repos arbeiten, die Sie nicht lokal haben
Remote Control	Ihr Computer, gesteuert von einem Browser	Verwenden Sie die Web-UI, während Sie alles lokal halten

Schnittstellen

Sie können auf Claude Code über das Terminal, die [Desktop-App](#), [IDE-Erweiterungen](#), [claude.ai/code](#), [Remote Control](#), [Slack](#) und [CI/CD-Pipelines](#) zugreifen. Die Schnittstelle bestimmt, wie Sie Claude sehen und damit interagieren, aber die zugrunde liegende agentengesteuerte Schleife ist identisch. Siehe [Claude Code überall verwenden](#) für die vollständige Liste.

Mit Sitzungen arbeiten

Claude Code speichert Ihre Konversation lokal, während Sie arbeiten. Jede Nachricht, Tool-Nutzung und jedes Ergebnis wird gespeichert, was [Zurückspulen](#), [Fortsetzen und Verzweigen](#) von Sitzungen ermöglicht. Bevor Claude Code-Änderungen vornimmt, erstellt es auch einen Snapshot der betroffenen Dateien, damit Sie bei Bedarf zurückrollen können.

Sitzungen sind unabhängig. Jede neue Sitzung beginnt mit einem frischen context window, ohne die Konversationshistorie aus vorherigen Sitzungen. Claude kann Erkenntnisse über Sitzungen hinweg mit [Auto-Speicher](#) beibehalten, und Sie können Ihre eigenen persistenten Anweisungen in [CLAUDE.md](#) hinzufügen.

Über Branches arbeiten

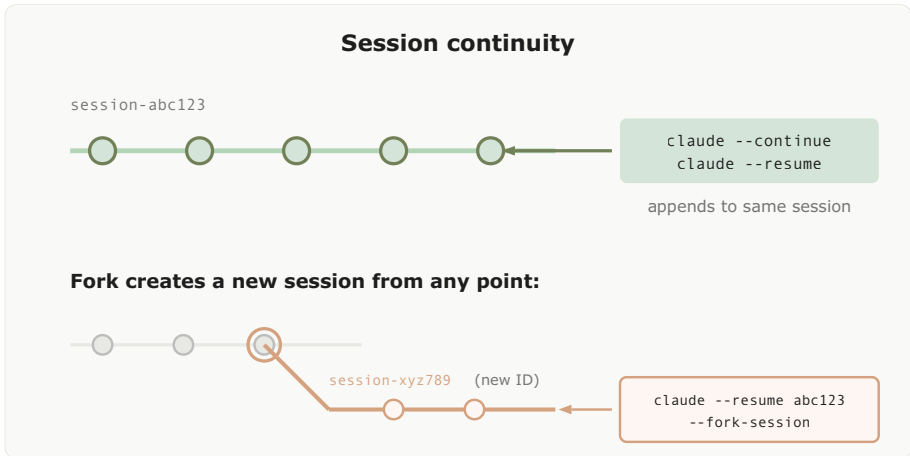
Jede Claude Code-Konversation ist eine Sitzung, die an Ihr aktuelles Verzeichnis gebunden ist. Wenn Sie fortsetzen, sehen Sie nur Sitzungen aus diesem Verzeichnis.

Claude sieht die Dateien Ihres aktuellen Branches. Wenn Sie Branches wechseln, sieht Claude die Dateien des neuen Branches, aber Ihre Konversationshistorie bleibt gleich. Claude erinnert sich an das, was Sie besprochen haben, auch nach dem Wechsel.

Da Sitzungen an Verzeichnisse gebunden sind, können Sie parallele Claude-Sitzungen ausführen, indem Sie [git worktrees](#) verwenden, die separate Verzeichnisse für einzelne Branches erstellen.

Sitzungen fortsetzen oder verzweigen

Wenn Sie eine Sitzung mit `claude --continue` oder `claude --resume` fortsetzen, setzen Sie dort fort, wo Sie aufgehört haben, und verwenden dieselbe Sitzungs-ID. Neue Nachrichten werden an die bestehende Konversation angehängt. Ihre vollständige Konversationshistorie wird wiederhergestellt, aber sitzungsspezifische Berechtigungen nicht. Sie müssen diese erneut genehmigen.



Sitzungskontinuität: Fortsetzen setzt dieselbe Sitzung fort, Verzweigung erstellt einen neuen Branch mit einer neuen ID.

Um abzuzweigen und einen anderen Ansatz zu versuchen, ohne die ursprüngliche Sitzung zu beeinflussen, verwenden Sie das Flag `--fork-session`:

```
claude --continue --fork-session
```

Dies erstellt eine neue Sitzungs-ID, während die Konversationshistorie bis zu diesem Punkt beibehalten wird. Die ursprüngliche Sitzung bleibt unverändert. Wie beim Fortsetzen erben verzweigte Sitzungen keine sitzungsspezifischen Berechtigungen.

Dieselbe Sitzung in mehreren Terminals: Wenn Sie dieselbe Sitzung in mehreren Terminals fortsetzen, schreiben beide Terminals in dieselbe Sitzungsdatei. Nachrichten von beiden werden verschachtelt, wie zwei Personen, die in dasselbe Notizbuch schreiben. Nichts wird beschädigt, aber die Konversation wird durcheinander. Jedes Terminal sieht während der Sitzung nur seine eigenen Nachrichten, aber wenn Sie diese Sitzung später fortsetzen, sehen Sie alles verschachtelt. Für parallele Arbeit vom selben Ausgangspunkt verwenden Sie `--fork-session`, um jedem Terminal seine eigene saubere Sitzung zu geben.

Das context window

Claudes context window enthält Ihre Konversationshistorie, Dateiinhalte, Befehlsausgaben, [CLAUDE.md](#), geladene skills und Systemanweisungen. Während Sie arbeiten, füllt sich der Kontext. Claude komprimiert automatisch, aber Anweisungen von früh in der Konversation können verloren gehen. Legen Sie persistente Regeln in [CLAUDE.md](#) ab, und führen Sie `/context` aus, um zu sehen, was Platz verbraucht.

Wenn der Kontext voll wird

Claude Code verwaltet den Kontext automatisch, wenn Sie sich dem Limit nähern. Es löscht zuerst ältere Tool-Ausgaben, dann fasst die Konversation zusammen, falls erforderlich. Ihre Anfragen und wichtige Code-Snippets werden beibehalten; detaillierte Anweisungen von früh in der Konversation können verloren gehen. Legen Sie persistente Regeln in [CLAUDE.md](#) ab, anstatt sich auf die Konversationshistorie zu verlassen.

Um zu kontrollieren, was während der Komprimierung beibehalten wird, fügen Sie einen Abschnitt „Compact Instructions“ zu [CLAUDE.md](#) hinzu oder führen Sie `/compact` mit einem Fokus aus (wie `/compact focus on the API changes`).

Führen Sie `/context` aus, um zu sehen, was Platz verbraucht. MCP-Server fügen Tool-Definitionen zu jeder Anfrage hinzu, daher können einige wenige Server erheblichen Kontext verbrauchen, bevor Sie mit der Arbeit beginnen. Führen Sie `/mcp` aus, um die Kosten pro Server zu überprüfen.

Kontext mit skills und subagents verwalten

Über die Komprimierung hinaus können Sie andere Funktionen verwenden, um zu kontrollieren, was in den Kontext geladen wird.

[Skills](#) werden bei Bedarf geladen. Claude sieht Skill-Beschreibungen zu Sitzungsbeginn, aber der vollständige Inhalt wird nur geladen, wenn ein Skill verwendet wird. Für Skills, die Sie manuell aufrufen, setzen Sie `disable-model-invocation: true`, um Beschreibungen aus dem Kontext zu halten, bis Sie sie benötigen.

[Subagents](#) erhalten ihren eigenen frischen Kontext, völlig getrennt von Ihrer Hauptkonversation. Ihre Arbeit bläht Ihren Kontext nicht auf. Wenn sie fertig sind, geben sie eine Zusammenfassung zurück. Diese Isolation ist der Grund, warum subagents bei langen Sitzungen helfen.

Siehe [Kontextkosten](#) für die Kosten jeder Funktion und [Token-Nutzung reduzieren](#) für Tipps zur Verwaltung des Kontexts.

Sicher bleiben mit Checkpoints und Berechtigungen

Claude hat zwei Sicherheitsmechanismen: Checkpoints ermöglichen es Ihnen, Dateiänderungen rückgängig zu machen, und Berechtigungen kontrollieren, was Claude ohne Nachfrage tun kann.

Änderungen mit Checkpoints rückgängig machen

Jede Dateibearbeitung ist reversibel. Bevor Claude eine Datei bearbeitet, erstellt er einen Snapshot des aktuellen Inhalts. Wenn etwas schief geht, drücken Sie zweimal `Esc`, um zu einem vorherigen Zustand zurückzuspulen, oder bitten Sie Claude, rückgängig zu machen.

Checkpoints sind lokal für Ihre Sitzung, getrennt von git. Sie decken nur Dateiänderungen ab. Aktionen, die sich auf Remote-Systeme auswirken (Datenbanken, APIs, Bereitstellung), können nicht checkpointed werden, weshalb Claude vor dem Ausführen von Befehlen mit externen Nebenwirkungen fragt.

Kontrollieren Sie, was Claude tun kann

Drücken Sie `Shift+Tab`, um durch die Berechtigungsmodi zu wechseln:

- **Standard:** Claude fragt vor Dateibearbeitungen und Shell-Befehlen
- **Auto-accept edits:** Claude bearbeitet Dateien ohne zu fragen, fragt aber immer noch nach Befehlen
- **Plan Mode:** Claude verwendet nur schreibgeschützte Tools und erstellt einen Plan, den Sie vor der Ausführung genehmigen können

Sie können auch spezifische Befehle in `.claude/settings.json` zulassen, damit Claude nicht jedes Mal fragt. Dies ist nützlich für vertrauenswürdige Befehle wie `npm test` oder `git status`. Einstellungen können von organisationsweiten Richtlinien bis zu persönlichen Vorlieben reichen. Siehe [Berechtigungen](#) für Details.

Effektiv mit Claude Code arbeiten

Diese Tipps helfen Ihnen, bessere Ergebnisse von Claude Code zu erhalten.

Fragen Sie Claude Code um Hilfe

Claude Code kann Ihnen beibringen, wie man ihn verwendet. Stellen Sie Fragen wie „Wie richte ich hooks ein?“ oder „Was ist der beste Weg, meine CLAUDE.md zu strukturieren?“ und Claude wird erklären.

Integrierte Befehle führen Sie auch durch die Einrichtung:

- `/init` führt Sie durch die Erstellung einer CLAUDE.md für Ihr Projekt
- `/agents` hilft Ihnen, benutzerdefinierte subagents zu konfigurieren
- `/doctor` diagnostiziert häufige Probleme mit Ihrer Installation

Es ist eine Konversation

Claude Code ist konversativ. Sie benötigen keine perfekten Eingabeaufforderungen. Beginnen Sie mit dem, was Sie möchten, und verfeinern Sie dann:

Beheben Sie den Login-Fehler

[Claude untersucht, versucht etwas]

Das ist nicht ganz richtig. Das Problem liegt in der Sitzungsverwaltung.

[Claude passt seinen Ansatz an]

Wenn der erste Versuch nicht richtig ist, müssen Sie nicht von vorne anfangen. Sie iterieren.

Unterbrechen und lenken

Sie können Claude jederzeit unterbrechen. Wenn er den falschen Weg geht, geben Sie einfach Ihre Korrektur ein und drücken Sie Enter. Claude wird stoppen, was er tut, und seinen Ansatz basierend auf Ihrer Eingabe anpassen. Sie müssen nicht warten, bis er fertig ist, oder von vorne anfangen.

Seien Sie von Anfang an spezifisch

Je präziser Ihre anfängliche Eingabeaufforderung ist, desto weniger Korrekturen benötigen Sie. Verweisen Sie auf spezifische Dateien, erwähnen Sie Einschränkungen und zeigen Sie auf Beispielmuster.

Der Checkout-Fluss ist für Benutzer mit abgelaufenen Karten unterbrochen. Überprüfen Sie `src/payments/` auf das Problem, besonders Token-Aktualisierung. Schreiben Sie zuerst einen fehlgeschlagenen Test, dann beheben Sie ihn.

Vage Eingabeaufforderungen funktionieren, aber Sie werden mehr Zeit mit Lenkung verbringen. Spezifische Eingabeaufforderungen wie die obige sind oft beim ersten Versuch erfolgreich.

Geben Sie Claude etwas zum Überprüfen

Claude funktioniert besser, wenn er seine eigene Arbeit überprüfen kann. Fügen Sie Testfälle ein, fügen Sie Screenshots des erwarteten UI ein oder definieren Sie die gewünschte Ausgabe.

Implementieren Sie `validateEmail`. Testfälle: `'user@example.com' → true`, `'invalid' → false`, `'user@.com' → false`. Führen Sie die Tests danach aus.

Für visuelle Arbeit fügen Sie einen Screenshot des Designs ein und bitten Sie Claude, seine Implementierung dagegen zu vergleichen.

Vor der Implementierung erkunden

Bei komplexen Problemen trennen Sie Forschung von Codierung. Verwenden Sie Plan Mode (`Shift+Tab` zweimal), um die Codebasis zuerst zu analysieren:

Lesen Sie `src/auth/` und verstehen Sie, wie wir Sitzungen handhaben. Erstellen Sie dann einen Plan zum Hinzufügen von OAuth-Unterstützung.

Überprüfen Sie den Plan, verfeinern Sie ihn durch Konversation, dann lassen Sie Claude implementieren. Dieser zweiphasige Ansatz erzeugt bessere Ergebnisse als direkt zum Code zu springen.

Delegieren, nicht diktieren

Denken Sie daran, an einen fähigen Kollegen zu delegieren. Geben Sie Kontext und Richtung, dann vertrauen Sie Claude, die Details herauszufinden:

Der Checkout-Fluss ist für Benutzer mit abgelaufenen Karten unterbrochen.
Der relevante Code ist in `src/payments/`. Können Sie ihn untersuchen und beheben?

Sie müssen nicht angeben, welche Dateien zu lesen sind oder welche Befehle auszuführen sind. Claude findet das heraus.

Nächste Schritte

- [Mit Funktionen erweitern](#) Fügen Sie Skills, MCP-Verbindungen und benutzerdefinierte Befehle hinzu
- [Häufige Workflows](#) Schritt-für-Schritt-Anleitungen für typische Aufgaben

Part 2: Core Usage

Claude Code erweitern

Verstehen Sie, wann Sie CLAUDE.md, Skills, Subagents, Hooks, MCP und Plugins verwenden.

Claude Code kombiniert ein Modell, das über Ihren Code nachdenkt, mit [integrierten Tools](#) für Dateivorgänge, Suche, Ausführung und Webzugriff. Die integrierten Tools decken die meisten Codierungsaufgaben ab. Dieses Handbuch behandelt die Erweiterungsebene: Funktionen, die Sie hinzufügen, um anzupassen, was Claude weiß, es mit externen Diensten zu verbinden und Workflows zu automatisieren.

Note:

Informationen zur Funktionsweise der Kern-Agentenschleife finden Sie unter [How Claude Code works](#).

Neu bei Claude Code? Beginnen Sie mit [CLAUDE.md](#) für Projektkonventionen. Fügen Sie andere Erweiterungen nach Bedarf hinzu.

Übersicht

Erweiterungen verbinden sich mit verschiedenen Teilen der Agentenschleife:

- [CLAUDE.md](#) fügt persistenten Kontext hinzu, den Claude in jeder Sitzung sieht
- [Skills](#) fügen wiederverwendbares Wissen und aufrufbare Workflows hinzu
- [MCP](#) verbindet Claude mit externen Diensten und Tools
- [Subagents](#) führen ihre eigenen Schleifen in isoliertem Kontext aus und geben Zusammenfassungen zurück
- [Agent teams](#) koordinieren mehrere unabhängige Sitzungen mit gemeinsamen Aufgaben und Peer-to-Peer-Messaging
- [Hooks](#) laufen vollständig außerhalb der Schleife als deterministische Skripte
- [Plugins](#) und [Marketplaces](#) verpacken und verteilen diese Funktionen

[Skills](#) sind die flexibelste Erweiterung. Ein Skill ist eine Markdown-Datei, die Wissen, Workflows oder Anweisungen enthält. Sie können Skills mit einem Befehl wie `/deploy` aufrufen, oder Claude kann sie automatisch laden, wenn sie relevant sind. Skills können in Ihrer aktuellen Konversation oder in einem isolierten Kontext über Subagents ausgeführt werden.

Funktionen an Ihr Ziel anpassen

Funktionen reichen von immer aktivem Kontext, den Claude in jeder Sitzung sieht, bis zu On-Demand-Funktionen, die Sie oder Claude aufrufen können, bis zu Hintergrundautomatisierung, die bei bestimmten Ereignissen ausgeführt wird. Die folgende Tabelle zeigt, was verfügbar ist und wann jede Funktion sinnvoll ist.

Funktion	Was sie tut	Wann man sie verwendet	Beispiel
CLAU-DE.md	Persistenter Kontext, der in jeder Konversation geladen wird	Projektkonventionen, „immer X tun“-Regeln	„Verwenden Sie pnpm, nicht npm. Führen Sie Tests vor dem Commit aus.“
Skill	Anweisungen, Wissen und Workflows, die Claude verwenden kann	Wiederverwendbarer Inhalt, Referenzdokumente, wiederholbare Aufgaben	<code>/deploy</code> führt Ihre Bereitstellungs-Checkliste aus; API-Docs-Skill mit Endpunkt-Mustern
Subagent	Isolierter Ausführungskontext, der zusammengefasste Ergebnisse zurückgibt	Kontextisolation, parallele Aufgaben, spezialisierte Worker	Recherche-Aufgabe, die viele Dateien liest, aber nur wichtige Erkenntnisse zurückgibt
Agent teams	Koordinieren Sie mehrere unabhängige Claude Code-Sitzungen	Parallele Recherche, neue Funktionsentwicklung, Debugging mit konkurrierenden Hypothesen	Spawnen Sie Reviewer, um Sicherheit, Leistung und Tests gleichzeitig zu überprüfen
MCP	Verbindung zu externen Diensten	Externe Daten oder Aktionen	Abfrage Ihrer Datenbank, Posten auf Slack, Steuerung eines Browsers

Funktion	Was sie tut	Wann man sie verwendet	Beispiel
Hook	Deterministisches Skript, das bei Ereignissen ausgeführt wird	Vorhersagbare Automatisierung, kein LLM beteiligt	Führen Sie ESLint nach jeder Dateibearbeitung aus

Plugins sind die Verpackungsebene. Ein Plugin bündelt Skills, Hooks, Subagents und MCP-Server in eine einzelne installierbare Einheit. Plugin-Skills sind namensgebunden (wie `/my-plugin:review`), sodass mehrere Plugins nebeneinander existieren können. Verwenden Sie Plugins, wenn Sie dasselbe Setup über mehrere Repositories hinweg wiederverwenden möchten oder es über einen **Marketplace** an andere verteilen möchten.

Ähnliche Funktionen vergleichen

Einige Funktionen können ähnlich wirken. Hier erfahren Sie, wie Sie sie unterscheiden.

Skill vs Subagent

Skills und Subagents lösen unterschiedliche Probleme:

- **Skills** sind wiederverwendbare Inhalte, die Sie in jeden Kontext laden können
- **Subagents** sind isolierte Worker, die separat von Ihrer Hauptkonversation ausgeführt werden

Aspekt	Skill	Subagent
Was es ist	Wiederverwendbare Anweisungen, Wissen oder Workflows	Isolierter Worker mit eigenem Kontext
Hauptvorteil	Inhalte über Kontexte hinweg teilen	Kontextisolation. Die Arbeit erfolgt separat, nur die Zusammenfassung wird zurückgegeben
Am besten für	Referenzmaterial, aufrufbare Workflows	Aufgaben, die viele Dateien lesen, parallele Arbeit, spezialisierte Worker

Skills können Referenz oder Aktion sein. Referenz-Skills bieten Wissen, das Claude während Ihrer Sitzung nutzt (wie Ihr API-Stilhandbuch). Action-Skills sagen Claude, etwas Bestimmtes zu tun (wie `/deploy`, das Ihren Bereitstellungs-Workflow ausführt).

Verwenden Sie einen Subagent, wenn Sie Kontextisolation benötigen oder wenn Ihr Kontextfenster voll wird. Der Subagent könnte Dutzende von Dateien lesen oder umfangreiche Suchen durchführen, aber Ihre Hauptkonversation erhält nur eine Zusammenfassung. Da die Arbeit des Subagent Ihren Hauptkontext nicht verbraucht, ist dies auch nützlich, wenn Sie nicht möchten, dass die Zwischenarbeit sichtbar bleibt. Benutzerdefinierte Subagents können ihre eigenen Anweisungen haben und Skills vorladen.

Sie können sich kombinieren. Ein Subagent kann spezifische Skills vorladen (`skills`:-Feld). Ein Skill kann in isoliertem Kontext mit `context: fork` ausgeführt werden. Weitere Informationen finden Sie unter [Skills](#).

CLAUDE.md vs Skill

Beide speichern Anweisungen, aber sie werden unterschiedlich geladen und dienen unterschiedlichen Zwecken.

Aspekt	CLAUDE.md	Skill
Lädt	Jede Sitzung, automatisch	On Demand
Kann Dateien enthalten	Ja, mit <code>@path</code> -Importen	Ja, mit <code>@path</code> -Importen
Kann Workflows auslösen	Nein	Ja, mit <code></name></code>
Am besten für	„Immer X tun“-Regeln	Referenzmaterial, aufrufbare Workflows

Fügen Sie es in CLAUDE.md ein, wenn Claude es immer wissen sollte: Codierungskonventionen, Build-Befehle, Projektstruktur, „niemals X tun“-Regeln.

Fügen Sie es in einen Skill ein, wenn es Referenzmaterial ist, das Claude manchmal benötigt (API-Docs, Stilhandbücher) oder ein Workflow, den Sie mit `</name>` auslösen (bereitstellen, überprüfen, freigeben).

Faustregel: Halten Sie CLAUDE.md unter 200 Zeilen. Wenn es wächst, verschieben Sie Referenzinhalte zu Skills oder teilen Sie sie in `.claude/rules/`-Dateien auf.

CLAUDE.md vs Rules vs Skills

Alle drei speichern Anweisungen, aber sie werden unterschiedlich geladen:

Aspekt	CLAUDE.md	.claude/rules/	Skill
Lädt	Jede Sitzung	Jede Sitzung oder wenn übereinstimmende Dateien geöffnet werden	On Demand, wenn aufgerufen oder relevant
Umfang	Ganzes Projekt	Kann auf Dateipfade begrenzt werden	Aufgabenspezifisch
Am besten für	Kernkonventionen und Build-Befehle	Sprachspezifische oder verzeichnisspezifische Richtlinien	Referenzmaterial, wiederholbare Workflows

Verwenden Sie CLAUDE.md für Anweisungen, die jede Sitzung benötigt: Build-Befehle, Test-Konventionen, Projektarchitektur.

Verwenden Sie Regeln, um CLAUDE.md fokussiert zu halten. Regeln mit `paths` [Frontmatter](#) werden nur geladen, wenn Claude mit übereinstimmenden Dateien arbeitet, was Kontext spart.

Verwenden Sie Skills für Inhalte, die Claude nur manchmal benötigt, wie API-Dokumentation oder eine Bereitstellungs-Checkliste, die Sie mit `</name>` auslösen.

Subagent vs Agent team

Beide parallelisieren Arbeit, aber sie sind architektonisch unterschiedlich:

- **Subagents** laufen in Ihrer Sitzung und berichten Ergebnisse an Ihren Hauptkontext zurück
- **Agent teams** sind unabhängige Claude Code-Sitzungen, die miteinander kommunizieren

Aspekt	Subagent	Agent team
Kontext	Eigenes Kontextfenster; Ergebnisse kehren zum Aufrufer zurück	Eigenes Kontextfenster; vollständig unabhängig
Kommunikation	Berichtet Ergebnisse nur an den Hauptagent zurück	Teammates senden sich gegenseitig direkt Nachrichten
Koordination	Hauptagent verwaltet alle Arbeiten	Gemeinsame Aufgabenliste mit Selbstkoordination

Aspekt	Subagent	Agent team
Am besten für	Fokussierte Aufgaben, bei denen nur das Ergebnis zählt	Komplexe Arbeit, die Diskussion und Zusammenarbeit erfordert
Token-Kosten	Niedriger: Ergebnisse werden zum Hauptkontext zusammengefasst	Höher: jeder Teammate ist eine separate Claude-Instanz

Verwenden Sie einen Subagent, wenn Sie einen schnellen, fokussierten Worker benötigen: eine Frage recherchieren, eine Behauptung überprüfen, eine Datei überprüfen. Der Subagent erledigt die Arbeit und gibt eine Zusammenfassung zurück. Ihre Hauptkonversation bleibt sauber.

Verwenden Sie ein Agent Team, wenn Teammates Erkenntnisse teilen, sich gegenseitig in Frage stellen und unabhängig koordinieren müssen. Agent Teams sind am besten für Recherche mit konkurrierenden Hypothesen, parallele Code-Überprüfung und neue Funktionsentwicklung, bei der jeder Teammate ein separates Stück besitzt.

Übergangspunkt: Wenn Sie parallele Subagents ausführen, aber auf Kontextgrenzen stoßen, oder wenn Ihre Subagents miteinander kommunizieren müssen, sind Agent Teams der natürliche nächste Schritt.

Note:

Agent Teams sind experimentell und standardmäßig deaktiviert. Weitere Informationen zu Setup und aktuellen Einschränkungen finden Sie unter [agent teams](#).

MCP vs Skill

MCP verbindet Claude mit externen Diensten. Skills erweitern das Wissen von Claude, einschließlich der Verwendung dieser Dienste effektiv.

Aspekt	MCP	Skill
Was es ist	Protokoll zur Verbindung mit externen Diensten	Wissen, Workflows und Referenzmaterial
Bietet	Tools und Datenzugriff	Wissen, Workflows, Referenzmaterial

Aspekt	MCP	Skill
Beispiele	Slack-Integration, Datenbankabfragen, Browser-Steuerung	Code-Review-Checkliste, Bereitstellungs-Workflow, API-Stilhandbuch

Diese lösen unterschiedliche Probleme und funktionieren gut zusammen:

MCP gibt Claude die Möglichkeit, mit externen Systemen zu interagieren. Ohne MCP kann Claude Ihre Datenbank nicht abfragen oder auf Slack posten.

Skills geben Claude Wissen darüber, wie diese Tools effektiv verwendet werden, plus Workflows, die Sie mit `<name>` auslösen können. Ein Skill könnte Ihr Team-Datenbankschema und Abfragemuster enthalten, oder einen `/post-to-slack`-Workflow mit Ihren Team-Nachrichtenformatierungsregeln.

Beispiel: Ein MCP-Server verbindet Claude mit Ihrer Datenbank. Ein Skill lehrt Claude Ihr Datenmodell, häufige Abfragemuster und welche Tabellen für verschiedene Aufgaben verwendet werden.

Verstehen Sie, wie Funktionen sich schichten

Funktionen können auf mehreren Ebenen definiert werden: benutzerübergreifend, pro Projekt, über Plugins oder durch verwaltete Richtlinien. Sie können auch CLAUDE.md-Dateien in Unterverzeichnissen verschachteln oder Skills in bestimmten Paketen eines Monorepos platzieren. Wenn dieselbe Funktion auf mehreren Ebenen vorhanden ist, so schichten sie sich:

- **CLAUDE.md-Dateien** sind additiv: alle Ebenen tragen gleichzeitig Inhalte zu Claudes Kontext bei. Dateien aus Ihrem Arbeitsverzeichnis und darüber werden beim Start geladen; Unterverzeichnisse werden geladen, wenn Sie darin arbeiten. Wenn Anweisungen in Konflikt geraten, nutzt Claude sein Urteilsvermögen, um sie zu reconcilieren, wobei spezifischere Anweisungen typischerweise Vorrang haben. Siehe [wie CLAUDE.md-Dateien geladen werden](#).
- **Skills und Subagents** überschreiben nach Name: wenn derselbe Name auf mehreren Ebenen vorhanden ist, gewinnt eine Definition basierend auf Priorität (verwaltet > Benutzer > Projekt für Skills; verwaltet > CLI-Flag > Projekt > Benutzer > Plugin für Subagents). Plugin-Skills sind [namensgebunden](#), um Konflikte zu vermeiden. Siehe [Skill-Erkennung](#) und [Subagent-Umfang](#).
- **MCP-Server** überschreiben nach Name: lokal > Projekt > Benutzer. Siehe [MCP-Umfang](#).

- **Hooks** zusammenführen: alle registrierten Hooks werden für ihre übereinstimmenden Ereignisse unabhängig von der Quelle ausgelöst. Siehe [Hooks](#).

Funktionen kombinieren

Jede Erweiterung löst ein anderes Problem: CLAUDE.md behandelt immer aktivem Kontext, Skills behandeln On-Demand-Wissen und Workflows, MCP behandelt externe Verbindungen, Subagents behandeln Isolation und Hooks behandeln Automatisierung. Echte Setups kombinieren sie basierend auf Ihrem Workflow.

Beispielsweise könnten Sie CLAUDE.md für Projektkonventionen, einen Skill für Ihren Bereitstellungs-Workflow, MCP zur Verbindung mit Ihrer Datenbank und einen Hook zum Ausführen von Linting nach jeder Bearbeitung verwenden. Jede Funktion behandelt das, wofür sie am besten geeignet ist.

Muster	Wie es funktioniert	Beispiel
Skill + MCP	MCP bietet die Verbindung; ein Skill lehrt Claude, sie gut zu nutzen	MCP verbindet sich mit Ihrer Datenbank, ein Skill dokumentiert Ihr Schema und Abfragemuster
Skill + Subagent	Ein Skill spawnt Subagents für parallele Arbeit	<code>/audit</code> -Skill startet Sicherheits-, Leistungs- und Style-Subagents, die in isoliertem Kontext arbeiten
CLAUDE.md + Skills	CLAUDE.md hält immer aktivem Regeln; Skills halten Referenzmaterial, das On Demand geladen wird	CLAUDE.md sagt "folgen Sie unseren API-Konventionen", ein Skill enthält das vollständige API-Stilhandbuch
Hook + MCP	Ein Hook löst externe Aktionen über MCP aus	Post-Edit-Hook sendet eine Slack-Benachrichtigung, wenn Claude kritische Dateien ändert

Verstehen Sie Kontextkosten

Jede Funktion, die Sie hinzufügen, verbraucht etwas von Claudes Kontext. Zu viel kann Ihr Kontextfenster füllen, aber es kann auch Rauschen hinzufügen, das Claude weniger effektiv macht; Skills werden möglicherweise nicht korrekt ausgelöst, oder Claude kann Ihre Konventionen aus den Augen verlieren. Das Verständnis dieser Kompromisse hilft Ihnen, ein effektives Setup zu erstellen.

Kontextkosten nach Funktion

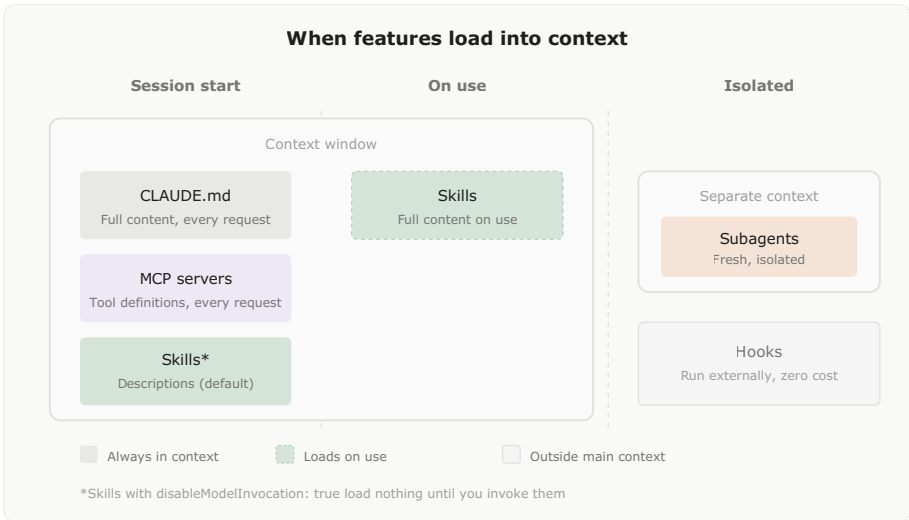
Jede Funktion hat eine andere Ladestrategie und Kontextkosten:

Funktion	Wann sie lädt	Was lädt	Kontextkosten
CLAU-DE.md	Sitzungsstart	Vollständiger Inhalt	Jede Anfrage
Skills	Sitzungsstart + wenn verwendet	Beschreibungen beim Start, vollständiger Inhalt bei Verwendung	Niedrig (Beschreibungen jede Anfrage)*
MCP-Server	Sitzungsstart	Alle Tool-Definitionen und Schemas	Jede Anfrage
Sub-agents	Wenn gespawnt	Frischer Kontext mit angegebenen Skills	Isoliert von Hauptsitzung
Hooks	Bei Auslösung	Nichts (läuft extern)	Null, es sei denn, Hook gibt zusätzlichen Kontext zurück

*Standardmäßig werden Skill-Beschreibungen beim Sitzungsstart geladen, damit Claude entscheiden kann, wann sie verwendet werden. Setzen Sie `disable-model-invocation: true` in das Frontmatter eines Skills, um es vollständig vor Claude zu verbergen, bis Sie es manuell aufrufen. Dies reduziert die Kontextkosten auf Null für Skills, die Sie nur selbst auslösen.

Verstehen Sie, wie Funktionen geladen werden

Jede Funktion wird an verschiedenen Punkten in Ihrer Sitzung geladen. Die folgenden Registerkarten erklären, wann jede geladen wird und was in den Kontext geht.



Kontextladung: CLAUDE.md und MCP werden beim Sitzungsstart geladen und bleiben in jeder Anfrage. Skills laden Beschreibungen beim Start, vollständigen Inhalt bei Aufruf. Subagents erhalten isolierten Kontext. Hooks laufen extern.

CLAUDE.md

Wann: Sitzungsstart

Was lädt: Vollständiger Inhalt aller CLAUDE.md-Dateien (verwaltet, Benutzer und Projektebenen).

Vererbung: Claude liest CLAUDE.md-Dateien aus Ihrem Arbeitsverzeichnis bis zur Wurzel und entdeckt verschachtelte in Unterverzeichnissen, wenn es auf diese Dateien zugreift. Weitere Informationen finden Sie unter [How CLAUDE.md files load](#).

Tip:

Halten Sie CLAUDE.md unter ~500 Zeilen. Verschieben Sie Referenzmaterial zu Skills, die On-Demand geladen werden.

Skills

Skills sind zusätzliche Funktionen in Claudes Toolkit. Sie können Referenzmaterial sein (wie ein API-Stilhandbuch) oder aufrufbare Workflows, die Sie mit `/<name>` auslösen (wie `/deploy`). Claude Code wird mit [gebündelten Skills](#) wie `/simplify`, `/batch` und `/debug` ausgeliefert, die sofort funktionieren. Sie können auch Ihre eigenen erstellen. Claude verwendet Skills, wenn angemessen, oder Sie können einen direkt aufrufen.

Wann: Hängt von der Konfiguration des Skills ab. Standardmäßig werden Beschreibungen beim Sitzungsstart geladen und vollständiger Inhalt bei Verwendung. Für nur-Benutzer-Skills (`disable-model-invocation: true`) wird nichts geladen, bis Sie sie aufrufen.

Was lädt: Für modell-aufrufbare Skills sieht Claude Namen und Beschreibungen in jeder Anfrage. Wenn Sie einen Skill mit `</name>` aufrufen oder Claude ihn automatisch lädt, wird der vollständige Inhalt in Ihre Konversation geladen.

Wie Claude Skills wählt: Claude gleicht Ihre Aufgabe gegen Skill-Beschreibungen ab, um zu entscheiden, welche relevant sind. Wenn Beschreibungen vage oder überlappend sind, kann Claude den falschen Skill laden oder einen verpassen, der helfen würde. Um Claude zu sagen, einen bestimmten Skill zu verwenden, rufen Sie ihn mit `</name>` auf. Skills mit `disable-model-invocation: true` sind für Claude unsichtbar, bis Sie sie aufrufen.

Kontextkosten: Niedrig bis verwendet. Nur-Benutzer-Skills haben Null-Kosten bis aufgerufen.

In Subagents: Skills funktionieren in Subagents anders. Anstelle von On-Demand-Laden werden Skills, die an einen Subagent übergeben werden, vollständig in seinen Kontext beim Start vorgeladen. Subagents erben Skills nicht von der Hauptsitzung; Sie müssen sie explizit angeben.

Tip:

Verwenden Sie `disable-model-invocation: true` für Skills mit Nebenwirkungen. Dies spart Kontext und stellt sicher, dass nur Sie sie auslösen.

MCP-Server

Wann: Sitzungsstart.

Was lädt: Alle Tool-Definitionen und JSON-Schemas von verbundenen Servern.

Kontextkosten: [Tool-Suche](#) (standardmäßig aktiviert) lädt MCP-Tools bis zu 10% des Kontexts und verschiebt den Rest bis zur Notwendigkeit.

Zuverlässigkeitshinweis: MCP-Verbindungen können während einer Sitzung stillschweigend fehlschlagen. Wenn ein Server die Verbindung trennt, verschwinden seine Tools ohne Warnung. Claude kann versuchen, ein Tool zu verwenden, das nicht mehr vorhanden ist. Wenn Sie bemerken, dass Claude ein MCP-Tool nicht verwenden kann, auf das es zuvor zugreifen konnte, überprüfen Sie die Verbindung mit `/mcp`.

Tip:

Führen Sie `/mcp` aus, um Token-Kosten pro Server zu sehen. Trennen Sie Server, die Sie nicht aktiv verwenden.

Subagents

Wann: On Demand, wenn Sie oder Claude einen für eine Aufgabe spawnt.

Was lädt: Frischer, isolierter Kontext, der Folgendes enthält:

- Der System-Prompt (geteilt mit Parent für Cache-Effizienz)
- Vollständiger Inhalt von Skills, die im `skills:`-Feld des Agenten aufgelistet sind
- CLAUDE.md und Git-Status (geerbt vom Parent)
- Welcher Kontext auch immer der Lead-Agent im Prompt übergibt

Kontextkosten: Isoliert von Hauptsitzung. Subagents erben Ihre Konversationshistorie oder aufgerufenen Skills nicht.

Tip:

Verwenden Sie Subagents für Arbeit, die Ihren vollständigen Konversationskontext nicht benötigt. Ihre Isolation verhindert, dass Ihre Hauptsitzung aufgebläht wird.

Hooks

Wann: Bei Auslösung. Hooks werden bei bestimmten Lebenszykluseignissen ausgelöst, wie Tool-Ausführung, Sitzungsgrenzen, Prompt-Einreichung, Berechtigungsanfragen und Komprimierung. Siehe [Hooks](#) für die vollständige Liste.

Was lädt: Standardmäßig nichts. Hooks laufen als externe Skripte.

Kontextkosten: Null, es sei denn, der Hook gibt Ausgabe zurück, die als Nachrichten zu Ihrer Konversation hinzugefügt wird.

Tip:

Hooks sind ideal für Nebenwirkungen (Linting, Logging), die Claudes Kontext nicht beeinflussen müssen.

Weitere Informationen

Jede Funktion hat ihr eigenes Handbuch mit Setup-Anweisungen, Beispielen und Konfigurationsoptionen.

- [CLAUDE.md](#) Speichern Sie Projektkontext, Konventionen und Anweisungen
- [Skills](#) Geben Sie Claude Fachkompetenz und wiederverwendbare Workflows
- [Subagents](#) Lagern Sie Arbeit in isoliertem Kontext aus
- [Agent teams](#) Koordinieren Sie mehrere Sitzungen, die parallel arbeiten
- [MCP](#) Verbinden Sie Claude mit externen Diensten
- [Hooks](#) Automatisieren Sie Workflows mit Hooks
- [Plugins](#) Bündeln und teilen Sie Feature-Sets
- [Marketplaces](#) Hosten und verteilen Sie Plugin-Sammlungen

Interaktiver Modus

Vollständige Referenz für Tastaturkürzel, Eingabemodi und interaktive Funktionen in Claude Code-Sitzungen.

Tastaturkürzel

Note:

Tastaturkürzel können je nach Plattform und Terminal variieren. Drücken Sie `?`, um die verfügbaren Kürzel für Ihre Umgebung anzuzeigen.

macOS-Benutzer: Option/Alt-Tastenkürzel (`Alt+B`, `Alt+F`, `Alt+Y`, `Alt+M`, `Alt+P`) erfordern die Konfiguration von Option als Meta in Ihrem Terminal:

- **iTerm2:** Einstellungen → Profile → Keys → Left/Right Option key auf „Esc+“ setzen
- **Terminal.app:** Einstellungen → Profile → Keyboard → „Use Option as Meta Key“ aktivieren
- **VS Code:** Einstellungen → Profile → Keys → Left/Right Option key auf „Esc+“ setzen

Weitere Informationen finden Sie unter [Terminal-Konfiguration](#).

Allgemeine Steuerelemente

Tastaturkürzel	Beschreibung	Kontext
<code>Ctrl+C</code>	Aktuelle Eingabe oder Generierung abbrechen	Standard-Interrupt
<code>Ctrl+F</code>	Alle Hintergrund-Agenten beenden. Zweimal innerhalb von 3 Sekunden drücken, um zu bestätigen	Steuerung von Hintergrund-Agenten
<code>Ctrl+D</code>	Claude Code-Sitzung beenden	EOF-Signal
<code>Ctrl+G</code>	Im Standard-Texteditor öffnen	Bearbeiten Sie Ihren Prompt oder benutzerdefinierte Antwort in Ihrem Standard-Texteditor

Tastaturkürzel	Beschreibung	Kontext
Ctrl+L	Terminal-Bildschirm löschen	Behält Gesprächsverlauf bei
Ctrl+O	Ausführliche Ausgabe umschalten	Zeigt detaillierte Tool-Nutzung und Ausführung
Ctrl+R	Reverse-Suche im Befehlsverlauf	Durchsuchen Sie vorherige Befehle interaktiv
Ctrl+V oder Cmd+V (iTerm2) oder Alt+V (Windows)	Bild aus Zwischenablage einfügen	Fügt ein Bild oder einen Pfad zu einer Bilddatei ein
Ctrl+B	Hintergrund-Ausführung von Aufgaben	Führt Bash-Befehle und Agenten im Hintergrund aus. Tmux-Benutzer drücken zweimal
Ctrl+T	Task-Liste umschalten	Zeigen oder verbergen Sie die Task-Liste im Terminal-Statusbereich
Left/Right arrows	Durch Dialog-Registerkarten navigieren	Navigieren Sie zwischen Registerkarten in Berechtigungsdialogen und Menüs
Up/Down arrows	Befehlsverlauf navigieren	Rufen Sie vorherige Eingaben ab
Esc + Esc	Zurückspulen oder zusammenfassen	Stellen Sie Code und/oder Konversation auf einen vorherigen Punkt wieder her, oder fassen Sie ab einer ausgewählten Nachricht zusammen
Shift+Tab oder Alt+M (einige Konfigurationen)	Berechtigungsmodi umschalten	Wechseln Sie zwischen Auto-Accept Mode, Plan Mode und normalem Modus.
Option+P (macOS) oder Alt+P (Windows/Linux)	Modell wechseln	Wechseln Sie Modelle, ohne Ihren Prompt zu löschen

Tastaturkürzel	Beschreibung	Kontext
<code>Option+T</code> (macOS) oder <code>Alt+T</code> (Windows/Linux)	Extended Thinking umschalten	Aktivieren oder deaktivieren Sie den Extended Thinking-Modus. Führen Sie zuerst <code>/terminal-setup</code> aus, um dieses Tastaturkürzel zu aktivieren

Textbearbeitung

Tastaturkürzel	Beschreibung	Kontext
<code>Ctrl+K</code>	Bis zum Zeilenende löschen	Speichert gelöschten Text zum Einfügen
<code>Ctrl+U</code>	Ganze Zeile löschen	Speichert gelöschten Text zum Einfügen
<code>Ctrl+Y</code>	Gelöschten Text einfügen	Fügen Sie Text ein, der mit <code>Ctrl+K</code> oder <code>Ctrl+U</code> gelöscht wurde
<code>Alt+Y</code> (nach <code>Ctrl+Y</code>)	Einfügen-Verlauf durchlaufen	Nach dem Einfügen können Sie zuvor gelöschten Text durchlaufen. Erfordert Option als Meta auf macOS
<code>Alt+B</code>	Cursor um ein Wort nach hinten bewegen	Wort-Navigation. Erfordert Option als Meta auf macOS
<code>Alt+F</code>	Cursor um ein Wort nach vorne bewegen	Wort-Navigation. Erfordert Option als Meta auf macOS

Design und Anzeige

Tastaturkürzel	Beschreibung	Kontext
<code>Ctrl+T</code>	Syntax-Hervorhebung für Code-Blöcke umschalten	Funktioniert nur im <code>/theme</code> -Auswahlmenü. Steuert, ob Code in Claudes Antworten Syntax-Färbung verwendet

Note:

Syntax-Hervorhebung ist nur in der nativen Version von Claude Code verfügbar.

Mehrzeilige Eingabe

Methode	Tastaturkürzel	Kontext
Schneller Escape	<code>\ + Enter</code>	Funktioniert in allen Terminals
macOS Standard	<code>Option+Enter</code>	Standard auf macOS
Shift+Enter	<code>Shift+Enter</code>	Funktioniert standardmäßig in iTerm2, WezTerm, Ghostty, Kitty
Steuersequenz	<code>Ctrl+J</code>	Zeilenumbruch-Zeichen für mehrzeilig
Einfügemodus	Direkt einfügen	Für Code-Blöcke, Protokolle

Tip:

Shift+Enter funktioniert ohne Konfiguration in iTerm2, WezTerm, Ghostty und Kitty. Für andere Terminals (VS Code, Alacritty, Zed, Warp) führen Sie `/terminal-setup` aus, um die Bindung zu installieren.

Schnellbefehle

Tastaturkürzel	Beschreibung	Notizen
<code>/</code> am Anfang	Befehl oder Skill	Siehe integrierte Befehle und Skills
<code>!</code> am Anfang	Bash-Modus	Führen Sie Befehle direkt aus und fügen Sie die Ausführungsausgabe zur Sitzung hinzu
<code>@</code>	Dateipfad-Erwähnung	Trigger-Dateipfad-Autovervollständigung

Integrierte Befehle

Geben Sie `/` in Claude Code ein, um alle verfügbaren Befehle anzuzeigen, oder geben Sie `/` gefolgt von beliebigen Buchstaben ein, um zu filtern. Nicht alle Befehle sind für jeden Benutzer sichtbar. Einige hängen von Ihrer Plattform, Ihrem Plan oder Ihrer Umgebung ab. Beispielsweise erscheint `/desktop` nur auf macOS und Windows, `/upgrade` und `/privacy-settings` sind nur für Pro- und Max-Pläne verfügbar, und `/terminal-setup` ist verborgen, wenn Ihr Terminal seine Tastenbindungen nativ unterstützt.

Claude Code wird auch mit [gebündelten Skills](#) wie `/simplify`, `/batch` und `/debug` ausgeliefert, die neben integrierten Befehlen angezeigt werden, wenn Sie `/` eingeben. Um Ihre eigenen Befehle zu erstellen, siehe [Skills](#).

In der folgenden Tabelle gibt `<arg>` ein erforderliches Argument an und `[arg]` ein optionales.

Befehl	Zweck
<code>/add-dir <path></code>	Fügen Sie ein neues Arbeitsverzeichnis zur aktuellen Sitzung hinzu
<code>/agents</code>	Verwalten Sie Agent -Konfigurationen
<code>/btw <question></code>	Stellen Sie eine schnelle Nebenfrage , ohne sie zum Gespräch hinzuzufügen
<code>/chrome</code>	Konfigurieren Sie Claude in Chrome -Einstellungen
<code>/clear</code>	Löschen Sie den Gesprächsverlauf und geben Sie Kontext frei. Aliase: <code>/reset</code> , <code>/new</code>

Befehl	Zweck
<code>/compact [instructions]</code>	Komprimieren Sie das Gespräch mit optionalen Fokus-Anweisungen
<code>/config</code>	Öffnen Sie die Einstellungen -Schnittstelle, um Design, Modell, Ausgabestil und andere Einstellungen anzupassen. Alias: <code>/settings</code>
<code>/context</code>	Visualisieren Sie die aktuelle Kontextnutzung als farbiges Gitter
<code>/copy</code>	Kopieren Sie die letzte Antwort des Assistenten in die Zwischenablage. Wenn Code-Blöcke vorhanden sind, zeigt eine interaktive Auswahl an, um einzelne Blöcke oder die vollständige Antwort auszuwählen
<code>/cost</code>	Zeigen Sie Token-Nutzungsstatistiken an. Siehe Kostenverfolgungs-Leitfaden für abonnementspezifische Details
<code>/desktop</code>	Setzen Sie die aktuelle Sitzung in der Claude Code Desktop-App fort. Nur macOS und Windows. Alias: <code>/app</code>
<code>/diff</code>	Öffnen Sie einen interaktiven Diff-Viewer, der nicht committete Änderungen und Pro-Turn-Diffs anzeigt. Verwenden Sie Links-/Rechts-Pfeile, um zwischen dem aktuellen Git-Diff und einzelnen Claude-Turns zu wechseln, und Auf-/Ab-Pfeile zum Durchsuchen von Dateien
<code>/doctor</code>	Diagnostizieren und überprüfen Sie Ihre Claude Code-Installation und -Einstellungen
<code>/exit</code>	Beenden Sie die CLI. Alias: <code>/quit</code>
<code>/export [filename]</code>	Exportieren Sie das aktuelle Gespräch als Klartext. Mit einem Dateinamen wird direkt in diese Datei geschrieben. Ohne öffnet einen Dialog zum Kopieren in die Zwischenablage oder Speichern in einer Datei
<code>/extra-usage</code>	Konfigurieren Sie zusätzliche Nutzung, um weiterzuarbeiten, wenn Ratenlimits erreicht werden
<code>/fast [on off]</code>	Schalten Sie Fast Mode ein oder aus
<code>/feedback [report]</code>	Senden Sie Feedback zu Claude Code. Alias: <code>/bug</code>
<code>/fork [name]</code>	Erstellen Sie einen Fork des aktuellen Gesprächs an diesem Punkt
<code>/help</code>	Zeigen Sie Hilfe und verfügbare Befehle an
<code>/hooks</code>	Verwalten Sie Hook -Konfigurationen für Tool-Ereignisse

Befehl	Zweck
<code>/ide</code>	Verwalten Sie IDE-Integrationen und zeigen Sie den Status an
<code>/init</code>	Initialisieren Sie das Projekt mit <code>CLAUDE.md</code> -Leitfaden
<code>/insights</code>	Generieren Sie einen Bericht, der Ihre Claude Code-Sitzungen analysiert, einschließlich Projektbereiche, Interaktionsmuster und Reibungspunkte
<code>/install-github-app</code>	Richten Sie die Claude GitHub Actions -App für ein Repository ein. Führt Sie durch die Auswahl eines Repos und die Konfiguration der Integration
<code>/install-slack-app</code>	Installieren Sie die Claude Slack-App. Öffnet einen Browser, um den OAuth-Flow abzuschließen
<code>/keybindings</code>	Öffnen oder erstellen Sie Ihre Tastenbindungs-Konfigurationsdatei
<code>/login</code>	Melden Sie sich bei Ihrem Anthropic-Konto an
<code>/logout</code>	Melden Sie sich von Ihrem Anthropic-Konto ab
<code>/mcp</code>	Verwalten Sie MCP-Serververbindungen und OAuth-Authentifizierung
<code>/memory</code>	Bearbeiten Sie <code>CLAUDE.md</code> -Speicherdateien, aktivieren oder deaktivieren Sie Auto-Memory , und zeigen Sie Auto-Memory-Einträge an
<code>/mobile</code>	Zeigen Sie QR-Code zum Herunterladen der Claude Mobile-App an. Aliase: <code>/ios</code> , <code>/android</code>
<code>/model [model]</code>	Wählen Sie das KI-Modell aus oder ändern Sie es. Für Modelle, die dies unterstützen, verwenden Sie Links-/Rechts-Pfeile, um Anstrengungsstufe anzupassen . Die Änderung wird sofort wirksam, ohne auf die Fertigstellung der aktuellen Antwort zu warten
<code>/passes</code>	Teilen Sie eine kostenlose Woche Claude Code mit Freunden. Nur sichtbar, wenn Ihr Konto berechtigt ist
<code>/permissions</code>	Zeigen Sie Berechtigungen an oder aktualisieren Sie sie. Alias: <code>/allowed-tools</code>
<code>/plan</code>	Geben Sie den Plan Mode direkt vom Prompt ein
<code>/plugin</code>	Verwalten Sie Claude Code Plugins
<code>/pr-comments [PR]</code>	Rufen Sie Kommentare aus einem GitHub Pull Request ab und zeigen Sie sie an. Erkennt automatisch den PR für den aktuellen Branch, oder übergeben Sie eine PR-URL oder -Nummer. Erfordert die <code>gh</code> CLI

Befehl	Zweck
<code>/privacy-settings</code>	Zeigen Sie Ihre Datenschutzeinstellungen an und aktualisieren Sie sie. Nur für Pro- und Max-Plan-Abonnenten verfügbar
<code>/release-notes</code>	Zeigen Sie das vollständige Änderungsprotokoll an, wobei die neueste Version am nächsten zu Ihrem Prompt liegt
<code>/reload-plugins</code>	Laden Sie alle aktiven Plugins neu, um ausstehende Änderungen anzuwenden, ohne neu zu starten. Meldet, was geladen wurde, und notiert alle Änderungen, die einen Neustart erfordern
<code>/remote-control</code>	Machen Sie diese Sitzung für Remote-Steuerung von claude.ai verfügbar. Alias: <code>/rc</code>
<code>/remote-env</code>	Konfigurieren Sie die Standard-Remote-Umgebung für Teleport-Sitzungen
<code>/rename [name]</code>	Benennen Sie die aktuelle Sitzung um. Ohne Namen wird automatisch eine aus dem Gesprächsverlauf generiert
<code>/resume [session]</code>	Setzen Sie ein Gespräch nach ID oder Name fort, oder öffnen Sie die Sitzungsauswahl. Alias: <code>/continue</code>
<code>/review</code>	Veraltet. Installieren Sie stattdessen das <code>code-review-Plugin</code> : <code>claude plugin install code-review@claude-code-marketplace</code>
<code>/rewind</code>	Spulen Sie das Gespräch und/oder den Code zu einem vorherigen Punkt zurück, oder fassen Sie ab einer ausgewählten Nachricht zusammen. Siehe Checkpointing . Alias: <code>/checkpoint</code>
<code>/sandbox</code>	Schalten Sie Sandbox-Modus um. Nur auf unterstützten Plattformen verfügbar
<code>/security-review</code>	Analysieren Sie ausstehende Änderungen im aktuellen Branch auf Sicherheitslücken. Überprüft den Git-Diff und identifiziert Risiken wie Injection, Auth-Probleme und Datenexposition
<code>/skills</code>	Listet verfügbare Skills auf
<code>/stats</code>	Visualisieren Sie tägliche Nutzung, Sitzungsverlauf, Streaks und Modelleinstellungen
<code>/status</code>	Öffnen Sie die Einstellungen-Schnittstelle (Status-Registerkarte) mit Version, Modell, Konto und Konnektivität

Befehl	Zweck
<code>/statusline</code>	Konfigurieren Sie Claude Codes Status-Zeile . Beschreiben Sie, was Sie möchten, oder führen Sie ohne Argumente aus, um automatisch von Ihrem Shell-Prompt zu konfigurieren
<code>/stickers</code>	Bestellen Sie Claude Code-Aufkleber
<code>/tasks</code>	Listet und verwaltet Hintergrund-Aufgaben
<code>/terminal-setup</code>	Konfigurieren Sie Terminal-Tastenbindungen für Shift+Enter und andere Kürzel. Nur in Terminals sichtbar, die dies benötigen, wie VS Code, Alacritty oder Warp
<code>/theme</code>	Ändern Sie das Farbschema. Umfasst helle und dunkle Varianten, farbenblind-zugängliche (daltonisierte) Designs und ANSI-Designs, die die Farbpalette Ihres Terminals verwenden
<code>/upgrade</code>	Öffnen Sie die Upgrade-Seite, um zu einem höheren Plan-Tier zu wechseln
<code>/usage</code>	Zeigen Sie Plan-Nutzungslimits und Ratenlimit-Status an
<code>/vim</code>	Wechseln Sie zwischen Vim- und Normal-Bearbeitungsmodi

MCP-Prompts

MCP-Server können Prompts verfügbar machen, die als Befehle angezeigt werden. Diese verwenden das Format `/mcp__<server>__<prompt>` und werden dynamisch von verbundenen Servern erkannt. Siehe [MCP-Prompts](#) für Details.

Vim-Editor-Modus

Aktivieren Sie Vim-ähnliche Bearbeitung mit dem `/vim`-Befehl oder konfigurieren Sie dauerhaft über `/config`.

Modusumschaltung

Befehl	Aktion	Aus Modus
<code>Esc</code>	Geben Sie den NORMAL-Modus ein	INSERT
<code>i</code>	Einfügen vor Cursor	NORMAL
<code>I</code>	Einfügen am Anfang der Zeile	NORMAL
<code>a</code>	Einfügen nach Cursor	NORMAL

Befehl	Aktion	Aus Modus
A	Einfügen am Ende der Zeile	NORMAL
o	Zeile unten öffnen	NORMAL
O	Zeile oben öffnen	NORMAL

Navigation (NORMAL-Modus)

Befehl	Aktion
h / j / k / l	Nach links/unten/oben/rechts bewegen
w	Nächstes Wort
e	Ende des Wortes
b	Vorheriges Wort
0	Anfang der Zeile
\$	Ende der Zeile
^	Erstes Nicht-Leerzeichen-Zeichen
gg	Anfang der Eingabe
G	Ende der Eingabe
f{char}	Zur nächsten Vorkommen des Zeichens springen
F{char}	Zum vorherigen Vorkommen des Zeichens springen
t{char}	Direkt vor das nächste Vorkommen des Zeichens springen
T{char}	Direkt nach das vorherige Vorkommen des Zeichens springen
;	Letzte f/F/t/T-Bewegung wiederholen
,	Letzte f/F/t/T-Bewegung in umgekehrter Reihenfolge wiederholen

Note:

Im Vim-Normal-Modus navigieren die Pfeiltasten den Befehlsverlauf, wenn sich der Cursor am Anfang oder Ende der Eingabe befindet und nicht weiter bewegt werden kann.

Bearbeitung (NORMAL-Modus)

Befehl	Aktion
x	Zeichen löschen
dd	Zeile löschen
D	Bis zum Zeilenende löschen
dw / de / db	Wort löschen/bis Ende/zurück
cc	Zeile ändern
C	Bis zum Zeilenende ändern
cw / ce / cb	Wort ändern/bis Ende/zurück
yy / Y	Zeile yanken (kopieren)
yw / ye / yb	Wort yanken/bis Ende/zurück
p	Nach Cursor einfügen
P	Vor Cursor einfügen
>>	Zeile einrücken
<<	Zeile ausrücken
J	Zeilen verbinden
.	Letzte Änderung wiederholen

Textobjekte (NORMAL-Modus)

Textobjekte funktionieren mit Operatoren wie **d**, **c** und **y**:

Befehl	Aktion
iw / aw	Inneres/um Wort
iW / aW	Inneres/um WORT (Leerzeichen-begrenzt)
i" / a"	Inneres/um doppelte Anführungszeichen
i' / a'	Inneres/um einfache Anführungszeichen

Befehl	Aktion
<code>i(/ a(</code>	Inneres/um Klammern
<code>i[/ a[</code>	Inneres/um eckige Klammern
<code>i{ / a{</code>	Inneres/um geschweifte Klammern

Befehlsverlauf

Claude Code verwaltet den Befehlsverlauf für die aktuelle Sitzung:

- Der Eingabeverlauf wird pro Arbeitsverzeichnis gespeichert
- Der Eingabeverlauf wird zurückgesetzt, wenn Sie `/clear` ausführen, um eine neue Sitzung zu starten. Das vorherige Gespräch der Sitzung wird beibehalten und kann fortgesetzt werden.
- Verwenden Sie Auf-/Ab-Pfeile zum Navigieren (siehe Tastaturkürzel oben)
- **Hinweis:** Verlaufserweiterung (`!`) ist standardmäßig deaktiviert

Reverse-Suche mit Ctrl+R

Drücken Sie `Ctrl+R`, um interaktiv durch Ihren Befehlsverlauf zu suchen:

1. **Suche starten:** Drücken Sie `Ctrl+R`, um die Reverse-Verlaufssuche zu aktivieren
2. **Abfrage eingeben:** Geben Sie Text ein, um in vorherigen Befehlen zu suchen. Der Suchbegriff wird in übereinstimmenden Ergebnissen hervorgehoben
3. **Übereinstimmungen navigieren:** Drücken Sie `Ctrl+R` erneut, um durch ältere Übereinstimmungen zu navigieren
4. **Übereinstimmung akzeptieren:**
 - Drücken Sie `Tab` oder `Esc`, um die aktuelle Übereinstimmung zu akzeptieren und die Bearbeitung fortzusetzen
 - Drücken Sie `Enter`, um die Übereinstimmung zu akzeptieren und den Befehl sofort auszuführen
5. **Suche abbrechen:**
 - Drücken Sie `Ctrl+C`, um abubrechen und Ihre ursprüngliche Eingabe wiederherzustellen
 - Drücken Sie `Backspace` bei leerer Suche, um abubrechen

Die Suche zeigt übereinstimmende Befehle mit dem hervorgehobenen Suchbegriff an, sodass Sie vorherige Eingaben finden und wiederverwenden können.

Hintergrund-Bash-Befehle

Claude Code unterstützt die Ausführung von Bash-Befehlen im Hintergrund, sodass Sie weiterarbeiten können, während lange laufende Prozesse ausgeführt werden.

Wie Backgrounding funktioniert

Wenn Claude Code einen Befehl im Hintergrund ausführt, führt es den Befehl asynchron aus und gibt sofort eine Hintergrund-Task-ID zurück. Claude Code kann auf neue Prompts reagieren, während der Befehl im Hintergrund weiter ausgeführt wird.

Um Befehle im Hintergrund auszuführen, können Sie entweder:

- Claude Code auffordern, einen Befehl im Hintergrund auszuführen
- Drücken Sie Ctrl+B, um eine reguläre Bash-Tool-Invokation in den Hintergrund zu verschieben. (Tmux-Benutzer müssen Ctrl+B zweimal drücken, da Tmux einen Präfix-Schlüssel hat.)

Wichtige Funktionen:

- Die Ausgabe wird gepuffert und Claude kann sie mit dem TaskOutput-Tool abrufen
- Hintergrund-Tasks haben eindeutige IDs zum Verfolgen und Abrufen von Ausgaben
- Hintergrund-Tasks werden automatisch bereinigt, wenn Claude Code beendet wird

Um alle Hintergrund-Task-Funktionalität zu deaktivieren, setzen Sie die Umgebungsvariable `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` auf `1`. Siehe [Umgebungsvariablen](#) für Details.

Häufig backgroundete Befehle:

- Build-Tools (webpack, vite, make)
- Paketmanager (npm, yarn, pnpm)
- Test-Runner (jest, pytest)
- Entwicklungsserver
- Lange laufende Prozesse (docker, terraform)

Bash-Modus mit `!`-Präfix

Führen Sie Bash-Befehle direkt aus, ohne Claude zu durchlaufen, indem Sie Ihre Eingabe mit `!` präfixieren:

```
! npm test
! git status
! ls -la
```

Bash-Modus:

- Fügt den Befehl und seine Ausgabe zum Gesprächskontext hinzu
- Zeigt Echtzeit-Fortschritt und Ausgabe
- Unterstützt das gleiche **Ctrl+B** -Backgrounding für lange laufende Befehle
- Erfordert nicht, dass Claude den Befehl interpretiert oder genehmigt
- Unterstützt verlaufs-basierte Autovervollständigung: Geben Sie einen Teilbefehl ein und drücken Sie **Tab**, um aus vorherigen **!** -Befehlen im aktuellen Projekt zu vervollständigen
- Beenden Sie mit **Escape**, **Backspace** oder **Ctrl+U** bei einer leeren Eingabeaufforderung

Dies ist nützlich für schnelle Shell-Operationen bei Beibehaltung des Gesprächskontexts.

Prompt-Vorschläge

Wenn Sie eine Sitzung zum ersten Mal öffnen, wird ein ausgegrauter Beispielfehl in der Prompt-Eingabe angezeigt, um Ihnen den Einstieg zu erleichtern. Claude Code wählt dies aus Ihrem Projekt-Git-Verlauf aus, sodass es Dateien widerspiegelt, an denen Sie kürzlich gearbeitet haben.

Nach Claudes Antwort werden Vorschläge weiterhin basierend auf Ihrem Gesprächsverlauf angezeigt, z. B. ein Folgeschritt aus einer mehrteiligen Anfrage oder eine natürliche Fortsetzung Ihres Workflows.

- Drücken Sie **Tab**, um den Vorschlag zu akzeptieren, oder drücken Sie **Enter**, um zu akzeptieren und einzureichen
- Beginnen Sie zu tippen, um ihn zu verwerfen

Der Vorschlag wird als Hintergrund-Anfrage ausgeführt, die den Prompt-Cache des übergeordneten Gesprächs wiederverwenden, sodass die zusätzlichen Kosten minimal sind. Claude Code überspringt die Vorschlagsgenerierung, wenn der Cache kalt ist, um unnötige Kosten zu vermeiden.

Vorschläge werden automatisch nach dem ersten Turn eines Gesprächs, im nicht-interaktiven Modus und im Plan Mode übersprungen.

Um Prompt-Vorschläge vollständig zu deaktivieren, setzen Sie die Umgebungsvariable oder schalten Sie die Einstellung in `/config` um:

```
export CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION=false
```

Nebenfragen mit `/btw`

Verwenden Sie `/btw`, um eine schnelle Frage zu Ihrer aktuellen Arbeit zu stellen, ohne sie zum Gesprächsverlauf hinzuzufügen. Dies ist nützlich, wenn Sie eine schnelle Antwort möchten, aber nicht den Hauptkontext unordentlich machen oder Claude von einer lange laufenden Aufgabe ablenken möchten.

```
/btw what was the name of that config file again?
```

Nebenfragen haben vollständige Sichtbarkeit des aktuellen Gesprächs, sodass Sie nach Code fragen können, den Claude bereits gelesen hat, Entscheidungen, die es früher getroffen hat, oder alles andere aus der Sitzung. Die Frage und Antwort sind flüchtig: Sie erscheinen in einer verwerfbaren Überlagerung und treten niemals in den Gesprächsverlauf ein.

- **Verfügbar während Claude arbeitet:** Sie können `/btw` ausführen, auch während Claude eine Antwort verarbeitet. Die Nebenfrage wird unabhängig ausgeführt und unterbricht den Hauptturn nicht.
- **Kein Tool-Zugriff:** Nebenfragen beantworten nur aus dem, was bereits im Kontext ist. Claude kann keine Dateien lesen, Befehle ausführen oder suchen, wenn eine Nebenfrage beantwortet wird.
- **Einzelne Antwort:** Es gibt keine Folgeversuche. Wenn Sie einen Hin- und Herwechsel benötigen, verwenden Sie stattdessen einen normalen Prompt.
- **Niedrige Kosten:** Die Nebenfrage verwendet den Prompt-Cache des übergeordneten Gesprächs wieder, sodass die zusätzlichen Kosten minimal sind.

Drücken Sie **Space**, **Enter** oder **Escape**, um die Antwort zu verwerfen und zur Eingabeaufforderung zurückzukehren.

`/btw` ist das Gegenteil eines [Subagenten](#): Es sieht Ihr vollständiges Gespräch, hat aber keine Tools, während ein Subagent vollständige Tools hat, aber mit einem leeren Kontext beginnt. Verwenden Sie `/btw`, um nach dem zu fragen, was Claude bereits aus dieser Sitzung weiß; verwenden Sie einen Subagenten, um etwas Neues herauszufinden.

Task-Liste

Bei der Arbeit an komplexen, mehrstufigen Arbeiten erstellt Claude eine Task-Liste, um den Fortschritt zu verfolgen. Aufgaben werden im Statusbereich Ihres Terminals mit Indikatoren angezeigt, die zeigen, was ausstehend, in Bearbeitung oder abgeschlossen ist.

- Drücken Sie `Ctrl+T`, um die Task-Listen-Ansicht umzuschalten. Die Anzeige zeigt bis zu 10 Aufgaben gleichzeitig
- Um alle Aufgaben anzuzeigen oder zu löschen, fragen Sie Claude direkt: „show me all tasks“ oder „clear all tasks“
- Aufgaben bleiben über Kontext-Komprimierungen hinweg bestehen und helfen Claude, bei größeren Projekten organisiert zu bleiben
- Um eine Task-Liste über Sitzungen hinweg zu teilen, setzen Sie `CLAUDE_CODE_TASK_LIST_ID`, um ein benanntes Verzeichnis in `~/.claude/tasks/` zu verwenden: `CLAUDE_CODE_TASK_LIST_ID=my-project claude`
- Um zur vorherigen TODO-Liste zurückzukehren, setzen Sie `CLAUDE_CODE_ENABLE_TASKS=false`.

PR-Review-Status

Bei der Arbeit an einem Branch mit einem offenen Pull Request zeigt Claude Code einen anklickbaren PR-Link in der Fußzeile an (z. B. „PR #446“). Der Link hat eine farbige Unterstreichung, die den Review-Status anzeigt:

- Grün: genehmigt
- Gelb: Review ausstehend
- Rot: Änderungen angefordert
- Grau: Entwurf
- Lila: zusammengeführt

`Cmd+click` (Mac) oder `Ctrl+click` (Windows/Linux) auf den Link, um den Pull Request in Ihrem Browser zu öffnen. Der Status wird automatisch alle 60 Sekunden aktualisiert.

Note:

Der PR-Status erfordert, dass die `gh` CLI installiert und authentifiziert ist (`gh auth login`).

Siehe auch

- [Skills](#) - Benutzerdefinierte Prompts und Workflows

- [Checkpointing](#) - Spulen Sie Claudes Änderungen zurück und stellen Sie vorherige Zustände wieder her
- [CLI-Referenz](#) - Befehlszeilenflags und Optionen
- [Einstellungen](#) - Konfigurationsoptionen
- [Speicherverwaltung](#) - Verwalten von CLAUDE.md-Dateien

Best Practices für Claude Code

Tipps und Muster, um das Beste aus Claude Code herauszuholen – von der Konfiguration Ihrer Umgebung bis zur Skalierung über parallele Sessions.

Claude Code ist eine agentengesteuerte Coding-Umgebung. Im Gegensatz zu einem Chatbot, der Fragen beantwortet und wartet, kann Claude Code Ihre Dateien lesen, Befehle ausführen, Änderungen vornehmen und autonom Probleme lösen, während Sie zuschauen, umleiten oder sich ganz zurückziehen.

Dies ändert Ihre Arbeitsweise. Anstatt Code selbst zu schreiben und Claude zur Überprüfung zu bitten, beschreiben Sie, was Sie möchten, und Claude findet heraus, wie es zu bauen ist. Claude erkundet, plant und implementiert.

Aber diese Autonomie bringt immer noch eine Lernkurve mit sich. Claude arbeitet innerhalb bestimmter Einschränkungen, die Sie verstehen müssen.

Dieser Leitfaden behandelt Muster, die sich in den internen Teams von Anthropic und bei Ingenieuren, die Claude Code in verschiedenen Codebases, Sprachen und Umgebungen nutzen, als wirksam erwiesen haben. Informationen zur Funktionsweise der agentengesteuerten Schleife finden Sie unter [How Claude Code works](#).

Die meisten Best Practices basieren auf einer Einschränkung: Claudes Kontextfenster füllt sich schnell, und die Leistung verschlechtert sich, wenn es sich füllt.

Claudes Kontextfenster enthält Ihre gesamte Konversation, einschließlich jeder Nachricht, jeder Datei, die Claude liest, und jeder Befehlsausgabe. Dies kann sich jedoch schnell füllen. Eine einzelne Debugging-Sitzung oder Codebase-Erkundung könnte Zehntausende von Tokens generieren und verbrauchen.

Dies ist wichtig, da die LLM-Leistung abnimmt, wenn sich der Kontext füllt. Wenn das Kontextfenster voll wird, könnte Claude anfangen, frühere Anweisungen zu „vergessen“ oder mehr Fehler zu machen. Das Kontextfenster ist die wichtigste Ressource, die verwaltet werden muss. Verfolgen Sie die Kontextnutzung kontinuierlich mit einer [benutzerdefinierten Statuszeile](#), und siehe [Token-Nutzung reduzieren](#) für Strategien zur Reduzierung der Token-Nutzung.

Geben Sie Claude eine Möglichkeit, seine Arbeit zu überprüfen

Tip:

Fügen Sie Tests, Screenshots oder erwartete Ausgaben ein, damit Claude sich selbst überprüfen kann. Dies ist das Wichtigste, das Sie tun können.

Claude funktioniert dramatisch besser, wenn er seine eigene Arbeit überprüfen kann, wie Tests ausführen, Screenshots vergleichen und Ausgaben validieren.

Ohne klare Erfolgskriterien könnte es etwas produzieren, das richtig aussieht, aber tatsächlich nicht funktioniert. Sie werden zur einzigen Feedback-Schleife, und jeder Fehler erfordert Ihre Aufmerksamkeit.

Strategie	Vorher	Nachher
Überprüfungskriterien bereitstellen	„implementiere eine Funktion, die E-Mail-Adressen validiert“	„schreibe eine validateE-mail-Funktion. Beispiel-Testfälle: user@example.com ist wahr, invalid.com ist falsch, user@.com ist falsch. führe die Tests nach der Implementierung aus“
UI-Änderungen visuell überprüfen	„mache das Dashboard besser aussehen“	„[Screenshot einfügen] implementiere dieses Design. mache einen Screenshot des Ergebnisses und vergleiche ihn mit dem Original. liste Unterschiede auf und behebe sie“
Grundursachen beheben, nicht Symptome	„der Build schlägt fehl“	„der Build schlägt mit diesem Fehler fehl: [Fehler einfügen]. behebe ihn und überprüfe, dass der Build erfolgreich ist. behebe die Grundursache, unterdrücke den Fehler nicht“

UI-Änderungen können mit der [Claude in Chrome-Erweiterung](#) überprüft werden. Sie öffnet neue Registerkarten in Ihrem Browser, testet die UI und iteriert, bis der Code funktioniert.

Ihre Überprüfung kann auch eine Test-Suite, ein Linter oder ein Bash-Befehl sein, der die Ausgabe überprüft. Investieren Sie darin, Ihre Überprüfung solide zu machen.

Erkunden Sie zuerst, dann planen Sie, dann codieren Sie

Tip:

Trennen Sie Forschung und Planung von der Implementierung, um zu vermeiden, das falsche Problem zu lösen.

Wenn Claude direkt zum Codieren springt, kann dies zu Code führen, der das falsche Problem löst. Verwenden Sie [Plan Mode](#), um Erkundung von Ausführung zu trennen.

Der empfohlene Workflow hat vier Phasen:

Step 1: Erkunden

Geben Sie Plan Mode ein. Claude liest Dateien und beantwortet Fragen, ohne Änderungen vorzunehmen.

```
read /src/auth and understand how we handle sessions and login.  
also look at how we manage environment variables for secrets.
```

Step 2: Planen

Bitten Sie Claude, einen detaillierten Implementierungsplan zu erstellen.

```
I want to add Google OAuth. What files need to change?  
What's the session flow? Create a plan.
```

Drücken Sie **Ctrl+G**, um den Plan in Ihrem Texteditor zur direkten Bearbeitung zu öffnen, bevor Claude fortfährt.

Step 3: Implementieren

Wechseln Sie zurück zum Normal Mode und lassen Sie Claude codieren, wobei Sie gegen seinen Plan überprüfen.

implement the OAuth flow from your plan. write tests for the callback handler, run the test suite and fix any failures.

Step 4: Commit

Bitten Sie Claude, mit einer aussagekräftigen Nachricht zu committen und einen PR zu erstellen.

commit with a descriptive message and open a PR

Plan Mode ist nützlich, bringt aber auch Overhead mit sich.

Für Aufgaben, bei denen der Umfang klar ist und die Lösung klein ist (wie das Beheben eines Tippfehlers, das Hinzufügen einer Log-Zeile oder das Umbenennen einer Variablen), bitten Sie Claude, es direkt zu tun.

Planung ist am nützlichsten, wenn Sie unsicher über den Ansatz sind, wenn die Änderung mehrere Dateien ändert, oder wenn Sie mit dem zu ändernden Code nicht vertraut sind. Wenn Sie den Diff in einem Satz beschreiben könnten, überspringen Sie den Plan.

Geben Sie spezifischen Kontext in Ihren Prompts an

Tip:

Je präziser Ihre Anweisungen sind, desto weniger Korrekturen benötigen Sie.

Claude kann Absichten ableiten, aber er kann nicht Ihre Gedanken lesen. Verweisen Sie auf spezifische Dateien, erwähnen Sie Einschränkungen und zeigen Sie auf Beispielmuster.

Strategie	Vorher	Nachher
Begrenzen Sie die Aufgabe. Geben Sie an, welche Datei, welches Szenario und Testpräferenzen.	<i>„füge Tests für foo.py hinzu“</i>	<i>„schreibe einen Test für foo.py, der den Edge Case abdeckt, in dem der Benutzer abgemeldet ist. vermeide Mocks.“</i>

Strategie	Vorher	Nachher
Zeigen Sie auf Quellen. Leiten Sie Claude zur Quelle, die eine Frage beantworten kann.	<i>„warum hat ExecutionFactory eine so seltsame API?“</i>	<i>„schaue dir die Git-Historie von ExecutionFactory an und fasse zusammen, wie seine API entstanden ist“</i>
Verweisen Sie auf vorhandene Muster. Zeigen Sie Claude Muster in Ihrer Codebase.	<i>„füge ein Calendar-Widget hinzu“</i>	<i>„schaue dir an, wie vorhandene Widgets auf der Startseite implementiert sind, um die Muster zu verstehen. HotDogWidget.php ist ein gutes Beispiel. folge dem Muster, um ein neues Calendar-Widget zu implementieren, das dem Benutzer ermöglicht, einen Monat auszuwählen und vorwärts/rückwärts zu blättern, um ein Jahr auszuwählen. baue von Grund auf ohne Bibliotheken außer denen, die bereits in der Codebase verwendet werden.“</i>
Beschreiben Sie das Symptom. Geben Sie das Symptom, den wahrscheinlichen Ort und an, wie „behoben“ aussieht.	<i>„behebe den Login-Bug“</i>	<i>„Benutzer berichten, dass Login nach Session-Timeout fehlschlägt. überprüfe den Auth-Flow in src/auth/, besonders Token-Refresh. schreibe einen fehlgeschlagenen Test, der das Problem reproduziert, dann behebe es“</i>

Vage Prompts können nützlich sein, wenn Sie erkunden und Kurskorrekturen vornehmen können. Ein Prompt wie **„was würdest du in dieser Datei verbessern?“** kann Dinge an die Oberfläche bringen, an die Sie nicht gedacht hätten zu fragen.

Geben Sie umfangreiche Inhalte an

Tip:

Verwenden Sie `@`, um auf Dateien zu verweisen, fügen Sie Screenshots/Bilder ein oder leiten Sie Daten direkt weiter.

Sie können Claude auf mehrere Arten umfangreiche Daten bereitstellen:

- **Verweisen Sie auf Dateien mit `@`** anstatt zu beschreiben, wo Code lebt. Claude liest die Datei, bevor er antwortet.
- **Fügen Sie Bilder direkt ein.** Kopieren/fügen Sie Bilder ein oder ziehen Sie sie in den Prompt.
- **Geben Sie URLs** für Dokumentation und API-Referenzen an. Verwenden Sie `/permissions`, um häufig verwendete Domains auf die Whitelist zu setzen.
- **Leiten Sie Daten weiter**, indem Sie `cat error.log | claude` ausführen, um Datei-inhalte direkt zu senden.
- **Lassen Sie Claude abrufen, was es braucht.** Sagen Sie Claude, dass es Kontext selbst mit Bash-Befehlen, MCP-Tools oder durch Lesen von Dateien abrufen soll.

Konfigurieren Sie Ihre Umgebung

Ein paar Einrichtungsschritte machen Claude Code über alle Ihre Sessions hinweg erheblich effektiver. Einen vollständigen Überblick über Erweiterungsfunktionen und wann Sie jede verwenden sollten, finden Sie unter [Extend Claude Code](#).

Schreiben Sie eine effektive CLAUDE.md

Tip:

Führen Sie `/init` aus, um eine Starter-CLAUDE.md-Datei basierend auf Ihrer aktuellen Projektstruktur zu generieren, und verfeinern Sie sie dann im Laufe der Zeit.

CLAUDE.md ist eine spezielle Datei, die Claude zu Beginn jeder Konversation liest. Fügen Sie Bash-Befehle, Code-Stil und Workflow-Regeln ein. Dies gibt Claude persistenten Kontext, den es nicht aus Code allein ableiten kann.

Der `/init`-Befehl analysiert Ihre Codebase, um Build-Systeme, Test-Frameworks und Code-Muster zu erkennen, und gibt Ihnen eine solide Grundlage zum Verfeinern.

Es gibt kein erforderliches Format für CLAUDE.md-Dateien, aber halten Sie es kurz und für Menschen lesbar. Zum Beispiel:

```
## Code style
- Use ES modules (import/export) syntax, not CommonJS (require)
- Destructure imports when possible (eg. import { foo } from 'bar')

## Workflow
- Be sure to typecheck when you're done making a series of code changes
- Prefer running single tests, and not the whole test suite, for performance
```

CLAUDE.md wird jede Session geladen, also fügen Sie nur Dinge ein, die weit verbreitet gelten. Für Domänenwissen oder Workflows, die nur manchmal relevant sind, verwenden Sie stattdessen [skills](#). Claude lädt sie bei Bedarf, ohne jede Konversation zu überlasten.

Halten Sie es prägnant. Fragen Sie sich für jede Zeile: „Würde das Entfernen dieser Zeile dazu führen, dass Claude Fehler macht?“ Wenn nicht, streichen Sie es. Überladene CLAUDE.md-Dateien führen dazu, dass Claude Ihre tatsächlichen Anweisungen ignoriert!

✓ Einschließen	✗ Ausschließen
Bash-Befehle, die Claude nicht erraten kann	Alles, was Claude durch Lesen von Code herausfinden kann
Code-Stil-Regeln, die von Standardwerten abweichen	Standard-Sprachkonventionen, die Claude bereits kennt
Test-Anweisungen und bevorzugte Test-Runner	Detaillierte API-Dokumentation (verlinken Sie stattdessen auf Docs)
Repository-Etikette (Branch-Naming, PR-Konventionen)	Informationen, die sich häufig ändern
Architektonische Entscheidungen, die für Ihr Projekt spezifisch sind	Lange Erklärungen oder Tutorials
Entwicklungsumgebungs-Eigenheiten (erforderliche Umgebungsvariablen)	Datei-für-Datei-Beschreibungen der Codebase
Häufige Fallstricke oder nicht offensichtliche Verhaltensweisen	Selbstverständliche Praktiken wie „schreibe sauberen Code“

Wenn Claude etwas tut, das Sie nicht möchten, obwohl es eine Regel dagegen gibt, ist die Datei wahrscheinlich zu lang und die Regel geht verloren. Wenn Claude Fragen stellt, die in CLAUDE.md beantwortet werden, könnte die Formulierung mehrdeutig sein. Behandeln Sie CLAUDE.md wie Code: überprüfen Sie es, wenn etwas schiefgeht, bereinigen Sie es regelmäßig, und testen Sie Änderungen, indem Sie beobachten, ob sich Claudes Verhalten tatsächlich ändert.

Sie können Anweisungen durch Hinzufügen von Betonung (z. B. „WICHTIG“ oder „DUMUSST“) abstimmen, um die Einhaltung zu verbessern. Überprüfen Sie CLAUDE.md in Git, damit Ihr Team beitragen kann. Die Datei nimmt im Laufe der Zeit an Wert zu.

CLAUDE.md-Dateien können zusätzliche Dateien mit der `@path/to/import`-Syntax importieren:

```
See @README.md for project overview and @package.json for available npm commands.
```

```
## Additional Instructions
```

- Git workflow: @docs/git-instructions.md
- Personal overrides: @~/ .claude/my-project-instructions.md

Sie können CLAUDE.md-Dateien an mehreren Orten platzieren:

- **Home-Ordner** (`~/ .claude/CLAUDE.md`): gilt für alle Claude-Sessions
- **Projekt-Root** (`./CLAUDE.md`): überprüfen Sie in Git, um mit Ihrem Team zu teilen
- **Übergeordnete Verzeichnisse**: nützlich für Monorepos, bei denen sowohl `root/CLAUDE.md` als auch `root/foo/CLAUDE.md` automatisch eingezogen werden
- **Untergeordnete Verzeichnisse**: Claude zieht untergeordnete CLAUDE.md-Dateien bei Bedarf ein, wenn mit Dateien in diesen Verzeichnissen gearbeitet wird

Konfigurieren Sie Berechtigungen

Tip:

Verwenden Sie `/permissions`, um sichere Befehle auf die Whitelist zu setzen, oder `/sandbox` für Isolation auf Betriebssystemebene. Dies reduziert Unterbrechungen, während Sie die Kontrolle behalten.

Standardmäßig fordert Claude Code Berechtigung für Aktionen an, die Ihr System ändern könnten: Dateischreibvorgänge, Bash-Befehle, MCP-Tools usw. Dies ist sicher, aber mühsam. Nach der zehnten Genehmigung überprüfen Sie nicht wirklich mehr, Sie klicken einfach durch. Es gibt zwei Möglichkeiten, diese Unterbrechungen zu reduzieren:

- **Berechtigungs-Whitelists:** erlauben Sie spezifische Tools, die Sie kennen und die sicher sind (wie `npm run lint` oder `git commit`)
- **Sandboxing:** aktivieren Sie Isolation auf Betriebssystemebene, die Dateisystem- und Netzwerkzugriff einschränkt und Claude ermöglicht, freier innerhalb definierter Grenzen zu arbeiten

Alternativ verwenden Sie `--dangerously-skip-permissions`, um alle Berechtigungsprüfungen für enthaltene Workflows wie das Beheben von Lint-Fehlern oder das Generieren von Boilerplate zu umgehen.

Warning:

Das Zulassen, dass Claude beliebige Befehle ausführt, kann zu Datenverlust, Systemkorruption oder Datenexfiltration durch Prompt-Injection führen. Verwenden Sie `--dangerously-skip-permissions` nur in einer Sandbox ohne Internetzugang.

Lesen Sie mehr über [Konfigurieren von Berechtigungen](#) und [Aktivieren von Sandboxing](#).

Verwenden Sie CLI-Tools

Tip:

Sagen Sie Claude Code, dass es CLI-Tools wie `gh`, `aws`, `gcloud` und `sentry-cli` bei der Interaktion mit externen Diensten verwenden soll.

CLI-Tools sind die kontexteffizienteste Möglichkeit, mit externen Diensten zu interagieren. Wenn Sie GitHub verwenden, installieren Sie die `gh`-CLI. Claude weiß, wie man sie zum Erstellen von Issues, Öffnen von Pull Requests und Lesen von Kommentaren verwendet. Ohne `gh` kann Claude immer noch die GitHub-API verwenden, aber un-authentifizierte Anfragen treffen oft auf Rate Limits.

Claude ist auch effektiv beim Erlernen von CLI-Tools, die er nicht bereits kennt.

Versuchen Sie Prompts wie `Use 'foo-cli-tool --help' to learn about foo tool, then use it to solve A, B, C.`

Verbinden Sie MCP-Server

Tip:

Führen Sie `claude mcp add` aus, um externe Tools wie Notion, Figma oder Ihre Datenbank zu verbinden.

Mit [MCP-Servern](#) können Sie Claude bitten, Funktionen von Issue-Trackern zu implementieren, Datenbanken abzufragen, Überwachungsdaten zu analysieren, Designs von Figma zu integrieren und Workflows zu automatisieren.

Richten Sie Hooks ein

Tip:

Verwenden Sie Hooks für Aktionen, die jedes Mal mit null Ausnahmen stattfinden müssen.

[Hooks](#) führen Skripte automatisch an bestimmten Punkten in Claudes Workflow aus. Im Gegensatz zu CLAUDE.md-Anweisungen, die beratend sind, sind Hooks deterministisch und garantieren, dass die Aktion stattfindet.

Claude kann Hooks für Sie schreiben. Versuchen Sie Prompts wie „Schreibe einen Hook, der eslint nach jeder Dateibearbeitung ausführt“ oder „Schreibe einen Hook, der Schreibvorgänge in den Migrations-Ordner blockiert.“ Führen Sie `/hooks` für interaktive Konfiguration aus, oder bearbeiten Sie `.claude/settings.json` direkt.

Erstellen Sie Skills

Tip:

Erstellen Sie `SKILL.md`-Dateien in `.claude/skills/`, um Claude Domänenwissen und wiederverwendbare Workflows zu geben.

[Skills](#) erweitern Claudes Wissen mit Informationen, die für Ihr Projekt, Team oder Ihre Domäne spezifisch sind. Claude wendet sie automatisch an, wenn relevant, oder Sie können sie direkt mit `/skill-name` aufrufen.

Erstellen Sie einen Skill, indem Sie ein Verzeichnis mit einer `SKILL.md` zu `.claude/skills/` hinzufügen:

```
---
name: api-conventions
description: REST API design conventions for our services
---

## API Conventions

- Use kebab-case for URL paths
- Use camelCase for JSON properties
- Always include pagination for list endpoints
- Version APIs in the URL path (/v1/, /v2/)
```

Skills können auch wiederverwendbare Workflows definieren, die Sie direkt aufrufen:

```
---
name: fix-issue
description: Fix a GitHub issue
disable-model-invocation: true
---

Analyze and fix the GitHub issue: $ARGUMENTS.

1. Use `gh issue view` to get the issue details
2. Understand the problem described in the issue
3. Search the codebase for relevant files
4. Implement the necessary changes to fix the issue
5. Write and run tests to verify the fix
6. Ensure code passes linting and type checking
7. Create a descriptive commit message
8. Push and create a PR
```

Führen Sie `/fix-issue 1234` aus, um es aufzurufen. Verwenden Sie `disable-model-invocation: true` für Workflows mit Nebenwirkungen, die Sie manuell auslösen möchten.

Erstellen Sie benutzerdefinierte Subagents

Tip:

Definieren Sie spezialisierte Assistenten in `.claude/agents/`, an die Claude für isolierte Aufgaben delegieren kann.

[Subagents](#) laufen in ihrem eigenen Kontext mit ihrem eigenen Satz erlaubter Tools. Sie sind nützlich für Aufgaben, die viele Dateien lesen oder spezialisierte Aufmerksamkeit benötigen, ohne Ihre Hauptkonversation zu überlasten.

```
---
name: security-reviewer
description: Reviews code for security vulnerabilities
tools: Read, Grep, Glob, Bash
model: opus
---

You are a senior security engineer. Review code for:
- Injection vulnerabilities (SQL, XSS, command injection)
- Authentication and authorization flaws
- Secrets or credentials in code
- Insecure data handling

Provide specific line references and suggested fixes.
```

Sagen Sie Claude explizit, dass es Subagents verwenden soll: „*Verwende einen Subagent, um diesen Code auf Sicherheitsprobleme zu überprüfen.*“

Installieren Sie Plugins

Tip:

Führen Sie `/plugin` aus, um den Marketplace zu durchsuchen. Plugins fügen Skills, Tools und Integrationen ohne Konfiguration hinzu.

[Plugins](#) bündeln Skills, Hooks, Subagents und MCP-Server in eine einzelne installierbare Einheit aus der Community und von Anthropic. Wenn Sie mit einer typisierten Sprache arbeiten, installieren Sie ein [Code-Intelligence-Plugin](#), um Claude präzise Symbol-Navigation und automatische Fehlererkennung nach Bearbeitungen zu geben.

Anleitungen zur Auswahl zwischen Skills, Subagents, Hooks und MCP finden Sie unter [Extend Claude Code](#).

Kommunizieren Sie effektiv

Die Art und Weise, wie Sie mit Claude Code kommunizieren, hat einen großen Einfluss auf die Qualität der Ergebnisse.

Stellen Sie Codebase-Fragen

Tip:

Stellen Sie Claude Fragen, die Sie einem Senior Engineer stellen würden.

Wenn Sie sich in eine neue Codebase einarbeiten, verwenden Sie Claude Code zum Lernen und Erkunden. Sie können Claude die gleichen Fragen stellen, die Sie einem anderen Engineer stellen würden:

- Wie funktioniert Logging?
- Wie erstelle ich einen neuen API-Endpunkt?
- Was macht `async move { ... }` auf Zeile 134 von `foo.rs`?
- Welche Edge Cases behandelt `CustomerOnboardingFlowImpl`?
- Warum ruft dieser Code `foo()` anstelle von `bar()` auf Zeile 333 auf?

Die Verwendung von Claude Code auf diese Weise ist ein effektiver Onboarding-Workflow, der die Einarbeitungszeit verbessert und die Belastung anderer Engineers reduziert. Keine spezielle Prompt-Formulierung erforderlich: stellen Sie Fragen direkt.

Lassen Sie Claude Sie interviewen

Tip:

Für größere Features lassen Sie Claude Sie zuerst interviewen. Beginnen Sie mit einem minimalen Prompt und bitten Sie Claude, Sie mit dem `AskUserQuestion`-Tool zu interviewen.

Claude stellt Fragen zu Dingen, die Sie möglicherweise noch nicht berücksichtigt haben, einschließlich technischer Implementierung, UI/UX, Edge Cases und Tradeoffs.

I want to build [brief description]. Interview me in detail using the AskUserQuestion tool.

Ask about technical implementation, UI/UX, edge cases, concerns, and tradeoffs. Don't ask obvious questions, dig into the hard parts I might not have considered.

Keep interviewing until we've covered everything, then write a complete spec to SPEC.md.

Sobald die Spezifikation fertig ist, starten Sie eine neue Session, um sie auszuführen. Die neue Session hat einen sauberen Kontext, der sich vollständig auf die Implementierung konzentriert, und Sie haben eine geschriebene Spezifikation zum Referenzieren.

Verwalten Sie Ihre Session

Konversationen sind persistent und reversibel. Nutzen Sie dies zu Ihrem Vorteil!

Korrigieren Sie früh und oft

Tip:

Korrigieren Sie Claude, sobald Sie bemerken, dass es vom Weg abkommt.

Die besten Ergebnisse kommen aus engen Feedback-Schleifen. Obwohl Claude gelegentlich Probleme beim ersten Versuch perfekt löst, führt eine schnelle Korrektur im Allgemeinen zu besseren Lösungen schneller.

- **Esc** : stoppen Sie Claude mitten in einer Aktion mit der **Esc** -Taste. Der Kontext wird beibehalten, sodass Sie umleiten können.
- **Esc + Esc** **oder** **/rewind** : drücken Sie **Esc** zweimal oder führen Sie **/rewind** aus, um das Rewind-Menü zu öffnen und die vorherige Konversation und den Code-Status wiederherzustellen, oder fassen Sie eine ausgewählte Nachricht zusammen.
- **"Undo that"** : lassen Sie Claude seine Änderungen rückgängig machen.
- **/clear** : setzen Sie den Kontext zwischen nicht verwandten Aufgaben zurück. Lange Sessions mit irrelevantem Kontext können die Leistung reduzieren.

Wenn Sie Claude mehr als zweimal bei demselben Problem in einer Session korrigiert haben, ist der Kontext mit fehlgeschlagenen Ansätzen überladen. Führen Sie `/clear` aus und beginnen Sie mit einem spezifischeren Prompt, der das Gelernte einbezieht. Eine saubere Session mit einem besseren Prompt übertrifft fast immer eine lange Session mit angesammelten Korrekturen.

Verwalten Sie den Kontext aggressiv

Tip:

Führen Sie `/clear` zwischen nicht verwandten Aufgaben aus, um den Kontext zurückzusetzen.

Claude Code komprimiert automatisch die Konversationshistorie, wenn Sie sich den Kontextlimits nähern, was wichtigen Code und Entscheidungen bewahrt und Platz freimacht.

Während langer Sessions kann sich Claudes Kontextfenster mit irrelevanten Konversationen, Dateiinhalten und Befehlen füllen. Dies kann die Leistung reduzieren und Claude manchmal ablenken.

- Verwenden Sie `/clear` häufig zwischen Aufgaben, um das Kontextfenster vollständig zurückzusetzen
- Wenn die automatische Komprimierung ausgelöst wird, fasst Claude zusammen, was am wichtigsten ist, einschließlich Code-Muster, Dateizustände und wichtige Entscheidungen
- Für mehr Kontrolle führen Sie `/compact <instructions>` aus, wie `/compact Focus on the API changes`
- Um nur einen Teil der Konversation zu komprimieren, verwenden Sie `Esc + Esc` oder `/rewind`, wählen Sie einen Nachricht-Checkpoint und wählen Sie **Summarize from here**. Dies verdichtet Nachrichten von diesem Punkt an, während der frühere Kontext erhalten bleibt.
- Passen Sie das Komprimierungsverhalten in CLAUDE.md mit Anweisungen wie `„When compacting, always preserve the full list of modified files and any test commands“` an, um sicherzustellen, dass kritischer Kontext die Zusammenfassung überlebt
- Für schnelle Fragen, die nicht im Kontext bleiben müssen, verwenden Sie `/btw`. Die Antwort erscheint in einer verworfaren Überlagerung und gelangt niemals in die Konversationshistorie, sodass Sie ein Detail überprüfen können, ohne den Kontext zu vergrößern.

Verwenden Sie Subagents für Untersuchungen

Tip:

Delegieren Sie Forschung mit „`use subagents to investigate X`“. Sie erkunden in einem separaten Kontext und halten Ihre Hauptkonversation sauber für die Implementierung.

Da der Kontext Ihre grundlegende Einschränkung ist, sind Subagents eines der mächtigsten verfügbaren Tools. Wenn Claude eine Codebase erforscht, liest er viele Dateien, die alle Ihren Kontext verbrauchen. Subagents laufen in separaten Kontextfenstern und berichten Zusammenfassungen zurück:

```
Use subagents to investigate how our authentication system handles token
refresh, and whether we have any existing OAuth utilities I should reuse.
```

Der Subagent erkundet die Codebase, liest relevante Dateien und berichtet Erkenntnisse zurück, alles ohne Ihre Hauptkonversation zu überlasten.

Sie können Subagents auch zur Überprüfung verwenden, nachdem Claude etwas implementiert hat:

```
use a subagent to review this code for edge cases
```

Rewind mit Checkpoints

Tip:

Jede Aktion, die Claude macht, erstellt einen Checkpoint. Sie können Konversation, Code oder beides zu jedem vorherigen Checkpoint wiederherstellen.

Claude erstellt automatisch Checkpoints vor Änderungen. Doppeltippen Sie auf `Escape` oder führen Sie `/rewind` aus, um das Rewind-Menü zu öffnen. Sie können nur Konversation wiederherstellen, nur Code wiederherstellen, beides wiederherstellen oder eine ausgewählte Nachricht zusammenfassen. Siehe [Checkpointing](#) für Details.

Anstatt jeden Schritt sorgfältig zu planen, können Sie Claude bitten, etwas Riskantes zu versuchen. Wenn es nicht funktioniert, rewind und versuchen Sie einen anderen Ansatz. Checkpoints bleiben über Sessions hinweg erhalten, sodass Sie Ihr Terminal schließen und später immer noch rewind können.

Warning:

Checkpoints verfolgen nur Änderungen, die *von Claude* vorgenommen wurden, nicht externe Prozesse. Dies ist kein Ersatz für Git.

Setzen Sie Konversationen fort

Tip:

Führen Sie `claude --continue` aus, um dort weiterzumachen, wo Sie aufgehört haben, oder `--resume`, um aus aktuellen Sessions auszuwählen.

Claude Code speichert Konversationen lokal. Wenn sich eine Aufgabe über mehrere Sessions erstreckt, müssen Sie den Kontext nicht erneut erklären:

```
claude --continue    # Resume the most recent conversation
claude --resume      # Select from recent conversations
```

Verwenden Sie `/rename`, um Sessions aussagekräftige Namen wie `„oauth-migration“` oder `„debugging-memory-leak“` zu geben, damit Sie sie später finden können. Behandeln Sie Sessions wie Branches: verschiedene Workstreams können separate, persistente Kontexte haben.

Automatisieren und skalieren Sie

Sobald Sie mit einem Claude effektiv sind, multiplizieren Sie Ihre Ausgabe mit parallelen Sessions, nicht-interaktivem Modus und Fan-Out-Mustern.

Alles bisher geht von einem Menschen, einem Claude und einer Konversation aus. Aber Claude Code skaliert horizontal. Die Techniken in diesem Abschnitt zeigen, wie Sie mehr erreichen können.

Führen Sie nicht-interaktiven Modus aus

Tip:

Verwenden Sie `claude -p "prompt"` in CI, Pre-Commit-Hooks oder Skripten. Fügen Sie `--output-format stream-json` für Streaming-JSON-Ausgabe hinzu.

Mit `claude -p "your prompt"` können Sie Claude nicht-interaktiv ohne eine Session ausführen. Der nicht-interaktive Modus ist, wie Sie Claude in CI-Pipelines, Pre-Commit-Hooks oder jeden automatisierten Workflow integrieren. Die Ausgabeformate ermöglichen es Ihnen, Ergebnisse programmgesteuert zu analysieren: Klartext, JSON oder Streaming-JSON.

```
## One-off queries
claude -p "Explain what this project does"

## Structured output for scripts
claude -p "List all API endpoints" --output-format json

## Streaming for real-time processing
claude -p "Analyze this log file" --output-format stream-json
```

Führen Sie mehrere Claude-Sessions aus

Tip:

Führen Sie mehrere Claude-Sessions parallel aus, um die Entwicklung zu beschleunigen, isolierte Experimente auszuführen oder komplexe Workflows zu starten.

Es gibt drei Hauptmöglichkeiten, parallele Sessions auszuführen:

- [Claude Code Desktop-App](#): Verwalten Sie mehrere lokale Sessions visuell. Jede Session erhält ihren eigenen isolierten Worktree.
- [Claude Code im Web](#): Führen Sie auf der sicheren Cloud-Infrastruktur von Anthropic in isolierten VMs aus.
- [Agent Teams](#): Automatisierte Koordination mehrerer Sessions mit gemeinsamen Aufgaben, Messaging und einem Team Lead.

Über die Parallelisierung von Arbeit hinaus ermöglichen mehrere Sessions qualitätsorientierte Workflows. Ein frischer Kontext verbessert die Code-Überprüfung, da Claude nicht durch Code, den es gerade geschrieben hat, voreingenommen ist.

Verwenden Sie beispielsweise ein Writer/Reviewer-Muster:

Session A (Writer)	Session B (Reviewer)
Implement a rate limiter for our API endpoints	
	Review the rate limiter implementation in @src/middleware/rateLimiter.ts. Look for edge cases, race conditions, and consistency with our existing middleware patterns.
Here's the review feedback: [Session B output]. Address these issues.	

Sie können etwas Ähnliches mit Tests tun: lassen Sie einen Claude Tests schreiben, dann schreiben Sie einen anderen Code, um sie zu bestehen.

Fan Out über Dateien

Tip:

Schleifen Sie durch Aufgaben, die `claude -p` für jede aufrufen. Verwenden Sie `--allowedTools`, um Berechtigungen für Batch-Operationen zu begrenzen.

Für große Migrationen oder Analysen können Sie Arbeit über viele parallele Claude-Aufrufe verteilen:

Step 1: Generieren Sie eine Aufgabenliste

Lassen Sie Claude alle Dateien auflisten, die migriert werden müssen (z. B. `list all 2,000 Python files that need migrating`)

Step 2: Schreiben Sie ein Skript, um die Liste zu durchlaufen

```
for file in $(cat files.txt); do
  claude -p "Migrate $file from React to Vue. Return OK or FAIL." \
    --allowedTools "Edit,Bash(git commit *)"
done
```

Step 3: Testen Sie auf ein paar Dateien, dann führen Sie in großem Maßstab aus

Verfeinern Sie Ihren Prompt basierend auf dem, was bei den ersten 2-3 Dateien schiefliegt, dann führen Sie auf dem vollständigen Satz aus. Das `--allowedTools` -Flag beschränkt, was Claude tun kann, was wichtig ist, wenn Sie unbeaufsichtigt laufen.

Sie können Claude auch in vorhandene Daten-/Verarbeitungs-Pipelines integrieren:

```
claude -p "<your prompt>" --output-format json | your_command
```

Verwenden Sie `--verbose` zum Debuggen während der Entwicklung und schalten Sie es in der Produktion aus.

Vermeiden Sie häufige Fehlermuster

Dies sind häufige Fehler. Sie früh zu erkennen spart Zeit:

- **Die Kitchen-Sink-Session.** Sie beginnen mit einer Aufgabe, dann fragen Claude etwas Unverwandtes, dann gehen Sie zurück zur ersten Aufgabe. Der Kontext ist voll mit irrelevanten Informationen. > **Lösung:** `/clear` zwischen nicht verwandten Aufgaben.
- **Immer wieder korrigieren.** Claude macht etwas falsch, Sie korrigieren es, es ist immer noch falsch, Sie korrigieren erneut. Der Kontext ist mit fehlgeschlagenen Ansätzen verschmutzt. > **Lösung:** Nach zwei fehlgeschlagenen Korrekturen `/clear` und schreiben Sie einen besseren anfänglichen Prompt, der das Gelernte einbezieht.
- **Die über-spezifizierte CLAUDE.md.** Wenn Ihre CLAUDE.md zu lang ist, ignoriert Claude die Hälfte davon, weil wichtige Regeln in dem Lärm verloren gehen. > **Lösung:** Rücksichtslos bereinigen. Wenn Claude etwas bereits ohne die Anweisung richtig macht, löschen Sie es oder konvertieren Sie es in einen Hook.
- **Die Trust-then-Verify-Lücke.** Claude produziert eine plausibel aussehende Implementierung, die Edge Cases nicht behandelt. > **Lösung:** Geben Sie immer Überprüfung an (Tests, Skripte, Screenshots). Wenn Sie es nicht überprüfen können, versenden Sie es nicht.
- **Die unendliche Erkundung.** Sie bitten Claude, etwas zu „untersuchen“, ohne es zu begrenzen. Claude liest Hunderte von Dateien und füllt den Kontext. > **Lösung:** Begrenzen Sie Untersuchungen eng oder verwenden Sie Subagents, damit die Erkundung Ihren Hauptkontext nicht verbraucht.

Entwickeln Sie Ihre Intuition

Die Muster in diesem Leitfaden sind nicht in Stein gemeißelt. Sie sind Ausgangspunkte, die im Allgemeinen gut funktionieren, aber möglicherweise nicht optimal für jede Situation sind.

Manchmal *sollten* Sie den Kontext ansammeln lassen, weil Sie tief in einem komplexen Problem stecken und die Geschichte wertvoll ist. Manchmal sollten Sie die Planung überspringen und Claude es herausfinden lassen, weil die Aufgabe explorativ ist. Manchmal ist ein vager Prompt genau richtig, weil Sie sehen möchten, wie Claude das Problem interpretiert, bevor Sie es einschränken.

Achten Sie auf das, was funktioniert. Wenn Claude großartige Ausgabe produziert, bemerken Sie, was Sie getan haben: die Prompt-Struktur, den Kontext, den Sie bereitgestellt haben, den Modus, in dem Sie waren. Wenn Claude kämpft, fragen Sie warum. War der Kontext zu laut? Der Prompt zu vage? Die Aufgabe zu groß für einen Pass?

Im Laufe der Zeit werden Sie Intuition entwickeln, die kein Leitfaden erfassen kann. Sie werden wissen, wann Sie spezifisch und wann offen sein sollten, wann Sie planen und wann Sie erkunden sollten, wann Sie den Kontext löschen und wann Sie ihn ansammeln lassen sollten.

Verwandte Ressourcen

- [How Claude Code works](#): die agentengesteuerte Schleife, Tools und Kontextverwaltung
- [Extend Claude Code](#): Skills, Hooks, MCP, Subagents und Plugins
- [Common workflows](#): Schritt-für-Schritt-Rezepte zum Debuggen, Testen, PRs und mehr
- [CLAUDE.md](#): speichern Sie Projektkonventionen und persistenten Kontext

Häufige Arbeitsabläufe

Schritt-für-Schritt-Anleitungen zum Erkunden von Codebases, Beheben von Fehlern, Refaktorisierung, Testen und anderen alltäglichen Aufgaben mit Claude Code.

Diese Seite behandelt praktische Arbeitsabläufe für die alltägliche Entwicklung: Erkunden unbekannter Code, Debugging, Refaktorisierung, Schreiben von Tests, Erstellen von PRs und Verwalten von Sitzungen. Jeder Abschnitt enthält Beispiel-Prompts, die Sie an Ihre eigenen Projekte anpassen können. Für übergeordnete Muster und Tipps siehe [Best Practices](#).

Neue Codebases verstehen

Schnelle Codebase-Übersicht erhalten

Angenommen, Sie sind gerade einem neuen Projekt beigetreten und müssen dessen Struktur schnell verstehen.

Step 1: Navigieren Sie zum Projektroot-Verzeichnis

```
cd /path/to/project
```

Step 2: Starten Sie Claude Code

```
claude
```

Step 3: Fordern Sie einen allgemeinen Überblick an

```
give me an overview of this codebase
```

Step 4: Tauchen Sie tiefer in spezifische Komponenten ein

```
explain the main architecture patterns used here
```

what are the key data models?

how is authentication handled?

Tip:

Tipps:

- Beginnen Sie mit allgemeinen Fragen und grenzen Sie dann auf spezifische Bereiche ein
- Fragen Sie nach Coding-Konventionen und Mustern, die im Projekt verwendet werden
- Fordern Sie ein Glossar mit projektspezifischen Begriffen an

Relevanten Code finden

Angenommen, Sie müssen Code finden, der sich auf eine bestimmte Funktion oder Funktionalität bezieht.

Step 1: Bitten Sie Claude, relevante Dateien zu finden

find the files that handle user authentication

Step 2: Erhalten Sie Kontext darüber, wie Komponenten zusammenwirken

how do these authentication files work together?

Step 3: Verstehen Sie den Ausführungsfluss

trace the login process from front-end to database

Tip:

Tipps:

- Seien Sie spezifisch bei dem, was Sie suchen
- Verwenden Sie Domänensprache aus dem Projekt
- Installieren Sie ein [Code-Intelligence-Plugin](#) für Ihre Sprache, um Claude präzise 'Go to Definition'- und „Find References“-Navigation zu geben

Fehler effizient beheben

Angenommen, Sie sind auf eine Fehlermeldung gestoßen und müssen deren Quelle finden und beheben.

Step 1: Teilen Sie den Fehler mit Claude

```
I'm seeing an error when I run npm test
```

Step 2: Fordern Sie Empfehlungen zur Fehlerbehebung an

```
suggest a few ways to fix the @ts-ignore in user.ts
```

Step 3: Wenden Sie die Fehlerbehebung an

```
update user.ts to add the null check you suggested
```

Tip:

Tipps:

- Teilen Sie Claude den Befehl mit, um das Problem zu reproduzieren und einen Stack Trace zu erhalten
- Erwähnen Sie alle Schritte, um den Fehler zu reproduzieren
- Lassen Sie Claude wissen, ob der Fehler intermittierend oder konsistent ist

Code refaktorisieren

Angenommen, Sie müssen alten Code aktualisieren, um moderne Muster und Praktiken zu verwenden.

Step 1: Identifizieren Sie Legacy-Code zur Refaktorisierung

```
find deprecated API usage in our codebase
```

Step 2: Erhalten Sie Refaktorisierungsempfehlungen

```
suggest how to refactor utils.js to use modern JavaScript features
```

Step 3: Wenden Sie die Änderungen sicher an

```
refactor utils.js to use ES2024 features while maintaining the same behavior
```

Step 4: Überprüfen Sie die Refaktorisierung

```
run tests for the refactored code
```

Tip:

Tipps:

- Bitten Sie Claude, die Vorteile des modernen Ansatzes zu erklären
- Fordern Sie an, dass Änderungen die Rückwärtskompatibilität beibehalten, wenn nötig
- Führen Sie Refaktorisierung in kleinen, testbaren Schritten durch

Spezialisierte subagents verwenden

Angenommen, Sie möchten spezialisierte KI-subagents verwenden, um bestimmte Aufgaben effektiver zu bewältigen.

Step 1: Verfügbare subagents anzeigen

```
/agents
```

Dies zeigt alle verfügbaren subagents und ermöglicht es Ihnen, neue zu erstellen.

Step 2: subagents automatisch verwenden

Claude Code delegiert automatisch geeignete Aufgaben an spezialisierte subagents:

```
review my recent code changes for security issues
```

```
run all tests and fix any failures
```

Step 3: Fordern Sie explizit spezifische subagents an

```
use the code-reviewer subagent to check the auth module
```

```
have the debugger subagent investigate why users can't log in
```

Step 4: Erstellen Sie benutzerdefinierte subagents für Ihren Arbeitsablauf

```
/agents
```

Wählen Sie dann 'Create New subagent' und folgen Sie den Aufforderungen, um Folgendes zu definieren:

- Eine eindeutige Kennung, die den Zweck des subagent beschreibt (z. B. `code-reviewer`, `api-designer`).
- Wann Claude diesen Agent verwenden sollte
- Welche Tools er verwenden kann
- Ein System-Prompt, der die Rolle und das Verhalten des Agents beschreibt

Tip:

Tipps:

- Erstellen Sie projektspezifische subagents in `.claude/agents/` für die Teamfreigabe
- Verwenden Sie beschreibende `description`-Felder, um automatische Delegation zu ermöglichen
- Begrenzen Sie den Tool-Zugriff auf das, was jeder subagent tatsächlich benötigt
- Lesen Sie die [subagents-Dokumentation](#) für detaillierte Beispiele

Plan Mode für sichere Code-Analyse verwenden

Plan Mode weist Claude an, einen Plan zu erstellen, indem die Codebase mit schreibgeschützten Operationen analysiert wird. Dies ist perfekt zum Erkunden von Codebases, Planen komplexer Änderungen oder sicheren Überprüfen von Code. Im Plan Mode verwendet Claude `AskUserQuestion`, um Anforderungen zu sammeln und Ihre Ziele zu klären, bevor ein Plan vorgeschlagen wird.

Wann Plan Mode verwendet werden sollte

- **Multi-Step-Implementierung:** Wenn Ihre Funktion Änderungen an vielen Dateien erfordert
- **Code-Exploration:** Wenn Sie die Codebase gründlich erforschen möchten, bevor Sie etwas ändern
- **Interaktive Entwicklung:** Wenn Sie die Richtung mit Claude iterieren möchten

Wie Plan Mode verwendet wird

Aktivieren Sie Plan Mode während einer Sitzung

Sie können während einer Sitzung mit **Shift+Tab** in Plan Mode wechseln, um durch Berechtigungsmodi zu wechseln.

Wenn Sie sich im Normal Mode befinden, wechselt **Shift+Tab** zunächst in Auto-Accept Mode, angezeigt durch `▶▶ accept edits on` am unteren Rand des Terminals. Ein nachfolgendes **Shift+Tab** wechselt in Plan Mode, angezeigt durch `🛡️ plan mode on`.

Starten Sie eine neue Sitzung im Plan Mode

Um eine neue Sitzung im Plan Mode zu starten, verwenden Sie das Flag `--permission-mode plan`:

```
claude --permission-mode plan
```

Führen Sie „Headless“-Abfragen im Plan Mode aus

Sie können auch eine Abfrage im Plan Mode direkt mit `-p` ausführen (d. h. im [“Headless-Modus”](#)):

```
claude --permission-mode plan -p "Analyze the authentication system and suggest improvements"
```

Beispiel: Planen einer komplexen Refaktorisierung

```
claude --permission-mode plan
```

I need to refactor our authentication system to use OAuth2. Create a detailed migration plan.

Claude analysiert die aktuelle Implementierung und erstellt einen umfassenden Plan. Verfeinern Sie mit Folgefragen:

What about backward compatibility?

How should we handle database migration?

Tip:

Drücken Sie **Ctrl+G**, um den Plan in Ihrem Standard-Texteditor zu öffnen, wo Sie ihn direkt bearbeiten können, bevor Claude fortfährt.

Konfigurieren Sie Plan Mode als Standard

```
// .claude/settings.json
{
  "permissions": {
    "defaultMode": "plan"
  }
}
```

Weitere Konfigurationsoptionen finden Sie in der [Einstellungsdokumentation](#).

Mit Tests arbeiten

Angenommen, Sie müssen Tests für nicht abgedeckten Code hinzufügen.

Step 1: Identifizieren Sie nicht getesteten Code

```
find functions in NotificationsService.swift that are not covered by tests
```

Step 2: Generieren Sie Test-Gerüste

```
add tests for the notification service
```

Step 3: Fügen Sie aussagekräftige Testfälle hinzu

```
add test cases for edge conditions in the notification service
```

Step 4: Führen Sie Tests aus und überprüfen Sie sie

```
run the new tests and fix any failures
```

Claude kann Tests generieren, die den vorhandenen Mustern und Konventionen Ihres Projekts entsprechen. Wenn Sie Tests anfordern, seien Sie spezifisch darüber, welches Verhalten Sie überprüfen möchten. Claude untersucht Ihre vorhandenen Testdateien, um den Stil, die Frameworks und die Assertion-Muster zu entsprechen, die bereits verwendet werden.

Für umfassende Abdeckung bitten Sie Claude, Grenzfälle zu identifizieren, die Sie möglicherweise übersehen haben. Claude kann Ihre Code-Pfade analysieren und Tests für Fehlerbedingungen, Grenzwerte und unerwartete Eingaben vorschlagen, die leicht zu übersehen sind.

Pull Requests erstellen

Sie können Pull Requests erstellen, indem Sie Claude direkt fragen („create a pr for my changes“), oder Claude Schritt für Schritt führen:

Step 1: Fassen Sie Ihre Änderungen zusammen

```
summarize the changes I've made to the authentication module
```

Step 2: Generieren Sie einen Pull Request


```
create a pr
```

Step 3: Überprüfen und verfeinern Sie

```
enhance the PR description with more context about the security improvements
```

Wenn Sie einen PR mit `gh pr create` erstellen, wird die Sitzung automatisch mit diesem PR verknüpft. Sie können sie später mit `claude --from-pr <number>` fortsetzen.

Tip:

Überprüfen Sie den von Claude generierten PR vor dem Einreichen und bitten Sie Claude, potenzielle Risiken oder Überlegungen hervorzuheben.

Dokumentation verwalten

Angenommen, Sie müssen Dokumentation für Ihren Code hinzufügen oder aktualisieren.

Step 1: Identifizieren Sie nicht dokumentierten Code

```
find functions without proper JSDoc comments in the auth module
```

Step 2: Generieren Sie Dokumentation

```
add JSDoc comments to the undocumented functions in auth.js
```

Step 3: Überprüfen und verbessern Sie

```
improve the generated documentation with more context and examples
```

Step 4: Überprüfen Sie die Dokumentation

```
check if the documentation follows our project standards
```

Tip:

Tipps:

- Geben Sie den Dokumentationsstil an, den Sie möchten (JSDoc, Docstrings usw.)
- Fordern Sie Beispiele in der Dokumentation an
- Fordern Sie Dokumentation für öffentliche APIs, Schnittstellen und komplexe Logik an

Mit Bildern arbeiten

Angenommen, Sie müssen mit Bildern in Ihrer Codebase arbeiten und möchten Claudes Hilfe bei der Analyse von Bildinhalten.

Step 1: Fügen Sie ein Bild zum Gespräch hinzu

Sie können eine dieser Methoden verwenden:

1. Ziehen Sie ein Bild per Drag & Drop in das Claude Code-Fenster
2. Kopieren Sie ein Bild und fügen Sie es in die CLI mit Strg+V ein (verwenden Sie nicht Cmd+V)
3. Geben Sie Claude einen Bildpfad an. Z. B. „Analyze this image: /path/to/your/image.png”

Step 2: Bitten Sie Claude, das Bild zu analysieren

What does this image show?

Describe the UI elements in this screenshot

Are there any problematic elements in this diagram?

Step 3: Verwenden Sie Bilder für Kontext

Here's a screenshot of the error. What's causing it?

This is our current database schema. How should we modify it for the new feature?

Step 4: Erhalten Sie Code-Vorschläge aus visuellem Inhalt

Generate CSS to match this design mockup

What HTML structure would recreate this component?

Tip:

Tipps:

- Verwenden Sie Bilder, wenn Textbeschreibungen unklar oder umständlich wären
- Fügen Sie Screenshots von Fehlern, UI-Designs oder Diagrammen für besseren Kontext ein
- Sie können mehrere Bilder in einem Gespräch verwenden
- Die Bildanalyse funktioniert mit Diagrammen, Screenshots, Mockups und mehr
- Wenn Claude auf Bilder verweist (z. B. `[Image #1]`), `Cmd+Click` (Mac) oder `Ctrl+Click` (Windows/Linux) den Link, um das Bild in Ihrem Standard-Viewer zu öffnen

Dateien und Verzeichnisse referenzieren

Verwenden Sie `@`, um schnell Dateien oder Verzeichnisse einzubeziehen, ohne auf Claude zu warten, um sie zu lesen.

Step 1: Referenzieren Sie eine einzelne Datei

Explain the logic in `@src/utils/auth.js`

Dies fügt den vollständigen Inhalt der Datei in das Gespräch ein.

Step 2: Referenzieren Sie ein Verzeichnis

What's the structure of `@src/components?`

Dies bietet eine Verzeichnisauflistung mit Dateiinformationen.

Step 3: Referenzieren Sie MCP-Ressourcen

Show me the data from `@github:repos/owner/repo/issues`

Dies ruft Daten von verbundenen MCP-Servern im Format `@server:resource` ab. Weitere Details finden Sie unter [MCP-Ressourcen](#).

Tip:

Tipps:

- Dateipfade können relativ oder absolut sein
- @ Dateireferenzen fügen **CLAUDE.md** im Verzeichnis der Datei und übergeordneten Verzeichnissen zum Kontext hinzu
- Verzeichnisreferenzen zeigen Dateiaufstellungen, keine Inhalte
- Sie können mehrere Dateien in einer einzelnen Nachricht referenzieren (z. B. „@file1.js and @file2.js“)

Erweitertes Denken verwenden (Thinking Mode)

[Erweitertes Denken](#) ist standardmäßig aktiviert und gibt Claude Raum, um komplexe Probleme Schritt für Schritt zu durchdenken, bevor er antwortet. Dieses Denken ist im ausführlichen Modus sichtbar, den Sie mit **Ctrl+0** umschalten können.

Darüber hinaus führt Opus 4.6 adaptives Denken ein: Anstelle eines festen Thinking-Token-Budgets weist das Modell Thinking dynamisch basierend auf Ihrer [Effort-Level](#)-Einstellung zu. Erweitertes Denken und adaptives Denken arbeiten zusammen, um Ihnen Kontrolle darüber zu geben, wie tief Claude denkt, bevor er antwortet.

Erweitertes Denken ist besonders wertvoll für komplexe architektonische Entscheidungen, schwierige Fehler, Multi-Step-Implementierungsplanung und Bewertung von Kompromissen zwischen verschiedenen Ansätzen.

Note:

Phrasen wie „think“, „think hard“ und „think more“ werden als reguläre Prompt-Anweisungen interpretiert und weisen keine Thinking-Tokens zu.

Konfigurieren Sie Thinking Mode

Thinking ist standardmäßig aktiviert, aber Sie können es anpassen oder deaktivieren.

Bereich	Wie man konfiguriert	Details
Effort Level	Passen Sie in <code>/model</code> an oder setzen Sie <code>CLAUDE_CODE_EFFORT_LEVEL</code>	Steuern Sie die Thinking-Tiefe für Opus 4.6 und Sonnet 4.6: niedrig, mittel, hoch. Siehe Effort Level anpassen
ultrathink Schlüsselwort	Fügen Sie „ultrathink“ irgendwo in Ihrem Prompt ein	Setzt Effort auf hoch für diesen Turn auf Opus 4.6 und Sonnet 4.6. Nützlich für einmalige Aufgaben, die tiefes Denken erfordern, ohne Ihre Effort-Einstellung dauerhaft zu ändern
Toggle-Verknüpfung	Drücken Sie <code>Option+T</code> (macOS) oder <code>Alt+T</code> (Windows/Linux)	Schalten Sie Thinking für die aktuelle Sitzung ein/aus (alle Modelle). Kann Terminal-Konfiguration erfordern, um Option-Tastaturkürzel zu aktivieren
Globaler Standard	Verwenden Sie <code>/config</code> , um Thinking Mode umzuschalten	Setzt Ihren Standard über alle Projekte (alle Modelle). Gespeichert als <code>alwaysThinkingEnabled</code> in <code>~/ .claude/settings.json</code>
Token-Budget begrenzen	Setzen Sie die Umgebungsvariable <code>MAX_THINKING_TOKENS</code>	Begrenzen Sie das Thinking-Budget auf eine bestimmte Anzahl von Tokens (ignoriert auf Opus 4.6, es sei denn, es ist auf 0 gesetzt). Beispiel: <code>export MAX_THINKING_TOKENS=10000</code>

Um Claudes Thinking-Prozess anzuzeigen, drücken Sie `Ctrl+0`, um den ausführlichen Modus umzuschalten und die interne Begründung als grauer kursiver Text angezeigt zu sehen.

Wie erweitertes Denken funktioniert

Erweitertes Denken steuert, wie viel interne Begründung Claude vor der Antwort durchführt. Mehr Denken bietet mehr Raum, um Lösungen zu erkunden, Grenzfälle zu analysieren und Fehler selbst zu korrigieren.

Mit Opus 4.6 verwendet Thinking adaptives Denken: Das Modell weist Thinking-Tokens dynamisch basierend auf dem [Effort Level](#) zu, den Sie auswählen (niedrig, mittel, hoch). Dies ist die empfohlene Methode, um den Kompromiss zwischen Geschwindigkeit und Reasoning-Tiefe zu optimieren.

Mit anderen Modellen verwendet Thinking ein festes Budget von bis zu 31.999 Tokens aus Ihrem Output-Budget. Sie können dies mit der Umgebungsvariable `MAX_THINKING_TOKENS` begrenzen oder Thinking vollständig über `/config` oder den Toggle `Option+T` / `Alt+T` deaktivieren.

`MAX_THINKING_TOKENS` wird auf Opus 4.6 und Sonnet 4.6 ignoriert, da adaptives Denken stattdessen die Thinking-Tiefe steuert. Die eine Ausnahme: Das Setzen von `MAX_THINKING_TOKENS=0` deaktiviert Thinking immer noch vollständig auf jedem Modell. Um adaptives Thinking zu deaktivieren und zum festen Thinking-Budget zurückzukehren, setzen Sie `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`. Siehe [Umgebungsvariablen](#).

Warning:

Ihnen werden alle verwendeten Thinking-Tokens berechnet, auch wenn Claude 4-Modelle zusammengefasstes Thinking anzeigen

Vorherige Gespräche fortsetzen

Wenn Sie Claude Code starten, können Sie eine vorherige Sitzung fortsetzen:

- `claude --continue` setzt das neueste Gespräch im aktuellen Verzeichnis fort
- `claude --resume` öffnet eine Gesprächsauswahl oder setzt nach Name fort
- `claude --from-pr 123` setzt Sitzungen fort, die mit einem bestimmten Pull Request verknüpft sind

Verwenden Sie innerhalb einer aktiven Sitzung `/resume`, um zu einem anderen Gespräch zu wechseln.

Sitzungen werden pro Projektverzeichnis gespeichert. Die `/resume`-Auswahl zeigt Sitzungen aus demselben Git-Repository, einschließlich worktrees.

Benennen Sie Ihre Sitzungen

Geben Sie Sitzungen beschreibende Namen, um sie später zu finden. Dies ist eine Best Practice, wenn Sie an mehreren Aufgaben oder Funktionen arbeiten.

Step 1: Benennen Sie die aktuelle Sitzung

Verwenden Sie `/rename` während einer Sitzung, um ihr einen einprägsamen Namen zu geben:

```
/rename auth-refactor
```

Sie können jede Sitzung auch aus der Auswahl umbenennen: Führen Sie `/resume` aus, navigieren Sie zu einer Sitzung und drücken Sie `R`.

Step 2: Später nach Name fortsetzen

Aus der Befehlszeile:

```
claude --resume auth-refactor
```

Oder innerhalb einer aktiven Sitzung:

```
/resume auth-refactor
```

Verwenden Sie die Sitzungsauswahl

Der Befehl `/resume` (oder `claude --resume` ohne Argumente) öffnet eine interaktive Sitzungsauswahl mit diesen Funktionen:

Tastaturkürzel in der Auswahl:

Tastaturkürzel	Aktion
<code>↑ / ↓</code>	Zwischen Sitzungen navigieren
<code>→ / ←</code>	Gruppierte Sitzungen erweitern oder reduzieren
<code>Enter</code>	Wählen Sie die hervorgehobene Sitzung aus und setzen Sie sie fort
<code>P</code>	Vorschau des Sitzungsinhalts
<code>R</code>	Benennen Sie die hervorgehobene Sitzung um

Tastaturkürzel	Aktion
<code>/</code>	Suchen Sie, um Sitzungen zu filtern
<code>A</code>	Zwischen aktuellem Verzeichnis und allen Projekten umschalten
<code>B</code>	Filtern Sie auf Sitzungen aus Ihrem aktuellen Git-Branch
<code>Esc</code>	Beenden Sie die Auswahl oder den Suchmodus

Sitzungsorganisation:

Die Auswahl zeigt Sitzungen mit hilfreichen Metadaten:

- Sitzungsname oder anfänglicher Prompt
- Verstrichene Zeit seit letzter Aktivität
- Nachrichtenanzahl
- Git-Branch (falls zutreffend)

Verzweigte Sitzungen (erstellt mit `/rewind` oder `--fork-session`) werden unter ihrer Root-Sitzung gruppiert, was es einfacher macht, verwandte Gespräche zu finden.

Tip:

Tipps:

- **Benennen Sie Sitzungen früh:** Verwenden Sie `/rename`, wenn Sie mit einer bestimmten Aufgabe beginnen – es ist viel einfacher, „payment-integration“ später zu finden als „explain this function“
- Verwenden Sie `--continue` für schnellen Zugriff auf Ihr letztes Gespräch im aktuellen Verzeichnis
- Verwenden Sie `--resume session-name`, wenn Sie wissen, welche Sitzung Sie benötigen
- Verwenden Sie `--resume` (ohne Namen), wenn Sie durchsuchen und auswählen müssen
- Verwenden Sie für Skripte `claude --continue --print "prompt"`, um im nicht-interaktiven Modus fortzufahren
- Drücken Sie `P` in der Auswahl, um eine Sitzung vor dem Fortsetzen in der Vorschau anzuzeigen
- Die fortgesetzte Konversation beginnt mit demselben Modell und der gleichen Konfiguration wie das Original

Wie es funktioniert:

1. **Gesprächsspeicherung:** Alle Gespräche werden automatisch lokal mit ihrer vollständigen Nachrichtenhistorie gespeichert

2. **Nachricht-Deserialisierung:** Beim Fortsetzen wird die gesamte Nachrichtenhistorie wiederhergestellt, um den Kontext zu bewahren
3. **Tool-Status:** Die Tool-Nutzung und Ergebnisse aus dem vorherigen Gespräch werden beibehalten
4. **Kontextwiederherstellung:** Das Gespräch wird mit allen vorherigen Kontexten intakt fortgesetzt

Führen Sie parallele Claude Code-Sitzungen mit Git worktrees aus

Wenn Sie an mehreren Aufgaben gleichzeitig arbeiten, benötigt jede Claude-Sitzung ihre eigene Kopie der Codebase, damit Änderungen nicht kollidieren. Git worktrees lösen dies, indem sie separate Arbeitsverzeichnisse erstellen, die jeweils ihre eigenen Dateien und Branches haben, während sie die gleiche Repository-Historie und Remote-Verbindungen teilen. Dies bedeutet, dass Sie Claude an einer Funktion in einem worktree arbeiten lassen können, während Sie einen Fehler in einem anderen beheben, ohne dass eine Sitzung die andere beeinträchtigt.

Verwenden Sie das Flag `--worktree` (`-w`), um einen isolierten worktree zu erstellen und Claude darin zu starten. Der Wert, den Sie übergeben, wird zum worktree-Verzeichnisnamen und Branch-Namen:

```
## Starten Sie Claude in einem worktree namens „feature-auth“
## Erstellt .claude/worktrees/feature-auth/ mit einem neuen Branch
claude --worktree feature-auth

## Starten Sie eine weitere Sitzung in einem separaten worktree
claude --worktree bugfix-123
```

Wenn Sie den Namen weglassen, generiert Claude automatisch einen zufälligen:

```
## Generiert automatisch einen Namen wie „bright-running-fox“
claude --worktree
```

Worktrees werden unter `<repo>/.claude/worktrees/<name>` erstellt und verzweigen sich vom Standard-Remote-Branch. Der worktree-Branch wird `worktree-<name>` genannt.

Sie können Claude auch während einer Sitzung bitten, „in einem worktree zu arbeiten“ oder „einen worktree zu starten“, und er erstellt automatisch einen.

subagent worktrees

subagents können auch Worktree-Isolation verwenden, um parallel ohne Konflikte zu arbeiten. Bitten Sie Claude, „worktrees für Ihre Agents zu verwenden“ oder konfigurieren Sie es in einem [benutzerdefinierten subagent](#), indem Sie `isolation: worktree` zum Frontmatter des Agents hinzufügen. Jeder subagent erhält seinen eigenen worktree, der automatisch bereinigt wird, wenn der subagent ohne Änderungen beendet wird.

Worktree-Bereinigung

Wenn Sie eine worktree-Sitzung beenden, handhabt Claude die Bereinigung basierend darauf, ob Sie Änderungen vorgenommen haben:

- **Keine Änderungen:** Der worktree und sein Branch werden automatisch entfernt
- **Änderungen oder Commits vorhanden:** Claude fordert Sie auf, den worktree zu behalten oder zu entfernen. Das Behalten bewahrt das Verzeichnis und den Branch, damit Sie später zurückkehren können. Das Entfernen löscht das worktree-Verzeichnis und seinen Branch und verwirft alle nicht committeten Änderungen und Commits

Um worktrees außerhalb einer Claude-Sitzung zu bereinigen, verwenden Sie [manuelle Worktree-Verwaltung](#).

Tip:

Fügen Sie `.claude/worktrees/` zu Ihrer `.gitignore` hinzu, um zu verhindern, dass worktree-Inhalte als nicht verfolgte Dateien in Ihrem Haupt-Repository angezeigt werden.

Verwalten Sie worktrees manuell

Für mehr Kontrolle über den worktree-Speicherort und die Branch-Konfiguration erstellen Sie worktrees direkt mit Git. Dies ist nützlich, wenn Sie einen bestimmten vorhandenen Branch auschecken oder den worktree außerhalb des Repositorys platzieren müssen.

```
## Erstellen Sie einen worktree mit einem neuen Branch
git worktree add ../project-feature-a -b feature-a

## Erstellen Sie einen worktree mit einem vorhandenen Branch
git worktree add ../project-bugfix bugfix-123

## Starten Sie Claude im worktree
cd ../project-feature-a && claude

## Bereinigen Sie, wenn Sie fertig sind
git worktree list
git worktree remove ../project-feature-a
```

Weitere Informationen finden Sie in der [offiziellen Git worktree-Dokumentation](#).

Tip:

Denken Sie daran, Ihre Entwicklungsumgebung in jedem neuen worktree gemäß dem Setup Ihres Projekts zu initialisieren. Je nach Ihrem Stack kann dies die Ausführung der Abhängigkeitsinstallation (`npm install` , `yarn`), das Einrichten virtueller Umgebungen oder das Befolgen des Standard-Setup-Prozesses Ihres Projekts umfassen.

Nicht-Git-Versionskontrolle

Worktree-Isolation funktioniert standardmäßig mit Git. Für andere Versionskontrollsysteme wie SVN, Perforce oder Mercurial konfigurieren Sie [WorktreeCreate](#) und [WorktreeRemove hooks](#), um benutzerdefinierte Worktree-Erstellungs- und Bereinigungslogik bereitzustellen. Wenn konfiguriert, ersetzen diese Hooks das Standard-Git-Verhalten, wenn Sie `--worktree` verwenden.

Für automatisierte Koordination paralleler Sitzungen mit gemeinsamen Aufgaben und Messaging siehe [Agent Teams](#).

Erhalten Sie eine Benachrichtigung, wenn Claude Ihre Aufmerksamkeit benötigt

Wenn Sie eine lange laufende Aufgabe starten und zu einem anderen Fenster wechseln, können Sie Desktop-Benachrichtigungen einrichten, damit Sie wissen, wenn Claude fertig ist oder Ihre Eingabe benötigt. Dies verwendet das **Notification Hook-Ereignis**, das immer dann ausgelöst wird, wenn Claude auf Berechtigung wartet, untätig ist und bereit für einen neuen Prompt ist, oder die Authentifizierung abgeschlossen ist.

Step 1: Öffnen Sie das Hooks-Menü

Geben Sie `/hooks` ein und wählen Sie **Notification** aus der Liste der Ereignisse.

Step 2: Konfigurieren Sie den Matcher

Wählen Sie **+ Match all (no filter)**, um bei allen Benachrichtigungstypen auszulösen. Um nur für bestimmte Ereignisse benachrichtigt zu werden, wählen Sie **+ Add new matcher...** und geben Sie einen dieser Werte ein:

Matcher	Wird ausgelöst, wenn
<code>permission_prompt</code>	Claude benötigt Ihre Genehmigung für eine Tool-Nutzung
<code>idle_prompt</code>	Claude ist fertig und wartet auf Ihren nächsten Prompt
<code>auth_success</code>	Die Authentifizierung ist abgeschlossen
<code>elicitation_dialog</code>	Claude stellt Ihnen eine Frage

Step 3: Fügen Sie Ihren Benachrichtigungsbefehl hinzu

Wählen Sie **+ Add new hook...** und geben Sie den Befehl für Ihr Betriebssystem ein:

macOS

Verwendet `osascript`, um eine native macOS-Benachrichtigung über AppleScript auszulösen:

```
osascript -e 'display notification "Claude Code needs your attention" with title "Claude Code"'
```

Linux

Verwendet `notify-send`, das auf den meisten Linux-Desktops mit einem Benachrichtigungsdaemon vorinstalliert ist:

```
notify-send 'Claude Code' 'Claude Code needs your attention'
```

Windows (PowerShell)

Verwendet PowerShell, um ein natives Nachrichtenfeld über .NET's Windows Forms anzuzeigen:

```
powershell.exe -Command  
"[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms');  
[System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention',  
'Claude Code')"
```

Step 4: In Benutzereinstellungen speichern

Wählen Sie **User settings**, um die Benachrichtigung auf alle Ihre Projekte anzuwenden.

Für die vollständige Anleitung mit JSON-Konfigurationsbeispielen siehe [Automatisieren Sie Arbeitsabläufe mit Hooks](#). Für das vollständige Ereignisschema und die Benachrichtigungstypen siehe die [Benachrichtigungsreferenz](#).

Verwenden Sie Claude als Unix-ähnliches Dienstprogramm

Fügen Sie Claude zu Ihrem Überprüfungsprozess hinzu

Angenommen, Sie möchten Claude Code als Linter oder Code-Reviewer verwenden.

Fügen Sie Claude zu Ihrem Build-Skript hinzu:

```
// package.json
{
  ...
  "scripts": {
    ...
    "lint:claude": "claude -p 'you are a linter. please look at the changes
vs. main and report any issues related to typos. report the filename and line
number on one line, and a description of the issue on the second line. do not
return any other text.'"
  }
}
```

Tip:

Tipps:

- Verwenden Sie Claude für automatisierte Code-Überprüfung in Ihrer CI/CD-Pipeline
- Passen Sie den Prompt an, um auf spezifische Probleme zu prüfen, die für Ihr Projekt relevant sind
- Erwägen Sie, mehrere Skripte für verschiedene Arten von Überprüfungen zu erstellen

Pipe in, Pipe out

Angenommen, Sie möchten Daten in Claude pipen und Daten in einem strukturierten Format zurückbekommen.

Pipen Sie Daten durch Claude:

```
cat build-error.txt | claude -p 'concisely explain the root cause of this build
error' > output.txt
```

Tip:

Tipps:

- Verwenden Sie Pipes, um Claude in vorhandene Shell-Skripte zu integrieren
- Kombinieren Sie mit anderen Unix-Tools für leistungsstarke Arbeitsabläufe
- Erwägen Sie die Verwendung von `–output-format` für strukturierte Ausgabe

Steuern Sie das Ausgabeformat

Angenommen, Sie benötigen Claudes Ausgabe in einem bestimmten Format, besonders wenn Sie Claude Code in Skripte oder andere Tools integrieren.

Step 1: Verwenden Sie Textformat (Standard)

```
cat data.txt | claude -p 'summarize this data' --output-format text > summary.txt
```

Dies gibt nur Claudes einfache Textantwort aus (Standardverhalten).

Step 2: Verwenden Sie JSON-Format

```
cat code.py | claude -p 'analyze this code for bugs' --output-format json > analysis.json
```

Dies gibt ein JSON-Array von Nachrichten mit Metadaten einschließlich Kosten und Dauer aus.

Step 3: Verwenden Sie Streaming-JSON-Format

```
cat log.txt | claude -p 'parse this log file for errors' --output-format stream-json
```

Dies gibt eine Reihe von JSON-Objekten in Echtzeit aus, während Claude die Anfrage verarbeitet. Jede Nachricht ist ein gültiges JSON-Objekt, aber die gesamte Ausgabe ist kein gültiges JSON, wenn es verkettet wird.

Tip:

Tipps:

- Verwenden Sie `--output-format text` für einfache Integrationen, bei denen Sie nur Claudes Antwort benötigen
- Verwenden Sie `--output-format json`, wenn Sie das vollständige Gesprächsprotokoll benötigen
- Verwenden Sie `--output-format stream-json` für Echtzeit-Ausgabe jedes Gesprächs-
turn

Fragen Sie Claude nach seinen Fähigkeiten

Claude hat integrierten Zugriff auf seine Dokumentation und kann Fragen zu seinen eigenen Funktionen und Einschränkungen beantworten.

Beispielfragen

can Claude Code create pull requests?

how does Claude Code handle permissions?

what skills are available?

how do I use MCP with Claude Code?

how do I configure Claude Code for Amazon Bedrock?

what are the limitations of Claude Code?

Note:

Claude bietet dokumentationsgestützte Antworten auf diese Fragen. Für ausführbare Beispiele und praktische Demonstrationen lesen Sie die spezifischen Workflow-Abschnitte oben.

Tip:

Tipps:

- Claude hat immer Zugriff auf die neueste Claude Code-Dokumentation, unabhängig von der Version, die Sie verwenden
- Stellen Sie spezifische Fragen, um detaillierte Antworten zu erhalten
- Claude kann komplexe Funktionen wie MCP-Integration, Enterprise-Konfigurationen und erweiterte Arbeitsabläufe erklären

Nächste Schritte

- [Best Practices](#) Muster, um das Beste aus Claude Code herauszuholen
- [Wie Claude Code funktioniert](#) Verstehen Sie die agentic Loop und Kontextverwaltung
- [Erweitern Sie Claude Code](#) Fügen Sie skills, hooks, MCP, subagents und plugins hinzu
- [Referenzimplementierung](#) Klonen Sie unsere Referenzimplementierung des Development Containers

Part 3: Commands & Reference

CLI-Referenz

Vollständige Referenz für die Claude Code Befehlszeilenschnittstelle, einschließlich Befehle und Flags.

CLI-Befehle

Sie können Sitzungen starten, Inhalte weiterleiten, Gespräche fortsetzen und Updates verwalten mit diesen Befehlen:

Befehl	Beschreibung	Beispiel
<code>claude</code>	Interaktive Sitzung starten	<code>claude</code>
<code>claude "query"</code>	Interaktive Sitzung mit initialem Prompt starten	<code>claude "explain this project"</code>
<code>claude -p "query"</code>	Abfrage über SDK, dann beenden	<code>claude -p "explain this function"</code>
<code>cat file claude -p "query"</code>	Weitergeleitete Inhalte verarbeiten	<code>cat logs.txt claude -p "explain"</code>
<code>claude -c</code>	Letzte Konversation im aktuellen Verzeichnis fortsetzen	<code>claude -c</code>
<code>claude -c -p "query"</code>	Über SDK fortsetzen	<code>claude -c -p "Check for type errors"</code>
<code>claude -r "<session>" "query"</code>	Sitzung nach ID oder Name fortsetzen	<code>claude -r "auth-refactor" "Finish this PR"</code>
<code>claude update</code>	Auf neueste Version aktualisieren	<code>claude update</code>

Befehl	Beschreibung	Beispiel
<code>claude auth login</code>	Melden Sie sich bei Ihrem Anthropic-Konto an. Verwenden Sie <code>--email</code> , um Ihre E-Mail-Adresse vorauszufüllen, und <code>--sso</code> , um SSO-Authentifizierung zu erzwingen	<code>claude auth login --email user@example.com --sso</code>
<code>claude auth logout</code>	Abmelden von Ihrem Anthropic-Konto	<code>claude auth logout</code>
<code>claude auth status</code>	Authentifizierungsstatus als JSON anzeigen. Verwenden Sie <code>--text</code> für benutzerfreundliche Ausgabe. Beendet mit Code 0, wenn angemeldet, 1, wenn nicht	<code>claude auth status</code>
<code>claude agents</code>	Alle konfigurierten subagents auflisten, gruppiert nach Quelle	<code>claude agents</code>
<code>claude mcp</code>	Model Context Protocol (MCP) Server konfigurieren	Siehe die Claude Code MCP-Dokumentation .
<code>claude remote-control</code>	Starten Sie eine Remote Control-Sitzung , um Claude Code von Claude.ai oder der Claude-App aus zu steuern, während sie lokal ausgeführt wird. Siehe Remote Control für Flags	<code>claude remote-control</code>

CLI-Flags

Passen Sie das Verhalten von Claude Code mit diesen Befehlszeilenflags an:

Flag	Beschreibung	Beispiel
<code>--add-dir</code>	Zusätzliche Arbeitsverzeichnisse hinzufügen, auf die Claude zugreifen kann (validiert, dass jeder Pfad als Verzeichnis vorhanden ist)	<code>claude --add-dir ../apps ../lib</code>
<code>--agent</code>	Geben Sie einen Agent für die aktuelle Sitzung an (überschreibt die <code>agent</code> -Einstellung)	<code>claude --agent my-custom-agent</code>

Flag	Beschreibung	Beispiel
<code>--agents</code>	Definieren Sie benutzerdefinierte subagents dynamisch über JSON (siehe unten für das Format)	<pre>claude --agents '{"reviewer": {"description":"Reviews code", "prompt":"You are a code reviewer"}}'</pre>
<code>--allow-dangerously-skip-permissions</code>	Aktivieren Sie das Umgehen von Berechtigungen als Option, ohne es sofort zu aktivieren. Ermöglicht die Zusammensetzung mit <code>--permission-mode</code> (mit Vorsicht verwenden)	<pre>claude --permission- mode plan --allow- dangerously-skip- permissions</pre>
<code>--allowedTools</code>	Tools, die ohne Aufforderung zur Berechtigung ausgeführt werden. Siehe Syntax der Berechtigungsregel für Musterabgleich. Um einzuschränken, welche Tools verfügbar sind, verwenden Sie stattdessen <code>--tools</code>	<pre>"Bash(git log *)" "Bash(git diff *)" "Read"</pre>
<code>--append-system-prompt</code>	Benutzerdefinierten Text am Ende des Standard-Systemprompts anhängen	<pre>claude --append- system-prompt "Always use TypeScript"</pre>
<code>--append-system-prompt-file</code>	Zusätzlichen Systemprompt-Text aus einer Datei laden und an den Standard-Prompt anhängen	<pre>claude --append- system-prompt-file ./ extra-rules.txt</pre>
<code>--betas</code>	Beta-Header, die in API-Anfragen einbezogen werden sollen (nur API-Schlüsselbenutzer)	<pre>claude --betas interleaved-thinking</pre>
<code>--chrome</code>	Aktivieren Sie Chrome-Browserintegration für Web-Automatisierung und Tests	<pre>claude --chrome</pre>
<code>--continue</code> , <code>-c</code>	Laden Sie die letzte Konversation im aktuellen Verzeichnis	<pre>claude --continue</pre>

Flag	Beschreibung	Beispiel
<code>--dangerously-skip-permissions</code>	Alle Berechtigungsaufforderungen überspringen (mit Vorsicht verwenden)	<code>claude --dangerously-skip-permissions</code>
<code>--debug</code>	Debug-Modus mit optionaler Kategoriefilterung aktivieren (z. B. <code>"api,hooks"</code> oder <code>"!statsig,!file"</code>)	<code>claude --debug "api,mcp"</code>
<code>--disable-slash-commands</code>	Alle skills und Befehle für diese Sitzung deaktivieren	<code>claude --disable-slash-commands</code>
<code>--disallowedTools</code>	Tools, die aus dem Kontext des Modells entfernt werden und nicht verwendet werden können	<code>"Bash(git log *)"</code> <code>"Bash(git diff *)"</code> <code>"Edit"</code>
<code>--fallback-model</code>	Automatisches Fallback auf das angegebene Modell aktivieren, wenn das Standardmodell überlastet ist (nur Druckmodus)	<code>claude -p --fallback-model sonnet "query"</code>
<code>--fork-session</code>	Beim Fortsetzen eine neue Sitzungs-ID erstellen, anstatt die ursprüngliche wiederzuverwenden (mit <code>--resume</code> oder <code>--continue</code> verwenden)	<code>claude --resume abc123 --fork-session</code>
<code>--from-pr</code>	Sitzungen fortsetzen, die mit einem bestimmten GitHub PR verknüpft sind. Akzeptiert eine PR-Nummer oder URL. Sitzungen werden automatisch verknüpft, wenn sie über <code>gh pr create</code> erstellt werden	<code>claude --from-pr 123</code>
<code>--ide</code>	Automatisch mit IDE beim Start verbinden, wenn genau eine gültige IDE verfügbar ist	<code>claude --ide</code>
<code>--init</code>	Initialisierungs-Hooks ausführen und interaktiven Modus starten	<code>claude --init</code>

Flag	Beschreibung	Beispiel
<code>--init-only</code>	Initialisierungs-Hooks ausführen und beenden (keine interaktive Sitzung)	<code>claude --init-only</code>
<code>--include-partial-messages</code>	Partielle Streaming-Ereignisse in die Ausgabe einbeziehen (erfordert <code>--print</code> und <code>--output-format=stream-json</code>)	<code>claude -p --output-format stream-json --include-partial-messages "query"</code>
<code>--input-format</code>	Eingabeformat für Druckmodus angeben (Optionen: <code>text</code> , <code>stream-json</code>)	<code>claude -p --output-format json --input-format stream-json</code>
<code>--json-schema</code>	Validierte JSON-Ausgabe abrufen, die einem JSON-Schema entspricht, nachdem der Agent seinen Workflow abgeschlossen hat (nur Druckmodus, siehe strukturierte Ausgaben)	<code>claude -p --json-schema '{"type":"object","properties":{...}}' "query"</code>
<code>--maintenance</code>	Wartungs-Hooks ausführen und beenden	<code>claude --maintenance</code>
<code>--max-budget-usd</code>	Maximaler Dollarbetrag, der für API-Aufrufe ausgegeben werden kann, bevor gestoppt wird (nur Druckmodus)	<code>claude -p --max-budget-usd 5.00 "query"</code>
<code>--max-turns</code>	Begrenzen Sie die Anzahl der agentischen Wendungen (nur Druckmodus). Beendet mit einem Fehler, wenn das Limit erreicht wird. Standardmäßig kein Limit	<code>claude -p --max-turns 3 "query"</code>
<code>--mcp-config</code>	Laden Sie MCP-Server aus JSON-Dateien oder Strings (durch Leerzeichen getrennt)	<code>claude --mcp-config ./mcp.json</code>
<code>--model</code>	Legt das Modell für die aktuelle Sitzung mit einem Alias für das neueste Modell (<code>sonnet</code> oder <code>opus</code>) oder den vollständigen Namen eines Modells fest	<code>claude --model claude-sonnet-4-6</code>

Flag	Beschreibung	Beispiel
<code>--no-chrome</code>	Deaktivieren Sie Chrome-Browserintegration für diese Sitzung	<code>claude --no-chrome</code>
<code>--no-session-persistence</code>	Deaktivieren Sie die Sitzungspersistenz, sodass Sitzungen nicht auf der Festplatte gespeichert werden und nicht fortgesetzt werden können (nur Druckmodus)	<code>claude -p --no-session-persistence "query"</code>
<code>--output-format</code>	Ausgabeformat für Druckmodus angeben (Optionen: <code>text</code> , <code>json</code> , <code>stream-json</code>)	<code>claude -p "query" --output-format json</code>
<code>--permission-mode</code>	Beginnen Sie in einem angegebenen Berechtigungsmodus	<code>claude --permission-mode plan</code>
<code>--permission-prompt-tool</code>	Geben Sie ein MCP-Tool an, um Berechtigungsaufforderungen im nicht-interaktiven Modus zu verarbeiten	<code>claude -p --permission-prompt-tool mcp_auth_tool "query"</code>
<code>--plugin-dir</code>	Laden Sie Plugins aus Verzeichnissen nur für diese Sitzung (wiederholbar)	<code>claude --plugin-dir ./my-plugins</code>
<code>--print</code> , <code>-p</code>	Antwort ohne interaktiven Modus drucken (siehe Agent SDK-Dokumentation für Details zur programm-gesteuerten Verwendung)	<code>claude -p "query"</code>
<code>--remote</code>	Erstellen Sie eine neue Web-Sitzung auf claude.ai mit der bereitgestellten Aufgabenbeschreibung	<code>claude --remote "Fix the login bug"</code>
<code>--resume</code> , <code>-r</code>	Setzen Sie eine bestimmte Sitzung nach ID oder Name fort, oder zeigen Sie eine interaktive Auswahl an, um eine Sitzung auszuwählen	<code>claude --resume auth-refactor</code>
<code>--session-id</code>	Verwenden Sie eine bestimmte Sitzungs-ID für die Konversation (muss eine gültige UUID sein)	<code>claude --session-id "550e8400-e29b-41d4-a716-446655440000"</code>

Flag	Beschreibung	Beispiel
<code>--setting-sources</code>	Durch Kommas getrennte Liste von Einstellungsquellen zum Laden (<code>user</code> , <code>project</code> , <code>local</code>)	<code>claude --setting-sources user,project</code>
<code>--settings</code>	Pfad zu einer Einstellungs-JSON-Datei oder eine JSON-Zeichenkette zum Laden zusätzlicher Einstellungen	<code>claude --settings ./settings.json</code>
<code>--strict-mcp-config</code>	Verwenden Sie nur MCP-Server aus <code>--mcp-config</code> , ignorieren Sie alle anderen MCP-Konfigurationen	<code>claude --strict-mcp-config --mcp-config ./mcp.json</code>
<code>--system-prompt</code>	Ersetzen Sie den gesamten Systemprompt durch benutzerdefinierten Text	<code>claude --system-prompt "You are a Python expert"</code>
<code>--system-prompt-file</code>	Laden Sie den Systemprompt aus einer Datei, um den Standard-Prompt zu ersetzen	<code>claude --system-prompt-file ./custom-prompt.txt</code>
<code>--teleport</code>	Setzen Sie eine Web-Sitzung in Ihrem lokalen Terminal fort	<code>claude --teleport</code>
<code>--teammate-mode</code>	Legen Sie fest, wie Agent-Team -Teamkollegen angezeigt werden: <code>auto</code> (Standard), <code>in-process</code> oder <code>tmux</code> . Siehe Agent-Teams einrichten	<code>claude --teammate-mode in-process</code>
<code>--tools</code>	Beschränken Sie, welche integrierten Tools Claude verwenden kann. Verwenden Sie <code>" "</code> , um alle zu deaktivieren, <code>"default"</code> für alle oder Tool-Namen wie <code>"Bash,Edit,Read"</code>	<code>claude --tools "Bash,Edit,Read"</code>
<code>--verbose</code>	Ausführliches Logging aktivieren, zeigt vollständige Ausgabe für jeden Wendung	<code>claude --verbose</code>
<code>--version</code> , <code>-v</code>	Versionsnummer ausgeben	<code>claude -v</code>

Flag	Beschreibung	Beispiel
<code>--worktree</code> , <code>-w</code>	Starten Sie Claude in einem isolierten <code>git worktree</code> bei <code><repo>/<code>.claude/worktrees/</code><code><name></code></code> . Wenn kein Name angegeben wird, wird einer automatisch generiert	<code>claude -w feature-auth</code>

Tip:

Das Flag `--output-format json` ist besonders nützlich für Scripting und Automatisierung und ermöglicht es Ihnen, Claudes Antworten programmgesteuert zu analysieren.

Format des Agents-Flags

Das Flag `--agents` akzeptiert ein JSON-Objekt, das einen oder mehrere benutzerdefinierte Subagents definiert. Jeder Subagent erfordert einen eindeutigen Namen (als Schlüssel) und ein Definitionsobjekt mit den folgenden Feldern:

Feld	Erforderlich	Beschreibung
<code>description</code>	Ja	Natürlichsprachige Beschreibung, wann der Subagent aufgerufen werden sollte
<code>prompt</code>	Ja	Der Systemprompt, der das Verhalten des Subagents lenkt
<code>tools</code>	Nein	Array von spezifischen Tools, die der Subagent verwenden kann, z. B. <code>["Read", "Edit", "Bash"]</code> . Falls weggelassen, erbt alle Tools. Unterstützt <code>Agent(agent_type)</code> -Syntax
<code>disallowedTools</code>	Nein	Array von Tool-Namen, die für diesen Subagent explizit verweigert werden

Feld	Erforderlich	Beschreibung
<code>model</code>	Nein	Modell-Alias zur Verwendung: <code>sonnet</code> , <code>opus</code> , <code>haiku</code> oder <code>inherit</code> . Falls weggelassen, wird standardmäßig <code>inherit</code> verwendet
<code>skills</code>	Nein	Array von skill -Namen, die in den Kontext des Sub-agents vorgeladen werden sollen
<code>mcpServers</code>	Nein	Array von MCP-Servern für diesen Subagent. Jeder Eintrag ist eine Server-Namenszeichenkette oder ein <code>{name: config}</code> -Objekt
<code>maxTurns</code>	Nein	Maximale Anzahl von agentischen Wendungen, bevor der Subagent stoppt

Beispiel:

```
claude --agents '{
  "code-reviewer": {
    "description": "Expert code reviewer. Use proactively after code changes.",
    "prompt": "You are a senior code reviewer. Focus on code quality, security,
and best practices.",
    "tools": ["Read", "Grep", "Glob", "Bash"],
    "model": "sonnet"
  },
  "debugger": {
    "description": "Debugging specialist for errors and test failures.",
    "prompt": "You are an expert debugger. Analyze errors, identify root causes,
and provide fixes."
  }
}'
```

Weitere Details zum Erstellen und Verwenden von Subagents finden Sie in der [Subagents-Dokumentation](#).

Systemprompt-Flags

Claude Code bietet vier Flags zum Anpassen des Systemprompts. Alle vier funktionieren sowohl im interaktiven als auch im nicht-interaktiven Modus.

Flag	Verhalten	Anwendungsfall
<code>--system-prompt</code>	Ersetzt den gesamten Standard-Prompt	Vollständige Kontrolle über Claudes Verhalten und Anweisungen
<code>--system-prompt-file</code>	Ersetzt mit Dateiinhalten	Laden Sie Prompts aus Dateien für Reproduzierbarkeit und Versionskontrolle
<code>--append-system-prompt</code>	Hängt an Standard-Prompt an	Fügen Sie spezifische Anweisungen hinzu, während Sie das Standard-Claude Code-Verhalten beibehalten
<code>--append-system-prompt-file</code>	Hängt Dateiinhalte an Standard-Prompt an	Laden Sie zusätzliche Anweisungen aus Dateien, während Sie die Standardwerte beibehalten

Wann jedes verwendet werden sollte:

- `--system-prompt` : Verwenden Sie, wenn Sie vollständige Kontrolle über Claudes Systemprompt benötigen. Dies entfernt alle Standard-Claude Code-Anweisungen und gibt Ihnen eine leere Leinwand.

```
claude --system-prompt "You are a Python expert who only writes type-annotated code"
```

- `--system-prompt-file` : Verwenden Sie, wenn Sie einen benutzerdefinierten Prompt aus einer Datei laden möchten, nützlich für Team-Konsistenz oder versionskontrollierte Prompt-Vorlagen.

```
claude --system-prompt-file ./prompts/code-review.txt
```

- **--append-system-prompt** : Verwenden Sie, wenn Sie spezifische Anweisungen hinzufügen möchten, während Sie die Standard-Funktionen von Claude Code beibehalten. Dies ist die sicherste Option für die meisten Anwendungsfälle.

```
claude --append-system-prompt "Always use TypeScript and include JSDoc comments"
```

- **--append-system-prompt-file** : Verwenden Sie, wenn Sie Anweisungen aus einer Datei anhängen möchten, während Sie die Claude Code-Standardwerte beibehalten. Nützlich für versionskontrollierte Ergänzungen.

```
claude --append-system-prompt-file ./prompts/style-rules.txt
```

--system-prompt und **--system-prompt-file** schließen sich gegenseitig aus. Die Append-Flags können zusammen mit einem der Ersetzungs-Flags verwendet werden.

Für die meisten Anwendungsfälle wird **--append-system-prompt** oder **--append-system-prompt-file** empfohlen, da sie die integrierten Funktionen von Claude Code beibehalten und gleichzeitig Ihre benutzerdefinierten Anforderungen hinzufügen. Verwenden Sie **--system-prompt** oder **--system-prompt-file** nur, wenn Sie vollständige Kontrolle über den Systemprompt benötigen.

Siehe auch

- [Chrome-Erweiterung](#) - Browser-Automatisierung und Web-Tests
- [Interaktiver Modus](#) - Tastenkombinationen, Eingabemodi und interaktive Funktionen
- [Schnellstart-Anleitung](#) - Erste Schritte mit Claude Code
- [Häufige Workflows](#) - Erweiterte Workflows und Muster
- [Einstellungen](#) - Konfigurationsoptionen
- [Agent SDK-Dokumentation](#) - Programmgesteuerte Verwendung und Integrationen

Tastaturkürzel anpassen

Passen Sie Tastaturkürzel in Claude Code mit einer Keybindings-Konfigurationsdatei an.

Note:

Anpassbare Tastaturkürzel erfordern Claude Code v2.1.18 oder später. Überprüfen Sie Ihre Version mit `claude --version`.

Claude Code unterstützt anpassbare Tastaturkürzel. Führen Sie `/keybindings` aus, um Ihre Konfigurationsdatei unter `~/.claude/keybindings.json` zu erstellen oder zu öffnen.

Konfigurationsdatei

Die Keybindings-Konfigurationsdatei ist ein Objekt mit einem `bindings`-Array. Jeder Block gibt einen Kontext und eine Zuordnung von Tastenkombinationen zu Aktionen an.

Note:

Änderungen an der Keybindings-Datei werden automatisch erkannt und angewendet, ohne Claude Code neu zu starten.

Feld	Beschreibung
<code>\$schema</code>	Optionale JSON-Schema-URL für Editor-Autovervollständigung
<code>\$docs</code>	Optionale Dokumentations-URL
<code>bindings</code>	Array von Binding-Blöcken nach Kontext

Dieses Beispiel bindet `Ctrl+E` zum Öffnen eines externen Editors im Chat-Kontext und hebt die Bindung von `Ctrl+U` auf:

```
{
  "$schema": "https://www.schemastore.org/claude-code-keybindings.json",
  "$docs": "https://code.claude.com/docs/en/keybindings",
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+e": "chat:externalEditor",
        "ctrl+u": null
      }
    }
  ]
}
```

Kontexte

Jeder Binding-Block gibt einen **Kontext** an, in dem die Bindings gelten:

Kontext	Beschreibung
Global	Gilt überall in der App
Chat	Haupteingabebereich für Chat
Autocomplete	Autovervollständigungsmenü ist offen
Settings	Einstellungsmenü (nur Escape zum Schließen)
Confirmation	Berechtigungs- und Bestätigungsdialoge
Tabs	Tab-Navigationskomponenten
Help	Hilfemenü ist sichtbar
Transcript	Transkript-Viewer
HistorySearch	Verlaufssuchmodus (Ctrl+R)
Task	Hintergrundaufgabe wird ausgeführt
ThemePicker	Design-Picker-Dialog
Attachments	Navigationsleiste für Bilder/Anhänge

Kontext	Beschreibung
Footer	Fußzeilen-Indikator-Navigation (Aufgaben, Teams, Diff)
MessageSelector	Nachrichtenauswahl für Rewind- und Zusammenfassungsdialog
DiffDialog	Diff-Viewer-Navigation
ModelPicker	Modell-Picker-Aufwandsstufe
Select	Generische Select/List-Komponenten
Plugin	Plugin-Dialog (durchsuchen, entdecken, verwalten)

Verfügbare Aktionen

Aktionen folgen einem `namespace:action` -Format, wie `chat:submit` zum Senden einer Nachricht oder `app:toggleTodos` zum Anzeigen der Aufgabenliste. Jeder Kontext hat spezifische verfügbare Aktionen.

App-Aktionen

Aktionen verfügbar im `Global` -Kontext:

Aktion	Standard	Beschreibung
<code>app:interrupt</code>	Ctrl+C	Aktuelle Operation abbrechen
<code>app:exit</code>	Ctrl+D	Claude Code beenden
<code>app:toggleTodos</code>	Ctrl+T	Sichtbarkeit der Aufgabenliste umschalten
<code>app:toggleTranscript</code>	Ctrl+O	Ausführliches Transkript umschalten

Verlaufsaktionen

Aktionen zum Navigieren im Befehlsverlauf:

Aktion	Standard	Beschreibung
<code>history:search</code>	Ctrl+R	Verlaufssuche öffnen
<code>history:previous</code>	Oben	Vorheriges Verlaufselement

Aktion	Standard	Beschreibung
<code>history:next</code>	Unten	Nächstes Verlaufselement

Chat-Aktionen

Aktionen verfügbar im `Chat` -Kontext:

Aktion	Standard	Beschreibung
<code>chat:cancel</code>	Escape	Aktuelle Eingabe abbrechen
<code>chat:cycleMode</code>	Shift+Tab*	Berechtigungsmodi durchlaufen
<code>chat:modelPicker</code>	Cmd+P / Meta+P	Modell-Picker öffnen
<code>chat:thinkingToggle</code>	Cmd+T / Meta+T	Erweitertes Denken umschalten
<code>chat:submit</code>	Enter	Nachricht senden
<code>chat:undo</code>	Ctrl+_	Letzte Aktion rückgängig machen
<code>chat:externalEditor</code>	Ctrl+G	In externem Editor öffnen
<code>chat:stash</code>	Ctrl+S	Aktuelle Eingabeaufforderung speichern
<code>chat:imagePaste</code>	Ctrl+V (Alt+V unter Windows)	Bild einfügen

*Unter Windows ohne VT-Modus (Node <24.2.0 / <22.17.0, Bun <1.2.23) Standard auf Meta+M.

Autovervollständigungsaktionen

Aktionen verfügbar im `Autocomplete` -Kontext:

Aktion	Standard	Beschreibung
<code>autocomplete:accept</code>	Tab	Vorschlag akzeptieren
<code>autocomplete:dismiss</code>	Escape	Menü schließen
<code>autocomplete:previous</code>	Oben	Vorheriger Vorschlag

Aktion	Standard	Beschreibung
<code>autocomplete:next</code>	Unten	Nächster Vorschlag

Bestätigungsaktionen

Aktionen verfügbar im `Confirmation`-Kontext:

Aktion	Standard	Beschreibung
<code>confirm:yes</code>	Y, Enter	Aktion bestätigen
<code>confirm:no</code>	N, Escape	Aktion ablehnen
<code>confirm:previous</code>	Oben	Vorherige Option
<code>confirm:next</code>	Unten	Nächste Option
<code>confirm:nextField</code>	Tab	Nächstes Feld
<code>confirm:previousField</code>	(nicht gebunden)	Vorheriges Feld
<code>confirm:cycleMode</code>	Shift+Tab	Berechtigungsmodi durchlaufen
<code>confirm:toggleExplanation</code>	Ctrl+E	Berechtigungserklärung umschalten

Berechtigungsaktionen

Aktionen verfügbar im `Confirmation`-Kontext für Berechtigungsdialoge:

Aktion	Standard	Beschreibung
<code>permission:toggleDebug</code>	Ctrl+D	Berechtigungs-Debug-Info umschalten

Transkript-Aktionen

Aktionen verfügbar im `Transcript`-Kontext:

Aktion	Standard	Beschreibung
<code>transcript:toggleShowAll</code>	Ctrl+E	Alle Inhalte anzeigen umschalten

Aktion	Standard	Beschreibung
<code>transcript:exit</code>	Ctrl+C, Escape	Transkript-Ansicht beenden

Verlaufssuch-Aktionen

Aktionen verfügbar im `HistorySearch` -Kontext:

Aktion	Standard	Beschreibung
<code>historySearch:next</code>	Ctrl+R	Nächster Treffer
<code>historySearch:accept</code>	Escape, Tab	Auswahl akzeptieren
<code>historySearch:cancel</code>	Ctrl+C	Suche abbrechen
<code>historySearch:execute</code>	Enter	Ausgewählten Befehl ausführen

Aufgaben-Aktionen

Aktionen verfügbar im `Task` -Kontext:

Aktion	Standard	Beschreibung
<code>task:background</code>	Ctrl+B	Aktuelle Aufgabe in den Hintergrund verschieben

Design-Aktionen

Aktionen verfügbar im `ThemePicker` -Kontext:

Aktion	Standard	Beschreibung
<code>theme:toggleSyntaxHighlighting</code>	Ctrl+T	Syntaxhervorhebung umschalten

Hilfe-Aktionen

Aktionen verfügbar im `Help` -Kontext:

Aktion	Standard	Beschreibung
<code>help:dismiss</code>	Escape	Hilfemenü schließen

Tabs-Aktionen

Aktionen verfügbar im **Tabs** -Kontext:

Aktion	Standard	Beschreibung
<code>tabs:next</code>	Tab, Rechts	Nächster Tab
<code>tabs:previous</code>	Shift+Tab, Links	Vorheriger Tab

Anhänge-Aktionen

Aktionen verfügbar im **Attachments** -Kontext:

Aktion	Standard	Beschreibung
<code>attachments:next</code>	Rechts	Nächster Anhang
<code>attachments:previous</code>	Links	Vorheriger Anhang
<code>attachments:remove</code>	Rücktaste, Löschen	Ausgewählten Anhang entfernen
<code>attachments:exit</code>	Unten, Escape	Anhang-Leiste beenden

Fußzeilen-Aktionen

Aktionen verfügbar im **Footer** -Kontext:

Aktion	Standard	Beschreibung
<code>footer:next</code>	Rechts	Nächstes Fußzeilen-Element
<code>footer:previous</code>	Links	Vorheriges Fußzeilen-Element
<code>footer:openSelected</code>	Enter	Ausgewähltes Fußzeilen-Element öffnen
<code>footer:clearSelection</code>	Escape	Fußzeilen-Auswahl löschen

Nachrichtenauswahl-Aktionen

Aktionen verfügbar im **MessageSelector** -Kontext:

Aktion	Standard	Beschreibung
<code>messageSelector:up</code>	Oben, K, Ctrl+P	In der Liste nach oben bewegen
<code>messageSelector:down</code>	Unten, J, Ctrl+N	In der Liste nach unten bewegen
<code>messageSelector:top</code>	Ctrl+Oben, Shift+Oben, Meta+Oben, Shift+K	Zum Anfang springen
<code>messageSelector:bottom</code>	Ctrl+Unten, Shift+Unten, Meta+Unten, Shift+J	Zum Ende springen
<code>messageSelector:select</code>	Enter	Nachricht auswählen

Diff-Aktionen

Aktionen verfügbar im `DiffDialog` -Kontext:

Aktion	Standard	Beschreibung
<code>diff:dismiss</code>	Escape	Diff-Viewer schließen
<code>diff:previousSource</code>	Links	Vorherige Diff-Quelle
<code>diff:nextSource</code>	Rechts	Nächste Diff-Quelle
<code>diff:previousFile</code>	Oben	Vorherige Datei im Diff
<code>diff:nextFile</code>	Unten	Nächste Datei im Diff
<code>diff:viewDetails</code>	Enter	Diff-Details anzeigen
<code>diff:back</code>	(kontextspezifisch)	Im Diff-Viewer zurückgehen

Modell-Picker-Aktionen

Aktionen verfügbar im `ModelPicker` -Kontext:

Aktion	Standard	Beschreibung
<code>modelPicker:decreaseEffort</code>	Links	Aufwandsstufe verringern
<code>modelPicker:increaseEffort</code>	Rechts	Aufwandsstufe erhöhen

Select-Aktionen

Aktionen verfügbar im **Select** -Kontext:

Aktion	Standard	Beschreibung
<code>select:next</code>	Unten, J, Ctrl+N	Nächste Option
<code>select:previous</code>	Oben, K, Ctrl+P	Vorherige Option
<code>select:accept</code>	Enter	Auswahl akzeptieren
<code>select:cancel</code>	Escape	Auswahl abbrechen

Plugin-Aktionen

Aktionen verfügbar im **Plugin** -Kontext:

Aktion	Standard	Beschreibung
<code>plugin:toggle</code>	Leertaste	Plugin-Auswahl umschalten
<code>plugin:install</code>	I	Ausgewählte Plugins installieren

Einstellungs-Aktionen

Aktionen verfügbar im **Settings** -Kontext:

Aktion	Standard	Beschreibung
<code>settings:search</code>	/	Suchmodus aktivieren
<code>settings:retry</code>	R	Nutzungsdaten neu laden (bei Fehler)

Tastenkombinations-Syntax

Modifizierer

Verwenden Sie Modifizierer-Tasten mit dem **+** -Trennzeichen:

- `ctrl` oder `control` - Strg-Taste
- `alt`, `opt`, oder `option` - Alt/Option-Taste
- `shift` - Umschalt-Taste
- `meta`, `cmd`, oder `command` - Meta/Befehlstaste

Beispiele:

<code>ctrl+k</code>	Einzelne Taste mit Modifizierer
<code>shift+tab</code>	Umschalt + Tab
<code>meta+p</code>	Befehl/Meta + P
<code>ctrl+shift+c</code>	Mehrere Modifizierer

Großbuchstaben

Ein eigenständiger Großbuchstabe impliziert Umschalt. Zum Beispiel ist `K` gleichbedeutend mit `shift+k`. Dies ist nützlich für Vim-ähnliche Bindings, bei denen Groß- und Kleinbuchstaben unterschiedliche Bedeutungen haben.

Großbuchstaben mit Modifizierern (z. B. `ctrl+K`) werden als stilistisch behandelt und implizieren **nicht** Umschalt — `ctrl+K` ist dasselbe wie `ctrl+k`.

Akkorde

Akkorde sind Sequenzen von Tastenkombinationen, die durch Leerzeichen getrennt sind:

```
ctrl+k ctrl+s   Drücken Sie Ctrl+K, loslassen, dann Ctrl+S
```

Spezielle Tasten

- `escape` oder `esc` - Escape-Taste
- `enter` oder `return` - Enter-Taste
- `tab` - Tab-Taste
- `space` - Leertaste
- `up`, `down`, `left`, `right` - Pfeiltasten
- `backspace`, `delete` - Lösch Tasten

Standardkürzel aufheben

Setzen Sie eine Aktion auf `null`, um ein Standardkürzel aufzuheben:

```
{
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+s": null
      }
    }
  ]
}
```

Reservierte Kürzel

Diese Kürzel können nicht neu gebunden werden:

Kürzel	Grund
Ctrl+C	Hardcodierter Interrupt/Abbruch
Ctrl+D	Hardcodierter Ausstieg

Terminal-Konflikte

Einige Kürzel können mit Terminal-Multiplexern in Konflikt geraten:

Kürzel	Konflikt
Ctrl+B	tmux-Präfix (zweimal drücken zum Senden)
Ctrl+A	GNU Screen-Präfix
Ctrl+Z	Unix-Prozess-Suspend (SIGTSTP)

Vim-Modus-Interaktion

Wenn der Vim-Modus aktiviert ist (`/vim`), arbeiten Keybindings und Vim-Modus unabhängig:

- **Vim-Modus** verarbeitet Eingaben auf der Texteingangsebene (Cursor-Bewegung, Modi, Bewegungen)

- **Keybindings** verarbeiten Aktionen auf der Komponentenebene (Aufgaben umschalten, senden usw.)
- Die Escape-Taste im Vim-Modus wechselt von INSERT zu NORMAL-Modus; sie löst nicht `chat:cancel` aus
- Die meisten Ctrl+Taste-Kürzel werden durch den Vim-Modus zum Keybinding-System weitergeleitet
- Im Vim-NORMAL-Modus zeigt `?` das Hilfemenü an (Vim-Verhalten)

Validierung

Claude Code validiert Ihre Keybindings und zeigt Warnungen für:

- Parse-Fehler (ungültiges JSON oder Struktur)
- Ungültige Kontextnamen
- Reservierte Kürzel-Konflikte
- Terminal-Multiplexer-Konflikte
- Doppelte Bindings im selben Kontext

Führen Sie `/doctor` aus, um Keybinding-Warnungen anzuzeigen.

Part 4: Configuration

Berechtigungen konfigurieren

Kontrollieren Sie, worauf Claude Code zugreifen kann und was es mit granularen Berechtigungsregeln, Modi und verwalteten Richtlinien tun kann.

Claude Code unterstützt granulare Berechtigungen, sodass Sie genau angeben können, was der Agent tun darf und was nicht. Berechtigungseinstellungen können in die Versionskontrolle eingecheckt und an alle Entwickler in Ihrer Organisation verteilt werden, sowie von einzelnen Entwicklern angepasst werden.

Berechtigungssystem

Claude Code verwendet ein gestuftes Berechtigungssystem, um Leistung und Sicherheit auszugleichen:

Werkzeugtyp	Beispiel	Genehmigung erforderlich	Verhalten „Ja, nicht mehr fragen“
Nur Lesen	Dateilesevorgänge, Grep	Nein	N/A
Bash-Befehle	Shell-Ausführung	Ja	Dauerhaft pro Projektverzeichnis und Befehl
Dateiänderung	Dateien bearbeiten/schreiben	Ja	Bis zum Ende der Sitzung

Berechtigungen verwalten

Sie können Claude Code’s Werkzeugberechtigungen mit `/permissions` anzeigen und verwalten. Diese Benutzeroberfläche listet alle Berechtigungsregeln und die `settings.json`-Dateien auf, aus denen sie stammen.

- **Allow**-Regeln ermöglichen Claude Code, das angegebene Werkzeug ohne manuelle Genehmigung zu verwenden.

- **Ask**-Regeln fordern eine Bestätigung auf, wenn Claude Code versucht, das angegebene Werkzeug zu verwenden.
- **Deny**-Regeln verhindern, dass Claude Code das angegebene Werkzeug verwendet.

Regeln werden in dieser Reihenfolge ausgewertet: **deny** -> **ask** -> **allow**. Die erste übereinstimmende Regel gewinnt, daher haben Deny-Regeln immer Vorrang.

Berechtigungsmodi

Claude Code unterstützt mehrere Berechtigungsmodi, die steuern, wie Werkzeuge genehmigt werden. Legen Sie den `defaultMode` in Ihren [Einstellungsdateien](#) fest:

Modus	Beschreibung
<code>default</code>	Standardverhalten: fordert Genehmigung bei der ersten Verwendung jedes Werkzeugs auf
<code>acceptEdits</code>	Akzeptiert automatisch Dateiberechtigungen für die Sitzung
<code>plan</code>	Plan Mode: Claude kann Dateien analysieren, aber nicht ändern oder Befehle ausführen
<code>dontAsk</code>	Verweigert Werkzeuge automatisch, es sei denn, sie sind vorab über <code>/permissions</code> oder <code>permissions.allow</code> -Regeln genehmigt
<code>bypassPermissions</code>	Überspringt alle Berechtigungsaufforderungen (erfordert sichere Umgebung, siehe Warnung unten)

Warning:

Der Modus `bypassPermissions` deaktiviert alle Berechtigungsprüfungen. Verwenden Sie diesen nur in isolierten Umgebungen wie Containern oder VMs, in denen Claude Code keinen Schaden anrichten kann. Administratoren können diesen Modus verhindern, indem sie `disableBypassPermissionsMode` in [verwalteten Einstellungen](#) auf `"disable"` setzen.

Berechtigungsregelsyntax

Berechtigungsregeln folgen dem Format `Tool` oder `Tool(specifier)`.

Alle Verwendungen eines Werkzeugs abgleichen

Um alle Verwendungen eines Werkzeugs abzugleichen, verwenden Sie einfach den Werkzeugnamen ohne Klammern:

Regel	Effekt
<code>Bash</code>	Gleicht alle Bash-Befehle ab
<code>WebFetch</code>	Gleicht alle Web-Fetch-Anfragen ab
<code>Read</code>	Gleicht alle Dateilesevorgänge ab

`Bash(*)` ist gleichwertig mit `Bash` und gleicht alle Bash-Befehle ab.

Verwenden Sie Spezifizierer für granulare Kontrolle

Fügen Sie einen Spezifizierer in Klammern hinzu, um bestimmte Werkzeugverwendungen abzugleichen:

Regel	Effekt
<code>Bash(npm run build)</code>	Gleicht den genauen Befehl <code>npm run build</code> ab
<code>Read(./env)</code>	Gleicht das Lesen der <code>.env</code> -Datei im aktuellen Verzeichnis ab
<code>WebFetch(domain:example.com)</code>	Gleicht Fetch-Anfragen an example.com ab

Wildcard-Muster

Bash-Regeln unterstützen Glob-Muster mit `*`. Platzhalter können an jeder Position im Befehl erscheinen. Diese Konfiguration ermöglicht npm- und git-Commit-Befehle, blockiert aber git push:

```
{
  "permissions": {
    "allow": [
      "Bash(npm run *)",
      "Bash(git commit *)",
      "Bash(git * main)",
      "Bash(* --version)",
      "Bash(* --help *)"
    ],
    "deny": [
      "Bash(git push *)"
    ]
  }
}
```

Das Leerzeichen vor `*` ist wichtig: `Bash(ls *)` gleicht `ls -la` ab, aber nicht `ls of`, während `Bash(ls*)` beide abgleicht. Die veraltete `:*`-Suffixsyntax ist gleichwertig mit `*`, wird aber nicht mehr empfohlen.

Werkzeugspezifische Berechtigungsregeln

Bash

Bash-Berechtigungsregeln unterstützen Wildcard-Abgleich mit `*`. Platzhalter können an jeder Position im Befehl erscheinen, einschließlich am Anfang, in der Mitte oder am Ende:

- `Bash(npm run build)` gleicht den genauen Bash-Befehl `npm run build` ab
- `Bash(npm run test *)` gleicht Bash-Befehle ab, die mit `npm run test` beginnen
- `Bash(npm *)` gleicht jeden Befehl ab, der mit `npm` beginnt
- `Bash(* install)` gleicht jeden Befehl ab, der mit `install` endet
- `Bash(git * main)` gleicht Befehle wie `git checkout main`, `git merge main` ab

Wenn `*` am Ende mit einem Leerzeichen davor erscheint (wie `Bash(ls *)`), wird eine Wortgrenze erzwungen, die erfordert, dass dem Präfix ein Leerzeichen oder das Ende der Zeichenkette folgt. Zum Beispiel gleicht `Bash(ls *)` `ls -la` ab, aber nicht `ls of`. Im Gegensatz dazu gleicht `Bash(ls*)` ohne Leerzeichen sowohl `ls -la` als auch `ls of` ab, da es keine Wortgrenzbeschränkung gibt.

Tip:

Claude Code ist sich Shell-Operatoren (wie `&&`) bewusst, daher gibt eine Präfixabgleichregel wie `Bash(safe-cmd *)` ihm nicht die Berechtigung, den Befehl `safe-cmd && other-cmd` auszuführen.

Warning:

Bash-Berechtigungsmuster, die versuchen, Befehlsargumente einzuschränken, sind fragil. Zum Beispiel beabsichtigt `Bash(curl http://github.com/ *)`, `curl` auf GitHub-URLs zu beschränken, wird aber Variationen nicht abgleichen wie:

- Optionen vor URL: `curl -X GET http://github.com/...`
- Anderes Protokoll: `curl https://github.com/...`
- Umleitungen: `curl -L http://bit.ly/xyz` (leitet zu github um)
- Variablen: `URL=http://github.com && curl $URL`
- Zusätzliche Leerzeichen: `curl http://github.com`

Für zuverlässigere URL-Filterung sollten Sie erwägen:

- **Bash-Netzwerkwerkzeuge einschränken:** Verwenden Sie Deny-Regeln, um `curl`, `wget` und ähnliche Befehle zu blockieren, verwenden Sie dann das WebFetch-Werkzeug mit `WebFetch(domain:github.com)`-Berechtigung für zulässige Domänen
- **PreToolUse-Hooks verwenden:** Implementieren Sie einen Hook, der URLs in Bash-Befehlen validiert und nicht zulässige Domänen blockiert
- Claude Code über Ihre zulässigen `curl`-Muster über `CLAUDE.md` informieren

Beachten Sie, dass die alleinige Verwendung von WebFetch keinen Netzwerkzugriff verhindert. Wenn Bash zulässig ist, kann Claude immer noch `curl`, `wget` oder andere Werkzeuge verwenden, um auf jede URL zuzugreifen.

Read und Edit

`Edit`-Regeln gelten für alle integrierten Werkzeuge, die Dateien bearbeiten. Claude versucht nach besten Kräften, `Read`-Regeln auf alle integrierten Werkzeuge anzuwenden, die Dateien lesen, wie Grep und Glob.

Read- und Edit-Regeln folgen beide der [gitignore](#)-Spezifikation mit vier unterschiedlichen Mustertypen:

Muster	Bedeutung	Beispiel	Gleicht ab
<code>//path</code>	Absoluter Pfad vom Dateisystem-Root	<code>Read(//Users/alice/secrets/**)</code>	<code>/Users/alice/secrets/**</code>

Muster	Bedeutung	Beispiel	Gleicht ab
<code>~/path</code>	Pfad vom Home -Verzeichnis	<code>Read(~/Documents/ *.pdf)</code>	<code>/Users/alice/ Documents/*.pdf</code>
<code>/path</code>	Pfad relativ zum Projekt-Root	<code>Edit(/src/**/ *.ts)</code>	<code><project root>/ src/**/*.ts</code>
<code>path</code> oder <code>./path</code>	Pfad relativ zum aktuellen Verzeichnis	<code>Read(*.env)</code>	<code><cwd>/*.env</code>

Warning:

Ein Muster wie `/Users/alice/file` ist KEIN absoluter Pfad. Es ist relativ zum Projekt-Root. Verwenden Sie `//Users/alice/file` für absolute Pfade.

Beispiele:

- `Edit(/docs/**)` : Bearbeitungen in `<project>/docs/` (NICHT `/docs/` und NICHT `<project>/.claude/docs/`)
- `Read(~/.zshrc)` : liest die `.zshrc` Ihres Home-Verzeichnisses
- `Edit(/tmp/scratch.txt)` : bearbeitet den absoluten Pfad `/tmp/scratch.txt`
- `Read(src/**)` : liest aus `<current-directory>/src/`

Note:

In gitignore-Mustern gleicht `*` Dateien in einem einzelnen Verzeichnis ab, während `**` rekursiv über Verzeichnisse hinweg abgleicht. Um allen Dateizugriff zu ermöglichen, verwenden Sie einfach den Werkzeugnamen ohne Klammern: `Read` , `Edit` oder `Write` .

WebFetch

- `WebFetch(domain:example.com)` gleicht Fetch-Anfragen an example.com ab

MCP

- `mcp__puppeteer` gleicht jedes Werkzeug ab, das vom `puppeteer` -Server bereitgestellt wird (Name in Claude Code konfiguriert)
- `mcp__puppeteer__*` Wildcard-Syntax, die auch alle Werkzeuge vom `puppeteer` -Server abgleicht
- `mcp__puppeteer__puppeteer_navigate` gleicht das `puppeteer_navigate` -Werkzeug ab, das vom `puppeteer` -Server bereitgestellt wird

Agent (subagents)

Verwenden Sie `Agent(AgentName)` -Regeln, um zu steuern, welche [subagents](#) Claude verwenden kann:

- `Agent(Explore)` gleicht den Explore-Subagent ab
- `Agent(Plan)` gleicht den Plan-Subagent ab
- `Agent(my-custom-agent)` gleicht einen benutzerdefinierten Subagent namens `my-custom-agent` ab

Fügen Sie diese Regeln zum `deny` -Array in Ihren Einstellungen hinzu oder verwenden Sie das `--disallowedTools` -CLI-Flag, um bestimmte Agenten zu deaktivieren. Um den Explore-Agenten zu deaktivieren:

```
{
  "permissions": {
    "deny": ["Agent(Explore)"]
  }
}
```

Berechtigungen mit Hooks erweitern

[Claude Code hooks](#) bieten eine Möglichkeit, benutzerdefinierte Shell-Befehle zu registrieren, um die Berechtigungsevaluierung zur Laufzeit durchzuführen. Wenn Claude Code einen Werkzeugaufwurf tätigt, werden PreToolUse-Hooks vor dem Berechtigungssystem ausgeführt, und die Hook-Ausgabe kann bestimmen, ob der Werkzeugaufwurf anstelle des Berechtigungssystems genehmigt oder abgelehnt wird.

Arbeitsverzeichnisse

Standardmäßig hat Claude Zugriff auf Dateien in dem Verzeichnis, in dem es gestartet wurde. Sie können diesen Zugriff erweitern:

- **Beim Start:** Verwenden Sie das CLI-Argument `--add-dir <path>`
- **Während der Sitzung:** Verwenden Sie den Befehl `/add-dir`
- **Persistente Konfiguration:** Fügen Sie zu `additionalDirectories` in [Einstellungsdateien](#) hinzu

Dateien in zusätzlichen Verzeichnissen folgen den gleichen Berechtigungsregeln wie das ursprüngliche Arbeitsverzeichnis: Sie werden lesbar ohne Aufforderungen, und Dateiberechtigungen folgen dem aktuellen Berechtigungsmodus.

Wie Berechtigungen mit Sandboxing interagieren

Berechtigungen und [Sandboxing](#) sind komplementäre Sicherheitsebenen:

- **Berechtigungen** steuern, welche Werkzeuge Claude Code verwenden kann und auf welche Dateien oder Domänen es zugreifen kann. Sie gelten für alle Werkzeuge (Bash, Read, Edit, WebFetch, MCP und andere).
- **Sandboxing** bietet OS-Ebenen-Durchsetzung, die den Zugriff des Bash-Werkzeugs auf das Dateisystem und das Netzwerk einschränkt. Es gilt nur für Bash-Befehle und ihre untergeordneten Prozesse.

Verwenden Sie beide für Defense-in-Depth:

- Berechtigungs-Deny-Regeln blockieren Claude daran, überhaupt zu versuchen, auf eingeschränkte Ressourcen zuzugreifen
- Sandbox-Einschränkungen verhindern, dass Bash-Befehle Ressourcen außerhalb definierter Grenzen erreichen, selbst wenn eine Prompt-Injection Claude's Entscheidungsfindung umgeht
- Dateisystem-Einschränkungen in der Sandbox verwenden Read- und Edit-Deny-Regeln, nicht separate Sandbox-Konfiguration
- Netzwerk-Einschränkungen kombinieren WebFetch-Berechtigungsregeln mit der `allowedDomains`-Liste der Sandbox

Verwaltete Einstellungen

Für Organisationen, die eine zentralisierte Kontrolle über die Claude Code-Konfiguration benötigen, können Administratoren verwaltete Einstellungen bereitstellen, die nicht von Benutzer- oder Projekteinstellungen überschrieben werden können. Diese Richtlinieneinstellungen folgen dem gleichen Format wie reguläre Einstellungsdateien und können über MDM/OS-Ebenen-Richtlinien, verwaltete Einstellungsdateien oder [servergesteuerte Einstellungen](#) bereitgestellt werden. Siehe [Einstellungsdateien](#) für Bereitstellungsmechanismen und Dateispeicherorte.

Nur verwaltete Einstellungen

Einige Einstellungen sind nur in verwalteten Einstellungen wirksam:

Einstellung	Beschreibung
<code>disableBypassPermissionsMode</code>	Auf <code>"disable"</code> setzen, um den Modus <code>bypassPermissions</code> und das Flag <code>--dangerously-skip-permissions</code> zu verhindern
<code>allowManagedPermissionRulesOnly</code>	Wenn <code>true</code> , verhindert, dass Benutzer- und Projekteinstellungen <code>allow</code> -, <code>ask</code> - oder <code>deny</code> -Berechtigungsregeln definieren. Nur Regeln in verwalteten Einstellungen gelten
<code>allowManagedHooksOnly</code>	Wenn <code>true</code> , verhindert das Laden von Benutzer-, Projekt- und Plugin-Hooks. Nur verwaltete Hooks und SDK-Hooks sind zulässig
<code>allowManagedMcpServersOnly</code>	Wenn <code>true</code> , werden nur <code>allowedMcpServers</code> aus verwalteten Einstellungen berücksichtigt. <code>deniedMcpServers</code> wird immer noch aus allen Quellen zusammengeführt. Siehe Verwaltete MCP-Konfiguration
<code>blockedMarketplaces</code>	Blocklist von Marketplace-Quellen. Blockierte Quellen werden vor dem Download überprüft, sodass sie das Dateisystem nie berühren. Siehe verwaltete Marketplace-Einschränkungen
<code>sandbox.network.allowManagedDomainsOnly</code>	Wenn <code>true</code> , werden nur <code>allowedDomains</code> und <code>WebFetch(domain: ...)</code> -Allow-Regeln aus verwalteten Einstellungen berücksichtigt. Nicht zulässige Domänen werden automatisch blockiert, ohne den Benutzer zu fragen. Verweigte Domänen werden immer noch aus allen Quellen zusammengeführt
<code>strictKnownMarketplaces</code>	Steuert, welche Plugin-Marketplaces Benutzer hinzufügen können. Siehe verwaltete Marketplace-Einschränkungen

Einstellung	Beschreibung
<code>allow_remote_sessions</code>	Wenn <code>true</code> , ermöglicht Benutzern, Remote Control und Web-Sitzungen zu starten. Standardmäßig <code>true</code> . Auf <code>false</code> setzen, um Remote-Sitzungszugriff zu verhindern

Einstellungspriorität

Berechtigungsregeln folgen der gleichen [Einstellungspriorität](#) wie alle anderen Claude Code-Einstellungen:

1. **Verwaltete Einstellungen:** können von keiner anderen Ebene überschrieben werden, einschließlich Befehlszeilenargumenten
2. **Befehlszeilenargumente:** temporäre Sitzungsüberschreibungen
3. **Lokale Projekteinstellungen** (`.claude/settings.local.json`)
4. **Gemeinsame Projekteinstellungen** (`.claude/settings.json`)
5. **Benutzereinstellungen** (`~/.claude/settings.json`)

Wenn ein Werkzeug auf einer beliebigen Ebene verweigert wird, kann keine andere Ebene es zulassen. Zum Beispiel kann eine verwaltete Einstellungs-Deny nicht durch `--allowedTools` überschrieben werden, und `--disallowedTools` kann Einschränkungen über das hinaus hinzufügen, was verwaltete Einstellungen definieren.

Wenn eine Berechtigung in Benutzereinstellungen zulässig ist, aber in Projekteinstellungen verweigert wird, hat die Projekteinstellung Vorrang und die Berechtigung wird blockiert.

Beispielkonfigurationen

Dieses [Repository](#) enthält Starter-Einstellungskonfigurationen für häufige Bereitstellungsszenarien. Verwenden Sie diese als Ausgangspunkte und passen Sie sie an Ihre Anforderungen an.

Siehe auch

- [Einstellungen](#): vollständige Konfigurationsreferenz einschließlich der Berechtigungseinstellungstabelle
- [Sandboxing](#): OS-Ebenen-Dateisystem- und Netzwerkisolation für Bash-Befehle
- [Authentifizierung](#): Richten Sie Benutzerzugriff auf Claude Code ein

- [Sicherheit](#): Sicherheitsvorkehrungen und Best Practices
- [Hooks](#): Automatisieren Sie Workflows und erweitern Sie die Berechtigungsevaluierung

Wie Claude sich Ihr Projekt merkt

Geben Sie Claude persistente Anweisungen mit CLAUDE.md-Dateien, und lassen Sie Claude automatisch Erkenntnisse mit Auto-Memory sammeln.

Jede Claude Code-Sitzung beginnt mit einem frischen Context Window. Zwei Mechanismen tragen Wissen über Sitzungen hinweg:

- **CLAUDE.md-Dateien:** Anweisungen, die Sie schreiben, um Claude persistenten Kontext zu geben
- **Auto-Memory:** Notizen, die Claude selbst basierend auf Ihren Korrekturen und Vorlieben schreibt

Diese Seite behandelt folgende Themen:

- [CLAUDE.md-Dateien schreiben und organisieren](#)
- [Regeln auf bestimmte Dateitypen beschränken](#) mit `.claude/rules/`
- [Auto-Memory konfigurieren](#), damit Claude automatisch Notizen macht
- [Fehlerbehebung](#), wenn Anweisungen nicht befolgt werden

CLAUDE.md vs. Auto-Memory

Claude Code hat zwei komplementäre Speichersysteme. Beide werden zu Beginn jeder Konversation geladen. Claude behandelt sie als Kontext, nicht als erzwungene Konfiguration. Je spezifischer und prägnanter Ihre Anweisungen sind, desto konsistenter folgt Claude ihnen.

	CLAUDE.md-Dateien	Auto-Memory
Wer schreibt es	Sie	Claude
Was es enthält	Anweisungen und Regeln	Erkenntnisse und Muster
Umfang	Projekt, Benutzer oder Organisation	Pro Worktree
Geladen in	Jede Sitzung	Jede Sitzung (erste 200 Zeilen)

	CLAUDE.md-Dateien	Auto-Memory
Verwenden für	Coding-Standards, Workflows, Projektarchitektur	Build-Befehle, Debugging-Erkenntnisse, Vorlieben, die Claude entdeckt

Verwenden Sie CLAUDE.md-Dateien, wenn Sie Claudes Verhalten lenken möchten. Auto-Memory lässt Claude aus Ihren Korrekturen lernen, ohne manuelle Arbeit.

Subagents können auch ihre eigene Auto-Memory pflegen. Weitere Informationen finden Sie unter [Subagent-Konfiguration](#).

CLAUDE.md-Dateien

CLAUDE.md-Dateien sind Markdown-Dateien, die Claude persistente Anweisungen für ein Projekt, Ihren persönlichen Workflow oder Ihre gesamte Organisation geben. Sie schreiben diese Dateien in Klartext; Claude liest sie zu Beginn jeder Sitzung.

Wählen Sie, wo Sie CLAUDE.md-Dateien ablegen

CLAUDE.md-Dateien können sich an mehreren Orten befinden, jeder mit einem anderen Umfang. Spezifischere Orte haben Vorrang vor breiteren.

Umfang	Ort	Zweck	Anwendungsbeispiele	Geteilt mit
Verwaltete Richtlinien	• macOS: <code>/Library/Application Support/ClaudeCode/CLAUDE.md</code> • Linux und WSL: <code>/etc/claude-code/CLAUDE.md</code> • Windows: <code>C:\Program Files\ClaudeCode\CLAUDE.md</code>	Organisationsweite Anweisungen, verwaltet von IT/DevOps	Unternehmens-Coding-Standards, Sicherheitsrichtlinien, Compliance-Anforderungen	Alle Benutzer in der Organisation
Projektanweisungen	<code>./CLAUDE.md</code> oder <code>./.claude/CLAUDE.md</code>	Team-gemeinsame Anweisungen für das Projekt	Projektarchitektur, Coding-Standards, häufige Workflows	Team-Mitglieder über Versionskontrolle

Um-fang	Ort	Zweck	Anwen-dungsbei-spiele	Geteilt mit
Benut-zeran-wei-sungen	<code>~/.claude/CLAUDE.md</code>	Persönliche Vor-lieben für alle Projekte	Code-Styling-Vorlieben, per-sönliche Too-ling-Shortcuts	Nur Sie (alle Projekte)

CLAUDE.md-Dateien in der Verzeichnishierarchie über dem Arbeitsverzeichnis werden beim Start vollständig geladen. CLAUDE.md-Dateien in Unterverzeichnissen werden bei Bedarf geladen, wenn Claude Dateien in diesen Verzeichnissen liest. Weitere Informationen finden Sie unter [Wie CLAUDE.md-Dateien geladen werden](#).

Bei großen Projekten können Sie Anweisungen in themaspezifische Dateien aufteilen, indem Sie [Projektregeln](#) verwenden. Regeln ermöglichen es Ihnen, Anweisungen auf bestimmte Dateitypen oder Unterverzeichnisse zu beschränken.

Richten Sie eine Projekt-CLAUDE.md ein

Eine Projekt-CLAUDE.md kann entweder in `./CLAUDE.md` oder `./.claude/CLAUDE.md` gespeichert werden. Erstellen Sie diese Datei und fügen Sie Anweisungen hinzu, die für jeden gelten, der am Projekt arbeitet: Build- und Test-Befehle, Coding-Standards, architektonische Entscheidungen, Namenskonventionen und häufige Workflows. Diese Anweisungen werden über Versionskontrolle mit Ihrem Team geteilt, daher konzentrieren Sie sich auf projektweite Standards statt auf persönliche Vorlieben.

Tip:

Führen Sie `/init` aus, um automatisch eine Start-CLAUDE.md zu generieren. Claude analysiert Ihre Codebasis und erstellt eine Datei mit Build-Befehlen, Test-Anweisungen und Projektkonventionen, die es entdeckt. Wenn bereits eine CLAUDE.md vorhanden ist, schlägt `/init` Verbesserungen vor, statt sie zu überschreiben. Verfeinern Sie sie von dort aus mit Anweisungen, die Claude nicht selbst entdecken würde.

Schreiben Sie effektive Anweisungen

CLAUDE.md-Dateien werden zu Beginn jeder Sitzung in das Context Window geladen und verbrauchen Token zusammen mit Ihrer Konversation. Da sie Kontext statt erzwungene Konfiguration sind, beeinflusst die Art, wie Sie Anweisungen schreiben, wie zuverlässig Claude ihnen folgt. Spezifische, prägnante, gut strukturierte Anweisungen funktionieren am besten.

Größe: Ziel unter 200 Zeilen pro CLAUDE.md-Datei. Längere Dateien verbrauchen mehr Kontext und reduzieren die Einhaltung. Wenn Ihre Anweisungen zu groß werden, teilen Sie sie mit [Importen](#) oder `.claude/rules/`-Dateien auf.

Struktur: Verwenden Sie Markdown-Header und Aufzählungszeichen, um verwandte Anweisungen zu gruppieren. Claude scannt die Struktur genauso wie Leser: organisierte Abschnitte sind leichter zu befolgen als dichte Absätze.

Spezifität: Schreiben Sie Anweisungen, die konkret genug sind, um überprüft zu werden. Zum Beispiel:

- „Verwenden Sie 2-Leerzeichen-Einrückung“ statt „Formatieren Sie Code ordnungsgemäß“
- „Führen Sie `npm test` vor dem Commit aus“ statt „Testen Sie Ihre Änderungen“
- „API-Handler befinden sich in `src/api/handlers/`“ statt „Halten Sie Dateien organisiert“

Konsistenz: Wenn zwei Regeln sich widersprechen, kann Claude eine willkürlich auswählen. Überprüfen Sie Ihre CLAUDE.md-Dateien, verschachtelte CLAUDE.md-Dateien in Unterverzeichnissen und `.claude/rules/` regelmäßig, um veraltete oder widersprüchliche Anweisungen zu entfernen. In Monorepos verwenden Sie `claudeMdExcludes`, um CLAUDE.md-Dateien von anderen Teams zu überspringen, die für Ihre Arbeit nicht relevant sind.

Importieren Sie zusätzliche Dateien

CLAUDE.md-Dateien können zusätzliche Dateien mit der Syntax `@path/to/import` importieren. Importierte Dateien werden erweitert und beim Start zusammen mit der CLAUDE.md, die sie referenziert, in den Kontext geladen.

Sowohl relative als auch absolute Pfade sind zulässig. Relative Pfade werden relativ zur Datei aufgelöst, die den Import enthält, nicht zum Arbeitsverzeichnis. Importierte Dateien können rekursiv andere Dateien importieren, mit einer maximalen Tiefe von fünf Hops.

Um eine README, package.json und einen Workflow-Leitfaden einzubeziehen, referenzieren Sie sie mit der `@`-Syntax überall in Ihrer CLAUDE.md:

```
Siehe @README für Projektübersicht und @package.json für verfügbare npm-Befehle  
für dieses Projekt.
```

```
## Zusätzliche Anweisungen  
- Git-Workflow @docs/git-instructions.md
```

Für persönliche Vorlieben, die Sie nicht einchecken möchten, importieren Sie eine Datei aus Ihrem Home-Verzeichnis. Der Import geht in die gemeinsame CLAUDE.md, aber die Datei, auf die er verweist, bleibt auf Ihrem Computer:

```
## Individuelle Vorlieben  
- @~/ .claude/my-project-instructions.md
```

Warning:

Wenn Claude Code zum ersten Mal externe Importe in einem Projekt antrifft, zeigt es einen Genehmigungsdialog an, der die Dateien auflistet. Wenn Sie ablehnen, bleiben die Importe deaktiviert und der Dialog wird nicht erneut angezeigt.

Für einen strukturierteren Ansatz zur Organisation von Anweisungen siehe [.claude/rules/](#).

Wie CLAUDE.md-Dateien geladen werden

Claude Code liest CLAUDE.md-Dateien, indem es die Verzeichnisstruktur von Ihrem aktuellen Arbeitsverzeichnis aus durchläuft und jedes Verzeichnis unterwegs überprüft. Das bedeutet, wenn Sie Claude Code in `foo/bar/` ausführen, lädt es Anweisungen aus `foo/bar/CLAUDE.md` und `foo/CLAUDE.md`.

Claude entdeckt auch CLAUDE.md-Dateien in Unterverzeichnissen unter Ihrem aktuellen Arbeitsverzeichnis. Statt sie beim Start zu laden, werden sie eingebunden, wenn Claude Dateien in diesen Unterverzeichnissen liest.

Wenn Sie in einem großen Monorepo arbeiten, in dem CLAUDE.md-Dateien anderer Teams aufgegriffen werden, verwenden Sie `claudeMdExcludes`, um sie zu überspringen.

Laden aus zusätzlichen Verzeichnissen

Das Flag `--add-dir` gibt Claude Zugriff auf zusätzliche Verzeichnisse außerhalb Ihres Hauptarbeitsverzeichnisses. Standardmäßig werden `CLAUDE.md`-Dateien aus diesen Verzeichnissen nicht geladen.

Um auch `CLAUDE.md`-Dateien aus zusätzlichen Verzeichnissen zu laden, einschließlich `CLAUDE.md`, `.claude/CLAUDE.md` und `.claude/rules/*.md`, setzen Sie die Umgebungsvariable `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD`:

```
CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1 claude --add-dir ../shared-config
```

Organisieren Sie Regeln mit `.claude/rules/`

Bei größeren Projekten können Sie Anweisungen in mehrere Dateien mit dem Verzeichnis `.claude/rules/` organisieren. Dies hält Anweisungen modular und leichter für Teams zu pflegen. Regeln können auch [auf bestimmte Dateipfade beschränkt werden](#), sodass sie nur in den Kontext geladen werden, wenn Claude mit übereinstimmenden Dateien arbeitet, was Rauschen reduziert und Kontextraum spart.

Note:

Regeln werden in jeder Sitzung oder beim Öffnen übereinstimmender Dateien in den Kontext geladen. Für aufgabenspezifische Anweisungen, die nicht ständig im Kontext sein müssen, verwenden Sie stattdessen [skills](#), die nur geladen werden, wenn Sie sie aufrufen oder wenn Claude bestimmt, dass sie für Ihren Prompt relevant sind.

Richten Sie Regeln ein

Platzieren Sie Markdown-Dateien im Verzeichnis `.claude/rules/` Ihres Projekts. Jede Datei sollte ein Thema abdecken, mit einem aussagekräftigen Dateinamen wie `testing.md` oder `api-design.md`. Alle `.md`-Dateien werden rekursiv entdeckt, sodass Sie Regeln in Unterverzeichnisse wie `frontend/` oder `backend/` organisieren können:

```
your-project/
├── .claude/
│   ├── CLAUDE.md          # Hauptprojktanweisungen
│   └── rules/
│       ├── code-style.md   # Code-Style-Richtlinien
│       ├── testing.md      # Test-Konventionen
│       └── security.md     # Sicherheitsanforderungen
```

Regeln ohne `paths` -Frontmatter werden beim Start mit der gleichen Priorität wie `.claude/CLAUDE.md` geladen.

Pfadspezifische Regeln

Regeln können mit YAML-Frontmatter mit dem Feld `paths` auf bestimmte Dateien beschränkt werden. Diese bedingten Regeln gelten nur, wenn Claude mit Dateien arbeitet, die den angegebenen Mustern entsprechen.

```
---
paths:
  - "src/api/**/*.ts"
---

## API-Entwicklungsregeln

- Alle API-Endpunkte müssen Eingabevalidierung enthalten
- Verwenden Sie das Standard-Fehlerantwortformat
- Fügen Sie OpenAPI-Dokumentationskommentare ein
```

Regeln ohne ein `paths` -Feld werden bedingungslos geladen und gelten für alle Dateien. Pfadgebundene Regeln werden ausgelöst, wenn Claude Dateien liest, die dem Muster entsprechen, nicht bei jedem Tool-Einsatz.

Verwenden Sie Glob-Muster im Feld `paths` , um Dateien nach Erweiterung, Verzeichnis oder einer beliebigen Kombination zu vergleichen:

Muster	Passt zu
<code>**/*.ts</code>	Alle TypeScript-Dateien in jedem Verzeichnis
<code>src/**/*.*</code>	Alle Dateien unter dem Verzeichnis <code>src/</code>

Muster	Passt zu
<code>*.md</code>	Markdown-Dateien im Projektstammverzeichnis
<code>src/components/*.tsx</code>	React-Komponenten in einem bestimmten Verzeichnis

Sie können mehrere Muster angeben und Klammer-Expansion verwenden, um mehrere Erweiterungen in einem Muster zu vergleichen:

```
---
paths:
  - "src/**/*.{ts,tsx}"
  - "lib/**/*.ts"
  - "tests/**/*.test.ts"
---
```

Teilen Sie Regeln über Projekte hinweg mit Symlinks

Das Verzeichnis `.claude/rules/` unterstützt Symlinks, sodass Sie einen gemeinsamen Satz von Regeln pflegen und in mehrere Projekte verlinken können. Symlinks werden aufgelöst und normal geladen, und zirkuläre Symlinks werden erkannt und elegant behandelt.

Dieses Beispiel verlinkt sowohl ein gemeinsames Verzeichnis als auch eine einzelne Datei:

```
ln -s ~/shared-claude-rules .claude/rules/shared
ln -s ~/company-standards/security.md .claude/rules/security.md
```

Benutzerebenen-Regeln

Persönliche Regeln in `~/.claude/rules/` gelten für jedes Projekt auf Ihrem Computer. Verwenden Sie sie für Vorlieben, die nicht projektspezifisch sind:

```
~/.claude/rules/
├─ preferences.md    # Ihre persönlichen Coding-Vorlieben
└─ workflows.md      # Ihre bevorzugten Workflows
```

Benutzerebenen-Regeln werden vor Projektregeln geladen, was Projektregeln höhere Priorität gibt.

Verwalten Sie CLAUDE.md für große Teams

Für Organisationen, die Claude Code über Teams bereitstellen, können Sie Anweisungen zentralisieren und steuern, welche CLAUDE.md-Dateien geladen werden.

Stellen Sie organisationsweite CLAUDE.md bereit

Organisationen können eine zentral verwaltete CLAUDE.md bereitstellen, die für alle Benutzer auf einem Computer gilt. Diese Datei kann durch individuelle Einstellungen nicht ausgeschlossen werden.

Step 1: Erstellen Sie die Datei am Ort der verwalteten Richtlinie

- macOS: `/Library/Application Support/ClaudeCode/CLAUDE.md`
- Linux und WSL: `/etc/claude-code/CLAUDE.md`
- Windows: `C:\Program Files\ClaudeCode\CLAUDE.md`

Step 2: Stellen Sie mit Ihrem Konfigurationsverwaltungssystem bereit

Verwenden Sie MDM, Group Policy, Ansible oder ähnliche Tools, um die Datei über Entwicklungsmaschinen zu verteilen. Weitere Informationen finden Sie unter [verwaltete Einstellungen](#) für andere organisationsweite Konfigurationsoptionen.

Schließen Sie bestimmte CLAUDE.md-Dateien aus

In großen Monorepos können Vorgänger-CLAUDE.md-Dateien Anweisungen enthalten, die für Ihre Arbeit nicht relevant sind. Die Einstellung `claudeMdExcludes` ermöglicht es Ihnen, bestimmte Dateien nach Pfad oder Glob-Muster zu überspringen.

Dieses Beispiel schließt eine CLAUDE.md auf oberster Ebene und ein Regelverzeichnis aus einem übergeordneten Ordner aus. Fügen Sie es zu `.claude/settings.local.json` hinzu, damit der Ausschluss lokal auf Ihrem Computer bleibt:

```
{
  "claudeMdExcludes": [
    "**/monorepo/CLAUDE.md",
    "/home/user/monorepo/other-team/.claude/rules/**"
  ]
}
```

Muster werden mit Glob-Syntax gegen absolute Dateipfade abgeglichen. Sie können `claudeMdExcludes` auf jeder [EinstellungsEbene](#) konfigurieren: Benutzer, Projekt, lokal oder verwaltete Richtlinie. Arrays werden über Ebenen hinweg zusammengeführt.

Verwaltete Richtlinie CLAUDE.md-Dateien können nicht ausgeschlossen werden. Dies stellt sicher, dass organisationsweite Anweisungen unabhängig von individuellen Einstellungen immer gelten.

Auto-Memory

Auto-Memory lässt Claude Wissen über Sitzungen hinweg sammeln, ohne dass Sie etwas schreiben müssen. Claude speichert Notizen für sich selbst, während es arbeitet: Build-Befehle, Debugging-Erkenntnisse, Architektur-Notizen, Code-Style-Vorlieben und Workflow-Gewohnheiten. Claude speichert nicht jede Sitzung etwas. Es entscheidet, was es sich merken sollte, basierend darauf, ob die Information in einer zukünftigen Konversation nützlich wäre.

Note:

Auto-Memory erfordert Claude Code v2.1.59 oder später. Überprüfen Sie Ihre Version mit `claude --version`.

Aktivieren oder deaktivieren Sie Auto-Memory

Auto-Memory ist standardmäßig aktiviert. Um es umzuschalten, öffnen Sie `/memory` in einer Sitzung und verwenden Sie den Auto-Memory-Schalter, oder setzen Sie `autoMemoryEnabled` in Ihren Projekteinstellungen:

```
{
  "autoMemoryEnabled": false
}
```

Um Auto-Memory über eine Umgebungsvariable zu deaktivieren, setzen Sie `CLAUDE_CODE_DISABLE_AUTO_MEMORY=1`.

Speicherort

Jedes Projekt erhält sein eigenes Speicherverzeichnis unter `~/.claude/projects/<project>/memory/`. Der Pfad `<project>` wird aus dem Git-Repository abgeleitet, sodass alle Worktrees und Unterverzeichnisse innerhalb desselben Repos ein Auto-Memory-Verzeichnis teilen. Außerhalb eines Git-Repos wird stattdessen das Projektstammverzeichnis verwendet.

Um Auto-Memory an einem anderen Ort zu speichern, setzen Sie `autoMemoryDirectory` in Ihren Benutzer- oder lokalen Einstellungen:

```
{  
  "autoMemoryDirectory": "~/my-custom-memory-dir"  
}
```

Diese Einstellung wird von Richtlinien-, lokalen und Benutzereinstellungen akzeptiert. Sie wird nicht von Projekteinstellungen (`.claude/settings.json`) akzeptiert, um zu verhindern, dass ein gemeinsames Projekt Auto-Memory-Schreibvorgänge an sensible Orte umleitet.

Das Verzeichnis enthält einen `MEMORY.md`-Einstiegspunkt und optionale Themadateien:

```
~/ .claude/projects/<project>/memory/  
├─ MEMORY.md           # Prägnanter Index, geladen in jede Sitzung  
├─ debugging.md        # Detaillierte Notizen zu Debugging-Mustern  
├─ api-conventions.md  # API-Designentscheidungen  
└─ ...                 # Alle anderen Themadateien, die Claude erstellt
```

`MEMORY.md` fungiert als Index des Speicherverzeichnisses. Claude liest und schreibt Dateien in diesem Verzeichnis während Ihrer Sitzung und verwendet `MEMORY.md`, um den Überblick zu behalten, was wo gespeichert ist.

Auto-Memory ist maschinenlokal. Alle Worktrees und Unterverzeichnisse innerhalb desselben Git-Repositories teilen ein Auto-Memory-Verzeichnis. Dateien werden nicht über Maschinen oder Cloud-Umgebungen hinweg geteilt.

Wie es funktioniert

Die ersten 200 Zeilen von `MEMORY.md` werden zu Beginn jeder Konversation geladen. Inhalte über Zeile 200 werden nicht beim Sitzungsstart geladen. Claude hält `MEMORY.md` prägnant, indem es detaillierte Notizen in separate Themadateien verschiebt.

Diese 200-Zeilen-Grenze gilt nur für `MEMORY.md`. `CLAUDE.md`-Dateien werden unabhängig von der Länge vollständig geladen, obwohl kürzere Dateien bessere Einhaltung erzeugen.

Themadateien wie `debugging.md` oder `patterns.md` werden nicht beim Start geladen. Claude liest sie bei Bedarf mit seinen Standard-Datei-Tools, wenn es die Informationen benötigt.

Claude liest und schreibt Speicherdateien während Ihrer Sitzung. Wenn Sie „Writing memory“ oder „Recalled memory“ in der Claude Code-Schnittstelle sehen, aktualisiert oder liest Claude aktiv aus `~/claude/projects/<project>/memory/`.

Überprüfen und bearbeiten Sie Ihr Speicher

Auto-Memory-Dateien sind einfaches Markdown, das Sie jederzeit bearbeiten oder löschen können. Führen Sie `/memory` aus, um Speicherdateien aus einer Sitzung heraus zu durchsuchen und zu öffnen.

Anzeigen und bearbeiten mit `/memory`

Der Befehl `/memory` listet alle CLAUDE.md- und Regelsdateien auf, die in Ihrer aktuellen Sitzung geladen sind, ermöglicht es Ihnen, Auto-Memory ein- oder auszuschalten, und bietet einen Link zum Öffnen des Auto-Memory-Ordners. Wählen Sie eine beliebige Datei aus, um sie in Ihrem Editor zu öffnen.

Wenn Sie Claude bitten, sich etwas zu merken, wie „immer pnpm verwenden, nicht npm“ oder „denken Sie daran, dass die API-Tests eine lokale Redis-Instanz erfordern“, speichert Claude es in Auto-Memory. Um Anweisungen stattdessen zu CLAUDE.md hinzuzufügen, bitten Sie Claude direkt, wie „fügen Sie dies zu CLAUDE.md hinzu“, oder bearbeiten Sie die Datei selbst über `/memory`.

Fehlerbehebung bei Speicherproblemen

Dies sind die häufigsten Probleme mit CLAUDE.md und Auto-Memory, zusammen mit Schritten zum Debuggen.

Claude folgt meiner CLAUDE.md nicht

CLAUDE.md ist Kontext, keine Erzwingung. Claude liest es und versucht, ihm zu folgen, aber es gibt keine Garantie für strikte Einhaltung, besonders bei vagen oder widersprüchlichen Anweisungen.

Zum Debuggen:

- Führen Sie `/memory` aus, um zu überprüfen, dass Ihre CLAUDE.md-Dateien geladen werden. Wenn eine Datei nicht aufgelistet ist, kann Claude sie nicht sehen.
- Überprüfen Sie, dass die relevante CLAUDE.md an einem Ort ist, der für Ihre Sitzung geladen wird (siehe [Wählen Sie, wo Sie CLAUDE.md-Dateien ablegen](#)).
- Machen Sie Anweisungen spezifischer. „Verwenden Sie 2-Leerzeichen-Einrückung“ funktioniert besser als „Formatieren Sie Code schön“.

- Suchen Sie nach widersprüchlichen Anweisungen über CLAUDE.md-Dateien hinweg. Wenn zwei Dateien unterschiedliche Anleitungen für das gleiche Verhalten geben, kann Claude eine willkürlich auswählen.

Tip:

Verwenden Sie den [InstructionsLoaded -Hook](#), um genau zu protokollieren, welche Anweisungsdateien geladen sind, wann sie geladen werden und warum. Dies ist nützlich zum Debuggen pfadspezifischer Regeln oder träge geladener Dateien in Unterverzeichnissen.

Ich weiß nicht, was Auto-Memory gespeichert hat

Führen Sie `/memory` aus und wählen Sie den Auto-Memory-Ordner aus, um zu durchsuchen, was Claude gespeichert hat. Alles ist einfaches Markdown, das Sie lesen, bearbeiten oder löschen können.

Meine CLAUDE.md ist zu groß

Dateien über 200 Zeilen verbrauchen mehr Kontext und können die Einhaltung reduzieren. Verschieben Sie detaillierte Inhalte in separate Dateien, auf die mit `@path`-Importen verwiesen wird (siehe [Importieren Sie zusätzliche Dateien](#)), oder teilen Sie Ihre Anweisungen über `.claude/rules/`-Dateien auf.

Anweisungen scheinen nach `/compact` verloren zu gehen

CLAUDE.md übersteht Komprimierung vollständig. Nach `/compact` liest Claude Ihre CLAUDE.md neu von der Festplatte und injiziert sie frisch in die Sitzung. Wenn eine Anweisung nach der Komprimierung verschwunden ist, wurde sie nur in der Konversation gegeben, nicht in CLAUDE.md geschrieben. Fügen Sie sie zu CLAUDE.md hinzu, um sie über Sitzungen hinweg zu erhalten.

Weitere Informationen finden Sie unter [Schreiben Sie effektive Anweisungen](#) für Anleitungen zu Größe, Struktur und Spezifität.

Verwandte Ressourcen

- [Skills](#): Verpacken Sie wiederholbare Workflows, die bei Bedarf geladen werden
- [Einstellungen](#): Konfigurieren Sie das Claude Code-Verhalten mit Einstellungsdateien
- [Verwalten Sie Sitzungen](#): Verwalten Sie Kontext, setzen Sie Konversationen fort und führen Sie parallele Sitzungen aus
- [Subagent-Speicher](#): Lassen Sie Subagents ihre eigene Auto-Memory pflegen

Claude Code-Einstellungen

Konfigurieren Sie Claude Code mit globalen und projektbezogenen Einstellungen sowie Umgebungsvariablen.

Claude Code bietet eine Vielzahl von Einstellungen, um sein Verhalten an Ihre Anforderungen anzupassen. Sie können Claude Code konfigurieren, indem Sie den Befehl `/config` in der interaktiven REPL ausführen, wodurch eine Einstellungsoberfläche mit Registerkarten geöffnet wird, auf der Sie Statusinformationen anzeigen und Konfigurationsoptionen ändern können.

Konfigurationsbereiche

Claude Code verwendet ein **Bereichssystem**, um zu bestimmen, wo Konfigurationen gelten und wer sie teilt. Das Verständnis von Bereichen hilft Ihnen zu entscheiden, wie Sie Claude Code für persönliche Nutzung, Teamzusammenarbeit oder Unternehmensbereitstellung konfigurieren.

Verfügbare Bereiche

Bereich	Speicherort	Wer ist betroffen	Mit Team geteilt?
Verwaltet	Serververwaltete Einstellungen, plist / Registry oder systemweite <code>managed-settings.json</code>	Alle Benutzer auf dem Computer	Ja (von IT bereitgestellt)
Benutzer	<code>~/.claude/</code> Verzeichnis	Sie, über alle Projekte hinweg	Nein
Projekt	<code>.claude/</code> im Repository	Alle Mitarbeiter in diesem Repository	Ja (in Git eingecheckt)
Lokal	<code>.claude/settings.local.json</code>	Sie, nur in diesem Repository	Nein (gitignored)

Wann Sie jeden Bereich verwenden sollten

Der **Verwaltungsbereich** ist für:

- Sicherheitsrichtlinien, die organisationsweit durchgesetzt werden müssen
- Compliance-Anforderungen, die nicht überschrieben werden können

- Standardisierte Konfigurationen, die von IT/DevOps bereitgestellt werden

Der **Benutzerbereich** ist am besten für:

- Persönliche Voreinstellungen, die Sie überall haben möchten (Designs, Editor-Einstellungen)
- Tools und Plugins, die Sie in allen Projekten verwenden
- API-Schlüssel und Authentifizierung (sicher gespeichert)

Der **Projektbereich** ist am besten für:

- Teamübergreifende Einstellungen (Berechtigungen, Hooks, MCP-Server)
- Plugins, die das ganze Team haben sollte
- Standardisierung von Tools über Mitarbeiter hinweg

Der **lokale Bereich** ist am besten für:

- Persönliche Übersreibungen für ein bestimmtes Projekt
- Testen von Konfigurationen vor dem Teilen mit dem Team
- Maschinenspezifische Einstellungen, die für andere nicht funktionieren

Wie Bereiche interagieren

Wenn die gleiche Einstellung in mehreren Bereichen konfiguriert ist, haben spezifischere Bereiche Vorrang:

1. **Verwaltet** (höchste) - kann von nichts überschrieben werden
2. **Befehlszeilenargumente** - temporäre Sitzungsübersreibungen
3. **Lokal** - überschreibt Projekt- und Benutzereinstellungen
4. **Projekt** - überschreibt Benutzereinstellungen
5. **Benutzer** (niedrigste) - gilt, wenn nichts anderes die Einstellung angibt

Wenn beispielsweise eine Berechtigung in Benutzereinstellungen erlaubt ist, aber in Projekteinstellungen verweigert wird, hat die Projekteinstellung Vorrang und die Berechtigung wird blockiert.

Was Bereiche verwendet

Bereiche gelten für viele Claude Code-Funktionen:

Funktion	Benutzerort	Projektort	Lokaler Ort
Einstellungen	<code>~/.claude/settings.json</code>	<code>.claude/settings.json</code>	<code>.claude/settings.local.json</code>
Sub-agents	<code>~/.claude/agents/</code>	<code>.claude/agents/</code>	—
MCP-Server	<code>~/.claude.json</code>	<code>.mcp.json</code>	<code>~/.claude.json</code> (pro Projekt)
Plugins	<code>~/.claude/settings.json</code>	<code>.claude/settings.json</code>	<code>.claude/settings.local.json</code>
CLAUDE.md	<code>~/.claude/CLAUDE.md</code>	<code>CLAUDE.md</code> oder <code>.claude/CLAUDE.md</code>	—

Einstellungsdateien

Die `settings.json` Datei ist unser offizieller Mechanismus zum Konfigurieren von Claude Code durch hierarchische Einstellungen:

- **Benutzereinstellungen** werden in `~/.claude/settings.json` definiert und gelten für alle Projekte.
- **Projekteinstellungen** werden in Ihrem Projektverzeichnis gespeichert:
 - `.claude/settings.json` für Einstellungen, die in die Versionskontrolle eingecheckt und mit Ihrem Team geteilt werden
 - `.claude/settings.local.json` für Einstellungen, die nicht eingecheckt werden, nützlich für persönliche Voreinstellungen und Experimente. Claude Code konfiguriert Git so, dass `.claude/settings.local.json` ignoriert wird, wenn sie erstellt wird.
- **Verwaltete Einstellungen:** Für Organisationen, die zentrale Kontrolle benötigen, unterstützt Claude Code mehrere Bereitstellungsmechanismen für verwaltete Einstellungen. Alle verwenden das gleiche JSON-Format und können nicht durch Benutzer- oder Projekteinstellungen überschrieben werden:
 - **Serververwaltete Einstellungen:** von Anthropic Servern über die Claude.ai Admin-Konsole bereitgestellt. Siehe [serververwaltete Einstellungen](#).

- **MDM/OS-Richtlinien:** über native Geräteverwaltung auf macOS und Windows bereitgestellt:
- macOS: `com.anthropic.claudecode` verwaltete Präferenzen-Domain (bereitgestellt über Konfigurationsprofile in Jamf, Kandji oder anderen MDM-Tools)
- Windows: `HKLM\SOFTWARE\Policies\ClaudeCode` Registry-Schlüssel mit einem `Settings` Wert (REG_SZ oder REG_EXPAND_SZ) mit JSON (bereitgestellt über Gruppenrichtlinie oder Intune)
- Windows (Benutzerebene): `HKCU\SOFTWARE\Policies\ClaudeCode` (niedrigste Richtlinienpriorität, wird nur verwendet, wenn keine Admin-Quelle vorhanden ist)
- **Dateibasiert:** `managed-settings.json` und `managed-mcp.json` in Systemverzeichnissen bereitgestellt:
- macOS: `/Library/Application Support/ClaudeCode/`
- Linux und WSL: `/etc/claude-code/`
- Windows: `C:\Program Files\ClaudeCode\`

Siehe [verwaltete Einstellungen](#) und [Verwaltete MCP-Konfiguration](#) für Details.

Note:

Verwaltete Bereitstellungen können auch **Plugin-Marketplace-Ergänzungen** mit `strictKnownMarketplaces` einschränken. Weitere Informationen finden Sie unter [Verwaltete Marketplace-Einschränkungen](#).

- **Andere Konfiguration** wird in `~/.claude.json` gespeichert. Diese Datei enthält Ihre Voreinstellungen (Design, Benachrichtigungseinstellungen, Editor-Modus), OAuth-Sitzung, [MCP-Server](#) Konfigurationen für Benutzer- und lokale Bereiche, projektbezogene Zustände (zulässige Tools, Vertrauenseinstellungen) und verschiedene Caches. Projektbezogene MCP-Server werden separat in `.mcp.json` gespeichert.

Note:

Claude Code erstellt automatisch zeitgestempelte Sicherungen von Konfigurationsdateien und behält die fünf neuesten Sicherungen bei, um Datenverlust zu verhindern.

```

{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "allow": [
      "Bash(npm run lint)",
      "Bash(npm run test *)",
      "Read(~/.zshrc)"
    ],
    "deny": [
      "Bash(curl *)",
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)"
    ]
  },
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_METRICS_EXPORTER": "otlp"
  },
  "companyAnnouncements": [
    "Welcome to Acme Corp! Review our code guidelines at docs.acme.com",
    "Reminder: Code reviews required for all PRs",
    "New security policy in effect"
  ]
}

```

Die `$schema` Zeile im obigen Beispiel verweist auf das [offizielle JSON-Schema](https://json.schemastore.org/claude-code-settings.json) für Claude Code-Einstellungen. Das Hinzufügen zu Ihrer `settings.json` ermöglicht Autovervollständigung und Inline-Validierung in VS Code, Cursor und jedem anderen Editor, der JSON-Schema-Validierung unterstützt.

Verfügbare Einstellungen

`settings.json` unterstützt eine Reihe von Optionen:

Schlüssel	Beschreibung	Beispiel
<code>apiKeyHelper</code>	Benutzerdefiniertes Skript, das in <code>/bin/sh</code> ausgeführt werden soll, um einen Auth-Wert zu generieren. Dieser Wert wird als <code>X-Api-Key</code> und <code>Authorization: Bearer</code> Header für Modellanfragen gesendet	<code>/bin/ generate_temp_api_key .sh</code>
<code>cleanupPeriodDays</code>	Sitzungen, die länger als dieser Zeitraum inaktiv sind, werden beim Start gelöscht. Wenn auf <code>0</code> gesetzt, werden alle Sitzungen sofort gelöscht. (Standard: 30 Tage)	<code>20</code>
<code>companyAnnouncements</code>	Ankündigung, die Benutzern beim Start angezeigt wird. Wenn mehrere Ankündigungen bereitgestellt werden, werden sie zufällig durchlaufen.	<code>["Welcome to Acme Corp! Review our code guidelines at docs.acme.com"]</code>
<code>env</code>	Umgebungsvariablen, die auf jede Sitzung angewendet werden	<code>{"F00": "bar"}</code>
<code>attribution</code>	Passen Sie die Zuschreibung für Git-Commits und Pull Requests an. Siehe Zuschreibungseinstellungen	<code>{"commit": " Generated with Claude Code", "pr": ""}</code>
<code>includeCoAuthoredBy</code>	Veraltet: Verwenden Sie stattdessen <code>attribution</code> . Ob die <code>co-authored-by Claude</code> Byline in Git-Commits und Pull Requests einbezogen werden soll (Standard: <code>true</code>)	<code>false</code>
<code>includeGitInstructions</code>	Integrierte Commit- und PR-Workflow-Anweisungen in Claudes System-Prompt einbeziehen (Standard: <code>true</code>). Auf <code>false</code> setzen, um diese Anweisungen zu entfernen, z. B. wenn Sie Ihre eigenen Git-Workflow-Skills verwenden. Die Umgebungsvariable <code>CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS</code> hat Vorrang vor dieser Einstellung, wenn gesetzt	<code>false</code>

Schlüssel	Beschreibung	Beispiel
<code>permissions</code>	Siehe Tabelle unten für die Struktur der Berechtigungen.	
<code>hooks</code>	Konfigurieren Sie benutzerdefinierte Befehle, die bei Lebenszyklusereignissen ausgeführt werden. Siehe Hooks-Dokumentation für das Format	Siehe Hooks
<code>disableAllHooks</code>	Deaktivieren Sie alle Hooks und alle benutzerdefinierten Statuszeilen	<code>true</code>
<code>allowManagedHooksOnly</code>	(Nur verwaltete Einstellungen) Verhindern Sie das Laden von Benutzer-, Projekt- und Plugin-Hooks. Erlaubt nur verwaltete Hooks und SDK-Hooks. Siehe Hook-Konfiguration	<code>true</code>
<code>allowedHttpHookUrls</code>	Whitelist von URL-Mustern, auf die HTTP-Hooks abzielen können. Unterstützt <code>*</code> als Wildcard. Wenn gesetzt, werden Hooks mit nicht übereinstimmenden URLs blockiert. Undefined = keine Einschränkung, leeres Array = alle HTTP-Hooks blockieren. Arrays werden über Einstellungsquellen zusammengeführt. Siehe Hook-Konfiguration	<code>["https://hooks.example.com/*"]</code>
<code>httpHookAllowedEnvVars</code>	Whitelist von Umgebungsvariablennamen, die HTTP-Hooks in Header interpolieren können. Wenn gesetzt, ist die effektive <code>allowedEnvVars</code> jedes Hooks der Schnittpunkt mit dieser Liste. Undefined = keine Einschränkung. Arrays werden über Einstellungsquellen zusammengeführt. Siehe Hook-Konfiguration	<code>["MY_TOKEN", "HOOK_SECRET"]</code>

Schlüssel	Beschreibung	Beispiel
<code>allowManagedPermissionsRulesOnly</code>	(Nur verwaltete Einstellungen) Verhindern Sie, dass Benutzer- und Projekteinstellungen <code>allow</code> , <code>ask</code> oder <code>deny</code> Berechtigungsregeln definieren. Nur Regeln in verwalteten Einstellungen gelten. Siehe Nur verwaltete Einstellungen	<code>true</code>
<code>allowManagedMcpServersOnly</code>	(Nur verwaltete Einstellungen) Nur <code>allowedMcpServers</code> aus verwalteten Einstellungen werden berücksichtigt. <code>deniedMcpServers</code> wird weiterhin aus allen Quellen zusammengeführt. Benutzer können weiterhin MCP-Server hinzufügen, aber nur die von Admin definierten Allowlist gilt. Siehe Verwaltete MCP-Konfiguration	<code>true</code>
<code>model</code>	Überschreiben Sie das Standard-Modell für Claude Code	<code>"claude-sonnet-4-6"</code>
<code>availableModels</code>	Beschränken Sie, welche Modelle Benutzer über <code>/model</code> , <code>--model</code> , Config-Tool oder <code>ANTHROPIC_MODEL</code> auswählen können. Beeinflusst nicht die Standard-Option. Siehe Modellauswahl einschränken	<code>["sonnet", "haiku"]</code>
<code>modelOverrides</code>	Ordnen Sie Anthropic-Modell-IDs Anbieter-spezifischen Modell-IDs wie Bedrock-Inferenzprofil-ARNs zu. Jeder Modellwähler-Eintrag verwendet seinen zugeordneten Wert beim Aufrufen der Anbieter-API. Siehe Modell-IDs pro Version überschreiben	<code>{"claude-opus-4-6": "arn:aws:bedrock:..."}</code>
<code>otelHeadersHelper</code>	Skript zum Generieren dynamischer OpenTelemetry-Header. Wird beim Start und regelmäßig ausgeführt (siehe Dynamische Header)	<code>/bin/generate_otel_headers.sh</code>

Schlüssel	Beschreibung	Beispiel
<code>statusLine</code>	Konfigurieren Sie eine benutzerdefinierte Statuszeile zur Anzeige von Kontext. Siehe statusLine Dokumentation	<pre>{"type": "command", "command": "~/ .claude/ statusline.sh"}</pre>
<code>fileSuggestion</code>	Konfigurieren Sie ein benutzerdefiniertes Skript für @ Datei-Autovervollständigung. Siehe Dateivorschlag-Einstellungen	<pre>{"type": "command", "command": "~/ .claude/file- suggestion.sh"}</pre>
<code>respectGitignore</code>	Steuern Sie, ob der @ Datei-Picker <code>.gitignore</code> Muster respektiert. Wenn <code>true</code> (Standard), werden Dateien, die <code>.gitignore</code> Mustern entsprechen, von Vorschlägen ausgeschlossen	<code>false</code>
<code>outputStyle</code>	Konfigurieren Sie einen Ausgabestil, um den System-Prompt anzupassen. Siehe Ausgabestil-Dokumentation	<code>"Explanatory"</code>
<code>forceLoginMethod</code>	Verwenden Sie <code>claudeai</code> , um die Anmeldung auf Claude.ai-Konten zu beschränken, <code>console</code> , um die Anmeldung auf Claude Console (API-Nutzungsabrechnung) Konten zu beschränken	<code>claudeai</code>
<code>forceLoginOrgUUID</code>	Geben Sie die UUID einer Organisation an, um sie während der Anmeldung automatisch auszuwählen und den Organisationsauswahlschritt zu umgehen. Erfordert, dass <code>forceLoginMethod</code> gesetzt ist	<code>"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"</code>
<code>enableAllProjectMcpServers</code>	Genehmigen Sie automatisch alle MCP-Server, die in Projekt <code>.mcp.json</code> Dateien definiert sind	<code>true</code>
<code>enabledMcpjsonServers</code>	Liste spezifischer MCP-Server aus <code>.mcp.json</code> Dateien zum Genehmigen	<code>["memory", "github"]</code>

Schlüssel	Beschreibung	Beispiel
<code>disabledMcpjsonServers</code>	Liste spezifischer MCP-Server aus <code>.mcp.json</code> Dateien zum Ablehnen	<code>["filesystem"]</code>
<code>allowedMcpServers</code>	Wenn in <code>managed-settings.json</code> gesetzt, Whitelist von MCP-Servern, die Benutzer konfigurieren können. Undefined = keine Einschränkungen, leeres Array = Lockdown. Gilt für alle Bereiche. Denylist hat Vorrang. Siehe Verwaltete MCP-Konfiguration	<code>[{ "serverName": "github" }]</code>
<code>deniedMcpServers</code>	Wenn in <code>managed-settings.json</code> gesetzt, Denylist von MCP-Servern, die explizit blockiert sind. Gilt für alle Bereiche einschließlich verwalteter Server. Denylist hat Vorrang vor Allowlist. Siehe Verwaltete MCP-Konfiguration	<code>[{ "serverName": "filesystem" }]</code>
<code>strictKnownMarketplaces</code>	Wenn in <code>managed-settings.json</code> gesetzt, Whitelist von Plugin-Marketplaces, die Benutzer hinzufügen können. Undefined = keine Einschränkungen, leeres Array = Lockdown. Gilt nur für Marketplace-Ergänzungen. Siehe Verwaltete Marketplace-Einschränkungen	<code>[{ "source": "github", "repo": "acme-corp/plugins" }]</code>
<code>blockedMarketplaces</code>	(Nur verwaltete Einstellungen) Blocklist von Marketplace-Quellen. Blockierte Quellen werden vor dem Download überprüft, sodass sie das Dateisystem nie berühren. Siehe Verwaltete Marketplace-Einschränkungen	<code>[{ "source": "github", "repo": "untrusted/plugins" }]</code>

Schlüssel	Beschreibung	Beispiel
<code>pluginTrustMessage</code>	(Nur verwaltete Einstellungen) Benutzerdefinierte Nachricht, die der vor der Installation angezeigten Plug-in-Vertrauenswarnung angehängt wird. Verwenden Sie dies, um organisationsspezifischen Kontext hinzuzufügen, z. B. um zu bestätigen, dass Plugins aus Ihrem internen Marketplace überprüft sind.	<code>"All plugins from our marketplace are approved by IT"</code>
<code>awsAuthRefresh</code>	Benutzerdefiniertes Skript, das das <code>.aws</code> Verzeichnis ändert (siehe erweiterte Anmeldedaten-Konfiguration)	<code>aws sso login --profile myprofile</code>
<code>awsCredentialExport</code>	Benutzerdefiniertes Skript, das JSON mit AWS-Anmeldedaten ausgibt (siehe erweiterte Anmeldedaten-Konfiguration)	<code>/bin/generate_aws_grant.sh</code>
<code>alwaysThinkingEnabled</code>	Aktivieren Sie erweitertes Denken standardmäßig für alle Sitzungen. Normalerweise über den Befehl <code>/config</code> konfiguriert, anstatt direkt zu bearbeiten	<code>true</code>
<code>plansDirectory</code>	Passen Sie an, wo Plandateien gespeichert werden. Der Pfad ist relativ zum Projektstamm. Standard: <code>~/.claude/plans</code>	<code>"./plans"</code>
<code>showTurnDuration</code>	Zeigen Sie Nachrichten zur Dauer der Runde nach Antworten an (z. B. "Cooked for 1m 6s"). Auf <code>false</code> setzen, um diese Nachrichten auszublen	<code>true</code>

Schlüssel	Beschreibung	Beispiel
<code>spinnerVerbs</code>	Passen Sie die Aktionsverben an, die im Spinner und in Nachrichten zur Dauer der Runde angezeigt werden. Setzen Sie <code>mode</code> auf <code>"replace"</code> , um nur Ihre Verben zu verwenden, oder <code>"append"</code> , um sie zu den Standardwerten hinzuzufügen	<pre>{ "mode": "append", "verbs": ["Pondering", "Crafting"] }</pre>
<code>language</code>	Konfigurieren Sie Claudes bevorzugte Antwortsprache (z. B. <code>"japanese"</code> , <code>"spanish"</code> , <code>"french"</code>). Claude wird standardmäßig in dieser Sprache antworten	<code>"japanese"</code>
<code>autoUpdatesChannel</code>	Release-Kanal zum Folgen von Updates. Verwenden Sie <code>"stable"</code> für eine Version, die normalerweise etwa eine Woche alt ist und Versionen mit großen Regressionen überspringt, oder <code>"latest"</code> (Standard) für die neueste Version	<code>"stable"</code>
<code>spinnerTipsEnabled</code>	Zeigen Sie Tipps im Spinner an, während Claude arbeitet. Auf <code>false</code> setzen, um Tipps zu deaktivieren (Standard: <code>true</code>)	<code>false</code>
<code>spinnerTipsOverride</code>	Überschreiben Sie Spinner-Tipps mit benutzerdefinierten Strings. <code>tips</code> : Array von Tip-Strings. <code>excludeDefault</code> : wenn <code>true</code> , nur benutzerdefinierte Tipps anzeigen; wenn <code>false</code> oder nicht vorhanden, werden benutzerdefinierte Tipps mit integrierten Tipps zusammengeführt	<pre>{ "excludeDefault": true, "tips": ["Use our internal tool x"] }</pre>
<code>terminalProgressBarEnabled</code>	Aktivieren Sie die Terminal-Fortschrittsleiste, die den Fortschritt in unterstützten Terminals wie Windows Terminal und iTerm2 anzeigt (Standard: <code>true</code>)	<code>false</code>

Schlüssel	Beschreibung	Beispiel
<code>prefersReducedMotion</code>	Reduzieren oder deaktivieren Sie UI-Animationen (Spinner, Shimmer, Flash-Effekte) für Barrierefreiheit	<code>true</code>
<code>fastModePerSessionOptIn</code>	Wenn <code>true</code> , bleibt der schnelle Modus nicht über Sitzungen hinweg bestehen. Jede Sitzung startet mit ausgeschaltetem schnellen Modus und erfordert, dass Benutzer ihn mit <code>/fast</code> aktivieren. Die Voreinstellung des Benutzers für den schnellen Modus wird weiterhin gespeichert. Siehe Opt-in pro Sitzung erforderlich	<code>true</code>
<code>teammateMode</code>	Wie Agent-Team Teamkollegen angezeigt werden: <code>auto</code> (wählt geteilte Fenster in tmux oder iTerm2, ansonsten in-process), <code>in-process</code> oder <code>tmux</code> . Siehe Agent-Teams einrichten	<code>"in-process"</code>

Berechtigungseinstellungen

Schlüssel	Beschreibung	Beispiel
<code>allow</code>	Array von Berechtigungsregeln, um Tool-Nutzung zu erlauben. Siehe Berechtigungsregelsyntax unten für Details zum Musterabgleich	<code>["Bash(git diff *)"]</code>
<code>ask</code>	Array von Berechtigungsregeln, um bei Tool-Nutzung um Bestätigung zu fragen. Siehe Berechtigungsregelsyntax unten	<code>["Bash(git push *)"]</code>
<code>deny</code>	Array von Berechtigungsregeln, um Tool-Nutzung zu verweigern. Verwenden Sie dies, um sensible Dateien vom Claude Code-Zugriff auszuschließen. Siehe Berechtigungsregelsyntax und Bash-Berechtigungsbeschränkungen	<code>["WebFetch", "Bash(curl *)", "Read(./env)", "Read(./secrets/**)"]</code>

Schlüssel	Beschreibung	Beispiel
<code>additionalDirectories</code>	Zusätzliche Arbeitsverzeichnisse , auf die Claude Zugriff hat	<code>["../docs/"]</code>
<code>defaultMode</code>	Standard- Berechtigungsmodus beim Öffnen von Claude Code	<code>"acceptEdits"</code>
<code>disableBypassPermissionsMode</code>	Auf <code>"disable"</code> setzen, um zu verhindern, dass der <code>bypassPermissions</code> Modus aktiviert wird. Dies deaktiviert das Befehlszeilenargument <code>--dangerously-skip-permissions</code> . Siehe verwaltete Einstellungen	<code>"disable"</code>

Berechtigungsregelsyntax

Berechtigungsregeln folgen dem Format `Tool` oder `Tool(specifier)`. Regeln werden in Reihenfolge ausgewertet: zuerst Deny-Regeln, dann Ask, dann Allow. Die erste übereinstimmende Regel gewinnt.

Schnelle Beispiele:

Regel	Effekt
<code>Bash</code>	Passt auf alle Bash-Befehle
<code>Bash(npm run *)</code>	Passt auf Befehle, die mit <code>npm run</code> beginnen
<code>Read(./env)</code>	Passt auf das Lesen der <code>.env</code> Datei
<code>WebFetch(domain:example.com)</code>	Passt auf Abrufanfragen an example.com

Für die vollständige Referenz der Regelsyntax, einschließlich Wildcard-Verhalten, Tool-spezifischer Muster für Read, Edit, WebFetch, MCP und Agent-Regeln sowie Sicherheitsbeschränkungen von Bash-Mustern, siehe [Berechtigungsregelsyntax](#).

Sandbox-Einstellungen

Konfigurieren Sie erweitertes Sandbox-Verhalten. Sandboxing isoliert Bash-Befehle von Ihrem Dateisystem und Netzwerk. Siehe [Sandboxing](#) für Details.

Schlüssel	Beschreibung	Beispiel
<code>enabled</code>	Aktivieren Sie Bash-Sandboxing (macOS, Linux und WSL2). Standard: false	<code>true</code>
<code>autoAllowBashIfSandboxed</code>	Genehmigen Sie Bash-Befehle automatisch, wenn sie in einer Sandbox ausgeführt werden. Standard: true	<code>true</code>
<code>excludedCommands</code>	Befehle, die außerhalb der Sandbox ausgeführt werden sollten	<code>["git", "docker"]</code>
<code>allowUnsandboxedCommands</code>	Erlauben Sie Befehlen, außerhalb der Sandbox über den Parameter <code>dangerouslyDisableSandbox</code> auszuführen. Wenn auf <code>false</code> gesetzt, ist die Escape-Hatch <code>dangerouslyDisableSandbox</code> vollständig deaktiviert und alle Befehle müssen in einer Sandbox ausgeführt werden (oder in <code>excludedCommands</code> sein). Nützlich für Unternehmensrichtlinien, die striktes Sandboxing erfordern. Standard: true	<code>false</code>
<code>filesystem.allowWrite</code>	Zusätzliche Pfade, in die Sandbox-Befehle schreiben können. Arrays werden über alle Einstellungsbereiche zusammengeführt: Benutzer-, Projekt- und verwaltete Pfade werden kombiniert, nicht ersetzt. Auch zusammengeführt mit Pfaden aus <code>Edit(...)</code> Berechtigungsregeln. Siehe Pfad-Präfixe unten.	<code>["//tmp/build", "~/.kube"]</code>
<code>filesystem.denyWrite</code>	Pfade, in die Sandbox-Befehle nicht schreiben können. Arrays werden über alle Einstellungsbereiche zusammengeführt. Auch zusammengeführt mit Pfaden aus <code>Edit(...)</code> Berechtigungsregeln.	<code>["//etc", "//usr/local/bin"]</code>

Schlüssel	Beschreibung	Beispiel
<code>filesystem.denyRead</code>	Pfade, aus denen Sandbox-Befehle nicht lesen können. Arrays werden über alle Einstellungsbereiche zusammgeführt. Auch zusammengeführt mit Pfaden aus <code>Read(...)</code> Berechtigungsregeln.	<code>["~/ .aws/credentials"]</code>
<code>network.allowUnixSockets</code>	Unix-Socket-Pfade, auf die in der Sandbox zugegriffen werden kann (für SSH-Agenten usw.)	<code>["~/ .ssh/agent-socket"]</code>
<code>network.allowAllUnixSockets</code>	Erlauben Sie alle Unix-Socket-Verbindungen in der Sandbox. Standard: false	<code>true</code>
<code>network.allowLocalBinding</code>	Erlauben Sie das Binden an Localhost-Ports (nur macOS). Standard: false	<code>true</code>
<code>network.allowedDomains</code>	Array von Domänen, um ausgehenden Netzwerkverkehr zu erlauben. Unterstützt Wildcards (z. B. <code>*.example.com</code>).	<code>["github.com", "*.npmjs.org"]</code>
<code>network.allowManagedDomainsOnly</code>	(Nur verwaltete Einstellungen) Nur <code>allowedDomains</code> und <code>WebFetch(domain: ...)</code> Berechtigungsregeln aus verwalteten Einstellungen werden berücksichtigt. Domänen aus Benutzer-, Projekt- und lokalen Einstellungen werden ignoriert. Nicht zulässige Domänen werden automatisch blockiert, ohne den Benutzer zu fragen. Verweigerte Domänen werden weiterhin aus allen Quellen berücksichtigt. Standard: false	<code>true</code>
<code>network.httpProxyPort</code>	HTTP-Proxy-Port, der verwendet wird, wenn Sie Ihren eigenen Proxy verwenden möchten. Wenn nicht angegeben, führt Claude seinen eigenen Proxy aus.	<code>8080</code>

Schlüssel	Beschreibung	Beispiel
<code>network.socksProxyPort</code>	SOCKS5-Proxy-Port, der verwendet wird, wenn Sie Ihren eigenen Proxy verwenden möchten. Wenn nicht angegeben, führt Claude seinen eigenen Proxy aus.	<code>8081</code>
<code>enableWeakerNestedSandbox</code>	Aktivieren Sie schwächere Sandbox für unprivilegierte Docker-Umgebungen (nur Linux und WSL2). Reduziert Sicherheit. Standard: false	<code>true</code>
<code>enableWeakerNetworkIsolation</code>	(Nur macOS) Erlauben Sie den Zugriff auf den System-TLS-Vertrauensdienst (<code>com.apple.trustd.agent</code>) in der Sandbox. Erforderlich für Go-basierte Tools wie <code>gh</code> , <code>gcloud</code> und <code>terraform</code> , um TLS-Zertifikate zu überprüfen, wenn <code>httpProxyPort</code> mit einem MITM-Proxy und benutzerdefinierter CA verwendet wird. Reduziert Sicherheit durch Öffnen eines möglichen Datenexfiltrationspfads. Standard: false	<code>true</code>

Sandbox-Pfad-Präfixe

Pfade in `filesystem.allowWrite`, `filesystem.denyWrite` und `filesystem.denyRead` unterstützen diese Präfixe:

Präfix	Bedeutung	Beispiel
<code>//</code>	Absoluter Pfad vom Dateisystem-Root	<code>//tmp/build</code> wird zu <code>/tmp/build</code>
<code>~/</code>	Relativ zum Home-Verzeichnis	<code>~/kube</code> wird zu <code>\$HOME/.kube</code>
<code>/</code>	Relativ zum Verzeichnis der Einstellungsdatei	<code>/build</code> wird zu <code>\$SETTINGS_DIR/build</code>

Präfix	Bedeutung	Beispiel
<code>./</code> oder kein Präfix	Relativer Pfad (aufgelöst durch Sandbox-Laufzeit)	<code>./output</code>

Konfigurationsbeispiel:

```
{
  "sandbox": {
    "enabled": true,
    "autoAllowBashIfSandboxed": true,
    "excludedCommands": ["docker"],
    "filesystem": {
      "allowWrite": ["//tmp/build", "~/kubernetes"],
      "denyRead": ["~/aws/credentials"]
    },
    "network": {
      "allowedDomains": ["github.com", "*.npmjs.org", "registry.yarnpkg.com"],
      "allowUnixSockets": [
        "/var/run/docker.sock"
      ],
      "allowLocalBinding": true
    }
  }
}
```

Dateisystem- und Netzwerkbeschränkungen können auf zwei Arten konfiguriert werden, die zusammengeführt werden:

- **sandbox.filesystem Einstellungen** (oben gezeigt): Steuern Sie Pfade an der OS-Sandbox-Grenze. Diese Einschränkungen gelten für alle Subprozess-Befehle (z. B. `kubectl`, `terraform`, `npm`), nicht nur für Claudes Datei-Tools.
- **Berechtigungsregeln:** Verwenden Sie `Edit` Allow/Deny-Regeln, um Claude's Datei-Tool-Zugriff zu steuern, `Read` Deny-Regeln, um Lesevorgänge zu blockieren, und `WebFetch` Allow/Deny-Regeln, um Netzwerk-Domänen zu steuern. Pfade aus diesen Regeln werden auch in die Sandbox-Konfiguration zusammengeführt.

Zuschreibungseinstellungen

Claude Code fügt Zuschreibung zu Git-Commits und Pull Requests hinzu. Diese werden separat konfiguriert:

- Commits verwenden [Git-Trailer](#) (wie `Co-Authored-By`) standardmäßig, die angepasst oder deaktiviert werden können
- Pull-Request-Beschreibungen sind Klartext

Schlüssel	Beschreibung
commit	Zuschreibung für Git-Commits, einschließlich aller Trailer. Leerer String verbirgt Commit-Zuschreibung
pr	Zuschreibung für Pull-Request-Beschreibungen. Leerer String verbirgt Pull-Request-Zuschreibung

Standard-Commit-Zuschreibung:

 Generated with [Claude Code](https://claude.com/claude-code)

Co-Authored-By: Claude Sonnet 4.6 <noreply@anthropic.com>

Standard-Pull-Request-Zuschreibung:

 Generated with [Claude Code](https://claude.com/claude-code)

Beispiel:

```
{
  "attribution": {
    "commit": "Generated with AI\n\nCo-Authored-By: AI <ai@example.com>",
    "pr": ""
  }
}
```

Note:

Die `attribution` Einstellung hat Vorrang vor der veralteten `includeCoAuthoredBy` Einstellung. Um alle Zuschreibungen auszublenden, setzen Sie `commit` und `pr` auf leere Strings.

Dateivorschlag-Einstellungen

Konfigurieren Sie einen benutzerdefinierten Befehl für `@` Dateipath-Autovervollständigung. Der integrierte Dateivorschlag verwendet schnelle Dateisystem-Durchquerung, aber große Monorepos können von projektspezifischer Indizierung wie einem vorgebauten Dateiindex oder benutzerdefinierten Tools profitieren.

```
{
  "fileSuggestion": {
    "type": "command",
    "command": "~/ .claude/file-suggestion.sh"
  }
}
```

Der Befehl wird mit den gleichen Umgebungsvariablen wie [Hooks](#) ausgeführt, einschließlich `CLAUDE_PROJECT_DIR`. Er empfängt JSON über stdin mit einem `query` Feld:

```
{"query": "src/comp"}
```

Geben Sie zeilengetrennte Dateipfade zu stdout aus (derzeit auf 15 begrenzt):

```
src/components/Button.tsx
src/components/Modal.tsx
src/components/Form.tsx
```

Beispiel:

```
#!/bin/bash
query=$(cat | jq -r '.query')
your-repo-file-index --query "$query" | head -20
```

Hook-Konfiguration

Diese Einstellungen steuern, welche Hooks ausgeführt werden dürfen und worauf HTTP-Hooks zugreifen können. Die `allowManagedHooksOnly` Einstellung kann nur in [verwalteten Einstellungen](#) konfiguriert werden. Die URL- und Env-Var-Whitelists können auf jeder Einstellungsebene gesetzt werden und werden über Quellen zusammengeführt.

Verhalten, wenn `allowManagedHooksOnly true` ist:

- Verwaltete Hooks und SDK-Hooks werden geladen
- Benutzer-Hooks, Projekt-Hooks und Plugin-Hooks werden blockiert

HTTP-Hook-URLs einschränken:

Begrenzen Sie, auf welche URLs HTTP-Hooks abzielen können. Unterstützt `*` als Wildcard zum Abgleichen. Wenn das Array definiert ist, werden HTTP-Hooks, die auf nicht übereinstimmende URLs abzielen, stillschweigend blockiert.

```
{
  "allowedHttpHookUrls": ["https://hooks.example.com/*", "http://localhost:*"]
}
```

HTTP-Hook-Umgebungsvariablen einschränken:

Begrenzen Sie, welche Umgebungsvariablenamen HTTP-Hooks in Header-Werte interpolieren können. Die effektive `allowedEnvVars` jedes Hooks ist der Schnittpunkt seiner eigenen Liste und dieser Einstellung.

```
{
  "httpHookAllowedEnvVars": ["MY_TOKEN", "HOOK_SECRET"]
}
```

Einstellungspriorität

Einstellungen werden in Prioritätsreihenfolge angewendet. Von höchster zu niedrigster:

1. **Verwaltete Einstellungen** ([serververwaltete](#), [MDM/OS-Richtlinien](#) oder [verwaltete Einstellungen](#))
 - Richtlinien, die von IT über Server-Bereitstellung, MDM-Konfigurationsprofile, Registry-Richtlinien oder verwaltete Einstellungsdateien bereitgestellt werden
 - Können nicht durch andere Ebenen überschrieben werden, einschließlich Befehlszeilenargumenten

- Innerhalb der verwalteten Ebene ist die Priorität: serververwaltete > MDM/OS-Richtlinien > `managed-settings.json` > HKCU Registry (nur Windows). Nur eine verwaltete Quelle wird verwendet; Quellen werden nicht zusammengeführt.

2. Befehlszeilenargumente

- Temporäre Überschreibungen für eine bestimmte Sitzung

3. Lokale Projekteinstellungen (`.claude/settings.local.json`)

- Persönliche projektspezifische Einstellungen

4. Gemeinsame Projekteinstellungen (`.claude/settings.json`)

- Teamübergreifende Projekteinstellungen in der Versionskontrolle

5. Benutzereinstellungen (`~/.claude/settings.json`)

- Persönliche globale Einstellungen

Diese Hierarchie stellt sicher, dass Organisationsrichtlinien immer durchgesetzt werden, während Teams und Einzelpersonen ihre Erfahrung anpassen können.

Wenn beispielsweise Ihre Benutzereinstellungen `Bash(npm run *)` erlauben, aber die gemeinsamen Einstellungen eines Projekts es verweigern, hat die Projekteinstellung Vorrang und der Befehl wird blockiert.

Note:

Array-Einstellungen werden über Bereiche zusammengeführt. Wenn die gleiche Array-wertige Einstellung (wie `sandbox.filesystem.allowWrite` oder `permissions.allow`) in mehreren Bereichen erscheint, werden die Arrays **verkettet und dedupliziert**, nicht ersetzt. Dies bedeutet, dass Bereiche mit niedrigerer Priorität Einträge hinzufügen können, ohne diejenigen mit höherer Priorität zu überschreiben, und umgekehrt. Wenn beispielsweise verwaltete Einstellungen `allowWrite` auf `["//opt/company-tools"]` setzen und ein Benutzer `["~/.kube"]` hinzufügt, sind beide Pfade in der endgültigen Konfiguration enthalten.

Aktive Einstellungen überprüfen

Führen Sie `/status` in Claude Code aus, um zu sehen, welche Einstellungsquellen aktiv sind und woher sie stammen. Die Ausgabe zeigt jede Konfigurationsebene (verwaltet, Benutzer, Projekt) zusammen mit ihrem Ursprung, wie `Enterprise managed settings (remote)`, `Enterprise managed settings (plist)`, `Enterprise managed settings (HKLM)` oder `Enterprise managed settings (file)`. Wenn eine Einstellungsdatei Fehler enthält, meldet `/status` das Problem, damit Sie es beheben können.

Wichtige Punkte zum Konfigurationssystem

- **Memory-Dateien (`CLAUDE.md`)**: Enthalten Anweisungen und Kontext, die Claude beim Start lädt
- **Einstellungsdateien (JSON)**: Konfigurieren Sie Berechtigungen, Umgebungsvariablen und Tool-Verhalten
- **Skills**: Benutzerdefinierte Prompts, die mit `/skill-name` aufgerufen oder von Claude automatisch geladen werden können
- **MCP-Server**: Erweitern Sie Claude Code mit zusätzlichen Tools und Integrationen
- **Priorität**: Höherrangige Konfigurationen (Verwaltet) überschreiben niedrigere (Benutzer/Projekt)
- **Vererbung**: Einstellungen werden zusammengeführt, wobei spezifischere Einstellungen breitere ergänzen oder überschreiben

System-Prompt

Claudes interner System-Prompt wird nicht veröffentlicht. Um benutzerdefinierte Anweisungen hinzuzufügen, verwenden Sie `CLAUDE.md` Dateien oder das Flag `--append-system-prompt`.

Sensible Dateien ausschließen

Um zu verhindern, dass Claude Code auf Dateien mit sensiblen Informationen wie API-Schlüsseln, Geheimnissen und Umgebungsdateien zugreift, verwenden Sie die `permissions.deny` Einstellung in Ihrer `.claude/settings.json` Datei:

```
{
  "permissions": {
    "deny": [
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)",
      "Read(./config/credentials.json)",
      "Read(./build)"
    ]
  }
}
```

Dies ersetzt die veraltete `ignorePatterns` Konfiguration. Dateien, die diesen Mustern entsprechen, werden von der Dateiermittlung und Suchergebnissen ausgeschlossen, und Lesevorgänge auf diesen Dateien werden verweigert.

Subagent-Konfiguration

Claude Code unterstützt benutzerdefinierte KI-Subagents, die auf Benutzer- und Projektebene konfiguriert werden können. Diese Subagents werden als Markdown-Dateien mit YAML-Frontmatter gespeichert:

- **Benutzer-Subagents:** `~/.claude/agents/` - Verfügbar über alle Ihre Projekte
- **Projekt-Subagents:** `.claude/agents/` - Spezifisch für Ihr Projekt und können mit Ihrem Team geteilt werden

Subagent-Dateien definieren spezialisierte KI-Assistenten mit benutzerdefinierten Prompts und Tool-Berechtigungen. Erfahren Sie mehr über das Erstellen und Verwenden von Subagents in der [Subagents-Dokumentation](#).

Plugin-Konfiguration

Claude Code unterstützt ein Plugin-System, das es Ihnen ermöglicht, die Funktionalität mit Skills, Agents, Hooks und MCP-Servern zu erweitern. Plugins werden über Marketplaces verteilt und können auf Benutzer- und Repository-Ebene konfiguriert werden.

Plugin-Einstellungen

Plugin-bezogene Einstellungen in `settings.json` :


```
{
  "enabledPlugins": {
    "formatter@acme-tools": true,
    "deployer@acme-tools": true,
    "analyzer@security-plugins": false
  },
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": "github",
      "repo": "acme-corp/claude-plugins"
    }
  }
}
```

enabledPlugins

Steuert, welche Plugins aktiviert sind. Format: `"plugin-name@marketplace-name": true/false`

Bereiche:

- **Benutzereinstellungen** (`~/.claude/settings.json`): Persönliche Plugin-Voreinstellungen
- **Projekteinstellungen** (`.claude/settings.json`): Projektspezifische Plugins, die mit dem Team geteilt werden
- **Lokale Einstellungen** (`.claude/settings.local.json`): Pro-Maschinen-Überschreibungen (nicht eingecheckt)

Beispiel:

```
{
  "enabledPlugins": {
    "code-formatter@team-tools": true,
    "deployment-tools@team-tools": true,
    "experimental-features@personal": false
  }
}
```

extraKnownMarketplaces

Definiert zusätzliche Marketplaces, die für das Repository verfügbar gemacht werden sollten. Normalerweise in Repository-Ebenen-Einstellungen verwendet, um sicherzustellen, dass Teamkollegen Zugriff auf erforderliche Plugin-Quellen haben.

Wenn ein Repository `extraKnownMarketplaces` enthält:

1. Teamkollegen werden aufgefordert, den Marketplace zu installieren, wenn sie den Ordner vertrauen
2. Teamkollegen werden dann aufgefordert, Plugins aus diesem Marketplace zu installieren
3. Benutzer können unerwünschte Marketplaces oder Plugins überspringen (in Benutzereinstellungen gespeichert)
4. Die Installation respektiert Vertrauensgrenzen und erfordert explizite Zustimmung

Beispiel:

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/claude-plugins"
      }
    },
    "security-plugins": {
      "source": {
        "source": "git",
        "url": "https://git.example.com/security/plugins.git"
      }
    }
  }
}
```

Marketplace-Quellentypen:

- `github` : GitHub-Repository (verwendet `repo`)
- `git` : Beliebige Git-URL (verwendet `url`)
- `directory` : Lokaler Dateisystem-Pfad (verwendet `path` , nur für Entwicklung)

- `hostPattern` : Regex-Muster zum Abgleichen von Marketplace-Hosts (verwendet `hostPattern`)

`strictKnownMarketplaces`

Nur verwaltete Einstellungen: Steuert, welche Plugin-Marketplaces Benutzer hinzufügen dürfen. Diese Einstellung kann nur in [verwalteten Einstellungen](#) konfiguriert werden und bietet Administratoren strikte Kontrolle über Marketplace-Quellen.

Verwaltete Einstellungsdatei-Speicherorte:

- **macOS:** `/Library/Application Support/ClaudeCode/managed-settings.json`
- **Linux und WSL:** `/etc/claude-code/managed-settings.json`
- **Windows:** `C:\Program Files\ClaudeCode\managed-settings.json`

Wichtige Merkmale:

- Nur in verwalteten Einstellungen verfügbar (`managed-settings.json`)
- Kann nicht durch Benutzer- oder Projekteinstellungen überschrieben werden (höchste Priorität)
- Durchgesetzt VOR Netzwerk-/Dateisystem-Operationen (blockierte Quellen werden nie ausgeführt)
- Verwendet exakten Abgleich für Quellspezifikationen (einschließlich `ref` , `path` für Git-Quellen), außer `hostPattern` , das Regex-Abgleich verwendet

Allowlist-Verhalten:

- `undefined` (Standard): Keine Einschränkungen - Benutzer können jeden Marketplace hinzufügen
- Leeres Array `[]` : Vollständiger Lockdown - Benutzer können keine neuen Marketplaces hinzufügen
- Liste von Quellen: Benutzer können nur Marketplaces hinzufügen, die genau übereinstimmen

Alle unterstützten Quellentypen:

Die Allowlist unterstützt sieben Marketplace-Quellentypen. Die meisten Quellen verwenden exakten Abgleich, während `hostPattern` Regex-Abgleich gegen den Marketplace-Host verwendet.

1. GitHub-Repositories:

```
{ "source": "github", "repo": "acme-corp/approved-plugins" }
{ "source": "github", "repo": "acme-corp/security-tools", "ref": "v2.0" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main", "path": "marketplace.json" }
```

Felder: **repo** (erforderlich), **ref** (optional: Branch/Tag/SHA), **path** (optional: Unterverzeichnis)

1. Git-Repositories:

```
{ "source": "git", "url": "https://gitlab.example.com/tools/plugins.git" }
{ "source": "git", "url": "https://bitbucket.org/acme-corp/plugins.git", "ref": "production" }
{ "source": "git", "url": "ssh://git@git.example.com/plugins.git", "ref": "v3.1", "path": "approved" }
```

Felder: **url** (erforderlich), **ref** (optional: Branch/Tag/SHA), **path** (optional: Unterverzeichnis)

1. URL-basierte Marketplaces:

```
{ "source": "url", "url": "https://plugins.example.com/marketplace.json" }
{ "source": "url", "url": "https://cdn.example.com/marketplace.json", "headers":
{ "Authorization": "Bearer ${TOKEN}" } }
```

Felder: **url** (erforderlich), **headers** (optional: HTTP-Header für authentifizierten Zugriff)

Note:

URL-basierte Marketplaces laden nur die **marketplace.json** Datei herunter. Sie laden keine Plugin-Dateien vom Server herunter. Plugins in URL-basierten Marketplaces müssen externe Quellen (GitHub, npm oder Git-URLs) verwenden, anstatt relative Pfade. Für Plugins mit relativen Pfaden verwenden Sie stattdessen einen Git-basierten Marketplace. Siehe [Troubleshooting](#) für Details.

1. NPM-Pakete:

```
{ "source": "npm", "package": "@acme-corp/claude-plugins" }
{ "source": "npm", "package": "@acme-corp/approved-marketplace" }
```

Felder: **package** (erforderlich, unterstützt scoped Pakete)

1. Dateipfade:

```
{ "source": "file", "path": "/usr/local/share/claude/acme-marketplace.json" }
{ "source": "file", "path": "/opt/acme-corp/plugins/marketplace.json" }
```

Felder: **path** (erforderlich: absoluter Pfad zur marketplace.json Datei)

1. Verzeichnispfade:

```
{ "source": "directory", "path": "/usr/local/share/claude/acme-plugins" }
{ "source": "directory", "path": "/opt/acme-corp/approved-marketplaces" }
```

Felder: **path** (erforderlich: absoluter Pfad zum Verzeichnis mit **.claude-plugin/marketplace.json**)

1. Host-Muster-Abgleich:

```
{ "source": "hostPattern", "hostPattern": "^github\\.example\\.com$" }
{ "source": "hostPattern", "hostPattern": "^gitlab\\.internal\\.example\\.com$" }
```

Felder: **hostPattern** (erforderlich: Regex-Muster zum Abgleich gegen den Marketplace-Host)

Verwenden Sie Host-Muster-Abgleich, wenn Sie alle Marketplaces von einem bestimmten Host erlauben möchten, ohne jedes Repository einzeln aufzuzählen. Dies ist nützlich für Organisationen mit internen GitHub Enterprise oder GitLab-Servern, auf denen Entwickler ihre eigenen Marketplaces erstellen.

Host-Extraktion nach Quellentyp:

- **github** : passt immer gegen **github.com**
- **git** : extrahiert Hostname aus der URL (unterstützt sowohl HTTPS als auch SSH-Formate)
- **url** : extrahiert Hostname aus der URL

- `npm`, `file`, `directory`: nicht unterstützt für Host-Muster-Abgleich

Konfigurationsbeispiele:

Beispiel: Nur bestimmte Marketplaces erlauben:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "github",
      "repo": "acme-corp/approved-plugins"
    },
    {
      "source": "github",
      "repo": "acme-corp/security-tools",
      "ref": "v2.0"
    },
    {
      "source": "url",
      "url": "https://plugins.example.com/marketplace.json"
    },
    {
      "source": "npm",
      "package": "@acme-corp/compliance-plugins"
    }
  ]
}
```

Beispiel - Alle Marketplace-Ergänzungen deaktivieren:

```
{
  "strictKnownMarketplaces": []
}
```

Beispiel: Alle Marketplaces von einem internen Git-Server erlauben:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

Anforderungen für exakten Abgleich:

Marketplace-Quellen müssen **genau** übereinstimmen, damit eine Benutzer-Ergänzung erlaubt wird. Für Git-basierte Quellen (`github` und `git`) umfasst dies alle optionalen Felder:

- Das `repo` oder `url` muss genau übereinstimmen
- Das `ref` Feld muss genau übereinstimmen (oder beide sind undefined)
- Das `path` Feld muss genau übereinstimmen (oder beide sind undefined)

Beispiele von Quellen, die **NICHT übereinstimmen**:

```
// Diese sind UNTERSCHIEDLICHE Quellen:
{ "source": "github", "repo": "acme-corp/plugins" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main" }

// Diese sind auch UNTERSCHIEDLICH:
{ "source": "github", "repo": "acme-corp/plugins", "path": "marketplace" }
{ "source": "github", "repo": "acme-corp/plugins" }
```

Vergleich mit `extraKnownMarketplaces` :

Aspekt	<code>strictKnownMarketplaces</code>	<code>extraKnownMarketplaces</code>
Zweck	Durchsetzung von Organisationsrichtlinien	Team-Komfort
Einstellungsdatei	Nur <code>managed-settings.json</code>	Beliebige Einstellungsdatei

Aspekt	<code>strictKnownMarketplaces</code>	<code>extraKnownMarketplaces</code>
Verhalten	Blockiert nicht-whitelisted Ergänzungen	Auto-installiert fehlende Marketplaces
Wann durchgesetzt	Vor Netzwerk-/Dateisystem-Operationen	Nach Benutzer-Vertrauens-Prompt
Kann überschrieben werden	Nein (höchste Priorität)	Ja (durch höherrangige Einstellungen)
Quellenformat	Direktes Quellobjekt	Benannter Marketplace mit verschachtelter Quelle
Anwendungsfall	Compliance, Sicherheitsbeschränkungen	Onboarding, Standardisierung

Format-Unterschied:

`strictKnownMarketplaces` verwendet direkte Quellobjekte:

```
{
  "strictKnownMarketplaces": [
    { "source": "github", "repo": "acme-corp/plugins" }
  ]
}
```

`extraKnownMarketplaces` erfordert benannte Marketplaces:

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": { "source": "github", "repo": "acme-corp/plugins" }
    }
  }
}
```

Wichtige Hinweise:

- Einschränkungen werden VOR Netzwerkanfragen oder Dateisystem-Operationen überprüft

- Wenn blockiert, sehen Benutzer klare Fehlermeldungen, die angeben, dass die Quelle durch verwaltete Richtlinie blockiert ist
- Die Einschränkung gilt nur für das Hinzufügen von NEUEN Marketplaces; zuvor installierte Marketplaces bleiben zugänglich
- Verwaltete Einstellungen haben die höchste Priorität und können nicht überschrieben werden

Siehe [Verwaltete Marketplace-Einschränkungen](#) für Dokumentation für Benutzer.

Plugins verwalten

Verwenden Sie den `/plugin` Befehl, um Plugins interaktiv zu verwalten:

- Durchsuchen Sie verfügbare Plugins aus Marketplaces
- Installieren/Deinstallieren Sie Plugins
- Aktivieren/Deaktivieren Sie Plugins
- Zeigen Sie Plugin-Details an (bereitgestellte Befehle, Agents, Hooks)
- Fügen Sie Marketplaces hinzu/entfernen Sie sie

Erfahren Sie mehr über das Plugin-System in der [Plugins-Dokumentation](#).

Umgebungsvariablen

Claude Code unterstützt die folgenden Umgebungsvariablen, um sein Verhalten zu steuern:

Note:

Alle Umgebungsvariablen können auch in `settings.json` konfiguriert werden. Dies ist nützlich, um Umgebungsvariablen automatisch für jede Sitzung zu setzen oder einen Satz von Umgebungsvariablen für Ihr ganzes Team oder Ihre ganze Organisation bereitzustellen.

Variable	Zweck	
<code>ANTHROPIC_API_KEY</code>	API-Schlüssel, der als <code>X-API-Key</code> Header gesendet wird, normalerweise für das Claude SDK (für interaktive Nutzung führen Sie <code>/login</code> aus)	

Variable	Zweck	
<code>ANTHROPIC_AUTH_TOKEN</code>	Benutzerdefinierter Wert für den <code>Authorization</code> Header (der Wert, den Sie hier setzen, wird mit <code>Bearer</code> vorangestellt)	
<code>ANTHROPIC_CUSTOM_HEADERS</code>	Benutzerdefinierte Header, die zu Anfragen hinzugefügt werden sollen (<code>Name: Value</code> Format, zeilengetrennt für mehrere Header)	
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL</code>	Siehe Modellkonfiguration	
<code>ANTHROPIC_DEFAULT_OPUS_MODEL</code>	Siehe Modellkonfiguration	
<code>ANTHROPIC_DEFAULT_SONNET_MODEL</code>	Siehe Modellkonfiguration	
<code>ANTHROPIC_FOUNDRY_API_KEY</code>	API-Schlüssel für Microsoft Foundry-Authentifizierung (siehe Microsoft Foundry)	
<code>ANTHROPIC_FOUNDRY_BASE_URL</code>	Vollständige Basis-URL für die Foundry-Ressource (z. B. <code>https://my-resource.services.ai.azure.com/anthropic</code>). Alternative zu <code>ANTHROPIC_FOUNDRY_RESOURCE</code> (siehe Microsoft Foundry)	
<code>ANTHROPIC_FOUNDRY_RESOURCE</code>	Foundry-Ressourcenname (z. B. <code>my-resource</code>). Erforderlich, wenn <code>ANTHROPIC_FOUNDRY_BASE_URL</code> nicht gesetzt ist (siehe Microsoft Foundry)	
<code>ANTHROPIC_MODEL</code>	Name der zu verwendenden Modelleinstellung (siehe Modellkonfiguration)	
<code>ANTHROPIC_SMALL_FAST_MODEL</code>	[VERALTET] Name des Haiku-Klasse-Modells für Hintergrundaufgaben	

Variable	Zweck	
<code>ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION</code>	Überschreiben Sie die AWS-Region für das Haiku-Klasse-Modell bei Verwendung von Bedrock	
<code>AWS_BEARER_TOKEN_BEDROCK</code>	Bedrock API-Schlüssel für Authentifizierung (siehe Bedrock API-Schlüssel)	
<code>BASH_DEFAULT_TIMEOUT_MS</code>	Standard-Timeout für lang laufende Bash-Befehle	
<code>BASH_MAX_OUTPUT_LENGTH</code>	Maximale Anzahl von Zeichen in Bash-Ausgaben, bevor sie in der Mitte gekürzt werden	
<code>BASH_MAX_TIMEOUT_MS</code>	Maximales Timeout, das das Modell für lang laufende Bash-Befehle setzen kann	
<code>CLAUDE_AUTOCOMPACT_PERCENT_OVERRIDE</code>	Setzen Sie den Prozentsatz der Kontextkapazität (1-100), bei dem Auto-Komprimierung ausgelöst wird. Standardmäßig wird Auto-Komprimierung bei etwa 95% Kapazität ausgelöst. Verwenden Sie niedrigere Werte wie <code>50</code> , um früher zu komprimieren. Werte über dem Standard-Schwellenwert haben keine Auswirkung. Gilt für Hauptkonversationen und Subagents. Dieser Prozentsatz entspricht dem Feld <code>context_window.used_percentage</code> , das in Statuszeile verfügbar ist	
<code>CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR</code>	Kehren Sie nach jedem Bash-Befehl zum ursprünglichen Arbeitsverzeichnis zurück	

Variable	Zweck	
<code>CLAUDE_CODE_ACCOUNT_UUID</code>	Konto-UUID für den authentifizierten Benutzer. Wird von SDK-Aufrufen verwendet, um Kontoinformationen synchron bereitzustellen und eine Racebedingung zu vermeiden, bei der frühe Telemetrie-Ereignisse keine Kontometadaten haben. Erfordert, dass auch <code>CLAUDE_CODE_USER_EMAIL</code> und <code>CLAUDE_CODE_ORGANIZATION_UUID</code> gesetzt sind	
<code>CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD</code>	Auf <code>1</code> setzen, um CLAUDE.md Dateien aus Verzeichnissen zu laden, die mit <code>--add-dir</code> angegeben sind. Standardmäßig laden zusätzliche Verzeichnisse keine Memory-Dateien	<code>1</code>
<code>CLAUDE_CODE_API_KEY_HELPER_TTL_MS</code>	Intervall in Millisekunden, in dem Anmeldedaten aktualisiert werden sollten (bei Verwendung von <code>apiKeyHelper</code>)	
<code>CLAUDE_CODE_CLIENT_CERT</code>	Pfad zur Client-Zertifikatsdatei für mTLS-Authentifizierung	
<code>CLAUDE_CODE_CLIENT_KEY</code>	Pfad zur privaten Client-Schlüsseldatei für mTLS-Authentifizierung	
<code>CLAUDE_CODE_CLIENT_KEY_PASSPHRASE</code>	Passphrase für verschlüsselten <code>CLAUDE_CODE_CLIENT_KEY</code> (optional)	
<code>CLAUDE_CODE_DISABLE_1M_CONTEXT</code>	Auf <code>1</code> setzen, um die Unterstützung des 1M-Kontextfensters zu deaktivieren. Wenn gesetzt, sind 1M-Modellvarianten im Modellwähler nicht verfügbar. Nützlich für Unternehmensumgebungen mit Compliance-Anforderungen	

Variable	Zweck	
<code>CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING</code>	Auf <code>1</code> setzen, um adaptives Denken für Opus 4.6 und Sonnet 4.6 zu deaktivieren. Wenn deaktiviert, fallen diese Modelle auf das feste Denkbudget zurück, das von <code>MAX_THINKING_TOKENS</code> gesteuert wird	
<code>CLAUDE_CODE_DISABLE_AUTO_MEMORY</code>	Auf <code>1</code> setzen, um Auto-Memory zu deaktivieren. Auf <code>0</code> setzen, um Auto-Memory während des schrittweisen Rollouts zu erzwingen. Wenn deaktiviert, erstellt oder lädt Claude keine Auto-Memory-Dateien	
<code>CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS</code>	Auf <code>1</code> setzen, um integrierte Commit- und PR-Workflow-Anweisungen aus Claudes System-Prompt zu entfernen. Nützlich bei Verwendung Ihrer eigenen Git-Workflow-Skills. Hat Vorrang vor der <code>includeGitInstructions</code> Einstellung, wenn gesetzt	
<code>CLAUDE_CODE_DISABLE_BACKGROUND_TASKS</code>	Auf <code>1</code> setzen, um alle Hintergrundaufgaben-Funktionalität zu deaktivieren, einschließlich des Parameters <code>run_in_background</code> auf Bash- und Subagent-Tools, Auto-Background und der Ctrl+B-Verknüpfung	
<code>CLAUDE_CODE_DISABLE_CRON</code>	Auf <code>1</code> setzen, um geplante Aufgaben zu deaktivieren. Der <code>/loop</code> Skill und Cron-Tools werden nicht verfügbar und alle bereits geplanten Aufgaben stoppen, einschließlich Aufgaben, die bereits mitten in der Sitzung laufen	

Variable	Zweck	
<code>CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS</code>	Auf 1 setzen, um Anthropic API-spezifische anthropic-beta Header zu deaktivieren. Verwenden Sie dies, wenn Sie Probleme wie “Unexpected value(s) for the anthropic-beta header” bei Verwendung eines LLM-Gateways mit Drittanbieter-Providern haben	
<code>CLAUDE_CODE_DISABLE_FAST_MODE</code>	Auf 1 setzen, um schnellen Modus zu deaktivieren	
<code>CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY</code>	Auf 1 setzen, um die Umfragen zur Sitzungsqualität “How is Claude doing?” zu deaktivieren. Auch deaktiviert bei Verwendung von Drittanbieter-Providern oder wenn Telemetrie deaktiviert ist. Siehe Umfragen zur Sitzungsqualität	
<code>CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC</code>	Äquivalent zum Setzen von <code>DISABLE_AUTOUPDATER</code> , <code>DISABLE_BUG_COMMAND</code> , <code>DISABLE_ERROR_REPORTING</code> und <code>DISABLE_TELEMETRY</code>	
<code>CLAUDE_CODE_DISABLE_TERMINAL_TITLE</code>	Auf 1 setzen, um automatische Terminal-Titel-Updates basierend auf Konversationskontext zu deaktivieren	
<code>CLAUDE_CODE_EFFORT_LEVEL</code>	Setzen Sie die Anstrengungsebene für unterstützte Modelle. Werte: low , medium , high . Niedrigere Anstrengung ist schneller und billiger, höhere Anstrengung bietet tieferes Denken. Unterstützt auf Opus 4.6 und Sonnet 4.6. Siehe Anstrengungsebene anpassen	

Variable	Zweck	
<code>CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION</code>	Auf <code>false</code> setzen, um Prompt-Vorschläge zu deaktivieren (der Umschalter “Prompt suggestions” in <code>/config</code>). Dies sind die ausgegraut angezeigten Vorhersagen, die nach Claudes Antwort in Ihrer Prompt-Eingabe erscheinen. Siehe Prompt-Vorschläge	
<code>CLAUDE_CODE_ENABLE_TASKS</code>	Auf <code>false</code> setzen, um vorübergehend zur vorherigen TODO-Liste anstelle des Task-Tracking-Systems zurückzukehren. Standard: <code>true</code> . Siehe Task-Liste	
<code>CLAUDE_CODE_ENABLE_TELEMETRY</code>	Auf <code>1</code> setzen, um OpenTelemetry-Datenerfassung für Metriken und Protokollierung zu aktivieren. Erforderlich, bevor OTel-Exporter konfiguriert werden. Siehe Monitoring	
<code>CLAUDE_CODE_EXIT_AFTER_STOP_DELAY</code>	Zeit in Millisekunden, die nach dem Leerlaufen der Abfrageschleife gewartet werden soll, bevor automatisch beendet wird. Nützlich für automatisierte Workflows und Skripte mit SDK-Modus	
<code>CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS</code>	Auf <code>1</code> setzen, um Agent-Teams zu aktivieren. Agent-Teams sind experimentell und standardmäßig deaktiviert	
<code>CLAUDE_CODE_FILE_READ_MAX_OUTPUT_TOKENS</code>	Überschreiben Sie das Standard-Token-Limit für Dateilesevorgänge. Nützlich, wenn Sie größere Dateien vollständig lesen müssen	
<code>CLAUDE_CODE_HIDE_ACCOUNT_INFO</code>	Auf <code>1</code> setzen, um Ihre E-Mail-Adresse und Organisationsnamen aus der Claude Code-Benutzeroberfläche auszublenden. Nützlich beim Streamen oder Aufzeichnen	

Variable	Zweck	
<code>CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL</code>	Überspringen Sie die automatische Installation von IDE-Erweiterungen	
<code>CLAUDE_CODE_MAX_OUTPUT_TOKENS</code>	Setzen Sie die maximale Anzahl von Ausgabe-Tokens für die meisten Anfragen. Standard: 32.000. Maximum: 64.000. Das Erhöhen dieses Wertes reduziert das verfügbare effektive Kontextfenster, bevor Auto-Komprimierung ausgelöst wird.	
<code>CLAUDE_CODE_ORGANIZATION_UUID</code>	Organisations-UUID für den authentifizierten Benutzer. Wird von SDK-Aufrufern verwendet, um Kontoinformationen synchron bereitzustellen. Erfordert, dass auch <code>CLAUDE_CODE_ACCOUNT_UUID</code> und <code>CLAUDE_CODE_USER_EMAIL</code> gesetzt sind	
<code>CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS</code>	Intervall zum Aktualisieren dynamischer OpenTelemetry-Header in Millisekunden (Standard: 1740000 / 29 Minuten). Siehe Dynamische Header	
<code>CLAUDE_CODE_PLAN_MODE_REQUIRED</code>	Auto-gesetzt auf <code>true</code> auf Agent-Team Teamkollegen, die Plan-Genehmigung erfordern. Schreibgeschützt: wird von Claude Code beim Spawnen von Teamkollegen gesetzt. Siehe Plan-Genehmigung erforderlich	
<code>CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS</code>	Timeout in Millisekunden für Git-Operationen beim Installieren oder Aktualisieren von Plugins (Standard: 120000). Erhöhen Sie diesen Wert für große Repositories oder langsame Netzwerkverbindungen. Siehe Git-Operationen Timeout	

Variable	Zweck	
<code>CLAUDE_CODE_PROXY_RESOLVES_HOSTS</code>	Auf <code>true</code> setzen, um dem Proxy zu erlauben, DNS-Auflösung durchzuführen, anstatt des Aufrufers. Opt-in für Umgebungen, in denen der Proxy die Hostname-Auflösung durchführen sollte	
<code>CLAUDE_CODE_SHELL</code>	Überschreiben Sie die automatische Shell-Erkennung. Nützlich, wenn sich Ihre Login-Shell von Ihrer bevorzugten Arbeits-Shell unterscheidet (z. B. <code>bash</code> vs <code>zsh</code>)	
<code>CLAUDE_CODE_SHELL_PREFIX</code>	Befehlspräfix zum Umhüllen aller Bash-Befehle (z. B. für Protokollierung oder Auditing). Beispiel: <code>/path/to/logger.sh</code> führt <code>/path/to/logger.sh <command></code> aus	
<code>CLAUDE_CODE_SIMPLE</code>	Auf <code>1</code> setzen, um mit einem minimalen System-Prompt und nur den Tools Bash, Dateilesevorgänge und Dateibearbeitung auszuführen. Deaktiviert MCP-Tools, Anhänge, Hooks und CLAUDE.md Dateien	
<code>CLAUDE_CODE_SKIP_BEDROCK_AUTH</code>	Überspringen Sie die AWS-Authentifizierung für Bedrock (z. B. bei Verwendung eines LLM-Gateways)	
<code>CLAUDE_CODE_SKIP_FOUNDRY_AUTH</code>	Überspringen Sie die Azure-Authentifizierung für Microsoft Foundry (z. B. bei Verwendung eines LLM-Gateways)	
<code>CLAUDE_CODE_SKIP_VERTEX_AUTH</code>	Überspringen Sie die Google-Authentifizierung für Vertex (z. B. bei Verwendung eines LLM-Gateways)	
<code>CLAUDE_CODE_SUBAGENT_MODEL</code>	Siehe Modellkonfiguration	

Variable	Zweck	
<code>CLAUDE_CODE_TASK_LIST_ID</code>	Teilen Sie eine Task-Liste über Sitzungen hinweg. Setzen Sie die gleiche ID in mehreren Claude Code-Instanzen, um an einer gemeinsamen Task-Liste zu koordinieren. Siehe Task-Liste	
<code>CLAUDE_CODE_TEAM_NAME</code>	Name des Agent-Teams, zu dem dieser Teamkollege gehört. Wird automatisch auf Agent-Team Mitgliedern gesetzt	
<code>CLAUDE_CODE_TMPDIR</code>	Überschreiben Sie das Temp-Verzeichnis, das für interne Temp-Dateien verwendet wird. Claude Code hängt <code>/claude/</code> an diesen Pfad an. Standard: <code>/tmp</code> auf Unix/macOS, <code>os.tmpdir()</code> auf Windows	
<code>CLAUDE_CODE_USER_EMAIL</code>	E-Mail-Adresse für den authentifizierten Benutzer. Wird von SDK-Aufrufen verwendet, um Kontoinformationen synchron bereitzustellen. Erfordert, dass auch <code>CLAUDE_CODE_ACCOUNT_UUID</code> und <code>CLAUDE_CODE_ORGANIZATION_UUID</code> gesetzt sind	
<code>CLAUDE_CODE_USE_BEDROCK</code>	Verwenden Sie Bedrock	
<code>CLAUDE_CODE_USE_FOUNDRY</code>	Verwenden Sie Microsoft Foundry	
<code>CLAUDE_CODE_USE_VERTEX</code>	Verwenden Sie Vertex	
<code>CLAUDE_CONFIG_DIR</code>	Passen Sie an, wo Claude Code seine Konfiguration und Datendateien speichert	
<code>DISABLE_AUTOUPDATER</code>	Auf <code>1</code> setzen, um automatische Updates zu deaktivieren.	

Variable	Zweck	
<code>DISABLE_BUG_COMMAND</code>	Auf <code>1</code> setzen, um den <code>/bug</code> Befehl zu deaktivieren	
<code>DISABLE_COST_WARNING</code>	Auf <code>1</code> setzen, um Kostenwarnnachrichten zu deaktivieren	
<code>DISABLE_ERROR_REPORTING</code>	Auf <code>1</code> setzen, um sich von Sentry-Fehlerberichten abzumelden	
<code>DISABLE_INSTALLATION_CHECKS</code>	Auf <code>1</code> setzen, um Installationswarnungen zu deaktivieren. Verwenden Sie nur, wenn Sie den Installationsort manuell verwalten, da dies Probleme mit Standardinstallationen maskieren kann	
<code>DISABLE_NON_ESSENTIAL_MODEL_CALLS</code>	Auf <code>1</code> setzen, um Modellaufrufe für nicht kritische Pfade wie Flavor-Text zu deaktivieren	
<code>DISABLE_PROMPT_CACHING</code>	Auf <code>1</code> setzen, um Prompt-Caching für alle Modelle zu deaktivieren (hat Vorrang vor Pro-Modell-Einstellungen)	
<code>DISABLE_PROMPT_CACHING_HAIKU</code>	Auf <code>1</code> setzen, um Prompt-Caching für Haiku-Modelle zu deaktivieren	
<code>DISABLE_PROMPT_CACHING_OPUS</code>	Auf <code>1</code> setzen, um Prompt-Caching für Opus-Modelle zu deaktivieren	
<code>DISABLE_PROMPT_CACHING_SONNET</code>	Auf <code>1</code> setzen, um Prompt-Caching für Sonnet-Modelle zu deaktivieren	
<code>DISABLE_TELEMETRY</code>	Auf <code>1</code> setzen, um sich von Statsig-Telemetrie abzumelden (beachten Sie, dass Statsig-Ereignisse keine Benutzerdaten wie Code, Dateipfade oder Bash-Befehle enthalten)	
<code>ENABLE_CLAUDEAI_MCP_SERVERS</code>	Auf <code>false</code> setzen, um claude.ai MCP-Server in Claude Code zu deaktivieren. Standardmäßig für angemeldete Benutzer aktiviert	

Variable	Zweck	
<code>ENABLE_TOOL_SEARCH</code>	Steuert MCP-Tool-Suche . Werte: <code>auto</code> (Standard, aktiviert bei 10% Kontext), <code>auto:N</code> (benutzerdefinierter Schwellenwert, z. B. <code>auto:5</code> für 5%), <code>true</code> (immer an), <code>false</code> (deaktiviert)	
<code>FORCE_AUTOUPDATE_PLUGINS</code>	Auf <code>true</code> setzen, um Plugin-Auto-Updates zu erzwingen, auch wenn der Haupt-Auto-Updater über <code>DISABLE_AUTOUPDATER</code> deaktiviert ist	
<code>HTTP_PROXY</code>	Geben Sie den HTTP-Proxy-Server für Netzwerkverbindungen an	
<code>HTTPS_PROXY</code>	Geben Sie den HTTPS-Proxy-Server für Netzwerkverbindungen an	
<code>IS_DEMO</code>	Auf <code>true</code> setzen, um Demo-Modus zu aktivieren: verbirgt E-Mail und Organisation aus der Benutzeroberfläche, überspringt Onboarding und verbirgt interne Befehle. Nützlich zum Streamen oder Aufzeichnen von Sitzungen	
<code>MAX_MCP_OUTPUT_TOKENS</code>	Maximale Anzahl von Tokens, die in MCP-Tool-Antworten zulässig sind. Claude Code zeigt eine Warnung an, wenn die Ausgabe 10.000 Tokens überschreitet (Standard: 25000)	

Variable	Zweck	
<code>MAX_THINKING_TOKENS</code>	Überschreiben Sie das erweitertes Denken Token-Budget. Denken ist standardmäßig mit maximalem Budget (31.999 Tokens) aktiviert. Verwenden Sie dies, um das Budget zu begrenzen (z. B. <code>MAX_THINKING_TOKENS=10000</code>) oder Denken vollständig zu deaktivieren (<code>MAX_THINKING_TOKENS=0</code>). Für Opus 4.6 wird die Denk-Tiefe durch Anstrengungsebene gesteuert, und diese Variable wird ignoriert, es sei denn, sie ist auf <code>0</code> gesetzt, um Denken zu deaktivieren.	
<code>MCP_CLIENT_SECRET</code>	OAuth-Client-Secret für MCP-Server, die vorkonfigurierte Anmeldedaten erfordern. Vermeidet die interaktive Eingabeaufforderung beim Hinzufügen eines Servers mit <code>--client-secret</code>	
<code>MCP_OAUTH_CALLBACK_PORT</code>	Fester Port für den OAuth-Redirect-Callback, als Alternative zu <code>--callback-port</code> beim Hinzufügen eines MCP-Servers mit vorkonfigurierten Anmeldedaten	
<code>MCP_TIMEOUT</code>	Timeout in Millisekunden für MCP-Server-Start	
<code>MCP_TOOL_TIMEOUT</code>	Timeout in Millisekunden für MCP-Tool-Ausführung	
<code>NO_PROXY</code>	Liste von Domänen und IPs, an die Anfragen direkt gestellt werden, wobei der Proxy umgangen wird	

Variable	Zweck	
<code>SLASH_COMMAND_TOOL_CHARACTER_BUDGET</code>	Überschreiben Sie das Zeichen-Budget für Skill-Metadaten, die dem Skill-Tool angezeigt werden. Das Budget skaliert dynamisch bei 2% des Kontextfensters, mit einem Fall-back von 16.000 Zeichen. Legacy-Name für Rückwärtskompatibilität beibehalten	
<code>USE_BUILTIN_RIPGREP</code>	Auf <code>0</code> setzen, um das systeminstallierte <code>rg</code> anstelle des mit Claude Code enthaltenen <code>rg</code> zu verwenden	
<code>VERTEX_REGION_CLAUDE_3_5_HAIKU</code>	Überschreiben Sie die Region für Claude 3.5 Haiku bei Verwendung von Vertex AI	
<code>VERTEX_REGION_CLAUDE_3_7_SONNET</code>	Überschreiben Sie die Region für Claude 3.7 Sonnet bei Verwendung von Vertex AI	
<code>VERTEX_REGION_CLAUDE_4_0_OPUS</code>	Überschreiben Sie die Region für Claude 4.0 Opus bei Verwendung von Vertex AI	
<code>VERTEX_REGION_CLAUDE_4_0_SONNET</code>	Überschreiben Sie die Region für Claude 4.0 Sonnet bei Verwendung von Vertex AI	
<code>VERTEX_REGION_CLAUDE_4_1_OPUS</code>	Überschreiben Sie die Region für Claude 4.1 Opus bei Verwendung von Vertex AI	

Tools, die Claude zur Verfügung stehen

Claude Code hat Zugriff auf eine Reihe leistungsstarker Tools, die ihm helfen, Ihre Codebasis zu verstehen und zu ändern:

Tool	Beschreibung	Berechtigung erforderlich
Agent	Spawnt einen Subagent mit eigenem Kontextfenster, um eine Aufgabe zu bewältigen	Nein
AskUserQuestion	Stellt Multiple-Choice-Fragen, um Anforderungen zu sammeln oder Mehrdeutigkeit zu klären	Nein
Bash	Führt Shell-Befehle in Ihrer Umgebung aus. Siehe Bash-Tool-Verhalten	Ja
CronCreate	Plant eine wiederkehrende oder einmalige Eingabeaufforderung innerhalb der aktuellen Sitzung (weg, wenn Claude beendet wird). Siehe geplante Aufgaben	Nein
CronDelete	Bricht eine geplante Aufgabe nach ID ab	Nein
CronList	Listet alle geplanten Aufgaben in der Sitzung auf	Nein
Edit	Nimmt gezielte Bearbeitungen an bestimmten Dateien vor	Ja
EnterPlanMode	Wechselt in den Plan-Modus, um einen Ansatz vor dem Codieren zu entwerfen	Nein
EnterWorktree	Erstellt einen isolierten Git-Worktree und wechselt hinein	Nein
ExitPlanMode	Präsentiert einen Plan zur Genehmigung und beendet den Plan-Modus	Ja
ExitWorktree	Beendet eine Worktree-Sitzung und kehrt zum ursprünglichen Verzeichnis zurück	Nein
Glob	Findet Dateien basierend auf Musterabgleich	Nein

Tool	Beschreibung	Berechtigung erforderlich
Grep	Sucht nach Mustern in Dateiinhalten	Nein
ListMcpResourcesTool	Listet Ressourcen auf, die von verbundenen MCP-Servern bereitgestellt werden	Nein
LSP	Code-Intelligenz über Sprachserver. Meldet Typfehler und Warnungen automatisch nach Dateibearbeitungen. Unterstützt auch Navigationsvorgänge: Sprung zu Definitionen, Suche nach Referenzen, Typinformationen abrufen, Symbole auflisten, Implementierungen finden, Call-Hierarchien verfolgen. Erfordert ein Code-Intelligence-Plugin und seine Sprachserver-Binärdatei	Nein
NotebookEdit	Ändert Jupyter-Notebook-Zellen	Ja
Read	Liest den Inhalt von Dateien	Nein
ReadMcpResourceTool	Liest eine bestimmte MCP-Ressource nach URI	Nein
Skill	Führt einen Skill innerhalb der Hauptkonversation aus	Ja
TaskCreate	Erstellt eine neue Aufgabe in der Task-Liste	Nein
TaskGet	Ruft vollständige Details für eine bestimmte Aufgabe ab	Nein
TaskList	Listet alle Aufgaben mit ihrem aktuellen Status auf	Nein
TaskOutput	Ruft Ausgabe von einer Hintergrundaufgabe ab	Nein
TaskStop	Beendet eine laufende Hintergrundaufgabe nach ID	Nein

Tool	Beschreibung	Berechtigung erforderlich
TaskUpdate	Aktualisiert Task-Status, Abhängigkeiten, Details oder löscht Aufgaben	Nein
TodoWrite	Verwaltet die Session-Task-Checkliste. Verfügbar im nicht-interaktiven Modus und dem Agent SDK ; interaktive Sitzungen verwenden stattdessen TaskCreate, TaskGet, TaskList und TaskUpdate	Nein
ToolSearch	Sucht nach und lädt aufgeschobene Tools, wenn Tool-Suche aktiviert ist	Nein
WebFetch	Ruft Inhalte von einer angegebenen URL ab	Ja
WebSearch	Führt Web-Suchen durch	Ja
Write	Erstellt oder überschreibt Dateien	Ja

Berechtigungsregeln können mit `/allowed-tools` oder in [Berechtigungseinstellungen](#) konfiguriert werden. Siehe auch [Tool-spezifische Berechtigungsregeln](#).

Bash-Tool-Verhalten

Das Bash-Tool führt Shell-Befehle mit folgendem Persistenz-Verhalten aus:

- **Arbeitsverzeichnis bleibt bestehen:** Wenn Claude das Arbeitsverzeichnis ändert (z. B. `cd /path/to/dir`), werden nachfolgende Bash-Befehle in diesem Verzeichnis ausgeführt. Sie können `CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR=1` verwenden, um nach jedem Befehl zum Projektverzeichnis zurückzukehren.
- **Umgebungsvariablen bleiben NICHT bestehen:** Umgebungsvariablen, die in einem Bash-Befehl gesetzt werden (z. B. `export MY_VAR=value`), sind **nicht** in nachfolgenden Bash-Befehlen verfügbar. Jeder Bash-Befehl wird in einer frischen Shell-Umgebung ausgeführt.

Um Umgebungsvariablen in Bash-Befehlen verfügbar zu machen, haben Sie **drei Optionen**:

Option 1: Aktivieren Sie die Umgebung vor dem Start von Claude Code (einfachster Ansatz)

Aktivieren Sie Ihre virtuelle Umgebung in Ihrem Terminal, bevor Sie Claude Code starten:

```
conda activate myenv
## oder: source /path/to/venv/bin/activate
claude
```

Dies funktioniert für Shell-Umgebungen, aber Umgebungsvariablen, die in Claudes Bash-Befehlen gesetzt werden, bleiben nicht zwischen Befehlen bestehen.

Option 2: Setzen Sie `CLAUDE_ENV_FILE` vor dem Start von Claude Code (persistente Umgebungseinrichtung)

Exportieren Sie den Pfad zu einem Shell-Skript, das Ihre Umgebungseinrichtung enthält:

```
export CLAUDE_ENV_FILE=/path/to/env-setup.sh
claude
```

Wobei `/path/to/env-setup.sh` enthält:

```
conda activate myenv
## oder: source /path/to/venv/bin/activate
## oder: export MY_VAR=value
```

Claude Code wird dieses Skript vor jedem Bash-Befehl sourcing, wodurch die Umgebung über alle Befehle hinweg persistent wird.

Option 3: Verwenden Sie einen `SessionStart-Hook` (projektspezifische Konfiguration)

Konfigurieren Sie in `.claude/settings.json`:

```
{
  "hooks": {
    "SessionStart": [{
      "matcher": "startup",
      "hooks": [{
        "type": "command",
        "command": "echo 'conda activate myenv' >> \"\$CLAUDE_ENV_FILE\""
      }]
    }]
  }
}
```

Der Hook schreibt in `$CLAUDE_ENV_FILE`, das dann vor jedem Bash-Befehl gesourced wird. Dies ist ideal für teamübergreifende Projektkonfigurationen.

Siehe [SessionStart-Hooks](#) für weitere Details zu Option 3.

Tools mit Hooks erweitern

Sie können benutzerdefinierte Befehle vor oder nach der Ausführung eines Tools ausführen, indem Sie [Claude Code Hooks](#) verwenden.

Sie könnten beispielsweise automatisch einen Python-Formatter ausführen, nachdem Claude Python-Dateien ändert, oder Änderungen an Produktionskonfigurationsdateien blockieren, indem Sie Write-Operationen auf bestimmte Pfade blockieren.

Siehe auch

- [Berechtigungen](#): Berechtigungssystem, Regelsyntax, Tool-spezifische Muster und verwaltete Richtlinien
- [Authentifizierung](#): Richten Sie Benutzerzugriff auf Claude Code ein
- [Troubleshooting](#): Lösungen für häufige Konfigurationsprobleme

Modellkonfiguration

Erfahren Sie mehr über die Claude Code-Modellkonfiguration, einschließlich Modellalias wie `opusplan`

Verfügbare Modelle

Für die `model`-Einstellung in Claude Code können Sie konfigurieren:

- Einen **Modellalias**
- Einen **Modellnamen**
 - Anthropic API: Ein vollständiger [Modellname](#)
 - Bedrock: ein Inference-Profil-ARN
 - Foundry: ein Bereitstellungsname
 - Vertex: ein Versionsname

Modellalias

Modellalias bieten eine bequeme Möglichkeit, Modelleinstellungen auszuwählen, ohne sich genaue Versionsnummern merken zu müssen:

Modellalias	Verhalten
<code>default</code>	Empfohlene Modelleinstellung, abhängig von Ihrem Kontotyp
<code>sonnet</code>	Verwendet das neueste Sonnet-Modell (derzeit Sonnet 4.6) für tägliche Codierungsaufgaben
<code>opus</code>	Verwendet das neueste Opus-Modell (derzeit Opus 4.6) für komplexe Reasoning-Aufgaben
<code>haiku</code>	Verwendet das schnelle und effiziente Haiku-Modell für einfache Aufgaben
<code>sonnet[1m]</code>	Verwendet Sonnet mit einem 1-Million-Token-Kontextfenster für lange Sitzungen
<code>opusplan</code>	Spezieller Modus, der <code>opus</code> während des Plan Mode verwendet und dann zu <code>sonnet</code> für die Ausführung wechselt

Aliase verweisen immer auf die neueste Version. Um eine bestimmte Version zu fixieren, verwenden Sie den vollständigen Modellnamen (z. B. `claude-opus-4-6`) oder setzen Sie die entsprechende Umgebungsvariable wie `ANTHROPIC_DEFAULT_OPUS_MODEL`.

Einstellung Ihres Modells

Sie können Ihr Modell auf mehrere Arten konfigurieren, aufgelistet nach Priorität:

1. **Während der Sitzung** - Verwenden Sie `/model <alias|name>`, um Modelle während der Sitzung zu wechseln
2. **Beim Start** - Starten Sie mit `claude --model <alias|name>`
3. **Umgebungsvariable** - Setzen Sie `ANTHROPIC_MODEL=<alias|name>`
4. **Einstellungen** - Konfigurieren Sie dauerhaft in Ihrer Einstellungsdatei mit dem `model`-Feld.

Beispielverwendung:

```
## Start mit Opus
claude --model opus

## Wechsel zu Sonnet während der Sitzung
/model sonnet
```

Beispiel-Einstellungsdatei:

```
{
  "permissions": {
    ...
  },
  "model": "opus"
}
```

Modellauswahl einschränken

Enterprise-Administratoren können `availableModels` in [verwalteten oder Richtlinieneinstellungen](#) verwenden, um einzuschränken, welche Modelle Benutzer auswählen können.

Wenn `availableModels` gesetzt ist, können Benutzer nicht über `/model`, das `--model`-Flag, das Config-Tool oder die `ANTHROPIC_MODEL`-Umgebungsvariable zu Modellen wechseln, die nicht in der Liste enthalten sind.

```
{
  "availableModels": ["sonnet", "haiku"]
}
```

Standardmodell-Verhalten

Die Option „Standard“ in der Modellauswahl wird von `availableModels` nicht beeinflusst. Sie bleibt immer verfügbar und stellt den Laufzeit-Standard des Systems dar, [basierend auf dem Abonnement-Tier des Benutzers](#).

Auch mit `availableModels: []` können Benutzer Claude Code weiterhin mit dem Standardmodell für ihren Tier verwenden.

Kontrollieren Sie das Modell, auf dem Benutzer ausgeführt werden

Um die Modellerfahrung vollständig zu kontrollieren, verwenden Sie `availableModels` zusammen mit der `model`-Einstellung:

- **availableModels:** schränkt ein, zu welchen Modellen Benutzer wechseln können
- **model:** setzt die explizite Modellüberschreibung, die Vorrang vor dem Standard hat

Dieses Beispiel stellt sicher, dass alle Benutzer Sonnet 4.6 ausführen und nur zwischen Sonnet und Haiku wählen können:

```
{
  "model": "sonnet",
  "availableModels": ["sonnet", "haiku"]
}
```

Merge-Verhalten

Wenn `availableModels` auf mehreren Ebenen gesetzt ist, z. B. in Benutzereinstellungen und Projekteinstellungen, werden Arrays zusammengeführt und dedupliziert. Um eine strikte Allowlist durchzusetzen, setzen Sie `availableModels` in verwalteten oder Richtlinieneinstellungen, die die höchste Priorität haben.

Spezielles Modellverhalten

default -Modelleinstellung

Das Verhalten von `default` hängt von Ihrem Kontotyp ab:

- **Max und Team Premium:** Standard ist Opus 4.6

- **Pro und Team Standard:** Standard ist Sonnet 4.6
- **Enterprise:** Opus 4.6 ist verfügbar, aber nicht der Standard

Claude Code kann automatisch auf Sonnet zurückfallen, wenn Sie einen Nutzungsschwellenwert mit Opus erreichen.

opusplan -Modelleinstellung

Der **opusplan** -Modellalias bietet einen automatisierten Hybrid-Ansatz:

- **Im Plan Mode** - Verwendet **opus** für komplexes Reasoning und Architekturentscheidungen
- **Im Execution Mode** - Wechselt automatisch zu **sonnet** für Code-Generierung und Implementierung

Dies gibt Ihnen das Beste aus beiden Welten: Opus's überlegenes Reasoning für die Planung und Sonnets Effizienz für die Ausführung.

Anpassung des Aufwandsniveaus

[Aufwandsniveaus](#) steuern adaptives Reasoning, das das Denken dynamisch basierend auf der Aufgabenkomplexität zuordnet. Niedrigerer Aufwand ist schneller und günstiger für unkomplizierte Aufgaben, während höherer Aufwand tieferes Reasoning für komplexe Probleme bietet.

Drei Ebenen sind verfügbar: **low**, **medium** und **high**. Opus 4.6 hat standardmäßig mittleren Aufwand für Max- und Team-Abonnenten.

Aufwand einstellen:

- **In `/model`** : Verwenden Sie die Pfeiltasten nach links/rechts, um den Aufwand-Schieberegler anzupassen, wenn Sie ein Modell auswählen
- **Umgebungsvariable:** Setzen Sie `CLAUDE_CODE_EFFORT_LEVEL=low|medium|high`
- **Einstellungen:** Setzen Sie `effortLevel` in Ihrer Einstellungsdatei

Der Aufwand wird auf Opus 4.6 und Sonnet 4.6 unterstützt. Der Aufwand-Schieberegler erscheint in `/model`, wenn ein unterstütztes Modell ausgewählt ist. Das aktuelle Aufwandsniveau wird auch neben dem Logo und Spinner angezeigt (z. B. „with low effort“), damit Sie bestätigen können, welche Einstellung aktiv ist, ohne `/model` zu öffnen.

Um adaptives Reasoning auf Opus 4.6 und Sonnet 4.6 zu deaktivieren und auf das vorherige feste Thinking-Budget zurückzukehren, setzen Sie

`CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`. Wenn deaktiviert, verwenden diese Modelle das feste Budget, das von `MAX_THINKING_TOKENS` gesteuert wird. Siehe [Umgebungsvariablen](#).

Erweiterter Kontext

Opus 4.6 und Sonnet 4.6 unterstützen ein [1-Million-Token-Kontextfenster](#) für lange Sitzungen mit großen Codebases.

Note:

Das 1M-Kontextfenster befindet sich derzeit in der Beta-Phase. Funktionen, Preise und Verfügbarkeit können sich ändern.

Erweiterter Kontext ist verfügbar für:

- **API- und Pay-as-you-go-Benutzer:** Vollzugriff auf 1M-Kontext
- **Pro-, Max-, Teams- und Enterprise-Abonnenten:** verfügbar mit [zusätzlicher Nutzung](#) aktiviert

Um 1M-Kontext vollständig zu deaktivieren, setzen Sie

`CLAUDE_CODE_DISABLE_1M_CONTEXT=1`. Dies entfernt 1M-Modellvarianten aus der Modellauswahl. Siehe [Umgebungsvariablen](#).

Die Auswahl eines 1M-Modells ändert die Abrechnung nicht sofort. Ihre Sitzung verwendet Standardsätze, bis sie 200K Token Kontext überschreitet. Jenseits von 200K Token werden Anfragen mit [Long-Context-Preisen](#) und dedizierten [Rate Limits](#) berechnet. Für Abonnenten werden Token jenseits von 200K als zusätzliche Nutzung statt durch das Abonnement abgerechnet.

Wenn Ihr Konto 1M-Kontext unterstützt, erscheint die Option in der Modellauswahl (`/model`) in den neuesten Versionen von Claude Code. Wenn Sie sie nicht sehen, versuchen Sie, Ihre Sitzung neu zu starten.

Sie können auch das `[1m]` -Suffix mit Modellaliasen oder vollständigen Modellnamen verwenden:

```
## Verwenden Sie den sonnet[1m]-Alias
/model sonnet[1m]

## Oder fügen Sie [1m] zu einem vollständigen Modellnamen hinzu
/model claude-sonnet-4-6[1m]
```


Überprüfung Ihres aktuellen Modells

Sie können sehen, welches Modell Sie derzeit verwenden, auf mehrere Arten:

1. In der [Statuszeile](#) (falls konfiguriert)
2. In `/status` , das auch Ihre Kontoinformationen anzeigt.

Umgebungsvariablen

Sie können die folgenden Umgebungsvariablen verwenden, die vollständige **Modellnamen** (oder Äquivalente für Ihren API-Anbieter) sein müssen, um die Modellnamen zu steuern, auf die die Aliase verweisen.

Umgebungsvariable	Beschreibung
<code>ANTHROPIC_DEFAULT_OPUS_MODEL</code>	Das Modell, das für <code>opus</code> verwendet werden soll, oder für <code>opusplan</code> , wenn Plan Mode aktiv ist.
<code>ANTHROPIC_DEFAULT_SONNET_MODEL</code>	Das Modell, das für <code>sonnet</code> verwendet werden soll, oder für <code>opusplan</code> , wenn Plan Mode nicht aktiv ist.
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL</code>	Das Modell, das für <code>haiku</code> verwendet werden soll, oder Hintergrundfunktionalität
<code>CLAUDE_CODE_SUBAGENT_MODEL</code>	Das Modell, das für <code>subagents</code> verwendet werden soll

Hinweis: `ANTHROPIC_SMALL_FAST_MODEL` ist veraltet zugunsten von `ANTHROPIC_DEFAULT_HAIKU_MODEL` .

Modelle für Drittanbieter-Bereitstellungen fixieren

Beim Bereitstellen von Claude Code über [Bedrock](#), [Vertex AI](#) oder [Foundry](#) sollten Sie Modellversionen vor dem Rollout für Benutzer fixieren.

Ohne Fixierung verwendet Claude Code Modellalias (`sonnet` , `opus` , `haiku`), die zur neuesten Version aufgelöst werden. Wenn Anthropic ein neues Modell veröffentlicht, werden Benutzer, deren Konten die neue Version nicht aktiviert haben, stillschweigend unterbrochen.

Warning:

Setzen Sie alle drei Modell-Umgebungsvariablen auf spezifische Versions-IDs als Teil Ihres anfänglichen Setups. Das Überspringen dieses Schritts bedeutet, dass ein Claude Code-Update Ihre Benutzer ohne Maßnahmen Ihrerseits unterbrechen kann.

Verwenden Sie die folgenden Umgebungsvariablen mit versionsspezifischen Modell-IDs für Ihren Anbieter:

Anbieter	Beispiel
Bedrock	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-6-v1'</code>
Vertex AI	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'</code>
Foundry	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'</code>

Wenden Sie das gleiche Muster auf `ANTHROPIC_DEFAULT_SONNET_MODEL` und `ANTHROPIC_DEFAULT_HAIKU_MODEL` an. Für aktuelle und ältere Modell-IDs über alle Anbieter hinweg siehe [Modellübersicht](#). Um Benutzer auf eine neue Modellversion zu aktualisieren, aktualisieren Sie diese Umgebungsvariablen und stellen Sie erneut bereit.

Note:

Die `settings.availableModels` -Allowlist gilt weiterhin bei Verwendung von Drittanbieter-Anbietern. Die Filterung stimmt mit dem Modellalias (`opus` , `sonnet` , `haiku`) überein, nicht mit der anbieterspezifischen Modell-ID.

Modell-IDs pro Version überschreiben

Die oben genannten Umgebungsvariablen auf Familienebene konfigurieren eine Modell-ID pro Familienalias. Wenn Sie mehrere Versionen innerhalb derselben Familie verschiedenen Anbieter-IDs zuordnen müssen, verwenden Sie stattdessen die `modelOverrides` -Einstellung.

`modelOverrides` ordnet einzelne Anthropic-Modell-IDs den anbieterspezifischen Strings zu, die Claude Code an die API Ihres Anbieters sendet. Wenn ein Benutzer ein zugeordnetes Modell in der `/model` -Auswahl auswählt, verwendet Claude Code Ihren konfigurierten Wert statt des integrierten Standards.

Dies ermöglicht es Enterprise-Administratoren, jede Modellversion zu einem bestimmten Bedrock-Inference-Profil-ARN, Vertex AI-Versionsnamen oder Foundry-Bereitstellungs-namen für Governance, Kostenzuteilung oder regionales Routing zu leiten.

Setzen Sie `modelOverrides` in Ihrer [Einstellungsdatei](#):

```
{
  "modelOverrides": {
    "claude-opus-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-
inference-profile/opus-prod",
    "claude-opus-4-5-20251101": "arn:aws:bedrock:us-
east-2:123456789012:application-inference-profile/opus-45-prod",
    "claude-sonnet-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-
inference-profile/sonnet-prod"
  }
}
```

Schlüssel müssen Anthropic-Modell-IDs sein, wie in der [Modellübersicht](#) aufgelistet. Für datierte Modell-IDs fügen Sie das Datumssuffix genau so ein, wie es dort angezeigt wird. Unbekannte Schlüssel werden ignoriert.

Überschreibungen ersetzen die integrierten Modell-IDs, die jeden Eintrag in der `/model`-Auswahl unterstützen. Bei Bedrock haben Überschreibungen Vorrang vor allen Inference-Profilen, die Claude Code beim Start automatisch erkennt. Werte, die Sie direkt über `ANTHROPIC_MODEL`, `--model` oder die `ANTHROPIC_DEFAULT_*_MODEL`-Umgebungsvariablen bereitstellen, werden unverändert an den Anbieter übergeben und werden nicht durch `modelOverrides` transformiert.

`modelOverrides` funktioniert zusammen mit `availableModels`. Die Allowlist wird gegen die Anthropic-Modell-ID ausgewertet, nicht gegen den Überschreibungswert, daher ein Eintrag wie `"opus"` in `availableModels` weiterhin übereinstimmt, auch wenn Opus-Versionen ARNs zugeordnet sind.

Prompt-Caching-Konfiguration

Claude Code verwendet automatisch [Prompt Caching](#), um die Leistung zu optimieren und Kosten zu senken. Sie können Prompt Caching global oder für bestimmte Modell-Tiers deaktivieren:

Umgebungsvariable	Beschreibung
<code>DISABLE_PROMPT_CACHING</code>	Setzen Sie auf <code>1</code> , um Prompt Caching für alle Modelle zu deaktivieren (hat Vorrang vor modellspezifischen Einstellungen)
<code>DISABLE_PROMPT_CACHING_HAIKU</code>	Setzen Sie auf <code>1</code> , um Prompt Caching nur für Haiku-Modelle zu deaktivieren

Umgebungsvariable	Beschreibung
<code>DISABLE_PROMPT_CACHING_SONNET</code>	Setzen Sie auf <code>1</code> , um Prompt Caching nur für Sonnet-Modelle zu deaktivieren
<code>DISABLE_PROMPT_CACHING_OPUS</code>	Setzen Sie auf <code>1</code> , um Prompt Caching nur für Opus-Modelle zu deaktivieren

Diese Umgebungsvariablen geben Ihnen eine feinkörnige Kontrolle über das Prompt-Caching-Verhalten. Die globale `DISABLE_PROMPT_CACHING`-Einstellung hat Vorrang vor den modellspezifischen Einstellungen, sodass Sie das gesamte Caching bei Bedarf schnell deaktivieren können. Die modellspezifischen Einstellungen sind nützlich für selektive Kontrolle, z. B. beim Debuggen bestimmter Modelle oder bei der Arbeit mit Cloud-Anbietern, die möglicherweise unterschiedliche Caching-Implementierungen haben.

Ausgabestile

Passen Sie Claude Code für Anwendungsfälle über Softwareentwicklung hinaus an

Ausgabestile ermöglichen es Ihnen, Claude Code als jeden Agententyp zu verwenden, während Sie seine Kernfunktionen wie das Ausführen lokaler Skripte, das Lesen/Schreiben von Dateien und das Nachverfolgen von TODOs beibehalten.

Integrierte Ausgabestile

Der **Standard**-Ausgabestil von Claude Code ist die vorhandene Systemaufforderung, die Ihnen helfen soll, Softwareentwicklungsaufgaben effizient zu bewältigen.

Es gibt zwei zusätzliche integrierte Ausgabestile, die sich auf das Unterrichten der Codebasis und der Funktionsweise von Claude konzentrieren:

- **Explanatory:** Bietet pädagogische „Insights“ zwischen der Unterstützung bei Softwareentwicklungsaufgaben. Hilft Ihnen, Implementierungsentscheidungen und Codebase-Muster zu verstehen.
- **Learning:** Kollaborativer, Lern-durch-Tun-Modus, in dem Claude nicht nur „Insights“ beim Codieren teilt, sondern Sie auch auffordert, kleine, strategische Codestücke selbst beizutragen. Claude Code fügt **TODO(human)** -Marker in Ihren Code ein, damit Sie diese implementieren können.

Wie Ausgabestile funktionieren

Ausgabestile ändern direkt die Systemaufforderung von Claude Code.

- Alle Ausgabestile schließen Anweisungen für effiziente Ausgabe aus (z. B. prägnante Antworten).
- Benutzerdefinierte Ausgabestile schließen Anweisungen zum Codieren aus (z. B. Überprüfung von Code mit Tests), es sei denn, **keep-coding-instructions** ist true.
- Alle Ausgabestile haben ihre eigenen benutzerdefinierten Anweisungen am Ende der Systemaufforderung hinzugefügt.
- Alle Ausgabestile lösen Erinnerungen für Claude aus, um die Ausgabestil-Anweisungen während des Gesprächs einzuhalten.

Ändern Sie Ihren Ausgabestil

Führen Sie `/config` aus und wählen Sie **Output style**, um einen Stil aus einem Menü auszuwählen. Ihre Auswahl wird in `.claude/settings.local.json` auf der [lokalen Projektebene](#) gespeichert.

Um einen Stil ohne Menü festzulegen, bearbeiten Sie das Feld `outputStyle` direkt in einer Einstellungsdatei:

```
{
  "outputStyle": "Explanatory"
}
```

Da der Ausgabestil in der Systemaufforderung beim Sitzungsstart festgelegt wird, werden Änderungen beim nächsten Start einer neuen Sitzung wirksam. Dies hält die Systemaufforderung während eines Gesprächs stabil, sodass Prompt Caching die Latenz und Kosten reduzieren kann.

Erstellen Sie einen benutzerdefinierten Ausgabestil

Benutzerdefinierte Ausgabestile sind Markdown-Dateien mit Frontmatter und dem Text, der zur Systemaufforderung hinzugefügt wird:

```
---
name: My Custom Style
description:
  A brief description of what this style does, to be displayed to the user
---

## Custom Style Instructions

You are an interactive CLI tool that helps users with software engineering
tasks. [Your custom instructions here...]

### Specific Behaviors

[Define how the assistant should behave in this style...]
```

Sie können diese Dateien auf Benutzerebene (`~/ .claude/output-styles`) oder Projektebene (`.claude/output-styles`) speichern.

Frontmatter

Ausgabestil-Dateien unterstützen Frontmatter zum Angeben von Metadaten:

Frontmatter	Zweck	Standard
<code>name</code>	Name des Ausgabestils, falls nicht der Dateiname	Wird vom Dateinamen ge- erbt
<code>description</code>	Beschreibung des Ausgabestils, angezeigt in der <code>/config</code> -Auswahl	Keine
<code>keep-coding- instructions</code>	Ob die Teile der Systemaufforderung von Claude Code bezüglich Codie- rung beibehalten werden sollen.	false

Vergleiche mit verwandten Funktionen

Ausgabestile vs. CLAUDE.md vs. --append-system-prompt

Ausgabestile schalten die Teile der Standard-Systemaufforderung von Claude Code, die spezifisch für Softwareentwicklung sind, vollständig aus. Weder CLAUDE.md noch `--append-system-prompt` bearbeiten die Standard-Systemaufforderung von Claude Code. CLAUDE.md fügt den Inhalt als Benutzernachricht *nach* der Standard-Systemaufforderung von Claude Code hinzu. `--append-system-prompt` hängt den Inhalt an die Systemaufforderung an.

Ausgabestile vs. Agents

Ausgabestile beeinflussen direkt die Hauptagentenschleife und beeinflussen nur die Systemaufforderung. Agents werden aufgerufen, um bestimmte Aufgaben zu bewältigen, und können zusätzliche Einstellungen wie das zu verwendende Modell, die verfügbaren Tools und einen Kontext darüber enthalten, wann der Agent verwendet werden soll.

Ausgabestile vs. Skills

Ausgabestile ändern, wie Claude antwortet (Formatierung, Ton, Struktur), und sind immer aktiv, sobald sie ausgewählt sind. Skills sind aufgabenspezifische Aufforderungen, die Sie mit `/skill-name` aufrufen oder die Claude automatisch lädt, wenn relevant. Verwenden Sie Ausgabestile für konsistente Formatierungspräferenzen; verwenden Sie Skills für wiederverwendbare Workflows und Aufgaben.

Beschleunigen Sie Antworten mit dem Schnellmodus

Erhalten Sie schnellere Opus 4.6-Antworten in Claude Code durch Aktivierung des Schnellmodus.

Note:

Der Schnellmodus befindet sich in [Forschungsvorschau](#). Die Funktion, Preisgestaltung und Verfügbarkeit können sich basierend auf Feedback ändern.

Der Schnellmodus ist eine Hochgeschwindigkeitskonfiguration für Claude Opus 4.6, die das Modell 2,5x schneller macht, allerdings zu höheren Kosten pro Token. Aktivieren Sie ihn mit `/fast`, wenn Sie Geschwindigkeit für interaktive Arbeiten wie schnelle Iteration oder Live-Debugging benötigen, und deaktivieren Sie ihn, wenn Kosten wichtiger sind als Latenz.

Der Schnellmodus ist kein anderes Modell. Er verwendet denselben Opus 4.6 mit einer anderen API-Konfiguration, die Geschwindigkeit über Kosteneffizienz priorisiert. Sie erhalten identische Qualität und Funktionen, nur schnellere Antworten.

Note:

Der Schnellmodus erfordert Claude Code v2.1.36 oder später. Überprüfen Sie Ihre Version mit `claude --version`.

Was Sie wissen sollten:

- Verwenden Sie `/fast`, um den Schnellmodus in Claude Code CLI ein- oder auszuschalten. Auch über `/fast` in der Claude Code VS Code Extension verfügbar.
- Die Preisgestaltung für Schnellmodus auf Opus 4.6 beginnt bei \$30/150 MTok. Der Schnellmodus ist bis 23:59 Uhr PT am 16. Februar mit 50 % Rabatt für alle Pläne verfügbar.
- Verfügbar für alle Claude Code-Benutzer auf Abonnementplänen (Pro/Max/Team/Enterprise) und Claude Console.

- Für Claude Code-Benutzer auf Abonnementplänen (Pro/Max/Team/Enterprise) ist der Schnellmodus nur über zusätzliche Nutzung verfügbar und nicht in den Abonnement-Ratenlimits enthalten.

Diese Seite behandelt, wie Sie [Schnellmodus umschalten](#), seine [Kostenabwägung](#), [wann Sie ihn verwenden](#), [Anforderungen](#), [Opt-in pro Sitzung](#) und [Ratenlimit-Verhalten](#).

Schnellmodus umschalten

Schalten Sie den Schnellmodus auf eine dieser Weisen um:

- Geben Sie `/fast` ein und drücken Sie Tab, um ihn ein- oder auszuschalten
- Setzen Sie `"fastMode": true` in Ihrer [Benutzereinstellungsdatei](#)

Standardmäßig bleibt der Schnellmodus über Sitzungen hinweg erhalten. Administratoren können den Schnellmodus so konfigurieren, dass er sich bei jeder Sitzung zurückgesetzt wird. Siehe [Opt-in pro Sitzung erforderlich](#) für Details.

Für die beste Kosteneffizienz aktivieren Sie den Schnellmodus am Anfang einer Sitzung, anstatt ihn mitten in einem Gespräch zu wechseln. Siehe [Kostenabwägung verstehen](#) für Details.

Wenn Sie den Schnellmodus aktivieren:

- Wenn Sie sich auf einem anderen Modell befinden, wechselt Claude Code automatisch zu Opus 4.6
- Sie sehen eine Bestätigungsmeldung: „Fast mode ON“
- Ein kleines `f`-Symbol erscheint neben der Eingabeaufforderung, während der Schnellmodus aktiv ist
- Führen Sie `/fast` jederzeit erneut aus, um zu überprüfen, ob der Schnellmodus aktiviert oder deaktiviert ist

Wenn Sie den Schnellmodus mit `/fast` erneut deaktivieren, bleiben Sie auf Opus 4.6. Das Modell wird nicht auf Ihr vorheriges Modell zurückgesetzt. Um zu einem anderen Modell zu wechseln, verwenden Sie `/model`.

Kostenabwägung verstehen

Der Schnellmodus hat höhere Pro-Token-Preise als Standard-Opus 4.6:

Modus	Eingabe (MTok)	Ausgabe (MTok)
Schnellmodus auf Opus 4.6 (<200K)	\$30	\$150

Modus	Eingabe (MTok)	Ausgabe (MTok)
Schnellmodus auf Opus 4.6 (>200K)	\$60	\$225

Der Schnellmodus ist mit dem erweiterten Kontextfenster mit 1M Token kompatibel.

Wenn Sie mitten in einem Gespräch in den Schnellmodus wechseln, zahlen Sie den vollständigen Schnellmodus-Preis für nicht zwischengespeicherte Eingabe-Token für den gesamten Gesprächskontext. Dies kostet mehr, als wenn Sie den Schnellmodus von Anfang an aktiviert hätten.

Entscheiden Sie, wann Sie den Schnellmodus verwenden

Der Schnellmodus ist am besten für interaktive Arbeiten geeignet, bei denen die Antwortlatenz wichtiger ist als die Kosten:

- Schnelle Iteration bei Code-Änderungen
- Live-Debugging-Sitzungen
- Zeitkritische Arbeiten mit engen Fristen

Der Standardmodus ist besser für:

- Lange autonome Aufgaben, bei denen Geschwindigkeit weniger wichtig ist
- Batch-Verarbeitung oder CI/CD-Pipelines
- Kostenempfindliche Arbeitslasten

Schnellmodus vs. Anstrengungsstufe

Der Schnellmodus und die Anstrengungsstufe beeinflussen beide die Antwortgeschwindigkeit, aber auf unterschiedliche Weise:

Einstellung	Effekt
Schnellmodus	Gleiche Modellqualität, niedrigere Latenz, höhere Kosten
Niedrigere Anstrengungsstufe	Weniger Denkzeit, schnellere Antworten, möglicherweise niedrigere Qualität bei komplexen Aufgaben

Sie können beide kombinieren: Verwenden Sie den Schnellmodus mit einer niedrigeren [Anstrengungsstufe](#) für maximale Geschwindigkeit bei einfachen Aufgaben.

Anforderungen

Der Schnellmodus erfordert alle folgenden Voraussetzungen:

- **Nicht auf Cloud-Anbietern von Drittanbietern verfügbar:** Der Schnellmodus ist nicht auf Amazon Bedrock, Google Vertex AI oder Microsoft Azure Foundry verfügbar. Der Schnellmodus ist über die Anthropic Console API und für Claude-Abonnementpläne mit zusätzlicher Nutzung verfügbar.
- **Zusätzliche Nutzung aktiviert:** Ihr Konto muss zusätzliche Nutzung aktiviert haben, die eine Abrechnung über die in Ihrem Plan enthaltene Nutzung hinaus ermöglicht. Aktivieren Sie dies für einzelne Konten in Ihren [Console-Abrechnungseinstellungen](#). Für Teams und Enterprise muss ein Administrator die zusätzliche Nutzung für die Organisation aktivieren.

Note:

Die Nutzung des Schnellmodus wird direkt zur zusätzlichen Nutzung abgerechnet, auch wenn Sie noch Nutzung in Ihrem Plan haben. Dies bedeutet, dass Schnellmodus-Token nicht gegen die in Ihrem Plan enthaltene Nutzung angerechnet werden und vom ersten Token an zum Schnellmodus-Tarif berechnet werden.

- **Admin-Aktivierung für Teams und Enterprise:** Der Schnellmodus ist standardmäßig für Teams- und Enterprise-Organisationen deaktiviert. Ein Administrator muss den Schnellmodus explizit [aktivieren](#), bevor Benutzer darauf zugreifen können.

Note:

Wenn Ihr Administrator den Schnellmodus für Ihre Organisation nicht aktiviert hat, zeigt der Befehl `/fast` „Fast mode has been disabled by your organization.“ an.

Schnellmodus für Ihre Organisation aktivieren

Administratoren können den Schnellmodus aktivieren in:

- **Console** (API-Kunden): [Claude Code-Einstellungen](#)
- **Claude AI** (Teams und Enterprise): [Admin-Einstellungen > Claude Code](#)

Eine weitere Option zum vollständigen Deaktivieren des Schnellmodus ist das Setzen von `CLAUDE_CODE_DISABLE_FAST_MODE=1`. Siehe [Umgebungsvariablen](#).

Opt-in pro Sitzung erforderlich

Standardmäßig bleibt der Schnellmodus über Sitzungen hinweg erhalten: Wenn ein Benutzer den Schnellmodus aktiviert, bleibt er in zukünftigen Sitzungen aktiviert. Administratoren auf [Teams](#)- oder [Enterprise](#)-Plänen können dies verhindern, indem sie `fastModePerSessionOptIn` in [verwalteten Einstellungen](#) oder [servergesteuerten Einstellungen](#) auf `true` setzen. Dies führt dazu, dass jede Sitzung mit deaktiviertem Schnellmodus beginnt und Benutzer ihn explizit mit `/fast` aktivieren müssen.

```
{
  "fastModePerSessionOptIn": true
}
```

Dies ist nützlich zur Kostenkontrolle in Organisationen, in denen Benutzer mehrere gleichzeitige Sitzungen ausführen. Benutzer können den Schnellmodus immer noch mit `/fast` aktivieren, wenn sie Geschwindigkeit benötigen, aber er wird zu Beginn jeder neuen Sitzung zurückgesetzt. Die Schnellmodus-Einstellung des Benutzers wird immer noch gespeichert, sodass das Entfernen dieser Einstellung das standardmäßige persistente Verhalten wiederherstellt.

Ratenlimits handhaben

Der Schnellmodus hat separate Ratenlimits vom Standard-Opus 4.6. Wenn Sie das Ratenlimit des Schnellmodus erreichen oder keine zusätzlichen Nutzungsguthaben mehr haben:

1. Der Schnellmodus fällt automatisch auf Standard-Opus 4.6 zurück
2. Das `₪`-Symbol wird grau, um eine Abkühlung anzuzeigen
3. Sie arbeiten weiterhin mit Standard-Geschwindigkeit und -Preisen
4. Wenn die Abkühlung abläuft, wird der Schnellmodus automatisch wieder aktiviert

Um den Schnellmodus manuell zu deaktivieren, anstatt auf die Abkühlung zu warten, führen Sie `/fast` erneut aus.

Forschungsvorschau

Der Schnellmodus ist eine Forschungsvorschau-Funktion. Dies bedeutet:

- Die Funktion kann sich basierend auf Feedback ändern
- Verfügbarkeit und Preisgestaltung können sich ändern
- Die zugrunde liegende API-Konfiguration kann sich weiterentwickeln

Melden Sie Probleme oder Feedback über Ihre üblichen Anthropic-Supportkanäle.

Siehe auch

- [Modellkonfiguration](#): Wechseln Sie Modelle und passen Sie Anstrengungsstufen an
- [Kosten effektiv verwalten](#): Verfolgen Sie die Token-Nutzung und reduzieren Sie Kosten
- [Statusleisten-Konfiguration](#): Zeigen Sie Modell- und Kontextinformationen an

Kosten effektiv verwalten

Verfolgen Sie die Token-Nutzung, legen Sie Ausgabenlimits für Teams fest und reduzieren Sie Claude Code-Kosten durch Kontextverwaltung, Modellauswahl, Einstellungen für erweitertes Denken und Preprocessing-Hooks.

Claude Code verbraucht Token für jede Interaktion. Die Kosten variieren je nach Codebasis-Größe, Abfragekomplexität und Gesprächslänge. Die durchschnittlichen Kosten betragen 6 USD pro Entwickler pro Tag, wobei die täglichen Kosten für 90 % der Benutzer unter 12 USD bleiben.

Für die Team-Nutzung werden Claude Code-Gebühren nach API-Token-Verbrauch berechnet. Im Durchschnitt kostet Claude Code etwa 100–200 USD pro Entwickler pro Monat mit Sonnet 4.6, es gibt jedoch große Unterschiede je nachdem, wie viele Instanzen Benutzer ausführen und ob sie diese in der Automatisierung verwenden.

Diese Seite behandelt, wie Sie [Ihre Kosten verfolgen](#), [Kosten für Teams verwalten](#) und [Token-Nutzung reduzieren](#).

Verfolgen Sie Ihre Kosten

Verwenden des `/cost`-Befehls

Note:

Der `/cost`-Befehl zeigt die API-Token-Nutzung an und ist für API-Benutzer vorgesehen. Claude Max und Pro-Abonnenten haben die Nutzung in ihrem Abonnement enthalten, daher sind `/cost`-Daten nicht relevant für Abrechnungszwecke. Abonnenten können `/stats` verwenden, um Nutzungsmuster anzuzeigen.

Der `/cost`-Befehl bietet detaillierte Token-Nutzungsstatistiken für Ihre aktuelle Sitzung:

```
Total cost:           $0.55
Total duration (API):  6m 19.7s
Total duration (wall): 6h 33m 10.2s
Total code changes:    0 lines added, 0 lines removed
```

Verwalten Sie Kosten für Teams

Bei Verwendung der Claude API können Sie [Workspace-Ausgabenlimits festlegen](#) für die gesamten Claude Code-Workspace-Ausgaben. Administratoren können [Kosten- und Nutzungsberichte anzeigen](#) in der Konsole.

Note:

Wenn Sie Claude Code zum ersten Mal mit Ihrem Claude Console-Konto authentifizieren, wird automatisch ein Workspace namens „Claude Code“ für Sie erstellt. Dieser Workspace bietet zentrale Kostenverfolgung und Verwaltung für alle Claude Code-Nutzung in Ihrer Organisation. Sie können keine API-Schlüssel für diesen Workspace erstellen; er ist ausschließlich für Claude Code-Authentifizierung und -Nutzung.

Bei Bedrock, Vertex und Foundry sendet Claude Code keine Metriken aus Ihrer Cloud. Um Kostenmetriken zu erhalten, berichteten mehrere große Unternehmen von der Verwendung von [LiteLLM](#), einem Open-Source-Tool, das Unternehmen hilft, [Ausgaben nach Schlüssel zu verfolgen](#). Dieses Projekt ist nicht mit Anthropic verbunden und wurde nicht auf Sicherheit überprüft.

Empfehlungen für Ratenlimits

Beim Einrichten von Claude Code für Teams sollten Sie diese Token Pro Minute (TPM) und Anfragen Pro Minute (RPM) pro Benutzer-Empfehlungen basierend auf Ihrer Organisationsgröße berücksichtigen:

Team-Größe	TPM pro Benutzer	RPM pro Benutzer
1–5 Benutzer	200.000–300.000	5–7
5–20 Benutzer	100.000–150.000	2,5–3,5
20–50 Benutzer	50.000–75.000	1,25–1,75
50–100 Benutzer	25.000–35.000	0,62–0,87
100–500 Benutzer	15.000–20.000	0,37–0,47
500+ Benutzer	10.000–15.000	0,25–0,35

Wenn Sie beispielsweise 200 Benutzer haben, könnten Sie 20.000 TPM für jeden Benutzer anfordern, oder insgesamt 4 Millionen TPM ($200 \times 20.000 = 4 \text{ Millionen}$).

Die TPM pro Benutzer sinkt mit zunehmender Team-Größe, da in größeren Organisationen weniger Benutzer Claude Code gleichzeitig verwenden. Diese Ratenlimits gelten auf Organisationsebene, nicht pro individuellem Benutzer, was bedeutet, dass einzelne Benutzer vorübergehend mehr als ihren berechneten Anteil verbrauchen können, wenn andere den Service nicht aktiv nutzen.

Note:

Wenn Sie Szenarien mit ungewöhnlich hoher gleichzeitiger Nutzung erwarten (z. B. Live-Schulungssitzungen mit großen Gruppen), benötigen Sie möglicherweise höhere TPM-Zuordnungen pro Benutzer.

Token-Kosten für Agent-Teams

[Agent-Teams](#) starten mehrere Claude Code-Instanzen, jede mit ihrem eigenen Kontextfenster. Die Token-Nutzung skaliert mit der Anzahl der aktiven Teammates und wie lange jeder läuft.

Um Agent-Team-Kosten überschaubar zu halten:

- Verwenden Sie Sonnet für Teammates. Es bietet ein Gleichgewicht zwischen Fähigkeit und Kosten für Koordinationsaufgaben.
- Halten Sie Teams klein. Jeder Teammate führt sein eigenes Kontextfenster aus, daher ist die Token-Nutzung ungefähr proportional zur Team-Größe.
- Halten Sie Spawn-Prompts fokussiert. Teammates laden CLAUDE.md, MCP-Server und Skills automatisch, aber alles im Spawn-Prompt trägt von Anfang an zu ihrem Kontext bei.
- Bereinigen Sie Teams, wenn die Arbeit erledigt ist. Aktive Teammates verbrauchen weiterhin Token, auch wenn sie untätig sind.
- Agent-Teams sind standardmäßig deaktiviert. Setzen Sie `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` in Ihrer [settings.json](#) oder Umgebung, um sie zu aktivieren. Siehe [Agent-Teams aktivieren](#).

Reduzieren Sie die Token-Nutzung

Token-Kosten skalieren mit der Kontextgröße: Je mehr Kontext Claude verarbeitet, desto mehr Token verwenden Sie. Claude Code optimiert Kosten automatisch durch Prompt Caching (das Kosten für wiederholte Inhalte wie Systemprompts reduziert) und Auto-Compaction (das Gesprächsverlauf zusammenfasst, wenn sich dem Kontextlimit genähert wird).

Die folgenden Strategien helfen Ihnen, den Kontext klein zu halten und die Kosten pro Nachricht zu reduzieren.

Verwalten Sie den Kontext proaktiv

Verwenden Sie `/cost`, um Ihre aktuelle Token-Nutzung zu überprüfen, oder [konfigurieren Sie Ihre Statuszeile](#), um sie kontinuierlich anzuzeigen.

- **Zwischen Aufgaben löschen:** Verwenden Sie `/clear`, um neu zu beginnen, wenn Sie zu nicht verwandter Arbeit wechseln. Veralteter Kontext verschwendet Token bei jeder nachfolgenden Nachricht. Verwenden Sie `/rename` vor dem Löschen, damit Sie die Sitzung später leicht finden können, dann `/resume`, um zu ihr zurückzukehren.
- **Fügen Sie benutzerdefinierte Compaction-Anweisungen hinzu:** `/compact Focus on code samples and API usage` teilt Claude mit, was während der Zusammenfassung beibehalten werden soll.

Sie können das Compaction-Verhalten auch in Ihrer CLAUDE.md anpassen:

```
## Compact instructions
```

```
When you are using compact, please focus on test output and code changes
```

Wählen Sie das richtige Modell

Sonnet bewältigt die meisten Codierungsaufgaben gut und kostet weniger als Opus. Reservieren Sie Opus für komplexe architektonische Entscheidungen oder mehrstufiges Denken. Verwenden Sie `/model`, um Modelle während einer Sitzung zu wechseln, oder legen Sie einen Standard in `/config` fest. Für einfache Subagent-Aufgaben geben Sie `model: haiku` in Ihrer [Subagent-Konfiguration](#) an.

Reduzieren Sie den MCP-Server-Overhead

Jeder MCP-Server fügt Tool-Definitionen zu Ihrem Kontext hinzu, auch wenn er untätig ist. Führen Sie `/context` aus, um zu sehen, was Platz verbraucht.

- **Bevorzugen Sie CLI-Tools, wenn verfügbar:** Tools wie `gh`, `aws`, `gcloud` und `sentry-cli` sind kontexteffektiver als MCP-Server, da sie keine persistenten Tool-Definitionen hinzufügen. Claude kann CLI-Befehle direkt ausführen, ohne den Overhead.
- **Deaktivieren Sie ungenutzte Server:** Führen Sie `/mcp` aus, um konfigurierte Server anzuzeigen und alle zu deaktivieren, die Sie nicht aktiv verwenden.

- **Tool-Suche ist automatisch:** Wenn MCP-Tool-Beschreibungen 10 % Ihres Kontextfensters überschreiten, verschiebt Claude Code sie automatisch und lädt Tools bei Bedarf über [Tool-Suche](#). Da verschobene Tools nur in den Kontext eintreten, wenn sie tatsächlich verwendet werden, bedeutet ein niedrigerer Schwellenwert weniger unnütze Tool-Definitionen, die Platz verbrauchen. Legen Sie einen niedrigeren Schwellenwert mit `ENABLE_TOOL_SEARCH=auto:<N>` fest (z. B. `auto:5` wird ausgelöst, wenn Tools 5 % Ihres Kontextfensters überschreiten).

Installieren Sie Code-Intelligence-Plugins für typisierte Sprachen

[Code-Intelligence-Plugins](#) geben Claude präzise Symbol-Navigation statt textbasierter Suche, wodurch unnötige Dateileser beim Erkunden unbekannten Codes reduziert werden. Ein einzelner „Gehe zu Definition“-Aufruf ersetzt, was sonst ein Grep gefolgt vom Lesen mehrerer Kandidatendateien sein könnte. Installierte Sprachserver melden auch Typfehler automatisch nach Bearbeitungen, sodass Claude Fehler erkennt, ohne einen Compiler auszuführen.

Verlagern Sie die Verarbeitung auf Hooks und Skills

Benutzerdefinierte [Hooks](#) können Daten vorverarbeiten, bevor Claude sie sieht. Anstatt dass Claude eine 10.000-Zeilen-Protokolldatei liest, um Fehler zu finden, kann ein Hook nach `ERROR` suchen und nur übereinstimmende Zeilen zurückgeben, wodurch der Kontext von Zehntausenden Token auf Hunderte reduziert wird.

Ein [Skill](#) kann Claude Domänenwissen geben, sodass es nicht erkunden muss. Beispielsweise könnte ein „codebase-overview“-Skill die Architektur Ihres Projekts, wichtige Verzeichnisse und Namenskonventionen beschreiben. Wenn Claude den Skill aufruft, erhält es diesen Kontext sofort, anstatt Token zu verschwenden, um mehrere Dateien zu lesen, um die Struktur zu verstehen.

Beispielsweise filtert dieser PreToolUse-Hook die Testausgabe, um nur Fehler anzuzeigen:

settings.json

Fügen Sie dies zu Ihrer [settings.json](#) hinzu, um den Hook vor jedem Bash-Befehl auszuführen:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/.claude/hooks/filter-test-output.sh"
          }
        ]
      }
    ]
  }
}
```

filter-test-output.sh

Der Hook ruft dieses Skript auf, das überprüft, ob der Befehl ein Test-Runner ist, und ihn ändert, um nur Fehler anzuzeigen:

```
#!/bin/bash
input=$(cat)
cmd=$(echo "$input" | jq -r '.tool_input.command')

## If running tests, filter to show only failures
if [[ "$cmd" =~ ^(npm test|pytest|go test) ]]; then
  filtered_cmd="$cmd 2>&1 | grep -A 5 -E '(FAIL|ERROR|error:)' | head -100"
  echo "{\"hookSpecificOutput\":{\"hookEventName\":\"PreToolUse\",\"permissionDecision\":\"allow\",\"updatedInput\":{\"command\":\"$filtered_cmd\"}}}"
else
  echo "{}"
fi
```

Verschieben Sie Anweisungen von CLAUDE.md zu Skills

Ihre [CLAUDE.md](#)-Datei wird beim Sitzungsstart in den Kontext geladen. Wenn sie detaillierte Anweisungen für spezifische Workflows enthält (wie PR-Reviews oder Datenbankmigrationen), sind diese Token vorhanden, auch wenn Sie nicht verwandte Arbeit erledigen:

gen. [Skills](#) werden bei Bedarf nur geladen, wenn sie aufgerufen werden, daher hält das Verschieben spezialisierter Anweisungen in Skills Ihren Basis-Kontext kleiner. Streben Sie danach, CLAUDE.md unter etwa 500 Zeilen zu halten, indem Sie nur das Wesentliche einbeziehen.

Passen Sie das erweiterte Denken an

Erweitertes Denken ist standardmäßig mit einem Budget von 31.999 Token aktiviert, da es die Leistung bei komplexen Planungs- und Denkaufgaben erheblich verbessert. Thinking-Token werden jedoch als Output-Token abgerechnet, daher können Sie für einfachere Aufgaben, bei denen tiefes Denken nicht erforderlich ist, Kosten reduzieren, indem Sie die [Anstrengungsstufe](#) in `/model` für Opus 4.6 senken, Denken in `/config` deaktivieren oder das Budget senken (z. B. `MAX_THINKING_TOKENS=8000`).

Delegieren Sie ausführliche Operationen an Subagents

Das Ausführen von Tests, das Abrufen von Dokumentation oder das Verarbeiten von Protokolldateien kann erheblichen Kontext verbrauchen. Delegieren Sie diese an [Subagents](#), sodass die ausführliche Ausgabe im Kontext des Subagent bleibt, während nur eine Zusammenfassung zu Ihrem Hauptgespräch zurückkehrt.

Verwalten Sie Agent-Team-Kosten

Agent-Teams verwenden ungefähr 7-mal mehr Token als Standard-Sitzungen, wenn Teammates im Plan Mode laufen, da jeder Teammate sein eigenes Kontextfenster verwaltet und als separate Claude-Instanz läuft. Halten Sie Team-Aufgaben klein und in sich geschlossen, um die Token-Nutzung pro Teammate zu begrenzen. Siehe [Agent-Teams](#) für Details.

Schreiben Sie spezifische Prompts

Vage Anfragen wie „Verbessern Sie diese Codebasis“ lösen breites Scannen aus. Spezifische Anfragen wie „Fügen Sie Eingabevalidierung zur Login-Funktion in auth.ts hinzu“ ermöglichen es Claude, effizient mit minimalen Dateileser zu arbeiten.

Arbeiten Sie effizient an komplexen Aufgaben

Für längere oder komplexere Arbeiten helfen diese Gewohnheiten, verschwendete Token durch das Gehen des falschen Weges zu vermeiden:

- **Verwenden Sie Plan Mode für komplexe Aufgaben:** Drücken Sie Shift+Tab, um [Plan Mode](#) vor der Implementierung zu betreten. Claude erkundet die Codebasis und schlägt einen Ansatz zur Genehmigung vor, was teure Überarbeitungen verhindert, wenn die anfängliche Richtung falsch ist.

- **Korrigieren Sie den Kurs früh:** Wenn Claude in die falsche Richtung geht, drücken Sie `Escape`, um sofort zu stoppen. Verwenden Sie `/rewind` oder doppelklicken Sie `Escape`, um das Gespräch und den Code zu einem vorherigen Checkpoint wiederherzustellen.
- **Geben Sie Verifizierungsziele an:** Fügen Sie Testfälle ein, fügen Sie Screenshots ein oder definieren Sie erwartete Ausgabe in Ihrem Prompt. Wenn Claude seine eigene Arbeit verifizieren kann, erkennt es Probleme, bevor Sie Korrekturen anfordern müssen.
- **Testen Sie schrittweise:** Schreiben Sie eine Datei, testen Sie sie, dann fahren Sie fort. Dies erkennt Probleme früh, wenn sie billig zu beheben sind.

Hintergrund-Token-Nutzung

Claude Code verwendet Token für einige Hintergrund-Funktionalität, auch wenn untätig:

- **Gesprächszusammenfassung:** Hintergrund-Jobs, die vorherige Gespräche für die `claude --resume`-Funktion zusammenfassen
- **Befehlsverarbeitung:** Einige Befehle wie `/cost` können Anfragen generieren, um den Status zu überprüfen

Diese Hintergrund-Prozesse verbrauchen eine kleine Menge Token (typischerweise unter 0,04 USD pro Sitzung), auch ohne aktive Interaktion.

Verstehen Sie Änderungen im Claude Code-Verhalten

Claude Code erhält regelmäßig Updates, die ändern können, wie Funktionen funktionieren, einschließlich Kostenberichterstattung. Führen Sie `claude --version` aus, um Ihre aktuelle Version zu überprüfen. Für spezifische Abrechnungsfragen kontaktieren Sie den Anthropic-Support über Ihr [Console-Konto](#). Für Team-Bereitstellungen beginnen Sie mit einer kleinen Pilotgruppe, um Nutzungsmuster zu etablieren, bevor Sie einen breiteren Rollout durchführen.

Part 5: Extensibility

Hooks-Referenz

Referenz für Claude Code Hook-Ereignisse, Konfigurationsschema, JSON-Ein-/Ausgabeformate, Exit-Codes, asynchrone Hooks, HTTP-Hooks, Prompt-Hooks und MCP-Tool-Hooks.

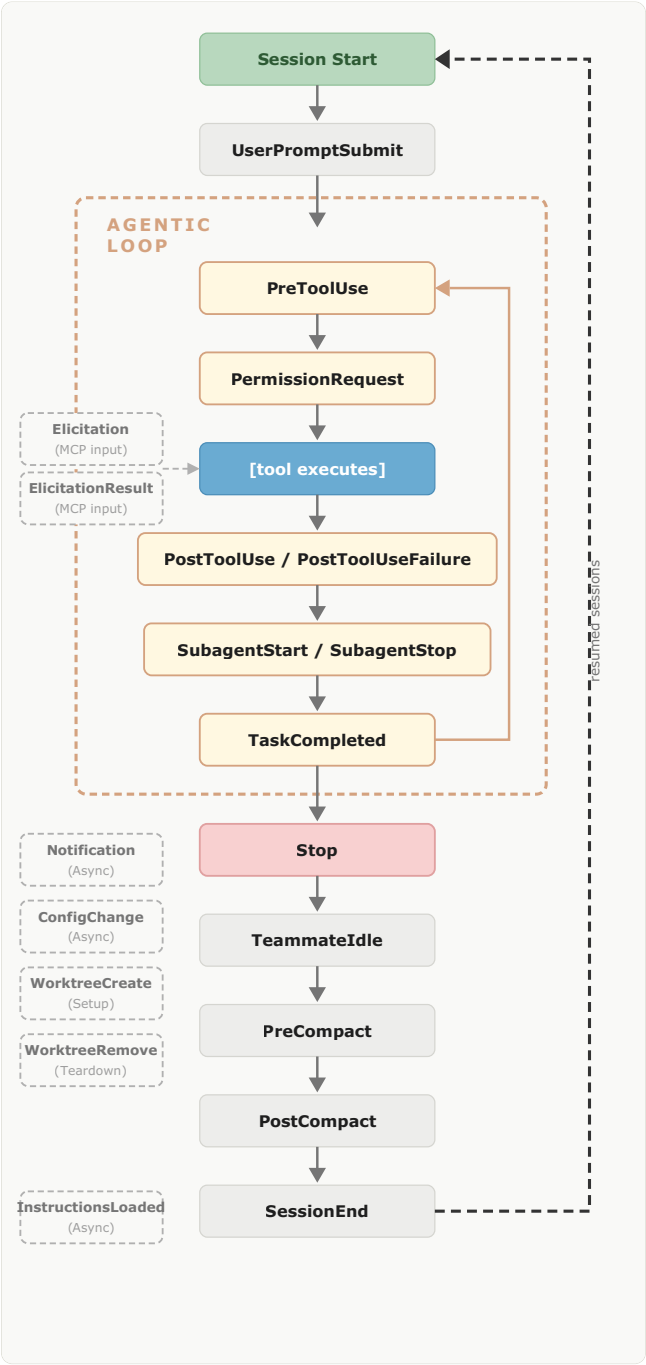
Tip:

Eine Schnellstartanleitung mit Beispielen finden Sie unter [Workflows mit Hooks automatisieren](#).

Hooks sind benutzerdefinierte Shell-Befehle, HTTP-Endpunkte oder LLM-Prompts, die automatisch an bestimmten Punkten im Lebenszyklus von Claude Code ausgeführt werden. Verwenden Sie diese Referenz, um Ereignisschemas, Konfigurationsoptionen, JSON-Ein-/Ausgabeformate und erweiterte Funktionen wie asynchrone Hooks, HTTP-Hooks und MCP-Tool-Hooks nachzuschlagen. Wenn Sie Hooks zum ersten Mal einrichten, beginnen Sie stattdessen mit der [Anleitung](#).

Hook-Lebenszyklus

Hooks werden an bestimmten Punkten während einer Claude Code-Sitzung ausgelöst. Wenn ein Ereignis ausgelöst wird und ein Matcher passt, übergibt Claude Code JSON-Kontext über das Ereignis an Ihren Hook-Handler. Bei Command-Hooks kommt die Eingabe über stdin an. Bei HTTP-Hooks kommt sie als POST-Request-Body an. Ihr Handler kann dann die Eingabe überprüfen, Maßnahmen ergreifen und optional eine Entscheidung zurückgeben. Einige Ereignisse werden einmal pro Sitzung ausgelöst, während andere wiederholt in der agentengesteuerten Schleife ausgelöst werden:



Hook-Lebenszyklus-Diagramm, das die Abfolge von Hooks von SessionStart durch die agentengesteuerte Schleife bis SessionEnd zeigt, mit WorktreeCreate, WorktreeRemove und InstructionsLoaded als eigenständige asynchrone Ereignisse

Die folgende Tabelle fasst zusammen, wann jedes Ereignis ausgelöst wird. Der Abschnitt [Hook-Ereignisse](#) dokumentiert das vollständige Eingabeschema und die Optionen zur Entscheidungskontrolle für jedes Ereignis.

Event	When it fires
SessionStart	When a session begins or resumes
UserPromptSubmit	When you submit a prompt, before Claude processes it
PreToolUse	Before a tool call executes. Can block it
PermissionRequest	When a permission dialog appears
PostToolUse	After a tool call succeeds
PostToolUseFailure	After a tool call fails
Notification	When Claude Code sends a notification
SubagentStart	When a subagent is spawned
SubagentStop	When a subagent finishes
Stop	When Claude finishes responding
TeammateIdle	When an agent team teammate is about to go idle
TaskCompleted	When a task is being marked as completed
InstructionsLoaded	When a CLAUDE.md or <code>.claude/rules/*.md</code> file is loaded into context. Fires at session start and when files are lazily loaded during a session
ConfigChange	When a configuration file changes during a session
WorktreeCreate	When a worktree is being created via <code>--worktree</code> or <code>isolation: "worktree"</code> . Replaces default git behavior
WorktreeRemove	When a worktree is being removed, either at session exit or when a subagent finishes
PreCompact	Before context compaction

Event	When it fires
<code>PostCompact</code>	After context compaction completes
<code>Elicitation</code>	When an MCP server requests user input during a tool call
<code>ElicitationResult</code>	After a user responds to an MCP elicitation, before the response is sent back to the server
<code>SessionEnd</code>	When a session terminates

Wie ein Hook aufgelöst wird

Um zu sehen, wie diese Teile zusammenpassen, betrachten Sie diesen `PreToolUse`-Hook, der destruktive Shell-Befehle blockiert. Der Hook führt `block-rm.sh` vor jedem Bash-Tool-Aufruf aus:

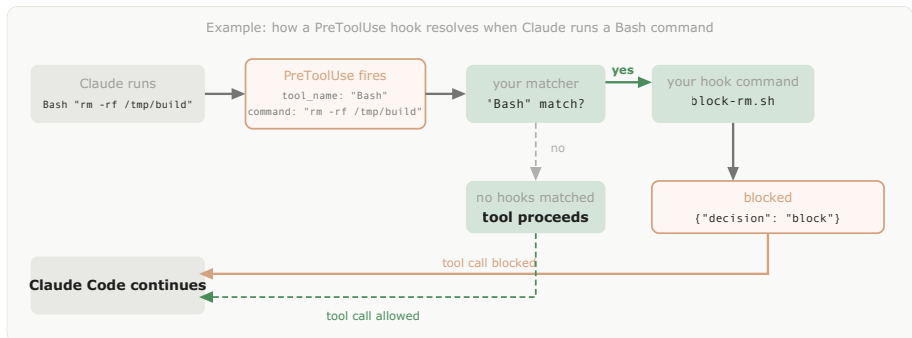
```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": ".claude/hooks/block-rm.sh"
          }
        ]
      }
    ]
  }
}
```

Das Skript liest die JSON-Eingabe von stdin, extrahiert den Befehl und gibt eine `permissionDecision` von `"deny"` zurück, wenn es `rm -rf` enthält:

```
#!/bin/bash
## .claude/hooks/block-rm.sh
COMMAND=$(jq -r '.tool_input.command')

if echo "$COMMAND" | grep -q 'rm -rf'; then
  jq -n '{
    hookSpecificOutput: {
      hookEventName: "PreToolUse",
      permissionDecision: "deny",
      permissionDecisionReason: "Destructive command blocked by hook"
    }
  }'
else
  exit 0 # allow the command
fi
```

Angenommen, Claude Code entscheidet sich, `Bash "rm -rf /tmp/build"` auszuführen. Hier ist, was passiert:



Hook-Auflösungsfluss: PreToolUse-Ereignis wird ausgelöst, Matcher prüft auf Bash-Übereinstimmung, Hook-Handler wird ausgeführt, Ergebnis wird an Claude Code zurückgegeben

Step 1: Ereignis wird ausgelöst

Das `PreToolUse`-Ereignis wird ausgelöst. Claude Code sendet die Tool-Eingabe als JSON über stdin an den Hook:

```
{ "tool_name": "Bash", "tool_input": { "command": "rm -rf /tmp/build" }, ... }
```

Step 2: Matcher prüft

Der Matcher `"Bash"` passt zum Tool-Namen, daher wird `block-rm.sh` ausgeführt. Wenn Sie den Matcher weglassen oder `"*"` verwenden, wird der Hook bei jedem Auftreten des Ereignisses ausgeführt. Hooks werden nur übersprungen, wenn ein Matcher definiert ist und nicht passt.

Step 3: Hook-Handler wird ausgeführt

Das Skript extrahiert `"rm -rf /tmp/build"` aus der Eingabe und findet `rm -rf`, daher gibt es eine Entscheidung auf stdout aus:

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Destructive command blocked by hook"
  }
}
```

Wenn der Befehl sicher gewesen wäre (wie `npm test`), würde das Skript stattdessen `exit 0` treffen, was Claude Code mitteilt, den Tool-Aufruf zuzulassen, ohne weitere Maßnahmen zu ergreifen.

Step 4: Claude Code handelt nach dem Ergebnis

Claude Code liest die JSON-Entscheidung, blockiert den Tool-Aufruf und zeigt Claude den Grund an.

Der Abschnitt [Konfiguration](#) unten dokumentiert das vollständige Schema, und jeder Abschnitt [Hook-Ereignis](#) dokumentiert, welche Eingabe Ihr Befehl erhält und welche Ausgabe er zurückgeben kann.

Konfiguration

Hooks werden in JSON-Einstellungsdateien definiert. Die Konfiguration hat drei Verschachtelungsebenen:

1. Wählen Sie ein [Hook-Ereignis](#) aus, auf das Sie reagieren möchten, wie `PreToolUse` oder `Stop`
2. Fügen Sie eine [Matcher-Gruppe](#) hinzu, um zu filtern, wann es ausgelöst wird, wie 'nur für das Bash-Tool'

3. Definieren Sie einen oder mehrere [Hook-Handler](#), die ausgeführt werden, wenn sie passen

Siehe [Wie ein Hook aufgelöst wird](#) oben für eine vollständige Anleitung mit einem kommentierten Beispiel.

Note:

Diese Seite verwendet spezifische Begriffe für jede Ebene: **Hook-Ereignis** für den Lebenszyklus-Punkt, **Matcher-Gruppe** für den Filter und **Hook-Handler** für den Shell-Befehl, HTTP-Endpunkt, Prompt oder Agent, der ausgeführt wird. „Hook“ allein bezieht sich auf die allgemeine Funktion.

Hook-Speicherorte

Der Ort, an dem Sie einen Hook definieren, bestimmt seinen Umfang:

Speicherort	Umfang	Freigegeben
<code>~/.claude/settings.json</code>	Alle Ihre Projekte	Nein, lokal auf Ihrem Computer
<code>.claude/settings.json</code>	Einzelnes Projekt	Ja, kann im Repo committed werden
<code>.claude/settings.local.json</code>	Einzelnes Projekt	Nein, gitignored
Verwaltete Richtlinien-einstellungen	Organisationsweit	Ja, von Admin kontrolliert
<code>Plugin hooks/hooks.json</code>	Wenn Plugin aktiviert ist	Ja, mit dem Plugin gebündelt
<code>Skill</code> oder <code>Agent</code> Frontmatter	Während die Komponente aktiv ist	Ja, in der Komponenten-datei definiert

Weitere Informationen zur Auflösung von Einstellungsdateien finden Sie unter [Einstellungen](#). Enterprise-Administratoren können `allowManagedHooksOnly` verwenden, um Benutzer-, Projekt- und Plugin-Hooks zu blockieren. Siehe [Hook-Konfiguration](#).

Matcher-Muster

Das Feld `matcher` ist eine Regex-Zeichenkette, die filtert, wann Hooks ausgelöst werden. Verwenden Sie `"*"`, `" "` oder lassen Sie `matcher` ganz weg, um alle Vorkommen zu treffen. Jeder Ereignistyp passt auf ein anderes Feld:

Ereignis	Worauf der Matcher filtert	Beispiel-Matcher-Werte
<code>PreToolUse</code> , <code>PostToolUse</code> , <code>PostToolUseFailure</code> , <code>PermissionRequest</code>	Tool-Name	<code>Bash</code> , <code>Edit Write</code> , <code>mcp_.*</code>
<code>SessionStart</code>	Wie die Sitzung gestartet wurde	<code>startup</code> , <code>resume</code> , <code>clear</code> , <code>compact</code>
<code>SessionEnd</code>	Warum die Sitzung endete	<code>clear</code> , <code>logout</code> , <code>prompt_input_exit</code> , <code>bypass_permissions_disabled</code> , <code>other</code>
<code>Notification</code>	Benachrichtigungstyp	<code>permission_prompt</code> , <code>idle_prompt</code> , <code>auth_success</code> , <code>elicitation_dialog</code>
<code>SubagentStart</code>	Agent-Typ	<code>Bash</code> , <code>Explore</code> , <code>Plan</code> oder benutzerdefinierte Agent-Namen
<code>PreCompact</code>	Was die Komprimierung ausgelöst hat	<code>manual</code> , <code>auto</code>
<code>SubagentStop</code>	Agent-Typ	gleiche Werte wie <code>SubagentStart</code>
<code>ConfigChange</code>	Konfigurationsquelle	<code>user_settings</code> , <code>project_settings</code> , <code>local_settings</code> , <code>policy_settings</code> , <code>skills</code>

Ereignis	Worauf der Matcher filtert	Beispiel-Matcher-Werte
UserPromptSubmit , Stop , TeammateIdle , TaskCompleted , WorktreeCreate , WorktreeRemove , InstructionsLoaded	Keine Matcher-Unterstützung	wird immer bei jedem Auftreten ausgelöst

Der Matcher ist ein Regex, daher passt `Edit|Write` zu beiden Tools und `Notebook.*` passt zu jedem Tool, das mit Notebook beginnt. Der Matcher wird gegen ein Feld aus der [JSON-Eingabe](#) ausgeführt, die Claude Code an Ihren Hook über stdin sendet. Bei Tool-Ereignissen ist dieses Feld `tool_name` . Jeder Abschnitt [Hook-Ereignis](#) listet den vollständigen Satz von Matcher-Werten und das Eingabeschema für dieses Ereignis auf.

Dieses Beispiel führt ein Linting-Skript nur aus, wenn Claude eine Datei schreibt oder bearbeitet:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/lint-check.sh"
          }
        ]
      }
    ]
  }
}
```

`UserPromptSubmit` , `Stop` , `TeammateIdle` , `TaskCompleted` , `WorktreeCreate` , `WorktreeRemove` und `InstructionsLoaded` unterstützen keine Matcher und werden immer bei jedem Auftreten ausgelöst. Wenn Sie ein `matcher` -Feld zu diesen Ereignissen hinzufügen, wird es stillschweigend ignoriert.

MCP-Tools abgleichen

[MCP](#) Server-Tools erscheinen als reguläre Tools in Tool-Ereignissen (`PreToolUse` , `PostToolUse` , `PostToolUseFailure` , `PermissionRequest`), daher können Sie sie auf die gleiche Weise abgleichen wie jeden anderen Tool-Namen.

MCP-Tools folgen dem Benennungsmuster `mcp__<server>__<tool>` , zum Beispiel:

- `mcp__memory__create_entities` : Memory-Server-Tool zum Erstellen von Entitäten
- `mcp__filesystem__read_file` : Filesystem-Server-Tool zum Lesen von Dateien
- `mcp__github__search_repositories` : GitHub-Server-Suchtool

Verwenden Sie Regex-Muster, um bestimmte MCP-Tools oder Gruppen von Tools zu targetieren:

- `mcp__memory__.*` passt zu allen Tools vom `memory` -Server
- `mcp__.*__write.*` passt zu jedem Tool, das „write“ von jedem Server enthält

Dieses Beispiel protokolliert alle Memory-Server-Operationen und validiert Schreibvorgänge von jedem MCP-Server:

```

{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__memory__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Memory operation initiated' >> ~/mcp-operations.log"
          }
        ]
      },
      {
        "matcher": "mcp__.*__write.*",
        "hooks": [
          {
            "type": "command",
            "command": "/home/user/scripts/validate-mcp-write.py"
          }
        ]
      }
    ]
  }
}

```

Hook-Handler-Felder

Jedes Objekt im inneren **hooks**-Array ist ein Hook-Handler: der Shell-Befehl, HTTP-Endpunkt, LLM-Prompt oder Agent, der ausgeführt wird, wenn der Matcher passt. Es gibt vier Typen:

- **Command-Hooks** (**type**: **"command"**): führen einen Shell-Befehl aus. Ihr Skript erhält die [JSON-Eingabe](#) des Ereignisses über stdin und kommuniziert Ergebnisse über Exit-Codes und stdout zurück.
- **HTTP-Hooks** (**type**: **"http"**): senden die [JSON-Eingabe](#) des Ereignisses als HTTP-POST-Request an eine URL. Der Endpunkt kommuniziert Ergebnisse über den Response-Body mit dem gleichen [JSON-Ausgabeformat](#) wie Command-Hooks zurück.

- **Prompt-Hooks** (`type: "prompt"`): senden einen Prompt an ein Claude-Modell zur Einzelrunden-Evaluierung. Das Modell gibt eine Ja/Nein-Entscheidung als JSON zurück. Siehe [Prompt-basierte Hooks](#).
- **Agent-Hooks** (`type: "agent"`): spawnen einen Subagenten, der Tools wie Read, Grep und Glob verwenden kann, um Bedingungen zu überprüfen, bevor eine Entscheidung zurückgegeben wird. Siehe [Agent-basierte Hooks](#).

Gemeinsame Felder

Diese Felder gelten für alle Hook-Typen:

Feld	Erforderlich	Beschreibung
<code>type</code>	ja	<code>"command"</code> , <code>"http"</code> , <code>"prompt"</code> oder <code>"agent"</code>
<code>timeout</code>	nein	Sekunden vor dem Abbruch. Standardwerte: 600 für Command, 30 für Prompt, 60 für Agent
<code>statusMessage</code>	nein	Benutzerdefinierte Spinner-Nachricht, die angezeigt wird, während der Hook ausgeführt wird
<code>once</code>	nein	Wenn <code>true</code> , wird nur einmal pro Sitzung ausgeführt und dann entfernt. Nur Skills, nicht Agents. Siehe Hooks in Skills und Agents

Command-Hook-Felder

Zusätzlich zu den [gemeinsamen Feldern](#) akzeptieren Command-Hooks diese Felder:

Feld	Erforderlich	Beschreibung
<code>command</code>	ja	Shell-Befehl zum Ausführen

Feld	Erforderlich	Beschreibung
<code>async</code>	nein	Wenn <code>true</code> , wird im Hintergrund ausgeführt, ohne zu blockieren. Siehe Hooks im Hintergrund ausführen

HTTP-Hook-Felder

Zusätzlich zu den [gemeinsamen Feldern](#) akzeptieren HTTP-Hooks diese Felder:

Feld	Erforderlich	Beschreibung
<code>url</code>	ja	URL, an die der POST-Request gesendet werden soll
<code>headers</code>	nein	Zusätzliche HTTP-Header als Schlüssel-Wert-Paare. Werte unterstützen Umgebungsvariablen-Interpolation mit <code>\$VAR_NAME</code> oder <code>\${VAR_NAME}</code> Syntax. Nur Variablen, die in <code>allowedEnvVars</code> aufgelistet sind, werden aufgelöst
<code>allowedEnvVars</code>	nein	Liste von Umgebungsvariablenennamen, die in Header-Werte interpoliert werden dürfen. Verweise auf nicht aufgelistete Variablen werden durch leere Zeichenketten ersetzt. Erforderlich für jede Umgebungsvariablen-Interpolation

Claude Code sendet die [JSON-Eingabe](#) des Hooks als POST-Request-Body mit `Content-Type: application/json`. Der Response-Body verwendet das gleiche [JSON-Ausgabeformat](#) wie Command-Hooks.

Die Fehlerbehandlung unterscheidet sich von Command-Hooks: Nicht-2xx-Antworten, Verbindungsfehler und Timeouts führen alle zu nicht-blockierenden Fehlern, die die Ausführung fortsetzen lassen. Um einen Tool-Aufruf zu blockieren oder eine Berechtigung zu verweigern, geben Sie eine 2xx-Antwort mit einem JSON-Body zurück, der `decision: "block"` oder ein `hookSpecificOutput` mit `permissionDecision: "deny"` enthält.

Dieses Beispiel sendet `PreToolUse`-Ereignisse an einen lokalen Validierungsdienst und authentifiziert sich mit einem Token aus der `MY_TOKEN`-Umgebungsvariable:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/pre-tool-use",
            "timeout": 30,
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}
```

Note:

HTTP-Hooks müssen durch direktes Bearbeiten von Einstellungs-JSON konfiguriert werden. Das interaktive `/hooks`-Menü unterstützt nur das Hinzufügen von Command-Hooks.

Prompt- und Agent-Hook-Felder

Zusätzlich zu den [gemeinsamen Feldern](#) akzeptieren Prompt- und Agent-Hooks diese Felder:

Feld	Erforderlich	Beschreibung
<code>prompt</code>	ja	Prompt-Text zum Senden an das Modell. Verwenden Sie <code>\$ARGUMENTS</code> als Platzhalter für die Hook-Eingabe JSON
<code>model</code>	nein	Modell zur Verwendung für die Evaluierung. Standardwert ist ein schnelles Modell

Alle passenden Hooks werden parallel ausgeführt, und identische Handler werden automatisch dedupliziert. Command-Hooks werden nach Befehlszeichenkette dedupliziert, und HTTP-Hooks werden nach URL dedupliziert. Handler werden im aktuellen Verzeichnis mit der Umgebung von Claude Code ausgeführt. Die `$CLAUDE_CODE_REMOTE`-Umgebungsvariable wird in Remote-Web-Umgebungen auf `"true"` gesetzt und ist in der lokalen CLI nicht gesetzt.

Referenzskripte nach Pfad

Verwenden Sie Umgebungsvariablen, um Hook-Skripte relativ zum Projekt- oder Plugin-Root zu referenzieren, unabhängig vom Arbeitsverzeichnis, wenn der Hook ausgeführt wird:

- `$CLAUDE_PROJECT_DIR` : das Projekt-Root. In Anführungszeichen setzen, um Pfade mit Leerzeichen zu handhaben.
- `${CLAUDE_PLUGIN_ROOT}` : das Root-Verzeichnis des Plugins, für Skripte, die mit einem [Plugin](#) gebündelt sind.

Projekt-Skripte

Dieses Beispiel verwendet `$CLAUDE_PROJECT_DIR`, um einen Style-Checker aus dem `.claude/hooks/`-Verzeichnis des Projekts nach jedem `Write`- oder `Edit`-Tool-Aufruf auszuführen:

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/check-style.sh"
          }
        ]
      }
    ]
  }
}

```

Plugin-Skripte

Definieren Sie Plugin-Hooks in `hooks/hooks.json` mit einem optionalen `description`-Feld auf oberster Ebene. Wenn ein Plugin aktiviert ist, werden seine Hooks mit Ihren Benutzer- und Projekt-Hooks zusammengeführt.

Dieses Beispiel führt ein Formatierungsskript aus, das mit dem Plugin gebündelt ist:

```

{
  "description": "Automatic code formatting",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format.sh",
            "timeout": 30
          }
        ]
      }
    ]
  }
}

```

Siehe die [Plugin-Komponenten-Referenz](#) für Details zum Erstellen von Plugin-Hooks.

Hooks in Skills und Agents

Zusätzlich zu Einstellungsdateien und Plugins können Hooks direkt in [Skills](#) und [Subagenten](#) mit Frontmatter definiert werden. Diese Hooks sind auf den Lebenszyklus der Komponente beschränkt und werden nur ausgeführt, wenn diese Komponente aktiv ist.

Alle Hook-Ereignisse werden unterstützt. Für Subagenten werden **Stop**-Hooks automatisch in **SubagentStop** konvertiert, da dies das Ereignis ist, das ausgelöst wird, wenn ein Subagent fertig ist.

Hooks verwenden das gleiche Konfigurationsformat wie einstellungsbasierte Hooks, sind aber auf die Lebensdauer der Komponente beschränkt und werden bereinigt, wenn sie fertig ist.

Dieser Skill definiert einen **PreToolUse**-Hook, der ein Sicherheitsvalidierungsskript vor jedem **Bash**-Befehl ausführt:

```

---
name: secure-operations
description: Perform operations with security checks
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/security-check.sh"
---

```

Agents verwenden das gleiche Format in ihrem YAML-Frontmatter.

Das `/hooks` -Menü

Geben Sie `/hooks` in Claude Code ein, um den interaktiven Hooks-Manager zu öffnen, in dem Sie Hooks anzeigen, hinzufügen und löschen können, ohne Einstellungsdateien direkt zu bearbeiten. Eine Schritt-für-Schritt-Anleitung finden Sie unter [Richten Sie Ihren ersten Hook ein](#) in der Anleitung.

Jeder Hook im Menü ist mit einem Klammer-Präfix gekennzeichnet, das seine Quelle angibt:

- `[User]` : aus `~/.claude/settings.json`
- `[Project]` : aus `.claude/settings.json`
- `[Local]` : aus `.claude/settings.local.json`
- `[Plugin]` : aus `hooks/hooks.json` eines Plugins, schreibgeschützt

Hooks deaktivieren oder entfernen

Um einen Hook zu entfernen, löschen Sie seinen Eintrag aus der Einstellungs-JSON-Datei, oder verwenden Sie das `/hooks` -Menü und wählen Sie den Hook aus, um ihn zu löschen.

Um alle Hooks vorübergehend zu deaktivieren, ohne sie zu entfernen, setzen Sie `"disableAllHooks": true` in Ihrer Einstellungsdatei oder verwenden Sie den Schalter im `/hooks` -Menü. Es gibt keine Möglichkeit, einen einzelnen Hook zu deaktivieren, während er in der Konfiguration bleibt.

Die `disableAllHooks` -Einstellung respektiert die verwaltete Einstellungshierarchie. Wenn ein Administrator Hooks durch verwaltete Richtlinieneinstellungen konfiguriert hat, kann `disableAllHooks`, das in Benutzer-, Projekt- oder lokalen Einstellungen gesetzt ist, diese verwalteten Hooks nicht deaktivieren. Nur `disableAllHooks`, das auf der verwalteten Einstellungsebene gesetzt ist, kann verwaltete Hooks deaktivieren.

Direkte Bearbeitungen von Hooks in Einstellungsdateien werden nicht sofort wirksam. Claude Code erstellt einen Snapshot von Hooks beim Start und verwendet ihn während der gesamten Sitzung. Dies verhindert, dass böswillige oder versehentliche Hook-Änderungen ohne Ihre Überprüfung mitten in der Sitzung wirksam werden. Wenn Hooks extern geändert werden, warnt Claude Code Sie und erfordert eine Überprüfung im `/ hooks` -Menü, bevor Änderungen wirksam werden.

Hook-Eingabe und -Ausgabe

Command-Hooks erhalten JSON-Daten über stdin und kommunizieren Ergebnisse über Exit-Codes, stdout und stderr. HTTP-Hooks erhalten die gleiche JSON als POST-Request-Body und kommunizieren Ergebnisse über den HTTP-Response-Body. Dieser Abschnitt behandelt Felder und Verhalten, die allen Ereignissen gemeinsam sind. Jeder Ereignisabschnitt unter [Hook-Ereignisse](#) enthält sein spezifisches Eingabeschema und Optionen zur Entscheidungskontrolle.

Gemeinsame Eingabefelder

Alle Hook-Ereignisse erhalten diese Felder als JSON, zusätzlich zu ereignisspezifischen Feldern, die in jedem Abschnitt [Hook-Ereignis](#) dokumentiert sind. Bei Command-Hooks kommt diese JSON über stdin an. Bei HTTP-Hooks kommt sie als POST-Request-Body an.

Feld	Beschreibung
<code>session_id</code>	Aktuelle Sitzungs-ID
<code>transcript_path</code>	Pfad zur Gesprächs-JSON
<code>cwd</code>	Aktuelles Arbeitsverzeichnis, wenn der Hook aufgerufen wird
<code>permission_mode</code>	Aktueller Berechtigungsmodus : <code>"default"</code> , <code>"plan"</code> , <code>"acceptEdits"</code> , <code>"dontAsk"</code> oder <code>"bypassPermissions"</code>
<code>hook_event_name</code>	Name des ausgelösten Ereignisses

Bei Ausführung mit `--agent` oder innerhalb eines Subagenten sind zwei zusätzliche Felder enthalten:

Feld	Beschreibung
<code>agent_id</code>	Eindeutige Kennung für den Subagenten. Nur vorhanden, wenn der Hook innerhalb eines Subagenten-Aufrufs ausgelöst wird. Verwenden Sie dies, um Subagenten-Hook-Aufrufe von Hauptthread-Aufrufen zu unterscheiden.
<code>agent_type</code>	Agent-Name (zum Beispiel <code>"Explore"</code> oder <code>"security-reviewer"</code>). Vorhanden, wenn die Sitzung <code>--agent</code> verwendet oder der Hook innerhalb eines Subagenten ausgelöst wird. Für Subagenten hat der Typ des Subagenten Vorrang vor dem <code>--agent</code> -Wert der Sitzung.

Beispielsweise erhält ein `PreToolUse` -Hook für einen Bash-Befehl dies über stdin:

```
{
  "session_id": "abc123",
  "transcript_path": "/home/user/.claude/projects/.../transcript.jsonl",
  "cwd": "/home/user/my-project",
  "permission_mode": "default",
  "hook_event_name": "PreToolUse",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test"
  }
}
```

Die Felder `tool_name` und `tool_input` sind ereignisspezifisch. Jeder Abschnitt [Hook-Ereignis](#) dokumentiert die zusätzlichen Felder für dieses Ereignis.

Exit-Code-Ausgabe

Der Exit-Code aus Ihrem Hook-Befehl teilt Claude Code mit, ob die Aktion fortgesetzt, blockiert oder ignoriert werden soll.

Exit 0 bedeutet Erfolg. Claude Code analysiert stdout auf [JSON-Ausgabefelder](#). JSON-Ausgabe wird nur bei Exit 0 verarbeitet. Bei den meisten Ereignissen wird stdout nur im ausführlichen Modus (`Ctrl+0`) angezeigt. Die Ausnahmen sind `UserPromptSubmit` und `SessionStart` , wo stdout als Kontext hinzugefügt wird, den Claude sehen und darauf reagieren kann.

Exit 2 bedeutet ein blockierender Fehler. Claude Code ignoriert stdout und jede JSON darin. Stattdessen wird stderr-Text an Claude als Fehlermeldung zurückgegeben. Die Auswirkung hängt vom Ereignis ab: `PreToolUse` blockiert den Tool-Aufruf, `UserPromptSubmit` lehnt den Prompt ab, und so weiter. Siehe [Exit-Code-2-Verhalten](#) für die vollständige Liste.

Jeder andere Exit-Code ist ein nicht-blockierender Fehler. stderr wird im ausführlichen Modus (`Ctrl+O`) angezeigt und die Ausführung wird fortgesetzt.

Beispielsweise ein Hook-Befehlsskript, das gefährliche Bash-Befehle blockiert:

```
#!/bin/bash
## Liest JSON-Eingabe von stdin, prüft den Befehl
command=$(jq -r '.tool_input.command' < /dev/stdin)

if [[ "$command" = rm* ]]; then
    echo "Blocked: rm commands are not allowed" >&2
    exit 2 # Blockierender Fehler: Tool-Aufruf wird verhindert
fi

exit 0 # Erfolg: Tool-Aufruf wird fortgesetzt
```

Exit-Code-2-Verhalten pro Ereignis

Exit-Code 2 ist die Art, wie ein Hook signalisiert „Stopp, mach das nicht.“ Die Auswirkung hängt vom Ereignis ab, da einige Ereignisse Aktionen darstellen, die blockiert werden können (wie ein Tool-Aufruf, der noch nicht stattgefunden hat), und andere Dinge darstellen, die bereits passiert sind oder nicht verhindert werden können.

Hook-Ereignis	Kann blockiert werden?	Was passiert bei Exit 2
<code>PreToolUse</code>	Ja	Blockiert den Tool-Aufruf
<code>PermissionRequest</code>	Ja	Verweigert die Berechtigung
<code>UserPromptSubmit</code>	Ja	Blockiert die Prompt-Verarbeitung und löscht den Prompt

Hook-Ereignis	Kann blockiert werden?	Was passiert bei Exit 2
<code>Stop</code>	Ja	Verhindert, dass Claude stoppt, setzt das Gespräch fort
<code>SubagentStop</code>	Ja	Verhindert, dass der Subagent stoppt
<code>TeammateIdle</code>	Ja	Verhindert, dass der Teammate untätig wird (Teammate arbeitet weiter)
<code>TaskCompleted</code>	Ja	Verhindert, dass die Aufgabe als abgeschlossen markiert wird
<code>ConfigChange</code>	Ja	Blockiert die Konfigurationsänderung von der Anwendung (außer <code>policy_settings</code>)
<code>PostToolUse</code>	Nein	Zeigt stderr Claude an (Tool wurde bereits ausgeführt)
<code>PostToolUseFailure</code>	Nein	Zeigt stderr Claude an (Tool ist bereits fehlgeschlagen)
<code>Notification</code>	Nein	Zeigt stderr nur dem Benutzer an
<code>SubagentStart</code>	Nein	Zeigt stderr nur dem Benutzer an
<code>SessionStart</code>	Nein	Zeigt stderr nur dem Benutzer an
<code>SessionEnd</code>	Nein	Zeigt stderr nur dem Benutzer an
<code>PreCompact</code>	Nein	Zeigt stderr nur dem Benutzer an

Hook-Ereignis	Kann blockiert werden?	Was passiert bei Exit 2
<code>WorktreeCreate</code>	Ja	Jeder Nicht-Null-Exit-Code führt zu Fehlschlag der Worktree-Erstellung
<code>WorktreeRemove</code>	Nein	Fehler werden nur im Debug-Modus protokolliert
<code>InstructionsLoaded</code>	Nein	Exit-Code wird ignoriert

HTTP-Response-Behandlung

HTTP-Hooks verwenden HTTP-Statuscodes und Response-Bodies anstelle von Exit-Codes und stdout:

- **2xx mit leerem Body:** Erfolg, äquivalent zu Exit-Code 0 ohne Ausgabe
- **2xx mit Plain-Text-Body:** Erfolg, der Text wird als Kontext hinzugefügt
- **2xx mit JSON-Body:** Erfolg, analysiert mit dem gleichen [JSON-Ausgabe](#)-Schema wie Command-Hooks
- **Nicht-2xx-Status:** Nicht-blockierender Fehler, Ausführung wird fortgesetzt
- **Verbindungsfehler oder Timeout:** Nicht-blockierender Fehler, Ausführung wird fortgesetzt

Im Gegensatz zu Command-Hooks können HTTP-Hooks nicht allein durch Statuscodes einen blockierenden Fehler signalisieren. Um einen Tool-Aufruf zu blockieren oder eine Berechtigung zu verweigern, geben Sie eine 2xx-Antwort mit einem JSON-Body zurück, der die entsprechenden Entscheidungsfelder enthält.

JSON-Ausgabe

Exit-Codes ermöglichen es Ihnen, zu erlauben oder zu blockieren, aber JSON-Ausgabe gibt Ihnen eine feinere Kontrolle. Anstatt mit Code 2 zu beenden, um zu blockieren, beenden Sie mit 0 und geben Sie ein JSON-Objekt auf stdout aus. Claude Code liest spezifische Felder aus dieser JSON, um das Verhalten zu steuern, einschließlich [Entscheidungskontrolle](#) zum Blockieren, Erlauben oder Eskalieren an den Benutzer.

Note:

Sie müssen einen Ansatz pro Hook wählen, nicht beide: Verwenden Sie entweder Exit-Codes allein zum Signalisieren, oder beenden Sie mit 0 und geben Sie JSON für strukturierte Kontrolle aus. Claude Code verarbeitet JSON nur bei Exit 0. Wenn Sie mit 2 beenden, wird jede JSON ignoriert.

Die stdout Ihres Hooks darf nur das JSON-Objekt enthalten. Wenn Ihr Shell-Profil beim Start Text ausgibt, kann dies die JSON-Analyse beeinträchtigen. Siehe [JSON-Validierung fehlgeschlagen](#) im Troubleshooting-Leitfaden.

Das JSON-Objekt unterstützt drei Arten von Feldern:

- **Universelle Felder** wie `continue` funktionieren über alle Ereignisse hinweg. Diese sind in der folgenden Tabelle aufgelistet.
- **Top-Level `decision` und `reason`** werden von einigen Ereignissen verwendet, um zu blockieren oder Feedback zu geben.
- **`hookSpecificOutput`** ist ein verschachteltes Objekt für Ereignisse, die reichere Kontrolle benötigen. Es erfordert ein `hookEventName` -Feld, das auf den Ereignisnamen gesetzt ist.

Feld	Standard	Beschreibung
<code>continue</code>	<code>true</code>	Wenn <code>false</code> , stoppt Claude die Verarbeitung vollständig, nachdem der Hook ausgeführt wurde. Hat Vorrang vor allen ereignisspezifischen Entscheidungsfeldern
<code>stopReason</code>	keine	Nachricht, die dem Benutzer angezeigt wird, wenn <code>continue false</code> ist. Wird Claude nicht angezeigt
<code>suppressOutput</code>	<code>false</code>	Wenn <code>true</code> , verbirgt stdout aus der Ausgabe des ausführlichen Modus
<code>systemMessage</code>	keine	Warnmeldung, die dem Benutzer angezeigt wird

Um Claude vollständig zu stoppen, unabhängig vom Ereignistyp:

```
{ "continue": false, "stopReason": "Build failed, fix errors before continuing" }
```

Entscheidungskontrolle

Nicht jedes Ereignis unterstützt das Blockieren oder Steuern des Verhaltens durch JSON. Die Ereignisse, die dies tun, verwenden jeweils einen anderen Satz von Feldern, um diese Entscheidung auszudrücken. Verwenden Sie diese Tabelle als schnelle Referenz, bevor Sie einen Hook schreiben:

Ereignisse	Entscheidungsmuster	Schlüsselfelder
UserPromptSubmit, Post-ToolUse, PostToolUseFailure, Stop, SubagentStop, ConfigChange	Top-Level decision	decision: "block", reason
TeammateIdle, TaskCompleted	Exit-Code oder continue: false	Exit-Code 2 blockiert die Aktion mit stderr-Feedback. JSON {"continue": false, "stopReason": "..."} stoppt auch den Teammate vollständig, was dem Stop -Hook-Verhalten entspricht
PreToolUse	hookSpecificOutput	permissionDecision (allow/deny/ask), permissionDecisionReason
PermissionRequest	hookSpecificOutput	decision.behavior (allow/deny)
WorktreeCreate	stdout-Pfad	Hook gibt absoluten Pfad zur erstellten Worktree aus. Nicht-Null-Exit schlägt die Erstellung fehl
WorktreeRemove, Notification, SessionEnd, Pre-Compact, InstructionsLoaded	Keine	Keine Entscheidungskontrolle. Wird für Nebenwirkungen wie Protokollierung oder Bereinigung verwendet

Hier sind Beispiele für jedes Muster in Aktion:

Top-Level-Entscheidung

Wird von `UserPromptSubmit`, `PostToolUse`, `PostToolUseFailure`, `Stop`, `SubagentStop` und `ConfigChange` verwendet. Der einzige Wert ist `"block"`. Um die Aktion fortzusetzen, lassen Sie `decision` aus Ihrer JSON weg, oder beenden Sie mit o ohne jede JSON:

```
{
  "decision": "block",
  "reason": "Test suite must pass before proceeding"
}
```

PreToolUse

Verwendet `hookSpecificOutput` für reichere Kontrolle: erlauben, verweigern oder an den Benutzer eskalieren. Sie können auch die Tool-Eingabe vor der Ausführung ändern oder zusätzlichen Kontext für Claude injizieren. Siehe [PreToolUse-Entscheidungskontrolle](#) für den vollständigen Satz von Optionen.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Database writes are not allowed"
  }
}
```

PermissionRequest

Verwendet `hookSpecificOutput`, um eine Berechtigungsanfrage im Namen des Benutzers zu erlauben oder zu verweigern. Beim Erlauben können Sie auch die Eingabe des Tools ändern oder Berechtigungsregeln anwenden, damit der Benutzer nicht erneut aufgefordert wird. Siehe [PermissionRequest-Entscheidungskontrolle](#) für den vollständigen Satz von Optionen.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

Erweiterte Beispiele einschließlich Bash-Befehlsvalidierung, Prompt-Filterung und Auto-Genehmigungsskripte finden Sie unter [Was Sie automatisieren können](#) in der Anleitung und der [Bash-Befehlsvalidierungs-Referenzimplementierung](#).

Hook-Ereignisse

Jedes Ereignis entspricht einem Punkt im Lebenszyklus von Claude Code, an dem Hooks ausgeführt werden können. Die folgenden Abschnitte sind in der Reihenfolge des Lebenszyklus angeordnet: von der Sitzungseinrichtung durch die agentengesteuerte Schleife bis zum Sitzungsende. Jeder Abschnitt beschreibt, wann das Ereignis ausgelöst wird, welche Matcher es unterstützt, die JSON-Eingabe, die es erhält, und wie das Verhalten durch die Ausgabe gesteuert wird.

SessionStart

Wird ausgeführt, wenn Claude Code eine neue Sitzung startet oder eine vorhandene Sitzung fortsetzt. Nützlich zum Laden von Entwicklungskontext wie vorhandenen Problemen oder kürzlichen Änderungen an Ihrer Codebasis oder zum Einrichten von Umgebungsvariablen. Für statischen Kontext, der kein Skript erfordert, verwenden Sie stattdessen [CLAUDE.md](#).

SessionStart wird bei jeder Sitzung ausgeführt, daher halten Sie diese Hooks schnell. Nur `type: "command"`-Hooks werden unterstützt.

Der Matcher-Wert entspricht der Art, wie die Sitzung initiiert wurde:

Matcher	Wann es ausgelöst wird
<code>startup</code>	Neue Sitzung

Matcher	Wann es ausgelöst wird
<code>resume</code>	<code>--resume</code> , <code>--continue</code> oder <code>/resume</code>
<code>clear</code>	<code>/clear</code>
<code>compact</code>	Auto- oder manuelle Komprimierung

SessionStart-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten SessionStart-Hooks `source` , `model` und optional `agent_type` . Das Feld `source` gibt an, wie die Sitzung gestartet wurde: `"startup"` für neue Sitzungen, `"resume"` für fortgesetzte Sitzungen, `"clear"` nach `/clear` oder `"compact"` nach Komprimierung. Das Feld `model` enthält die Modellkennung. Wenn Sie Claude Code mit `claude --agent <name>` starten, enthält ein `agent_type` -Feld den Agent-Namen.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SessionStart",
  "source": "startup",
  "model": "claude-sonnet-4-6"
}
```

SessionStart-Entscheidungskontrolle

Jeder Text, den Ihr Hook-Skript auf stdout ausgibt, wird als Kontext für Claude hinzugefügt. Zusätzlich zu den [JSON-Ausgabefeldern](#), die für alle Hooks verfügbar sind, können Sie diese ereignisspezifischen Felder zurückgeben:

Feld	Beschreibung
<code>additionalContext</code>	Zeichenkette, die zu Claudes Kontext hinzugefügt wird. Werte mehrerer Hooks werden verkettet

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "My additional context here"
  }
}
```

Umgebungsvariablen beibehalten

SessionStart-Hooks haben Zugriff auf die `CLAUDE_ENV_FILE`-Umgebungsvariable, die einen Dateipfad bereitstellt, in dem Sie Umgebungsvariablen für nachfolgende Bash-Befehle beibehalten können.

Um einzelne Umgebungsvariablen zu setzen, schreiben Sie `export`-Anweisungen in `CLAUDE_ENV_FILE`. Verwenden Sie Anhängen (`>>`), um Variablen zu bewahren, die von anderen Hooks gesetzt wurden:

```
#!/bin/bash

if [ -n "$CLAUDE_ENV_FILE" ]; then
  echo 'export NODE_ENV=production' >> "$CLAUDE_ENV_FILE"
  echo 'export DEBUG_LOG=true' >> "$CLAUDE_ENV_FILE"
  echo 'export PATH="$PATH:./node_modules/.bin"' >> "$CLAUDE_ENV_FILE"
fi

exit 0
```

Um alle Umgebungsänderungen von Setup-Befehlen zu erfassen, vergleichen Sie die exportierten Variablen vorher und nachher:

```
#!/bin/bash

ENV_BEFORE=$(export -p | sort)

## Führen Sie Ihre Setup-Befehle aus, die die Umgebung ändern
source ~/.nvm/nvm.sh
nvm use 20

if [ -n "$CLAUDE_ENV_FILE" ]; then
    ENV_AFTER=$(export -p | sort)
    comm -13 <(echo "$ENV_BEFORE") <(echo "$ENV_AFTER") >> "$CLAUDE_ENV_FILE"
fi

exit 0
```

Alle Variablen, die in diese Datei geschrieben werden, sind in allen nachfolgenden Bash-Befehlen verfügbar, die Claude Code während der Sitzung ausführt.

Note:

`CLAUDE_ENV_FILE` ist für SessionStart-Hooks verfügbar. Andere Hook-Typen haben keinen Zugriff auf diese Variable.

InstructionsLoaded

Wird ausgelöst, wenn eine `CLAUDE.md` - oder `.claude/rules/*.md` -Datei in den Kontext geladen wird. Dieses Ereignis wird beim Sitzungsstart für eifrig geladene Dateien ausgelöst und später erneut, wenn Dateien träge geladen werden, zum Beispiel wenn Claude auf ein Unterverzeichnis zugreift, das eine verschachtelte `CLAUDE.md` enthält, oder wenn bedingte Regeln mit `paths:` -Frontmatter passen. Der Hook unterstützt keine Blockierung oder Entscheidungskontrolle. Er wird asynchron zu Beobachtungszwecken ausgeführt.

InstructionsLoaded unterstützt keine Matcher und wird bei jedem Ladevorgang ausgelöst.

InstructionsLoaded-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten InstructionsLoaded-Hooks diese Felder:

Feld	Beschreibung
<code>file_path</code>	Absoluter Pfad zur Anweisungsdatei, die geladen wurde
<code>memory_type</code>	Umfang der Datei: <code>"User"</code> , <code>"Project"</code> , <code>"Local"</code> oder <code>"Managed"</code>
<code>load_reason</code>	Warum die Datei geladen wurde: <code>"session_start"</code> , <code>"nested_travel"</code> , <code>"path_glob_match"</code> oder <code>"include"</code>
<code>globs</code>	Pfad-Glob-Muster aus dem <code>paths:</code> -Frontmatter der Datei, falls vorhanden. Nur für <code>path_glob_match</code> -Ladevorgänge vorhanden
<code>trigger_file_path</code>	Pfad zur Datei, deren Zugriff diesen Ladevorgang ausgelöst hat, für träge Ladevorgänge
<code>parent_file_path</code>	Pfad zur übergeordneten Anweisungsdatei, die diese eingebunden hat, für <code>include</code> -Ladevorgänge

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../transcript.jsonl",
  "cwd": "/Users/my-project",
  "permission_mode": "default",
  "hook_event_name": "InstructionsLoaded",
  "file_path": "/Users/my-project/CLAUDE.md",
  "memory_type": "Project",
  "load_reason": "session_start"
}
```

InstructionsLoaded-Entscheidungskontrolle

InstructionsLoaded-Hooks haben keine Entscheidungskontrolle. Sie können das Laden von Anweisungen nicht blockieren oder ändern. Verwenden Sie dieses Ereignis für Audit-Protokollierung, Compliance-Tracking oder Beobachtbarkeit.

UserPromptSubmit

Wird ausgeführt, wenn der Benutzer einen Prompt einreicht, bevor Claude ihn verarbeitet. Dies ermöglicht es Ihnen, zusätzlichen Kontext basierend auf dem Prompt/Gespräch hinzuzufügen, Prompts zu validieren oder bestimmte Arten von Prompts zu blockieren.

UserPromptSubmit-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten UserPromptSubmit-Hooks das Feld `prompt`, das den Text enthält, den der Benutzer eingereicht hat.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "UserPromptSubmit",
  "prompt": "Write a function to calculate the factorial of a number"
}
```

UserPromptSubmit-Entscheidungskontrolle

`UserPromptSubmit`-Hooks können steuern, ob ein Benutzer-Prompt verarbeitet wird, und Kontext hinzufügen. Alle [JSON-Ausgabefelder](#) sind verfügbar.

Es gibt zwei Möglichkeiten, Kontext zum Gespräch bei Exit-Code 0 hinzuzufügen:

- **Plain-Text-stdout:** Jeder Nicht-JSON-Text, der auf stdout geschrieben wird, wird als Kontext hinzugefügt
- **JSON mit `additionalContext`:** Verwenden Sie das JSON-Format unten für mehr Kontrolle. Das Feld `additionalContext` wird als Kontext hinzugefügt

Plain-stdout wird als Hook-Ausgabe im Transkript angezeigt. Das Feld `additionalContext` wird diskreter hinzugefügt.

Um einen Prompt zu blockieren, geben Sie ein JSON-Objekt mit `decision` auf `"block"` zurück:

Feld	Beschreibung
<code>decision</code>	<code>"block"</code> verhindert die Verarbeitung des Prompts und löscht ihn aus dem Kontext. Weglassen, um den Prompt fortzusetzen
<code>reason</code>	Wird dem Benutzer angezeigt, wenn <code>decision</code> <code>"block"</code> ist. Wird nicht zum Kontext hinzugefügt
<code>additionalContext</code>	Zeichenkette, die zu Claudes Kontext hinzugefügt wird

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "My additional context here"
  }
}
```

Note:

Das JSON-Format ist nicht erforderlich für einfache Anwendungsfälle. Um Kontext hinzuzufügen, können Sie einfach Plain-Text auf stdout mit Exit-Code 0 ausgeben. Verwenden Sie JSON, wenn Sie Prompts blockieren oder mehr strukturierte Kontrolle benötigen.

PreToolUse

Wird ausgeführt, nachdem Claude Tool-Parameter erstellt hat und bevor der Tool-Aufruf verarbeitet wird. Passt auf Tool-Namen: `Bash`, `Edit`, `Write`, `Read`, `Glob`, `Grep`, `Agent`, `WebFetch`, `WebSearch` und alle [MCP-Tool-Namen](#).

Verwenden Sie [PreToolUse-Entscheidungskontrolle](#), um die Verwendung des Tools zu erlauben, zu verweigern oder um Berechtigung zu bitten.

PreToolUse-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten PreToolUse-Hooks `tool_name`, `tool_input` und `tool_use_id`. Die `tool_input`-Felder hängen vom Tool ab:

Bash

Führt Shell-Befehle aus.

Feld	Typ	Beispiel	Beschreibung
<code>command</code>	Zeichenkette	<code>"npm test"</code>	Der auszuführende Shell-Befehl
<code>description</code>	Zeichenkette	<code>"Run test suite"</code>	Optionale Beschreibung, was der Befehl tut

Feld	Typ	Beispiel	Beschreibung
<code>timeout</code>	Zahl	<code>120000</code>	Optionales Timeout in Millisekunden
<code>run_in_background</code>	Boolesch	<code>false</code>	Ob der Befehl im Hintergrund ausgeführt werden soll

Write

Erstellt oder überschreibt eine Datei.

Feld	Typ	Beispiel	Beschreibung
<code>file_path</code>	Zeichenkette	<code>"/path/to/file.txt"</code>	Absoluter Pfad zur zu schreibenden Datei
<code>content</code>	Zeichenkette	<code>"file content"</code>	Inhalt zum Schreiben in die Datei

Edit

Ersetzt eine Zeichenkette in einer vorhandenen Datei.

Feld	Typ	Beispiel	Beschreibung
<code>file_path</code>	Zeichenkette	<code>"/path/to/file.txt"</code>	Absoluter Pfad zur zu bearbeitenden Datei
<code>old_string</code>	Zeichenkette	<code>"original text"</code>	Text zum Suchen und Ersetzen
<code>new_string</code>	Zeichenkette	<code>"replacement text"</code>	Ersatztext
<code>replace_all</code>	Boolesch	<code>false</code>	Ob alle Vorkommen ersetzt werden sollen

Read

Liest Dateiinhalte.

Feld	Typ	Beispiel	Beschreibung
<code>file_path</code>	Zeichenkette	<code>"/path/to/file.txt"</code>	Absoluter Pfad zur zu lesenden Datei
<code>offset</code>	Zahl	<code>10</code>	Optionale Zeilennummer zum Starten des Lesens
<code>limit</code>	Zahl	<code>50</code>	Optionale Anzahl der zu lesenden Zeilen

Glob

Findet Dateien, die einem Glob-Muster entsprechen.

Feld	Typ	Beispiel	Beschreibung
<code>pattern</code>	Zeichenkette	<code>"**/*.ts"</code>	Glob-Muster zum Abgleich von Dateien
<code>path</code>	Zeichenkette	<code>"/path/to/dir"</code>	Optionales Verzeichnis zum Durchsuchen. Standardwert ist aktuelles Arbeitsverzeichnis

Grep

Durchsucht Dateiinhalte mit regulären Ausdrücken.

Feld	Typ	Beispiel	Beschreibung
<code>pattern</code>	Zeichenkette	<code>"TODO.*fix"</code>	Regex-Muster zum Suchen
<code>path</code>	Zeichenkette	<code>"/path/to/dir"</code>	Optionale Datei oder Verzeichnis zum Durchsuchen
<code>glob</code>	Zeichenkette	<code>"*.ts"</code>	Optionales Glob-Muster zum Filtern von Dateien

Feld	Typ	Beispiel	Beschreibung
<code>output_mode</code>	Zeichenkette	<code>"content"</code>	<code>"content"</code> , <code>"files_with_matches"</code> oder <code>"count"</code> . Standardwert ist <code>"files_with_matches"</code>
<code>-i</code>	Boolesch	<code>true</code>	Groß-/Kleinschreibung ignorieren
<code>multiline</code>	Boolesch	<code>false</code>	Mehrzeilige Übereinstimmung aktivieren

WebFetch

Ruft Web-Inhalte ab und verarbeitet sie.

Feld	Typ	Beispiel	Beschreibung
<code>url</code>	Zeichenkette	<code>"https://example.com/api"</code>	URL zum Abrufen von Inhalten
<code>prompt</code>	Zeichenkette	<code>"Extract the API endpoints"</code>	Prompt zum Ausführen auf dem abgerufenen Inhalt

WebSearch

Durchsucht das Web.

Feld	Typ	Beispiel	Beschreibung
<code>query</code>	Zeichenkette	<code>"react hooks best practices"</code>	Suchanfrage
<code>allowed_domains</code>	Array	<code>["docs.example.com"]</code>	Optional: Nur Ergebnisse von diesen Domains einschließen
<code>blocked_domains</code>	Array	<code>["spam.example.com"]</code>	Optional: Ergebnisse von diesen Domains ausschließen

Agent

Spawnt einen [Subagenten](#).

Feld	Typ	Beispiel	Beschreibung
<code>prompt</code>	Zeichenkette	<code>"Find all API endpoints"</code>	Die Aufgabe für den Agent
<code>description</code>	Zeichenkette	<code>"Find API endpoints"</code>	Kurze Beschreibung der Aufgabe
<code>subagent_type</code>	Zeichenkette	<code>"Explore"</code>	Typ des zu verwendenden spezialisierten Agenten
<code>model</code>	Zeichenkette	<code>"sonnet"</code>	Optionaler Modell-Alias zum Überschreiben des Standards

PreToolUse-Entscheidungskontrolle

`PreToolUse` -Hooks können steuern, ob ein Tool-Aufruf fortgesetzt wird. Im Gegensatz zu anderen Hooks, die ein Top-Level- `decision` -Feld verwenden, gibt `PreToolUse` seine Entscheidung in einem `hookSpecificOutput` -Objekt zurück. Dies gibt ihm reichere Kontrolle: drei Ergebnisse (erlauben, verweigern oder fragen) plus die Möglichkeit, die Tool-Eingabe vor der Ausführung zu ändern.

Feld	Beschreibung
<code>permissionDecision</code>	<code>"allow"</code> umgeht das Berechtigungssystem, <code>"deny"</code> verhindert den Tool-Aufruf, <code>"ask"</code> fordert den Benutzer zur Bestätigung auf
<code>permissionDecisionReason</code>	Für <code>"allow"</code> und <code>"ask"</code> , dem Benutzer angezeigt, aber nicht Claude. Für <code>"deny"</code> , Claude angezeigt
<code>updatedInput</code>	Ändert die Tool-Eingabeparameter vor der Ausführung. Kombinieren Sie mit <code>"allow"</code> , um automatisch zu genehmigen, oder mit <code>"ask"</code> , um die geänderte Eingabe dem Benutzer zu zeigen
<code>additionalContext</code>	Zeichenkette, die zu Claudes Kontext vor der Tool-Ausführung hinzugefügt wird

```

{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow",
    "permissionDecisionReason": "My reason here",
    "updatedInput": {
      "field_to_modify": "new value"
    },
    "additionalContext": "Current environment: production. Proceed with caution."
  }
}

```

Note:

PreToolUse verwendete zuvor Top-Level- **decision** - und **reason** -Felder, aber diese sind für dieses Ereignis veraltet. Verwenden Sie stattdessen `hookSpecificOutput.permissionDecision` und `hookSpecificOutput.permissionDecisionReason`. Die veralteten Werte **"approve"** und **"block"** werden auf **"allow"** und **"deny"** abgebildet. Andere Ereignisse wie PostToolUse und Stop verwenden weiterhin Top-Level- **decision** und **reason** als ihr aktuelles Format.

PermissionRequest

Wird ausgeführt, wenn dem Benutzer ein Berechtigungsdialog angezeigt wird. Verwenden Sie [PermissionRequest-Entscheidungskontrolle](#), um im Namen des Benutzers zu erlauben oder zu verweigern.

Passt auf Tool-Namen, gleiche Werte wie PreToolUse.

PermissionRequest-Eingabe

PermissionRequest-Hooks erhalten `tool_name` - und `tool_input` -Felder wie PreToolUse-Hooks, aber ohne `tool_use_id`. Ein optionales `permission_suggestions` -Array enthält die „Immer erlauben“-Optionen, die der Benutzer normalerweise im Berechtigungsdialog sehen würde. Der Unterschied liegt darin, wann der Hook ausgelöst wird: PermissionRequest-Hooks werden ausgeführt, wenn ein Berechtigungsdialog dem Benutzer angezeigt werden soll, während PreToolUse-Hooks vor der Tool-Ausführung unabhängig vom Berechtigungsstatus ausgeführt werden.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PermissionRequest",
  "tool_name": "Bash",
  "tool_input": {
    "command": "rm -rf node_modules",
    "description": "Remove node_modules directory"
  },
  "permission_suggestions": [
    { "type": "toolAlwaysAllow", "tool": "Bash" }
  ]
}
```

PermissionRequest-Entscheidungskontrolle

PermissionRequest-Hooks können Berechtigungsanfragen erlauben oder verweigern. Zusätzlich zu den [JSON-Ausgabefeldern](#), die für alle Hooks verfügbar sind, kann Ihr Hook-Skript ein **decision**-Objekt mit diesen ereignisspezifischen Feldern zurückgeben:

Feld	Beschreibung
behavior	"allow" gewährt die Berechtigung, "deny" verweigert sie
updatedInput	Nur für "allow" : ändert die Tool-Eingabeparameter vor der Ausführung
updatedPermissions	Nur für "allow" : wendet Berechtigungsregel-Updates an, äquivalent zur Auswahl einer „Immer erlauben“-Option durch den Benutzer
message	Nur für "deny" : teilt Claude mit, warum die Berechtigung verweigert wurde
interrupt	Nur für "deny" : wenn true , stoppt Claude

```

{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}

```

PostToolUse

Wird unmittelbar nach erfolgreichem Abschluss eines Tools ausgeführt.

Passt auf Tool-Namen, gleiche Werte wie PreToolUse.

PostToolUse-Eingabe

PostToolUse -Hooks werden ausgelöst, nachdem ein Tool bereits erfolgreich ausgeführt wurde. Die Eingabe enthält sowohl **tool_input**, die an das Tool gesendeten Argumente, als auch **tool_response**, das Ergebnis, das es zurückgegeben hat. Das genaue Schema für beide hängt vom Tool ab.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "PostToolUse",
  "tool_name": "Write",
  "tool_input": {
    "file_path": "/path/to/file.txt",
    "content": "file content"
  },
  "tool_response": {
    "filePath": "/path/to/file.txt",
    "success": true
  },
  "tool_use_id": "toolu_01ABC123..."
}
```

PostToolUse-Entscheidungskontrolle

`PostToolUse` -Hooks können Feedback an Claude nach der Tool-Ausführung geben. Zusätzlich zu den [JSON-Ausgabefeldern](#), die für alle Hooks verfügbar sind, kann Ihr Hook-Skript diese ereignisspezifischen Felder zurückgeben:

Feld	Beschreibung
<code>decision</code>	<code>"block"</code> fordert Claude mit dem <code>reason</code> auf. Weglassen, um die Aktion fortzusetzen
<code>reason</code>	Erklärung, die Claude angezeigt wird, wenn <code>decision</code> <code>"block"</code> ist
<code>additionalContext</code>	Zusätzlicher Kontext für Claude zu berücksichtigen
<code>updatedMCPToolOutput</code>	Nur für MCP-Tools : ersetzt die Ausgabe des Tools durch den bereitgestellten Wert

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "additionalContext": "Additional information for Claude"
  }
}
```

PostToolUseFailure

Wird ausgeführt, wenn eine Tool-Ausführung fehlschlägt. Dieses Ereignis wird für Tool-Aufrufe ausgelöst, die Fehler werfen oder Fehlerergebnisse zurückgeben. Verwenden Sie dies, um Fehler zu protokollieren, Warnungen zu senden oder korrektes Feedback an Claude zu geben.

Passt auf Tool-Namen, gleiche Werte wie PreToolUse.

PostToolUseFailure-Eingabe

PostToolUseFailure-Hooks erhalten die gleichen `tool_name` - und `tool_input` -Felder wie PostToolUse, zusammen mit Fehlerinformationen als Top-Level-Felder:

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PostToolUseFailure",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test",
    "description": "Run test suite"
  },
  "tool_use_id": "toolu_01ABC123...",
  "error": "Command exited with non-zero status code 1",
  "is_interrupt": false
}
```

Feld	Beschreibung
<code>error</code>	Zeichenkette, die beschreibt, was schief gelaufen ist
<code>is_interrupt</code>	Optionaler Boolesch, der angibt, ob der Fehler durch Benutzerunterbrechung verursacht wurde

PostToolUseFailure-Entscheidungskontrolle

`PostToolUseFailure` -Hooks können Kontext an Claude nach einem Tool-Fehler geben. Zusätzlich zu den [JSON-Ausgabefeldern](#), die für alle Hooks verfügbar sind, kann Ihr Hook-Skript diese ereignisspezifischen Felder zurückgeben:

Feld	Beschreibung
<code>additionalContext</code>	Zusätzlicher Kontext für Claude zu berücksichtigen neben dem Fehler

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUseFailure",
    "additionalContext": "Additional information about the failure for Claude"
  }
}
```

Notification

Wird ausgeführt, wenn Claude Code Benachrichtigungen sendet. Passt auf Benachrichtigungstyp: `permission_prompt`, `idle_prompt`, `auth_success`, `elicitation_dialog`. Matcher weglassen, um Hooks für alle Benachrichtigungstypen auszuführen.

Verwenden Sie separate Matcher, um verschiedene Handler je nach Benachrichtigungstyp auszuführen. Diese Konfiguration löst ein berechtigungsspezifisches Warnungsskript aus, wenn Claude Genehmigung benötigt, und eine andere Benachrichtigung, wenn Claude untätig war:


```

{
  "hooks": {
    "Notification": [
      {
        "matcher": "permission_prompt",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/permission-alert.sh"
          }
        ]
      },
      {
        "matcher": "idle_prompt",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/idle-notification.sh"
          }
        ]
      }
    ]
  }
}

```

Notification-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten Notification-Hooks `message` mit dem Benachrichtigungstext, ein optionales `title` und `notification_type`, das angibt, welcher Typ ausgelöst wurde.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "Notification",
  "message": "Claude needs your permission to use Bash",
  "title": "Permission needed",
  "notification_type": "permission_prompt"
}
```

Notification-Hooks können Benachrichtigungen nicht blockieren oder ändern. Zusätzlich zu den [JSON-Ausgabefeldern](#), die für alle Hooks verfügbar sind, können Sie `additionalContext` zurückgeben, um Kontext zum Gespräch hinzuzufügen:

Feld	Beschreibung
<code>additionalContext</code>	Zeichenkette, die zu Claudes Kontext hinzugefügt wird

SubagentStart

Wird ausgeführt, wenn ein Claude Code-Subagent über das Agent-Tool spawn't wird. Unterstützt Matcher zum Filtern nach Agent-Typname (integrierte Agents wie `Bash`, `Explore`, `Plan` oder benutzerdefinierte Agent-Namen aus `.claude/agents/`).

SubagentStart-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten SubagentStart-Hooks `agent_id` mit der eindeutigen Kennung für den Subagenten und `agent_type` mit dem Agent-Namen (integrierte Agents wie `"Bash"`, `"Explore"`, `"Plan"` oder benutzerdefinierte Agent-Namen).

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "SubagentStart",
  "agent_id": "agent-abc123",
  "agent_type": "Explore"
}
```

SubagentStart-Hooks können die Subagenten-Erstellung nicht blockieren, aber sie können Kontext in den Subagenten injizieren. Zusätzlich zu den [JSON-Ausgabefeldern](#), die für alle Hooks verfügbar sind, können Sie zurückgeben:

Feld	Beschreibung
<code>additionalContext</code>	Zeichenkette, die zum Kontext des Subagenten hinzugefügt wird

```
{
  "hookSpecificOutput": {
    "hookEventName": "SubagentStart",
    "additionalContext": "Follow security guidelines for this task"
  }
}
```

SubagentStop

Wird ausgeführt, wenn ein Claude Code-Subagent fertig mit der Antwort ist. Passt auf Agent-Typ, gleiche Werte wie SubagentStart.

SubagentStop-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten SubagentStop-Hooks `stop_hook_active`, `agent_id`, `agent_type`, `agent_transcript_path` und `last_assistant_message`. Das Feld `agent_type` ist der Wert, der zum Filtern von Matchern verwendet wird. Der `transcript_path` ist das Transkript der Hauptsitzung, während `agent_transcript_path` das eigene Transkript des Subagenten ist, das in einem ver-

schachtelten `subagents/` -Ordner gespeichert ist. Das Feld `last_assistant_message` enthält den Textinhalt der letzten Antwort des Subagenten, daher können Hooks darauf zugreifen, ohne die Transkriptdatei zu analysieren.

```
{
  "session_id": "abc123",
  "transcript_path": "~/\.claude/projects/.../abc123.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SubagentStop",
  "stop_hook_active": false,
  "agent_id": "def456",
  "agent_type": "Explore",
  "agent_transcript_path": "~/\.claude/projects/.../abc123/subagents/agent-
def456.jsonl",
  "last_assistant_message": "Analysis complete. Found 3 potential issues..."
}
```

SubagentStop-Hooks verwenden das gleiche Entscheidungskontrollformat wie [Stop-Hooks](#).

Stop

Wird ausgeführt, wenn der Haupt-Claude Code-Agent fertig mit der Antwort ist. Wird nicht ausgeführt, wenn der Stopp durch eine Benutzerunterbrechung verursacht wurde.

Stop-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten Stop-Hooks `stop_hook_active` und `last_assistant_message`. Das Feld `stop_hook_active` ist `true`, wenn Claude Code bereits als Ergebnis eines Stop-Hooks fortgesetzt wird. Überprüfen Sie diesen Wert oder verarbeiten Sie das Transkript, um zu verhindern, dass Claude Code unbegrenzt läuft. Das Feld `last_assistant_message` enthält den Textinhalt von Claudes letzter Antwort, daher können Hooks darauf zugreifen, ohne die Transkriptdatei zu analysieren.

```
{
  "session_id": "abc123",
  "transcript_path": "~/claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "Stop",
  "stop_hook_active": true,
  "last_assistant_message": "I've completed the refactoring. Here's a summary..."
}
```

Stop-Entscheidungskontrolle

Stop - und **SubagentStop** -Hooks können steuern, ob Claude fortgesetzt wird. Zusätzlich zu den [JSON-Ausgabefeldern](#), die für alle Hooks verfügbar sind, kann Ihr Hook-Skript diese ereignisspezifischen Felder zurückgeben:

Feld	Beschreibung
decision	"block" verhindert, dass Claude stoppt. Weglassen, um Claude zu stoppen
reason	Erforderlich, wenn decision "block" ist. Teilt Claude mit, warum es fortgesetzt werden sollte

```
{
  "decision": "block",
  "reason": "Must be provided when Claude is blocked from stopping"
}
```

TeammateIdle

Wird ausgeführt, wenn ein [Agent-Team](#)-Teammate nach Abschluss seiner Runde untätig werden soll. Verwenden Sie dies, um Qualitätsgates vor dem Stoppen eines Teammates durchzusetzen, wie das Erfordern von bestandenen Lint-Checks oder das Überprüfen, dass Ausgabedateien vorhanden sind.

Wenn ein `TeammateIdle`-Hook mit Code 2 beendet wird, erhält der Teammate die `stderr`-Nachricht als Feedback und arbeitet weiter, anstatt untätig zu werden. Um den Teammate vollständig zu stoppen, anstatt ihn erneut auszuführen, geben Sie JSON mit `{"continue": false, "stopReason": "..."}` zurück. `TeammateIdle`-Hooks unterstützen keine Matcher und werden bei jedem Auftreten ausgelöst.

TeammateIdle-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten `TeammateIdle`-Hooks `teammate_name` und `team_name`.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "TeammateIdle",
  "teammate_name": "researcher",
  "team_name": "my-project"
}
```

Feld	Beschreibung
<code>teammate_name</code>	Name des Teammates, das untätig werden soll
<code>team_name</code>	Name des Teams

TeammateIdle-Entscheidungskontrolle

`TeammateIdle`-Hooks unterstützen zwei Möglichkeiten, das Teammate-Verhalten zu steuern:

- **Exit-Code 2:** Der Teammate erhält die `stderr`-Nachricht als Feedback und arbeitet weiter, anstatt untätig zu werden.
- **JSON** `{"continue": false, "stopReason": "..."}` : Stoppt den Teammate vollständig, was dem `Stop`-Hook-Verhalten entspricht. Der `stopReason` wird dem Benutzer angezeigt.

Dieses Beispiel prüft, dass ein Build-Artefakt vorhanden ist, bevor ein Teammate untätig werden darf:

```
#!/bin/bash

if [ ! -f "./dist/output.js" ]; then
    echo "Build artifact missing. Run the build before stopping." >&2
    exit 2
fi

exit 0
```

TaskCompleted

Wird ausgeführt, wenn eine Aufgabe als abgeschlossen markiert wird. Dies wird in zwei Situationen ausgelöst: wenn ein Agent eine Aufgabe explizit über das TaskUpdate-Tool als abgeschlossen markiert, oder wenn ein [Agent-Team](#)-Teammate seine Runde mit laufenden Aufgaben beendet. Verwenden Sie dies, um Abschluss-Kriterien wie bestandene Tests oder Lint-Checks durchzusetzen, bevor eine Aufgabe geschlossen werden kann.

Wenn ein `TaskCompleted`-Hook mit Code 2 beendet wird, wird die Aufgabe nicht als abgeschlossen markiert und die stderr-Nachricht wird dem Modell als Feedback zurückgegeben. Um den Teammate vollständig zu stoppen, anstatt ihn erneut auszuführen, geben Sie JSON mit `{"continue": false, "stopReason": "...}"` zurück. TaskCompleted-Hooks unterstützen keine Matcher und werden bei jedem Auftreten ausgelöst.

TaskCompleted-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten TaskCompleted-Hooks `task_id`, `task_subject` und optional `task_description`, `teammate_name` und `team_name`.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "TaskCompleted",
  "task_id": "task-001",
  "task_subject": "Implement user authentication",
  "task_description": "Add login and signup endpoints",
  "teammate_name": "implementer",
  "team_name": "my-project"
}
```

Feld	Beschreibung
<code>task_id</code>	Kennung der abgeschlossenen Aufgabe
<code>task_subject</code>	Titel der Aufgabe
<code>task_description</code>	Detaillierte Beschreibung der Aufgabe. Kann fehlen
<code>teammate_name</code>	Name des Teammates, das die Aufgabe abschließt. Kann fehlen
<code>team_name</code>	Name des Teams. Kann fehlen

TaskCompleted-Entscheidungskontrolle

TaskCompleted-Hooks unterstützen zwei Möglichkeiten, die Aufgabenvervollständigung zu steuern:

- **Exit-Code 2:** Die Aufgabe wird nicht als abgeschlossen markiert und die stderr-Nachricht wird dem Modell als Feedback zurückgegeben.
- **JSON** `{"continue": false, "stopReason": "..."}:` Stoppt den Teammate vollständig, was dem `Stop`-Hook-Verhalten entspricht. Der `stopReason` wird dem Benutzer angezeigt.

Dieses Beispiel führt Tests aus und blockiert die Aufgabenvervollständigung, wenn sie fehlschlagen:


```
#!/bin/bash
INPUT=$(cat)
TASK_SUBJECT=$(echo "$INPUT" | jq -r '.task_subject')

## Führen Sie die Test-Suite aus
if ! npm test 2>&1; then
    echo "Tests not passing. Fix failing tests before completing: $TASK_SUBJECT" >&2
    exit 2
fi

exit 0
```

ConfigChange

Wird ausgeführt, wenn sich eine Konfigurationsdatei während einer Sitzung ändert. Verwenden Sie dies, um Einstellungsänderungen zu überprüfen, Sicherheitsrichtlinien durchzusetzen oder nicht autorisierte Änderungen an Konfigurationsdateien zu blockieren.

ConfigChange-Hooks werden für Änderungen an Einstellungsdateien, verwalteten Richtlinienereinstellungen und Skill-Dateien ausgelöst. Das Feld `source` in der Eingabe teilt Ihnen mit, welche Art von Konfiguration sich geändert hat, und das optionale Feld `file_path` gibt den Pfad zur geänderten Datei an.

Der Matcher filtert auf die Konfigurationsquelle:

Matcher	Wann es ausgelöst wird
<code>user_settings</code>	<code>~/.claude/settings.json</code> ändert sich
<code>project_settings</code>	<code>.claude/settings.json</code> ändert sich
<code>local_settings</code>	<code>.claude/settings.local.json</code> ändert sich
<code>policy_settings</code>	Verwaltete Richtlinienereinstellungen ändern sich
<code>skills</code>	Eine Skill-Datei in <code>.claude/skills/</code> ändert sich

Dieses Beispiel protokolliert alle Konfigurationsänderungen für Sicherheitsaudits:

```
{
  "hooks": {
    "ConfigChange": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/audit-config-
change.sh"
          }
        ]
      }
    ]
  }
}
```

ConfigChange-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten ConfigChange-Hooks `source` und optional `file_path`. Das Feld `source` gibt an, welche Konfigurationsart sich geändert hat, und `file_path` gibt den Pfad zur spezifischen Datei an, die geändert wurde.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "ConfigChange",
  "source": "project_settings",
  "file_path": "/Users/.../my-project/.claude/settings.json"
}
```

ConfigChange-Entscheidungskontrolle

ConfigChange-Hooks können Konfigurationsänderungen von der Anwendung blockieren. Verwenden Sie Exit-Code 2 oder ein JSON- `decision`, um die Änderung zu verhindern. Wenn blockiert, werden die neuen Einstellungen nicht auf die laufende Sitzung angewendet.

Feld	Beschreibung
<code>decision</code>	<code>"block"</code> verhindert die Anwendung der Konfigurationsänderung. Weglassen, um die Änderung zu erlauben
<code>reason</code>	Erklärung, die dem Benutzer angezeigt wird, wenn <code>decision</code> <code>"block"</code> ist

```
{
  "decision": "block",
  "reason": "Configuration changes to project settings require admin approval"
}
```

`policy_settings`-Änderungen können nicht blockiert werden. Hooks werden immer noch für `policy_settings`-Quellen ausgelöst, daher können Sie sie für Audit-Protokollierung verwenden, aber jede Blockierungsentscheidung wird ignoriert. Dies stellt sicher, dass von Unternehmen verwaltete Einstellungen immer wirksam werden.

WorktreeCreate

Wenn Sie `claude --worktree` ausführen oder ein `Subagent isolation: "worktree"` verwendet, erstellt Claude Code eine isolierte Arbeitskopie mit `git worktree`. Wenn Sie einen WorktreeCreate-Hook konfigurieren, ersetzt er das Standard-Git-Verhalten und ermöglicht es Ihnen, ein anderes Versionskontrollsystem wie SVN, Perforce oder Mercurial zu verwenden.

Der Hook muss den absoluten Pfad zum erstellten Worktree-Verzeichnis auf stdout ausgeben. Claude Code verwendet diesen Pfad als Arbeitsverzeichnis für die isolierte Sitzung.

Dieses Beispiel erstellt eine SVN-Arbeitskopie und gibt den Pfad aus, damit Claude Code ihn verwenden kann. Ersetzen Sie die Repository-URL durch Ihre eigene:

```
{
  "hooks": {
    "WorktreeCreate": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'NAME=$(jq -r .name); DIR=\"$HOME/.claude/
worktrees/$NAME\"; svn checkout https://svn.example.com/repo/trunk
\\$DIR\" >&2 && echo \\$DIR\"'"
          }
        ]
      }
    ]
  }
}
```

Der Hook liest den Worktree- **name** aus der JSON-Eingabe auf stdin, checkt eine frische Kopie in ein neues Verzeichnis aus und gibt den Verzeichnispfad aus. Das **echo** in der letzten Zeile ist das, was Claude Code als Worktree-Pfad liest. Leiten Sie jede andere Ausgabe zu stderr um, damit sie nicht mit dem Pfad interferiert.

WorktreeCreate-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten WorktreeCreate-Hooks das Feld **name**. Dies ist eine Slug-Kennung für den neuen Worktree, entweder vom Benutzer angegeben oder automatisch generiert (zum Beispiel **bold-oak-a3f2**).

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "WorktreeCreate",
  "name": "feature-auth"
}
```

WorktreeCreate-Ausgabe

Der Hook muss den absoluten Pfad zum erstellten Worktree-Verzeichnis auf stdout ausgeben. Wenn der Hook fehlschlägt oder keine Ausgabe erzeugt, schlägt die Worktree-Erstellung mit einem Fehler fehl.

WorktreeCreate-Hooks verwenden nicht das Standard-Allow/Block-Entscheidungsmodell. Stattdessen bestimmt der Erfolg oder Misserfolg des Hooks das Ergebnis. Nur `type: "command"`-Hooks werden unterstützt.

WorktreeRemove

Das Bereinigungsgegenstück zu [WorktreeCreate](#). Dieser Hook wird ausgelöst, wenn ein Worktree entfernt wird, entweder wenn Sie eine `--worktree`-Sitzung beenden und wählen, sie zu entfernen, oder wenn ein Subagent mit `isolation: "worktree"` fertig ist. Für Git-basierte Worktrees handhabt Claude die Bereinigung automatisch mit `git worktree remove`. Wenn Sie einen WorktreeCreate-Hook für ein nicht-Git-Versionskontrollsystem konfiguriert haben, koppeln Sie ihn mit einem WorktreeRemove-Hook, um die Bereinigung zu handhaben. Ohne einen wird das Worktree-Verzeichnis auf der Festplatte belassen.

Claude Code übergibt den Pfad, den WorktreeCreate auf stdout ausgegeben hat, als `worktree_path` in der Hook-Eingabe. Dieses Beispiel liest diesen Pfad und entfernt das Verzeichnis:

```
{
  "hooks": {
    "WorktreeRemove": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'jq -r .worktree_path | xargs rm -rf'"
          }
        ]
      }
    ]
  }
}
```

WorktreeRemove-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten WorktreeRemove-Hooks das Feld `worktree_path`, das der absolute Pfad zum entfernten Worktree ist.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "WorktreeRemove",
  "worktree_path": "/Users/.../my-project/.claude/worktrees/feature-auth"
}
```

WorktreeRemove-Hooks haben keine Entscheidungskontrolle. Sie können die Worktree-Entfernung nicht blockieren, können aber Bereinigungsaufgaben wie das Entfernen von Versionskontrollstatus oder das Archivieren von Änderungen durchführen. Hook-Fehler werden nur im Debug-Modus protokolliert. Nur `type: "command"`-Hooks werden unterstützt.

PreCompact

Wird ausgeführt, bevor Claude Code einen Komprimierungsvorgang ausführen soll.

Der Matcher-Wert gibt an, ob die Komprimierung manuell oder automatisch ausgelöst wurde:

Matcher	Wann es ausgelöst wird
<code>manual</code>	<code>/compact</code>
<code>auto</code>	Auto-Komprimierung, wenn das Kontextfenster voll ist

PreCompact-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten PreCompact-Hooks `trigger` und `custom_instructions`. Für `manual` enthält `custom_instructions` das, was der Benutzer in `/compact` übergibt. Für `auto` ist `custom_instructions` leer.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "PreCompact",
  "trigger": "manual",
  "custom_instructions": ""
}
```

SessionEnd

Wird ausgeführt, wenn eine Claude Code-Sitzung endet. Nützlich für Bereinigungsaufgaben, Protokollierung von Sitzungsstatistiken oder Speicherung des Sitzungsstatus. Unterstützt Matcher zum Filtern nach Ausstiegsgrund.

Das Feld `reason` in der Hook-Eingabe gibt an, warum die Sitzung endete:

Grund	Beschreibung
<code>clear</code>	Sitzung mit <code>/clear</code> -Befehl gelöscht
<code>logout</code>	Benutzer hat sich abgemeldet
<code>prompt_input_exit</code>	Benutzer hat beendet, während die Prompt-Eingabe sichtbar war
<code>bypass_permissions_disabled</code>	Bypass-Berechtigungsmodus wurde deaktiviert
<code>other</code>	Andere Ausstiegsgründe

SessionEnd-Eingabe

Zusätzlich zu den [gemeinsamen Eingabefeldern](#) erhalten SessionEnd-Hooks ein `reason`-Feld, das angibt, warum die Sitzung endete. Siehe die [Grund-Tabelle](#) oben für alle Werte.

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/... ",
  "permission_mode": "default",
  "hook_event_name": "SessionEnd",
  "reason": "other"
}
```

SessionEnd-Hooks haben keine Entscheidungskontrolle. Sie können die Sitzungsbeendigung nicht blockieren, können aber Bereinigungsaufgaben durchführen.

Prompt-basierte Hooks

Zusätzlich zu Command- und HTTP-Hooks unterstützt Claude Code Prompt-basierte Hooks (`type: "prompt"`), die ein LLM verwenden, um zu evaluieren, ob eine Aktion erlaubt oder blockiert werden soll, und Agent-Hooks (`type: "agent"`), die einen agenten-gesteuerten Verifizierer mit Tool-Zugriff spawnen. Nicht alle Ereignisse unterstützen jeden Hook-Typ.

Ereignisse, die alle vier Hook-Typen unterstützen (`command` , `http` , `prompt` und `agent`):

- `PermissionRequest`
- `PostToolUse`
- `PostToolUseFailure`
- `PreToolUse`
- `Stop`
- `SubagentStop`
- `TaskCompleted`
- `UserPromptSubmit`

Ereignisse, die nur `type: "command"` -Hooks unterstützen:

- `ConfigChange`
- `InstructionsLoaded`
- `Notification`
- `PreCompact`
- `SessionEnd`

- `SessionStart`
- `SubagentStart`
- `TeammateIdle`
- `WorktreeCreate`
- `WorktreeRemove`

Wie Prompt-basierte Hooks funktionieren

Anstatt einen Bash-Befehl auszuführen, Prompt-basierte Hooks:

1. Senden die Hook-Eingabe und Ihren Prompt an ein Claude-Modell, standardmäßig Haiku
2. Das LLM antwortet mit strukturiertem JSON, das eine Entscheidung enthält
3. Claude Code verarbeitet die Entscheidung automatisch

Prompt-Hook-Konfiguration

Setzen Sie `type` auf `"prompt"` und geben Sie eine `prompt`-Zeichenkette anstelle eines `command` an. Verwenden Sie den Platzhalter `$ARGUMENTS`, um die Hook-Eingabedaten in Ihren Prompt-Text einzufügen. Claude Code sendet den kombinierten Prompt und die Eingabe an ein schnelles Claude-Modell, das eine JSON-Entscheidung zurückgibt.

Dieser `Stop`-Hook fragt das LLM, ob Claude stoppen soll, bevor es fertig ist:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if Claude should stop: $ARGUMENTS. Check if all
tasks are complete."
          }
        ]
      }
    ]
  }
}
```

Feld	Erforderlich	Beschreibung
<code>type</code>	ja	Muss <code>"prompt"</code> sein
<code>prompt</code>	ja	Der Prompt-Text zum Senden an das LLM. Verwenden Sie <code>\$ARGUMENTS</code> als Platzhalter für die Hook-Eingabe JSON. Wenn <code>\$ARGUMENTS</code> nicht vorhanden ist, wird die Eingabe JSON an den Prompt angehängt
<code>model</code>	nein	Modell zur Verwendung für die Evaluierung. Standardwert ist ein schnelles Modell
<code>timeout</code>	nein	Timeout in Sekunden. Standard: 30

Response-Schema

Das LLM muss mit JSON antworten, das Folgendes enthält:

```
{
  "ok": true | false,
  "reason": "Explanation for the decision"
}
```

Feld	Beschreibung
<code>ok</code>	<code>true</code> erlaubt die Aktion, <code>false</code> verhindert sie
<code>reason</code>	Erforderlich, wenn <code>ok false</code> ist. Erklärung, die Claude angezeigt wird

Beispiel: Multi-Kriterien-Stop-Hook

Dieser `Stop`-Hook verwendet einen detaillierten Prompt, um drei Bedingungen zu überprüfen, bevor Claude stoppen darf. Wenn `"ok" false` ist, setzt Claude die Arbeit mit der bereitgestellten Begründung als nächste Anweisung fort. `SubagentStop`-Hooks verwenden das gleiche Format, um zu evaluieren, ob ein `Subagent` stoppen soll:

```

{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "You are evaluating whether Claude should stop working.
Context: $ARGUMENTS\n\nAnalyze the conversation and determine if:\n1. All user-
requested tasks are complete\n2. Any errors need to be addressed\n3. Follow-up
work is needed\n\nRespond with JSON: {\"ok\": true} to allow stopping, or
{\"ok\": false, \"reason\": \"your explanation\"} to continue working.",
            "timeout": 30
          }
        ]
      }
    ]
  }
}

```

Agent-basierte Hooks

Agent-basierte Hooks (`type: "agent"`) sind wie Prompt-basierte Hooks, aber mit Multi-Turn-Tool-Zugriff. Anstelle eines einzelnen LLM-Aufrufs spawnnt ein Agent-Hook einen Subagenten, der Dateien lesen, Code durchsuchen und die Codebasis überprüfen kann, um Bedingungen zu überprüfen. Agent-Hooks unterstützen die gleichen Ereignisse wie Prompt-basierte Hooks.

Wie Agent-Hooks funktionieren

Wenn ein Agent-Hook ausgelöst wird:

1. Claude Code spawnnt einen Subagenten mit Ihrem Prompt und der Hook-Eingabe
2. Der Subagent kann Tools wie Read, Grep und Glob verwenden, um zu untersuchen
3. Nach bis zu 50 Turns gibt der Subagent eine strukturierte `{ "ok": true/false }`-Entscheidung zurück
4. Claude Code verarbeitet die Entscheidung auf die gleiche Weise wie ein Prompt-Hook

Agent-Hooks sind nützlich, wenn die Überprüfung das Überprüfen tatsächlicher Dateien oder Test-Ausgabe erfordert, nicht nur die Evaluierung der Hook-Eingabedaten allein.

Agent-Hook-Konfiguration

Setzen Sie `type` auf `"agent"` und geben Sie eine `prompt`-Zeichenkette an. Die Konfigurationsfelder sind die gleichen wie [Prompt-Hooks](#), mit einem längeren Standard-Timeout:

Feld	Erforderlich	Beschreibung
<code>type</code>	ja	Muss <code>"agent"</code> sein
<code>prompt</code>	ja	Prompt, der beschreibt, was zu überprüfen ist. Verwenden Sie <code>\$ARGUMENTS</code> als Platzhalter für die Hook-Eingabe JSON
<code>model</code>	nein	Modell zur Verwendung. Standardwert ist ein schnelles Modell
<code>timeout</code>	nein	Timeout in Sekunden. Standard: 60

Das Response-Schema ist das gleiche wie Prompt-Hooks: `{ "ok": true }` zum Erlauben oder `{ "ok": false, "reason": "..." }` zum Blockieren.

Dieser `Stop`-Hook überprüft, dass alle Unit-Tests bestanden sind, bevor Claude fertig wird:

```

{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

Hooks im Hintergrund ausführen

Standardmäßig blockieren Hooks die Ausführung von Claude, bis sie abgeschlossen sind. Für lang laufende Aufgaben wie Bereitstellungen, Test-Suites oder externe API-Aufrufe setzen Sie `"async": true`, um den Hook im Hintergrund auszuführen, während Claude weiterarbeitet. Asynchrone Hooks können nicht blockieren oder das Verhalten von Claude steuern: Response-Felder wie `decision`, `permissionDecision` und `continue` haben keine Auswirkung, da die Aktion, die sie steuern würden, bereits abgeschlossen ist.

Konfigurieren Sie einen asynchronen Hook

Fügen Sie `"async": true` zur Konfiguration eines Command-Hooks hinzu, um ihn im Hintergrund auszuführen, ohne Claude zu blockieren. Dieses Feld ist nur auf `type: "command"`-Hooks verfügbar.

Dieser Hook führt ein Test-Skript nach jedem `Write`-Tool-Aufruf aus. Claude arbeitet sofort weiter, während `run-tests.sh` für bis zu 120 Sekunden ausgeführt wird. Wenn das Skript fertig ist, wird seine Ausgabe beim nächsten Gesprächsturn geliefert:

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/run-tests.sh",
            "async": true,
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

Das Feld `timeout` setzt die maximale Zeit in Sekunden für den Hintergrundprozess. Wenn nicht angegeben, verwenden asynchrone Hooks den gleichen 10-Minuten-Standard wie synchrone Hooks.

Wie asynchrone Hooks ausgeführt werden

Wenn ein asynchroner Hook ausgelöst wird, startet Claude Code den Hook-Prozess und setzt sofort fort, ohne auf den Abschluss zu warten. Der Hook erhält die gleiche JSON-Eingabe über stdin wie ein synchroner Hook.

Nachdem der Hintergrundprozess beendet ist, wenn der Hook eine JSON-Response mit einem `systemMessage`- oder `additionalContext`-Feld erzeugt hat, wird dieser Inhalt Claude beim nächsten Gesprächsturn als Kontext geliefert.

Beispiel: Tests nach Dateänderungen ausführen

Dieser Hook startet eine Test-Suite im Hintergrund, wenn Claude eine Datei schreibt, und meldet die Ergebnisse Claude, wenn die Tests fertig sind. Speichern Sie dieses Skript unter `.claude/hooks/run-tests-async.sh` in Ihrem Projekt und machen Sie es mit `chmod +x` ausführbar:

```
#!/bin/bash
## run-tests-async.sh

## Hook-Eingabe von stdin lesen
INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

## Tests nur für Quelldateien ausführen
if [[ "$FILE_PATH" != *.ts && "$FILE_PATH" != *.js ]]; then
    exit 0
fi

## Tests ausführen und Ergebnisse über systemMessage melden
RESULT=$(npm test 2>&1)
EXIT_CODE=$?

if [ $EXIT_CODE -eq 0 ]; then
    echo "{\"systemMessage\": \"Tests passed after editing $FILE_PATH\"}"
else
    echo "{\"systemMessage\": \"Tests failed after editing $FILE_PATH: $RESULT\"}"
fi
```

Fügen Sie dann diese Konfiguration zu `.claude/settings.json` im Projekt-Root hinzu. Das Flag `async: true` ermöglicht es Claude, weiterarbeiten zu können, während Tests ausgeführt werden:

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/run-tests-async.sh",
            "async": true,
            "timeout": 300
          }
        ]
      }
    ]
  }
}

```

Einschränkungen

Asynchrone Hooks haben mehrere Einschränkungen im Vergleich zu synchronen Hooks:

- Nur **type: "command"** -Hooks unterstützen **async**. Prompt-basierte Hooks können nicht asynchron ausgeführt werden.
- Asynchrone Hooks können Tool-Aufrufe nicht blockieren oder Entscheidungen zurückgeben. Zu dem Zeitpunkt, an dem der Hook abgeschlossen ist, hat die auslösende Aktion bereits stattgefunden.
- Hook-Ausgabe wird beim nächsten Gesprächsturn geliefert. Wenn die Sitzung untätig ist, wartet die Response, bis die nächste Benutzerinteraktion erfolgt.
- Jede Ausführung erstellt einen separaten Hintergrundprozess. Es gibt keine Deduplizierung über mehrere Auslösungen des gleichen asynchronen Hooks.

Sicherheitsüberlegungen

Haftungsausschluss

Command-Hooks werden mit den vollständigen Berechtigungen Ihres System-Benutzers ausgeführt.

Warning:

Command-Hooks führen Shell-Befehle mit Ihren vollständigen Berechtigungen aus. Sie können alle Dateien ändern, löschen oder zugreifen, auf die Ihr Benutzerkonto zugreifen kann. Überprüfen und testen Sie alle Hook-Befehle, bevor Sie sie zu Ihrer Konfiguration hinzufügen.

Best Practices für Sicherheit

Beachten Sie diese Praktiken beim Schreiben von Hooks:

- **Validieren und bereinigen Sie Eingaben:** Vertrauen Sie Eingabedaten niemals blind
- **Zitieren Sie immer Shell-Variablen:** Verwenden Sie `"$VAR"` nicht `$VAR`
- **Blockieren Sie Pfad-Traversal:** Prüfen Sie auf `..` in Dateipfaden
- **Verwenden Sie absolute Pfade:** Geben Sie vollständige Pfade für Skripte an, verwenden Sie `"$CLAUDE_PROJECT_DIR"` für das Projekt-Root
- **Überspringen Sie sensible Dateien:** Vermeiden Sie `.env`, `.git/`, Schlüssel, etc.

Debug-Hooks

Führen Sie `claude --debug` aus, um Hook-Ausführungsdetails zu sehen, einschließlich welche Hooks passten, ihre Exit-Codes und Ausgabe. Schalten Sie den ausführlichen Modus mit `Ctrl+O` um, um Hook-Fortschritt im Transkript zu sehen.

```
[DEBUG] Executing hooks for PostToolUse:Write
[DEBUG] Getting matching hook commands for PostToolUse with query: Write
[DEBUG] Found 1 hook matchers in settings
[DEBUG] Matched 1 hooks for query "Write"
[DEBUG] Found 1 hook commands to execute
[DEBUG] Executing hook command: <Your command> with timeout 600000ms
[DEBUG] Hook command completed with status 0: <Your stdout>
```

Für Troubleshooting häufiger Probleme wie Hooks, die nicht ausgelöst werden, unendliche Stop-Hook-Schleifen oder Konfigurationsfehler, siehe [Einschränkungen und Troubleshooting](#) in der Anleitung.

Automatisieren Sie Workflows mit Hooks

Führen Sie Shell-Befehle automatisch aus, wenn Claude Code Dateien bearbeitet, Aufgaben abschließt oder Eingaben benötigt. Formatieren Sie Code, senden Sie Benachrichtigungen, validieren Sie Befehle und erzwingen Sie Projektregeln.

Hooks sind benutzerdefinierte Shell-Befehle, die an bestimmten Punkten im Lebenszyklus von Claude Code ausgeführt werden. Sie bieten deterministische Kontrolle über das Verhalten von Claude Code und stellen sicher, dass bestimmte Aktionen immer stattfinden, anstatt sich darauf zu verlassen, dass das LLM sie ausführt. Verwenden Sie Hooks, um Projektregeln durchzusetzen, sich wiederholende Aufgaben zu automatisieren und Claude Code mit Ihren vorhandenen Tools zu integrieren.

Für Entscheidungen, die Urteilsvermögen erfordern, anstatt deterministischer Regeln, können Sie auch [Prompt-basierte Hooks](#) oder [Agent-basierte Hooks](#) verwenden, die ein Claude-Modell zur Bewertung von Bedingungen nutzen.

Für andere Möglichkeiten, Claude Code zu erweitern, siehe [skills](#) zum Geben zusätzlicher Anweisungen und ausführbarer Befehle, [subagents](#) zum Ausführen von Aufgaben in isolierten Kontexten und [plugins](#) zum Verpacken von Erweiterungen, die über Projekte hinweg freigegeben werden können.

Tip:

Dieser Leitfaden behandelt häufige Anwendungsfälle und wie Sie beginnen. Für vollständige Event-Schemas, JSON-Ein-/Ausgabeformate und erweiterte Funktionen wie asynchrone Hooks und MCP-Tool-Hooks siehe die [Hooks-Referenz](#).

Richten Sie Ihren ersten Hook ein

Der schnellste Weg, einen Hook zu erstellen, ist über das interaktive Menü `/hooks` in Claude Code. Diese Anleitung erstellt einen Desktop-Benachrichtigungs-Hook, damit Sie benachrichtigt werden, wenn Claude auf Ihre Eingabe wartet, anstatt das Terminal zu beobachten.

Step 1: Öffnen Sie das Hooks-Menü

Geben Sie `/hooks` in die Claude Code CLI ein. Sie sehen eine Liste aller verfügbaren Hook-Events sowie eine Option zum Deaktivieren aller Hooks. Jedes Event entspricht einem Punkt im Lebenszyklus von Claude, an dem Sie benutzerdefinierten Code ausführen können. Wählen Sie `Notification` aus, um einen Hook zu erstellen, der aktiviert wird, wenn Claude Ihre Aufmerksamkeit benötigt.

Step 2: Konfigurieren Sie den Matcher

Das Menü zeigt eine Liste von Matchern, die filtern, wann der Hook aktiviert wird. Setzen Sie den Matcher auf `*`, um bei allen Benachrichtigungstypen aktiviert zu werden. Sie können ihn später eingrenzen, indem Sie den Matcher auf einen bestimmten Wert wie `permission_prompt` oder `idle_prompt` ändern.

Step 3: Fügen Sie Ihren Befehl hinzu

Wählen Sie `+ Add new hook...`. Das Menü fordert Sie auf, einen Shell-Befehl einzugeben, der ausgeführt wird, wenn das Event aktiviert wird. Hooks führen jeden Shell-Befehl aus, den Sie bereitstellen, sodass Sie das integrierte Benachrichtigungstool Ihrer Plattform verwenden können. Kopieren Sie den Befehl für Ihr Betriebssystem:

macOS

Verwendet `osascript`, um eine native macOS-Benachrichtigung über AppleScript auszulösen:

```
osascript -e 'display notification "Claude Code needs your attention" with title "Claude Code"'
```

Linux

Verwendet `notify-send`, das auf den meisten Linux-Desktops mit einem Benachrichtigungsdaemon vorinstalliert ist:

```
notify-send 'Claude Code' 'Claude Code needs your attention'
```

Windows (PowerShell)

Verwendet PowerShell, um ein natives Meldungsfeld über .NET Windows Forms anzuzeigen:

```
powershell.exe -Command "[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms'); [System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention', 'Claude Code')"
```

Step 4: Wählen Sie einen Speicherort

Das Menü fragt, wo die Hook-Konfiguration gespeichert werden soll. Wählen Sie **User settings** aus, um sie in `~/.claude/settings.json` zu speichern, was den Hook auf alle Ihre Projekte anwendet. Sie könnten auch **Project settings** wählen, um ihn auf das aktuelle Projekt zu beschränken. Siehe [Hook-Speicherort konfigurieren](#) für alle verfügbaren Bereiche.

Step 5: Testen Sie den Hook

Drücken Sie **Esc**, um zur CLI zurückzukehren. Bitten Sie Claude, etwas zu tun, das eine Berechtigung erfordert, und wechseln Sie dann weg vom Terminal. Sie sollten eine Desktop-Benachrichtigung erhalten.

Was Sie automatisieren können

Hooks ermöglichen es Ihnen, Code an Schlüsselpunkten im Lebenszyklus von Claude Code auszuführen: Dateien nach Bearbeitungen formatieren, Befehle vor der Ausführung blockieren, Benachrichtigungen senden, wenn Claude Eingaben benötigt, Kontext beim Sitzungsstart injizieren und vieles mehr. Für die vollständige Liste der Hook-Events siehe die [Hooks-Referenz](#).

Jedes Beispiel enthält einen einsatzbereiten Konfigurationsblock, den Sie einer [Einstellungsdatei](#) hinzufügen. Die häufigsten Muster:

- [Benachrichtigung erhalten, wenn Claude Eingaben benötigt](#)
- [Code nach Bearbeitungen automatisch formatieren](#)
- [Bearbeitungen geschützter Dateien blockieren](#)
- [Kontext nach Komprimierung erneut injizieren](#)
- [Konfigurationsänderungen prüfen](#)

Benachrichtigung erhalten, wenn Claude Eingaben benötigt

Erhalten Sie eine Desktop-Benachrichtigung, wenn Claude die Arbeit beendet und Ihre Eingabe benötigt, damit Sie zu anderen Aufgaben wechseln können, ohne das Terminal zu überprüfen.

Dieser Hook verwendet das **Notification**-Event, das aktiviert wird, wenn Claude auf Eingaben oder Berechtigungen wartet. Jede Registerkarte unten verwendet den nativen Benachrichtigungsbefehl der Plattform. Fügen Sie dies zu `~/.claude/settings.json` hinzu, oder verwenden Sie die [interaktive Anleitung](#) oben, um es mit `/hooks` zu konfigurieren:

macOS

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "osascript -e 'display notification \"Claude Code needs your attention\" with title \"Claude Code\"'"
          }
        ]
      }
    ]
  }
}
```

Linux

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "notify-send 'Claude Code' 'Claude Code needs your
attention'"
          }
        ]
      }
    ]
  }
}
```

Windows (PowerShell)

```

{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "powershell.exe -Command \"[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms'); [System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention', 'Claude Code')\"";
          }
        ]
      }
    ]
  }
}

```

Code nach Bearbeitungen automatisch formatieren

Führen Sie [Prettier](#) automatisch auf jeder Datei aus, die Claude bearbeitet, damit die Formatierung konsistent bleibt, ohne manuelle Eingriffe.

Dieser Hook verwendet das `PostToolUse` -Event mit einem `Edit|Write` -Matcher, sodass er nur nach Datei-Bearbeitungs-Tools ausgeführt wird. Der Befehl extrahiert den bearbeiteten Dateipfad mit `jq` und übergibt ihn an Prettier. Fügen Sie dies zu `.claude/settings.json` in Ihrem Projektverzeichnis hinzu:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --
write"
          }
        ]
      }
    ]
  }
}
```

Note:

Die Bash-Beispiele auf dieser Seite verwenden `jq` zum JSON-Parsing. Installieren Sie es mit `brew install jq` (macOS), `apt-get install jq` (Debian/Ubuntu), oder siehe [jq -Dow-nloads](#).

Bearbeitungen geschützter Dateien blockieren

Verhindern Sie, dass Claude sensible Dateien wie `.env`, `package-lock.json` oder alles in `.git/` ändert. Claude erhält Feedback, das erklärt, warum die Bearbeitung blockiert wurde, sodass es seinen Ansatz anpassen kann.

Dieses Beispiel verwendet eine separate Skriptdatei, die der Hook aufruft. Das Skript überprüft den Zielpfad gegen eine Liste geschützter Muster und beendet sich mit Code 2, um die Bearbeitung zu blockieren.

Step 1: Erstellen Sie das Hook-Skript

Speichern Sie dies unter `.claude/hooks/protect-files.sh` :


```
#!/bin/bash
## protect-files.sh

INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

PROTECTED_PATTERNS=( ".env" "package-lock.json" ".git/" )

for pattern in "${PROTECTED_PATTERNS[@]}; do
  if [ "$FILE_PATH" = "$pattern" ]; then
    echo "Blocked: $FILE_PATH matches protected pattern '$pattern'" >&2
    exit 2
  fi
done

exit 0
```

Step 2: Machen Sie das Skript ausführbar (macOS/Linux)

Hook-Skripte müssen ausführbar sein, damit Claude Code sie ausführen kann:

```
chmod +x .claude/hooks/protect-files.sh
```

Step 3: Registrieren Sie den Hook

Fügen Sie einen **PreToolUse**-Hook zu **.claude/settings.json** hinzu, der das Skript vor jedem **Edit** - oder **Write** -Tool-Aufruf ausführt:

```

{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/protect-files.sh"
          }
        ]
      }
    ]
  }
}

```

Kontext nach Komprimierung erneut injizieren

Wenn Claudes Kontextfenster voll wird, fasst die Komprimierung das Gespräch zusammen, um Platz freizugeben. Dies kann wichtige Details verlieren. Verwenden Sie einen `SessionStart`-Hook mit einem `compact`-Matcher, um nach jeder Komprimierung kritischen Kontext erneut zu injizieren.

Jeder Text, den Ihr Befehl auf stdout schreibt, wird zu Claudes Kontext hinzugefügt. Dieses Beispiel erinnert Claude an Projektkonventionen und aktuelle Arbeiten. Fügen Sie dies zu `.claude/settings.json` in Ihrem Projektverzeichnis hinzu:

```

{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Reminder: use Bun, not npm. Run bun test before committing. Current sprint: auth refactor.'"
          }
        ]
      }
    ]
  }
}

```

Sie können das `echo` durch jeden Befehl ersetzen, der dynamische Ausgabe erzeugt, wie `git log --oneline -5`, um aktuelle Commits anzuzeigen. Zum Injizieren von Kontext bei jedem Sitzungsstart sollten Sie stattdessen [CLAUDE.md](#) verwenden. Für Umgebungsvariablen siehe [CLAUDE_ENV_FILE](#) in der Referenz.

Konfigurationsänderungen prüfen

Verfolgen Sie, wenn sich Einstellungs- oder Skills-Dateien während einer Sitzung ändern. Das `ConfigChange`-Event wird aktiviert, wenn ein externer Prozess oder Editor eine Konfigurationsdatei ändert, sodass Sie Änderungen für Compliance protokollieren oder nicht autorisierte Änderungen blockieren können.

Dieses Beispiel hängt jede Änderung an ein Audit-Protokoll an. Fügen Sie dies zu `~/.claude/settings.json` hinzu:

```
{
  "hooks": {
    "ConfigChange": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "jq -c '{timestamp: now | todate, source: .source,
file: .file_path}' >> ~/claude-config-audit.log"
          }
        ]
      }
    ]
  }
}
```

Der Matcher filtert nach Konfigurationstyp: `user_settings`, `project_settings`, `local_settings`, `policy_settings` oder `skills`. Um eine Änderung zu blockieren, beenden Sie mit Code 2 oder geben Sie `{"decision": "block"}` zurück. Siehe die [Config-Change-Referenz](#) für das vollständige Eingabe-Schema.

Wie Hooks funktionieren

Hook-Events werden an bestimmten Lebenszykluspunkten in Claude Code aktiviert. Wenn ein Event aktiviert wird, werden alle übereinstimmenden Hooks parallel ausgeführt, und identische Hook-Befehle werden automatisch dedupliziert. Die folgende Tabelle zeigt jedes Event und wann es ausgelöst wird:

Event	When it fires
<code>SessionStart</code>	When a session begins or resumes
<code>UserPromptSubmit</code>	When you submit a prompt, before Claude processes it
<code>PreToolUse</code>	Before a tool call executes. Can block it
<code>PermissionRequest</code>	When a permission dialog appears
<code>PostToolUse</code>	After a tool call succeeds

Event	When it fires
<code>PostToolUseFailure</code>	After a tool call fails
<code>Notification</code>	When Claude Code sends a notification
<code>SubagentStart</code>	When a subagent is spawned
<code>SubagentStop</code>	When a subagent finishes
<code>Stop</code>	When Claude finishes responding
<code>TeammateIdle</code>	When an agent team teammate is about to go idle
<code>TaskCompleted</code>	When a task is being marked as completed
<code>InstructionsLoaded</code>	When a <code>CLAUDE.md</code> or <code>.claude/rules/*.md</code> file is loaded into context. Fires at session start and when files are lazily loaded during a session
<code>ConfigChange</code>	When a configuration file changes during a session
<code>WorktreeCreate</code>	When a worktree is being created via <code>--worktree</code> or <code>isolation: "worktree"</code> . Replaces default git behavior
<code>WorktreeRemove</code>	When a worktree is being removed, either at session exit or when a subagent finishes
<code>PreCompact</code>	Before context compaction
<code>PostCompact</code>	After context compaction completes
<code>Elicitation</code>	When an MCP server requests user input during a tool call
<code>ElicitationResult</code>	After a user responds to an MCP elicitation, before the response is sent back to the server
<code>SessionEnd</code>	When a session terminates

Jeder Hook hat einen `type`, der bestimmt, wie er ausgeführt wird. Die meisten Hooks verwenden `"type": "command"`, was einen Shell-Befehl ausführt. Drei weitere Typen sind verfügbar:

- `"type": "http"`: POST-Event-Daten an eine URL. Siehe [HTTP-Hooks](#).
- `"type": "prompt"`: Single-Turn-LLM-Bewertung. Siehe [Prompt-basierte Hooks](#).

- **"type": "agent"** : Multi-Turn-Verifizierung mit Tool-Zugriff. Siehe [Agent-basierte Hooks](#).

Eingabe lesen und Ausgabe zurückgeben

Hooks kommunizieren mit Claude Code über stdin, stdout, stderr und Exit-Codes. Wenn ein Event aktiviert wird, übergibt Claude Code ereignisspezifische Daten als JSON an stdin Ihres Skripts. Ihr Skript liest diese Daten, führt seine Arbeit aus und teilt Claude Code mit, was als nächstes zu tun ist, über den Exit-Code.

Hook-Eingabe

Jedes Event enthält gemeinsame Felder wie `session_id` und `cwd`, aber jeder Event-Typ fügt unterschiedliche Daten hinzu. Wenn Claude beispielsweise einen Bash-Befehl ausführt, erhält ein `PreToolUse`-Hook etwa folgendes auf stdin:

```
{
  "session_id": "abc123",           // unique ID for this session
  "cwd": "/Users/sarah/myproject", // working directory when the event fired
  "hook_event_name": "PreToolUse", // which event triggered this hook
  "tool_name": "Bash",             // the tool Claude is about to use
  "tool_input": {                  // the arguments Claude passed to the tool
    "command": "npm test"          // for Bash, this is the shell command
  }
}
```

Ihr Skript kann dieses JSON analysieren und auf alle diese Felder reagieren.

`UserPromptSubmit`-Hooks erhalten stattdessen den `prompt`-Text, `SessionStart`-Hooks erhalten die `source` (startup, resume, clear, compact) und so weiter. Siehe [Gemeinsame Eingabefelder](#) in der Referenz für gemeinsame Felder und jeden Event-Abschnitt für ereignisspezifische Schemas.

Hook-Ausgabe

Ihr Skript teilt Claude Code mit, was als nächstes zu tun ist, indem es auf stdout oder stderr schreibt und mit einem bestimmten Code beendet wird. Beispielsweise ein `PreToolUse`-Hook, der einen Befehl blockieren möchte:

```
#!/bin/bash
INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command')

if echo "$COMMAND" | grep -q "drop table"; then
    echo "Blocked: dropping tables is not allowed" >&2 # stderr becomes Claude's
    feedback
    exit 2 # exit 2 = block the action
fi

exit 0 # exit 0 = let it proceed
```

Der Exit-Code bestimmt, was als nächstes passiert:

- **Exit 0:** Die Aktion wird fortgesetzt. Für `UserPromptSubmit` - und `SessionStart` -Hooks wird alles, was Sie auf stdout schreiben, zu Claudes Kontext hinzugefügt.
- **Exit 2:** Die Aktion wird blockiert. Schreiben Sie einen Grund auf stderr, und Claude erhält ihn als Feedback, sodass er sich anpassen kann.
- **Jeder andere Exit-Code:** Die Aktion wird fortgesetzt. Stderr wird protokolliert, aber nicht Claude angezeigt. Schalten Sie den ausführlichen Modus mit `Ctrl+0` um, um diese Meldungen im Transkript zu sehen.

Strukturierte JSON-Ausgabe

Exit-Codes geben Ihnen zwei Optionen: Zulassen oder Blockieren. Für mehr Kontrolle beenden Sie mit 0 und geben stattdessen ein JSON-Objekt auf stdout aus.

Note:

Verwenden Sie Exit 2, um mit einer stderr-Meldung zu blockieren, oder Exit 0 mit JSON für strukturierte Kontrolle. Mischen Sie sie nicht: Claude Code ignoriert JSON, wenn Sie mit 2 beenden.

Beispielsweise kann ein `PreToolUse` -Hook einen Tool-Aufruf ablehnen und Claude mitteilen, warum, oder ihn dem Benutzer zur Genehmigung eskalieren:

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Use rg instead of grep for better performance"
  }
}
```

Claude Code liest `permissionDecision` und bricht den Tool-Aufruf ab, dann gibt `permissionDecisionReason` an Claude als Feedback zurück. Diese drei Optionen sind spezifisch für `PreToolUse` :

- `"allow"` : Fortfahren ohne Berechtigungsaufforderung anzuzeigen
- `"deny"` : Tool-Aufruf abbrechen und den Grund an Claude senden
- `"ask"` : Berechtigungsaufforderung dem Benutzer wie gewohnt anzeigen

Andere Events verwenden unterschiedliche Entscheidungsmuster. Beispielsweise verwenden `PostToolUse` - und `Stop` -Hooks ein Top-Level-Feld `decision: "block"` , während `PermissionRequest hookSpecificOutput.decision.behavior` verwendet. Siehe die [Zusammenfassungstabelle](#) in der Referenz für eine vollständige Aufschlüsselung nach Event.

Für `UserPromptSubmit` -Hooks verwenden Sie stattdessen `additionalContext` , um Text in Claudes Kontext zu injizieren. Prompt-basierte Hooks (`type: "prompt"`) handhaben die Ausgabe anders: siehe [Prompt-basierte Hooks](#).

Hooks mit Matchern filtern

Ohne einen Matcher wird ein Hook bei jedem Auftreten seines Events aktiviert. Matcher ermöglichen es Ihnen, das einzugrenzen. Wenn Sie beispielsweise einen Formatter nur nach Datei-Bearbeitungen ausführen möchten (nicht nach jedem Tool-Aufruf), fügen Sie einen Matcher zu Ihrem `PostToolUse` -Hook hinzu:


```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "prettier --write ..." }
        ]
      }
    ]
  }
}
```

Der `"Edit|Write"`-Matcher ist ein Regex-Muster, das den Tool-Namen abgleicht. Der Hook wird nur aktiviert, wenn Claude das `Edit`- oder `Write`-Tool verwendet, nicht wenn es `Bash`, `Read` oder ein anderes Tool verwendet.

Jeder Event-Typ gleicht ein bestimmtes Feld ab. Matcher unterstützen exakte Strings und Regex-Muster:

Event	Was der Matcher filtert	Beispiel-Matcher-Werte
<code>PreToolUse</code> , <code>PostToolUse</code> , <code>PostToolUseFailure</code> , <code>PermissionRequest</code>	Tool-Name	<code>Bash</code> , <code>Edit Write</code> , <code>mcp_.*</code>
<code>SessionStart</code>	Wie die Sitzung gestartet wurde	<code>startup</code> , <code>resume</code> , <code>clear</code> , <code>compact</code>
<code>SessionEnd</code>	Warum die Sitzung endete	<code>clear</code> , <code>logout</code> , <code>prompt_input_exit</code> , <code>bypass_permissions_disabled</code> , <code>other</code>
<code>Notification</code>	Benachrichtigungstyp	<code>permission_prompt</code> , <code>idle_prompt</code> , <code>auth_success</code> , <code>elicitation_dialog</code>

Event	Was der Matcher filtert	Beispiel-Matcher-Werte
<code>SubagentStart</code>	Agent-Typ	<code>Bash</code> , <code>Explore</code> , <code>Plan</code> oder benutzerdefinierte Agent-Namen
<code>PreCompact</code>	Was die Komprimierung ausgelöst hat	<code>manual</code> , <code>auto</code>
<code>SubagentStop</code>	Agent-Typ	gleiche Werte wie <code>SubagentStart</code>
<code>ConfigChange</code>	Konfigurationsquelle	<code>user_settings</code> , <code>project_settings</code> , <code>local_settings</code> , <code>policy_settings</code> , <code>skills</code>
<code>UserPromptSubmit</code> , <code>Stop</code> , <code>TeammateIdle</code> , <code>TaskCompleted</code> , <code>WorktreeCreate</code> , <code>WorktreeRemove</code>	Keine Matcher-Unterstützung	Wird immer bei jedem Auftreten aktiviert

Ein paar weitere Beispiele, die Matcher auf verschiedene Event-Typen zeigen:

Jeden Bash-Befehl protokollieren

Gleichen Sie nur `Bash`-Tool-Aufrufe ab und protokollieren Sie jeden Befehl in einer Datei. Das `PostToolUse`-Event wird aktiviert, nachdem der Befehl abgeschlossen ist, sodass `tool_input.command` enthält, was ausgeführt wurde. Der Hook erhält die Event-Daten als JSON auf stdin, und `jq -r '.tool_input.command'` extrahiert nur die Befehlszeichenfolge, die `>>` an die Protokolldatei anhängt:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.command' >> ~/.claude/command-log.txt"
          }
        ]
      }
    ]
  }
}
```

MCP-Tools abgleichen

MCP-Tools verwenden eine andere Namenskonvention als integrierte Tools:

`mcp_<server>__<tool>`, wobei `<server>` der MCP-Servername und `<tool>` das Tool ist, das er bereitstellt. Beispielsweise `mcp_github__search_repositories` oder `mcp_filesystem__read_file`. Verwenden Sie einen Regex-Matcher, um alle Tools von einem bestimmten Server zu erfassen, oder gleichen Sie Server übergreifend mit einem Muster wie `mcp_.*__write.*` ab. Siehe [MCP-Tools abgleichen](#) in der Referenz für die vollständige Liste der Beispiele.

Der folgende Befehl extrahiert den Tool-Namen aus der Hook-JSON-Eingabe mit `jq` und schreibt ihn auf stderr, wo er im ausführlichen Modus (`Ctrl+O`) angezeigt wird:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp_github_.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"GitHub tool called: $(jq -r '.tool_name')\" >&2"
          }
        ]
      }
    ]
  }
}
```

Beim Sitzungsende aufräumen

Das `SessionEnd` -Event unterstützt Matcher auf den Grund, warum die Sitzung endete. Dieser Hook wird nur bei `clear` aktiviert (wenn Sie `/clear` ausführen), nicht bei normalen Exits:

```
{
  "hooks": {
    "SessionEnd": [
      {
        "matcher": "clear",
        "hooks": [
          {
            "type": "command",
            "command": "rm -f /tmp/claude-scratch-*.txt"
          }
        ]
      }
    ]
  }
}
```

Für die vollständige Matcher-Syntax siehe die [Hooks-Referenz](#).

Hook-Speicherort konfigurieren

Wo Sie einen Hook hinzufügen, bestimmt seinen Bereich:

Speicherort	Bereich	Freigegeben
<code>~/.claude/settings.json</code>	Alle Ihre Projekte	Nein, lokal auf Ihrem Computer
<code>.claude/settings.json</code>	Einzelnes Projekt	Ja, kann im Repo committed werden
<code>.claude/settings.local.json</code>	Einzelnes Projekt	Nein, gitignoriert
Verwaltete Richtlinien-einstellungen	Organisationsweit	Ja, von Admin kontrolliert
<code>Plugin hooks/hooks.json</code>	Wenn Plugin aktiviert ist	Ja, mit dem Plugin gebündelt
<code>Skill</code> oder <code>Agent</code> Frontmatter	Während der Skill oder Agent aktiv ist	Ja, in der Komponentendatei definiert

Sie können auch das Menü `/hooks` in Claude Code verwenden, um Hooks interaktiv hinzuzufügen, zu löschen und anzuzeigen. Um alle Hooks auf einmal zu deaktivieren, verwenden Sie den Schalter am unteren Ende des Menüs `/hooks` oder setzen Sie `"disableAllHooks": true` in Ihrer Einstellungsdatei.

Hooks, die über das Menü `/hooks` hinzugefügt werden, werden sofort wirksam. Wenn Sie Einstellungsdateien direkt bearbeiten, während Claude Code ausgeführt wird, werden die Änderungen erst wirksam, wenn Sie sie im Menü `/hooks` überprüfen oder Ihre Sitzung neu starten.

Prompt-basierte Hooks

Für Entscheidungen, die Urteilsvermögen erfordern, anstatt deterministischer Regeln, verwenden Sie `type: "prompt"`-Hooks. Anstatt einen Shell-Befehl auszuführen, sendet Claude Code Ihren Prompt und die Hook-Eingabedaten an ein Claude-Modell (standardmäßig Haiku), um die Entscheidung zu treffen. Sie können ein anderes Modell mit dem Feld `model` angeben, wenn Sie mehr Leistung benötigen.

Die einzige Aufgabe des Modells ist, eine Ja/Nein-Entscheidung als JSON zurückzugeben:

- `"ok": true`: Die Aktion wird fortgesetzt
- `"ok": false`: Die Aktion wird blockiert. Der `"reason"` des Modells wird an Claude zurückgegeben, sodass es sich anpassen kann.

Dieses Beispiel verwendet einen `Stop`-Hook, um das Modell zu fragen, ob alle angeforderten Aufgaben abgeschlossen sind. Wenn das Modell `"ok": false` zurückgibt, arbeitet Claude weiter und verwendet den `reason` als nächste Anweisung:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Check if all tasks are complete. If not, respond with\n{\"ok\": false, \"reason\": \"what remains to be done\"}."
          }
        ]
      }
    ]
  }
}
```

Für vollständige Konfigurationsoptionen siehe [Prompt-basierte Hooks](#) in der Referenz.

Agent-basierte Hooks

Wenn die Verifizierung des Inspizierens von Dateien oder das Ausführen von Befehlen erfordert, verwenden Sie `type: "agent"`-Hooks. Im Gegensatz zu Prompt-Hooks, die einen einzelnen LLM-Aufruf tätigen, spawnen Agent-Hooks einen Subagent, der Dateien lesen, Code durchsuchen und andere Tools verwenden kann, um Bedingungen zu überprüfen, bevor eine Entscheidung zurückgegeben wird.

Agent-Hooks verwenden das gleiche `"ok"` / `"reason"`-Antwortformat wie Prompt-Hooks, aber mit einem längeren Standard-Timeout von 60 Sekunden und bis zu 50 Tool-Use-Turns.

Dieses Beispiel überprüft, dass Tests bestanden werden, bevor Claude beendet werden darf:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

Verwenden Sie Prompt-Hooks, wenn die Hook-Eingabedaten allein ausreichen, um eine Entscheidung zu treffen. Verwenden Sie Agent-Hooks, wenn Sie etwas gegen den tatsächlichen Zustand der Codebasis überprüfen müssen.

Für vollständige Konfigurationsoptionen siehe [Agent-basierte Hooks](#) in der Referenz.

HTTP-Hooks

Verwenden Sie `type: "http"`-Hooks, um Event-Daten per POST an einen HTTP-Endpunkt zu senden, anstatt einen Shell-Befehl auszuführen. Der Endpunkt erhält die gleichen JSON-Daten, die ein Command-Hook auf stdin erhalten würde, und gibt Ergebnisse über den HTTP-Antwortkörper mit dem gleichen JSON-Format zurück.

HTTP-Hooks sind nützlich, wenn Sie möchten, dass ein Webserver, eine Cloud-Funktion oder ein externer Service die Hook-Logik handhabt: Beispielsweise ein gemeinsamer Audit-Service, der Tool-Use-Events über ein Team hinweg protokolliert.

Dieses Beispiel postet jeden Tool-Use an einen lokalen Logging-Service:

```

{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/tool-use",
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}

```

Der Endpunkt sollte einen JSON-Antwortkörper mit dem gleichen [Ausgabeformat](#) wie Command-Hooks zurückgeben. Um einen Tool-Aufruf zu blockieren, geben Sie eine 2xx-Antwort mit den entsprechenden `hookSpecificOutput`-Feldern zurück. HTTP-Statuscodes allein können Aktionen nicht blockieren.

Header-Werte unterstützen Umgebungsvariablen-Interpolation mit `$VAR_NAME` oder `${VAR_NAME}`-Syntax. Nur Variablen, die in der `allowedEnvVars`-Liste aufgeführt sind, werden aufgelöst; alle anderen `$VAR`-Referenzen bleiben leer.

Note:

HTTP-Hooks müssen durch direktes Bearbeiten Ihrer Einstellungs-JSON konfiguriert werden. Das interaktive Menü `/hooks` unterstützt nur das Hinzufügen von Command-Hooks.

Für vollständige Konfigurationsoptionen und Response-Handling siehe [HTTP-Hooks](#) in der Referenz.

Einschränkungen und Fehlerbehebung

Einschränkungen

- Command-Hooks kommunizieren nur über stdout, stderr und Exit-Codes. Sie können Befehle oder Tool-Aufrufe nicht direkt auslösen. HTTP-Hooks kommunizieren stattdessen über den Response-Body.
- Hook-Timeout beträgt standardmäßig 10 Minuten, konfigurierbar pro Hook mit dem Feld `timeout` (in Sekunden).
- `PostToolUse` -Hooks können Aktionen nicht rückgängig machen, da das Tool bereits ausgeführt wurde.
- `PermissionRequest` -Hooks werden nicht im [nicht-interaktiven Modus](#) (`-p`) aktiviert. Verwenden Sie `PreToolUse` -Hooks für automatisierte Berechtigungsentscheidungen.
- `Stop` -Hooks werden aktiviert, wenn Claude antwortet, nicht nur bei Aufgabenabschluss. Sie werden nicht bei Benutzerunterbrechungen aktiviert.

Hook wird nicht aktiviert

Der Hook ist konfiguriert, wird aber nie ausgeführt.

- Führen Sie `/hooks` aus und bestätigen Sie, dass der Hook unter dem richtigen Event angezeigt wird
- Überprüfen Sie, dass das Matcher-Muster den Tool-Namen genau abgleicht (Matcher sind case-sensitiv)
- Überprüfen Sie, dass Sie den richtigen Event-Typ auslösen (z. B. `PreToolUse` wird vor der Tool-Ausführung aktiviert, `PostToolUse` wird danach aktiviert)
- Wenn Sie `PermissionRequest` -Hooks im nicht-interaktiven Modus (`-p`) verwenden, wechseln Sie stattdessen zu `PreToolUse`

Hook-Fehler in der Ausgabe

Sie sehen eine Meldung wie “PreToolUse hook error: ...” im Transkript.

- Ihr Skript wurde unerwartet mit einem Nicht-Null-Code beendet. Testen Sie es manuell, indem Sie Beispiel-JSON pipen:

```
echo '{"tool_name":"Bash","tool_input":{"command":"ls"}}' | ./my-hook.sh
echo $? # Check the exit code
```

- Wenn Sie “command not found” sehen, verwenden Sie absolute Pfade oder `$CLAUDE_PROJECT_DIR`, um Skripte zu referenzieren

- Wenn Sie “jq: command not found” sehen, installieren Sie `jq` oder verwenden Sie Python/Node.js zum JSON-Parsing
- Wenn das Skript überhaupt nicht ausgeführt wird, machen Sie es ausführbar: `chmod +x ./my-hook.sh`

`/hooks` zeigt keine konfigurierten Hooks

Sie haben eine Einstellungsdatei bearbeitet, aber die Hooks werden nicht im Menü angezeigt.

- Starten Sie Ihre Sitzung neu oder öffnen Sie `/hooks`, um neu zu laden. Hooks, die über das Menü `/hooks` hinzugefügt werden, werden sofort wirksam, aber manuelle Datei-Bearbeitungen erfordern ein Neuladen.
- Überprüfen Sie, dass Ihr JSON gültig ist (nachfolgende Kommas und Kommentare sind nicht zulässig)
- Bestätigen Sie, dass die Einstellungsdatei am richtigen Speicherort ist: `.claude/settings.json` für Projekt-Hooks, `~/.claude/settings.json` für globale Hooks

Stop-Hook läuft endlos

Claude arbeitet in einer Endlosschleife weiter, anstatt zu stoppen.

Ihr Stop-Hook-Skript muss überprüfen, ob es bereits eine Fortsetzung ausgelöst hat. Analysieren Sie das Feld `stop_hook_active` aus der JSON-Eingabe und beenden Sie früh, wenn es `true` ist:

```
#!/bin/bash
INPUT=$(cat)
if [ "$(echo "$INPUT" | jq -r '.stop_hook_active')" = "true" ]; then
    exit 0 # Allow Claude to stop
fi
## ... rest of your hook logic
```

JSON-Validierung fehlgeschlagen

Claude Code zeigt einen JSON-Parsing-Fehler an, obwohl Ihr Hook-Skript gültiges JSON ausgibt.

Wenn Claude Code einen Hook ausführt, spawnt es eine Shell, die Ihr Profil sourced (`~/.zshrc` oder `~/.bashrc`). Wenn Ihr Profil bedingungslose `echo`-Anweisungen enthält, wird diese Ausgabe Ihrem Hook-JSON vorangestellt:

```
Shell ready on arm64
{"decision": "block", "reason": "Not allowed"}
```

Claude Code versucht, dies als JSON zu analysieren, und schlägt fehl. Um dies zu beheben, wickeln Sie Echo-Anweisungen in Ihrem Shell-Profil ein, sodass sie nur in interaktiven Shells ausgeführt werden:

```
## In ~/.zshrc or ~/.bashrc
if [[ $- == *i* ]]; then
  echo "Shell ready"
fi
```

Die Variable `$-` enthält Shell-Flags, und `i` bedeutet interaktiv. Hooks werden in nicht-interaktiven Shells ausgeführt, sodass das Echo übersprungen wird.

Debug-Techniken

Schalten Sie den ausführlichen Modus mit `Ctrl+0` um, um Hook-Ausgabe im Transkript zu sehen, oder führen Sie `claude --debug` aus, um vollständige Ausführungsdetails einschließlich der Hooks zu sehen, die abgeglichen wurden, und ihrer Exit-Codes.

Weitere Informationen

- [Hooks-Referenz](#): Vollständige Event-Schemas, JSON-Ausgabeformat, asynchrone Hooks und MCP-Tool-Hooks
- [Sicherheitsüberlegungen](#): Überprüfen Sie vor der Bereitstellung von Hooks in gemeinsamen oder Produktionsumgebungen
- [Bash-Befehlsvalidator-Beispiel](#): Vollständige Referenzimplementierung

Claude Code mit Tools über MCP verbinden

Erfahren Sie, wie Sie Claude Code mit Ihren Tools über das Model Context Protocol verbinden.

Claude Code kann sich über das [Model Context Protocol \(MCP\)](#), einen offenen Standard für KI-Tool-Integrationen, mit Hunderten von externen Tools und Datenquellen verbinden. MCP-Server geben Claude Code Zugriff auf Ihre Tools, Datenbanken und APIs.

Was Sie mit MCP tun können

Mit verbundenen MCP-Servern können Sie Claude Code auffordern:

- **Funktionen aus Issue-Trackern implementieren:** „Füge die in JIRA-Issue ENG-4521 beschriebene Funktion hinzu und erstelle einen PR auf GitHub.“
- **Überwachungsdaten analysieren:** „Überprüfe Sentry und Statsig, um die Nutzung der in ENG-4521 beschriebenen Funktion zu überprüfen.“
- **Datenbanken abfragen:** „Finde E-Mail-Adressen von 10 zufälligen Benutzern, die die Funktion ENG-4521 verwendet haben, basierend auf unserer PostgreSQL-Datenbank.“
- **Designs integrieren:** „Aktualisiere unsere Standard-E-Mail-Vorlage basierend auf den neuen Figma-Designs, die in Slack gepostet wurden“
- **Workflows automatisieren:** „Erstelle Gmail-Entwürfe, die diese 10 Benutzer zu einer Feedback-Sitzung zur neuen Funktion einladen.“

Beliebte MCP-Server

Hier sind einige häufig verwendete MCP-Server, die Sie mit Claude Code verbinden können:

Warning:

Verwenden Sie MCP-Server von Drittanbietern auf eigenes Risiko – Anthropic hat die Korrektheit oder Sicherheit aller dieser Server nicht überprüft. Stellen Sie sicher, dass Sie den MCP-Servern vertrauen, die Sie installieren. Seien Sie besonders vorsichtig bei der Verwendung von MCP-Servern, die nicht vertrauenswürdige Inhalte abrufen könnten, da diese Sie dem Risiko von Prompt-Injection aussetzen können.

Server	Description	Command
Notion	Connect your Notion workspace to search, update, and power workflows across tools	<code>claude mcp add --transport http notion https://mcp.notion.com/mcp</code>
Canva	Search, create, autofill, and export Canva designs	<code>claude mcp add --transport http canva https://mcp.canva.com/mcp</code>
Figma	Generate diagrams and better code from Figma context	<code>claude mcp add --transport http figma-remote-mcp https://mcp.figma.com/mcp</code>
Atlassian	Access Jira & Confluence from Claude	<code>claude mcp add --transport http atlassian https://mcp.atlassian.com/v1/mcp</code>
Linear	Manage issues, projects & team workflows in Linear	<code>claude mcp add --transport http linear https://mcp.linear.app/mcp</code>
monday.com	Manage projects, boards, and workflows in monday.com	<code>claude mcp add --transport http monday https://mcp.monday.com/mcp</code>
Intercom	Access to Intercom data for better customer insights	<code>claude mcp add --transport http intercom https://mcp.intercom.com/mcp</code>
Vercel	Analyze, debug, and manage projects and deployments	<code>claude mcp add --transport http vercel https://mcp.vercel.com</code>

Server	Description	Command
Granola	The AI notepad for meetings	<code>claude mcp add --transport http granola https://mcp.granola.ai/mcp</code>
Asana	Connect to Asana to coordinate tasks, projects, and goals	<code>claude mcp add --transport streamable-http asana https://mcp.asana.com/v2/mcp</code>
Miro	Access and create new content on Miro boards	<code>claude mcp add --transport http miro https://mcp.miro.com/</code>
Sentry	Search, query, and debug errors intelligently	<code>claude mcp add --transport http sentry https://mcp.sentry.dev/mcp</code>
Supabase	Manage databases, authentication, and storage	<code>claude mcp add --transport http supabase https://mcp.supabase.com/mcp</code>
Hugging Face	Access the Hugging Face Hub and thousands of Gradio Apps	<code>claude mcp add --transport http hugging-face https://huggingface.co/mcp</code>
Context7	Up-to-date docs for LLMs and AI code editors	<code>claude mcp add --transport http context7 https://mcp.context7.com/mcp</code>
Stripe	Payment processing and financial infrastructure tools	<code>claude mcp add --transport http stripe https://mcp.stripe.com</code>

Server	Description	Command
Microsoft Learn	Search trusted Microsoft docs to power your development	<code>claude mcp add --transport http microsoft-learn https://learn.microsoft.com/api/mcp</code>
Clay	Find prospects. Research accounts. Personalize outreach	<code>claude mcp add --transport http clay https://api.clay.com/v3/mcp</code>
Webflow	Manage Webflow CMS, pages, assets and sites	<code>claude mcp add --transport http webflow https://mcp.webflow.com/mcp</code>
Cloudflare	Build applications with compute, storage, and AI	<code>claude mcp add --transport http cloudflare https://bindings.mcp.cloudflare.com/mcp</code>
Ramp	Search, access, and analyze your Ramp financial data	<code>claude mcp add --transport http ramp https://ramp-mcp-remote.ramp.com/mcp</code>
ZoomInfo	Enrich contacts & accounts with GTM intelligence	<code>claude mcp add --transport http zoominfo https://mcp.zoominfo.com/mcp</code>
Netlify	Create, deploy, manage, and secure websites on Netlify	<code>claude mcp add --transport http netlify https://netlify-mcp.netlify.app/mcp</code>
Make	Run Make scenarios and manage your Make account	<code>claude mcp add --transport http make https://mcp.make.com</code>

Server	Description	Command
GoDaddy	Search domains and check availability	<pre>claude mcp add -- transport http godaddy https:// api.godaddy.com/v1/ domains/mcp</pre>
Google Cloud BigQuery	BigQuery: Advanced analytical insights for agents	<pre>claude mcp add -- transport http bigquery https:// bigquery.googleapis.c om/mcp</pre>
PayPal	Access PayPal payments platform	<pre>claude mcp add -- transport http paypal https:// mcp.paypal.com/mcp</pre>
PostHog	Query, analyze, and manage your PostHog insights	<pre>claude mcp add -- transport http posthog https:// mcp.posthog.com/mcp</pre>
Similarweb	Real time web, mobile app, and market data	<pre>claude mcp add -- transport http similarweb https:// mcp.similarweb.com</pre>
Crypto.com	Real time prices, orders, charts, and more for crypto	<pre>claude mcp add -- transport http crypto.com https:// mcp.crypto.com/ market-data/mcp</pre>
Attio	Search, manage, and update your Attio CRM from Claude	<pre>claude mcp add -- transport http attio https:// mcp.attio.com/mcp</pre>
Trivago	Find your ideal hotel at the best price	<pre>claude mcp add -- transport http trivago https:// mcp.trivago.com/mcp</pre>

Server	Description	Command
Jam	Record screen and collect automatic context for issues	<code>claude mcp add --transport http jam https://mcp.jam.dev/mcp</code>
Consensus	Explore scientific research	<code>claude mcp add --transport http consensus https://mcp.consensus.app/mcp</code>
Clockwise	Advanced scheduling and time management for work	<code>claude mcp add --transport http clockwise https://mcp.getclockwise.com/mcp</code>
Square	Search and manage transaction, merchant, and payment data	<code>claude mcp add --transport sse square https://mcp.squareup.com/sse</code>
Egnyte	Securely access and analyze Egnyte content	<code>claude mcp add --transport http egnyte https://mcp-server.egnyte.com/mcp</code>
Pylon	Search and manage Pylon support issues	<code>claude mcp add --transport http pylon https://mcp.usepylon.com/</code>
Honeycomb	Query and explore observability data and SLOs	<code>claude mcp add --transport http honeycomb https://mcp.honeycomb.io/mcp</code>

Note:

Benötigen Sie eine spezifische Integration? [Finden Sie Hunderte weitere MCP-Server auf GitHub](#), oder erstellen Sie Ihren eigenen mit dem [MCP SDK](#).

MCP-Server installieren

MCP-Server können je nach Ihren Anforderungen auf drei verschiedene Arten konfiguriert werden:

Option 1: Einen Remote-HTTP-Server hinzufügen

HTTP-Server sind die empfohlene Option für die Verbindung mit Remote-MCP-Servern. Dies ist das am weitesten unterstützte Transportprotokoll für Cloud-basierte Dienste.

```
## Grundlegende Syntax
claude mcp add --transport http <name> <url>

## Echtes Beispiel: Mit Notion verbinden
claude mcp add --transport http notion https://mcp.notion.com/mcp

## Beispiel mit Bearer-Token
claude mcp add --transport http secure-api https://api.example.com/mcp \
  --header "Authorization: Bearer your-token"
```

Option 2: Einen Remote-SSE-Server hinzufügen

Warning:

Das SSE-Transportprotokoll (Server-Sent Events) ist veraltet. Verwenden Sie stattdessen HTTP-Server, wo verfügbar.

```
## Grundlegende Syntax
claude mcp add --transport sse <name> <url>

## Echtes Beispiel: Mit Asana verbinden
claude mcp add --transport sse asana https://mcp.asana.com/sse

## Beispiel mit Authentifizierungs-Header
claude mcp add --transport sse private-api https://api.company.com/sse \
  --header "X-API-Key: your-key-here"
```

Option 3: Einen lokalen Stdio-Server hinzufügen

Stdio-Server werden als lokale Prozesse auf Ihrem Computer ausgeführt. Sie sind ideal für Tools, die direkten Systemzugriff oder benutzerdefinierte Skripte benötigen.

```
## Grundlegende Syntax
claude mcp add [options] <name> -- <command> [args...]

## Echtes Beispiel: Airtable-Server hinzufügen
claude mcp add --transport stdio --env AIRTABLE_API_KEY=YOUR_KEY airtable \
  -- npx -y airtable-mcp-server
```

Note:

Wichtig: Reihenfolge der Optionen

Alle Optionen (`--transport` , `--env` , `--scope` , `--header`) müssen **vor** dem Servernamen kommen. Der `--` (Doppelstrich) trennt dann den Servernamen von dem Befehl und den Argumenten, die an den MCP-Server übergeben werden.

Zum Beispiel:

- `claude mcp add --transport stdio myserver -- npx server` → führt `npx server` aus
- `claude mcp add --transport stdio --env KEY=value myserver -- python server.py --port 8080` → führt `python server.py --port 8080` mit `KEY=value` in der Umgebung aus

Dies verhindert Konflikte zwischen Claudes Flags und den Flags des Servers.

Verwalten Ihrer Server

Nach der Konfiguration können Sie Ihre MCP-Server mit diesen Befehlen verwalten:

```
## Alle konfigurierten Server auflisten
claude mcp list

## Details für einen bestimmten Server abrufen
claude mcp get github

## Einen Server entfernen
claude mcp remove github

## (innerhalb von Claude Code) Serverstatus überprüfen
/mcp
```

Dynamische Tool-Updates

Claude Code unterstützt MCP- `list_changed` -Benachrichtigungen, die es MCP-Servern ermöglichen, ihre verfügbaren Tools, Prompts und Ressourcen dynamisch zu aktualisieren, ohne dass Sie die Verbindung trennen und erneut verbinden müssen. Wenn ein MCP-Server eine `list_changed` -Benachrichtigung sendet, aktualisiert Claude Code automatisch die verfügbaren Funktionen von diesem Server.

Tip:

Tipps:

- Verwenden Sie das Flag `--scope` , um anzugeben, wo die Konfiguration gespeichert wird:
- `local` (Standard): Nur für Sie im aktuellen Projekt verfügbar (hieß in älteren Versionen `project`)
- `project` : Geteilt mit allen im Projekt über die Datei `.mcp.json`
- `user` : Für Sie über alle Projekte hinweg verfügbar (hieß in älteren Versionen `global`)
- Legen Sie Umgebungsvariablen mit `--env` -Flags fest (zum Beispiel `--env KEY=value`)
- Konfigurieren Sie das Startup-Timeout des MCP-Servers mit der Umgebungsvariablen `MCP_TIMEOUT` (zum Beispiel `MCP_TIMEOUT=10000` `claude` setzt ein 10-Sekunden-Timeout)
- Claude Code zeigt eine Warnung an, wenn die MCP-Tool-Ausgabe 10.000 Token überschreitet. Um dieses Limit zu erhöhen, setzen Sie die Umgebungsvariable `MAX_MCP_OUTPUT_TOKENS` (zum Beispiel `MAX_MCP_OUTPUT_TOKENS=50000`)
- Verwenden Sie `/mcp` , um sich bei Remote-Servern zu authentifizieren, die OAuth 2.0-Authentifizierung erfordern

Warning:

Windows-Benutzer: Auf nativem Windows (nicht WSL) erfordern lokale MCP-Server, die `npm` verwenden, den `cmd /c`-Wrapper, um eine ordnungsgemäße Ausführung zu gewährleisten.

```
## Dies erstellt command="cmd", das Windows ausführen kann
claude mcp add --transport stdio my-server -- cmd /c npm -y @some/package
```

Ohne den `cmd /c`-Wrapper treten „Connection closed“-Fehler auf, da Windows `npm` nicht direkt ausführen kann. (Siehe die obige Notiz für eine Erklärung des `--` -Parameters.)

Von Plugins bereitgestellte MCP-Server

[Plugins](#) können MCP-Server bündeln und automatisch Tools und Integrationen bereitstellen, wenn das Plugin aktiviert ist. Plugin-MCP-Server funktionieren identisch mit benutzerkonfigurierten Servern.

Wie Plugin-MCP-Server funktionieren:

- Plugins definieren MCP-Server in `.mcp.json` im Plugin-Root oder inline in `plugin.json`
- Wenn ein Plugin aktiviert ist, starten seine MCP-Server automatisch
- Plugin-MCP-Tools erscheinen neben manuell konfigurierten MCP-Tools
- Plugin-Server werden durch die Plugin-Installation verwaltet (nicht durch `/mcp`-Befehle)

Beispiel-Plugin-MCP-Konfiguration:

In `.mcp.json` im Plugin-Root:

```
{
  "database-tools": {
    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
    "env": {
      "DB_URL": "${DB_URL}"
    }
  }
}
```

Oder inline in `plugin.json` :

```
{
  "name": "my-plugin",
  "mcpServers": {
    "plugin-api": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/api-server",
      "args": ["--port", "8080"]
    }
  }
}
```

Plugin-MCP-Funktionen:

- **Automatischer Lebenszyklus:** Server starten, wenn das Plugin aktiviert wird, aber Sie müssen Claude Code neu starten, um MCP-Server-Änderungen anzuwenden (Aktivierung oder Deaktivierung)
- **Umgebungsvariablen:** Verwenden Sie `${CLAUDE_PLUGIN_ROOT}` für Plugin-relative Pfade
- **Zugriff auf Benutzerumgebung:** Zugriff auf die gleichen Umgebungsvariablen wie manuell konfigurierte Server
- **Mehrere Transporttypen:** Unterstützung für Stdio-, SSE- und HTTP-Transporte (die Transportunterstützung kann je nach Server variieren)

Anzeigen von Plugin-MCP-Servern:

```
## Innerhalb von Claude Code alle MCP-Server einschließlich Plugin-Server anzeigen
/mcp
```

Plugin-Server erscheinen in der Liste mit Indikatoren, die zeigen, dass sie von Plugins stammen.

Vorteile von Plugin-MCP-Servern:

- **Gebündelte Verteilung:** Tools und Server zusammen verpackt
- **Automatische Einrichtung:** Keine manuelle MCP-Konfiguration erforderlich
- **Team-Konsistenz:** Alle erhalten die gleichen Tools, wenn das Plugin installiert ist

Siehe die [Plugin-Komponenten-Referenz](#) für Details zum Bündeln von MCP-Servern mit Plugins.

MCP-Installationsbereiche

MCP-Server können auf drei verschiedenen Bereichsebenen konfiguriert werden, die jeweils unterschiedliche Zwecke für die Verwaltung der Serverzugänglichkeit und des Austauschs erfüllen. Das Verständnis dieser Bereiche hilft Ihnen, die beste Methode zur Konfiguration von Servern für Ihre spezifischen Anforderungen zu bestimmen.

Lokaler Bereich

Lokal begrenzte Server stellen die Standard-Konfigurationsebene dar und werden in `~/.claude.json` unter dem Pfad Ihres Projekts gespeichert. Diese Server bleiben privat für Sie und sind nur zugänglich, wenn Sie im aktuellen Projektverzeichnis arbeiten. Dieser Bereich ist ideal für persönliche Entwicklungsserver, experimentelle Konfigurationen oder Server, die vertrauliche Anmeldedaten enthalten, die nicht geteilt werden sollten.

Note:

Der Begriff „lokaler Bereich“ für MCP-Server unterscheidet sich von allgemeinen lokalen Einstellungen. Lokal begrenzte MCP-Server werden in `~/.claude.json` (Ihr Home-Verzeichnis) gespeichert, während allgemeine lokale Einstellungen `.claude/settings.local.json` (im Projektverzeichnis) verwenden. Siehe [Einstellungen](#) für Details zu Einstellungsdatei-Speicherorten.

```
## Einen lokal begrenzten Server hinzufügen (Standard)
claude mcp add --transport http stripe https://mcp.stripe.com

## Lokalen Bereich explizit angeben
claude mcp add --transport http stripe --scope local https://mcp.stripe.com
```

Projektbereich

Projektbegrenzte Server ermöglichen Teamzusammenarbeit durch das Speichern von Konfigurationen in einer `.mcp.json`-Datei im Root-Verzeichnis Ihres Projekts. Diese Datei ist dazu bestimmt, in die Versionskontrolle eingecheckt zu werden, um sicherzustellen, dass alle Teammitglieder Zugriff auf die gleichen MCP-Tools und -Dienste haben. Wenn Sie einen projektbegrenzten Server hinzufügen, erstellt oder aktualisiert Claude Code automatisch diese Datei mit der entsprechenden Konfigurationsstruktur.

```
## Einen projektbegrenzten Server hinzufügen
claude mcp add --transport http paypal --scope project https://mcp.paypal.com/mcp
```

Die resultierende `.mcp.json`-Datei folgt einem standardisierten Format:

```
{
  "mcpServers": {
    "shared-server": {
      "command": "/path/to/server",
      "args": [],
      "env": {}
    }
  }
}
```

Aus Sicherheitsgründen fordert Claude Code eine Genehmigung an, bevor projektbegrenzte Server aus `.mcp.json`-Dateien verwendet werden. Wenn Sie diese Genehmigungswahlmöglichkeiten zurücksetzen müssen, verwenden Sie den Befehl `claude mcp reset-project-choices`.

Benutzerbereich

Benutzerbegrenzte Server werden in `~/.claude.json` gespeichert und bieten projektübergreifende Zugänglichkeit, wodurch sie über alle Projekte auf Ihrem Computer verfügbar sind und gleichzeitig privat für Ihr Benutzerkonto bleiben. Dieser Bereich funktioniert gut für persönliche Utility-Server, Entwicklungstools oder Dienste, die Sie häufig über verschiedene Projekte hinweg verwenden.

```
## Einen Benutzer-Server hinzufügen
claude mcp add --transport http hubspot --scope user https://mcp.hubspot.com/anthropic
```

Den richtigen Bereich wählen

Wählen Sie Ihren Bereich basierend auf:

- **Lokaler Bereich:** Persönliche Server, experimentelle Konfigurationen oder vertrauliche Anmeldedaten, die spezifisch für ein Projekt sind

- **Projektbereich:** Von Teams gemeinsam genutzte Server, projektspezifische Tools oder Dienste, die für die Zusammenarbeit erforderlich sind
- **Benutzerbereich:** Persönliche Utilities, die über mehrere Projekte hinweg benötigt werden, Entwicklungstools oder häufig verwendete Dienste

Note:

Wo werden MCP-Server gespeichert?

- **Benutzer- und lokaler Bereich:** `~/.claude.json` (im Feld `mcpServers` oder unter Projektpfaden)
- **Projektbereich:** `.mcp.json` im Projekt-Root (eingchecked in die Versionskontrolle)
- **Verwaltet:** `managed-mcp.json` in Systemverzeichnissen (siehe [Verwaltete MCP-Konfiguration](#))

Bereichshierarchie und Vorrang

MCP-Server-Konfigurationen folgen einer klaren Vorranghierarchie. Wenn Server mit dem gleichen Namen auf mehreren Bereichen vorhanden sind, löst das System Konflikte durch Priorisierung lokal begrenzter Server zuerst, gefolgt von projektbegrenzten Servern und schließlich benutzerbegrenzten Servern. Dieses Design stellt sicher, dass persönliche Konfigurationen gemeinsame Konfigurationen bei Bedarf überschreiben können.

Umgebungsvariablen-Erweiterung in `.mcp.json`

Claude Code unterstützt die Umgebungsvariablen-Erweiterung in `.mcp.json`-Dateien, die es Teams ermöglicht, Konfigurationen zu teilen und gleichzeitig Flexibilität für maschinenspezifische Pfade und vertrauliche Werte wie API-Schlüssel zu bewahren.

Unterstützte Syntax:

- `${VAR}` - Erweitert sich zum Wert der Umgebungsvariablen `VAR`
- `${VAR:-default}` - Erweitert sich zu `VAR`, wenn gesetzt, andernfalls wird `default` verwendet

Erweiterungsorte: Umgebungsvariablen können erweitert werden in:

- `command` - Der Server-Ausführungspfad
- `args` - Befehlszeilenargumente
- `env` - Umgebungsvariablen, die an den Server übergeben werden
- `url` - Für HTTP-Server-Typen
- `headers` - Für HTTP-Server-Authentifizierung

Beispiel mit Variablenerweiterung:

```
{
  "mcpServers": {
    "api-server": {
      "type": "http",
      "url": "${API_BASE_URL:-https://api.example.com}/mcp",
      "headers": {
        "Authorization": "Bearer ${API_KEY}"
      }
    }
  }
}
```

Wenn eine erforderliche Umgebungsvariable nicht gesetzt ist und keinen Standardwert hat, kann Claude Code die Konfiguration nicht analysieren.

Praktische Beispiele

{/* ### Beispiel: Browser-Tests mit Playwright automatisieren

```
claude mcp add --transport stdio playwright -- npx -y @playwright/mcp@latest
```

Dann schreiben und führen Sie Browser-Tests aus:

```
Test if the login flow works with test@example.com
```

```
Take a screenshot of the checkout page on mobile
```

```
Verify that the search feature returns results
``` */}

Beispiel: Fehler mit Sentry überwachen

```bash theme={null}
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Authentifizieren Sie sich mit Ihrem Sentry-Konto:

```
/mcp
```

Debuggen Sie dann Produktionsprobleme:

```
What are the most common errors in the last 24 hours?
```

```
Show me the stack trace for error ID abc123
```

```
Which deployment introduced these new errors?
```

Beispiel: Mit GitHub für Code-Reviews verbinden

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

Authentifizieren Sie sich bei Bedarf, indem Sie „Authenticate“ für GitHub auswählen:

```
/mcp
```

Arbeiten Sie dann mit GitHub:

```
Review PR #456 and suggest improvements
```

```
Create a new issue for the bug we just found
```

```
Show me all open PRs assigned to me
```

Beispiel: Ihre PostgreSQL-Datenbank abfragen

```
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \
--dsn "postgresql://readonly:pass@prod.db.com:5432/analytics"
```

Fragen Sie dann Ihre Datenbank natürlich ab:

```
What's our total revenue this month?
```

```
Show me the schema for the orders table
```

```
Find customers who haven't made a purchase in 90 days
```

Mit Remote-MCP-Servern authentifizieren

Viele Cloud-basierte MCP-Server erfordern Authentifizierung. Claude Code unterstützt OAuth 2.0 für sichere Verbindungen.

Step 1: Fügen Sie den Server hinzu, der Authentifizierung erfordert

Zum Beispiel:

```
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Step 2: Verwenden Sie den /mcp-Befehl innerhalb von Claude Code

In Claude Code verwenden Sie den Befehl:

```
/mcp
```

Folgen Sie dann den Schritten in Ihrem Browser, um sich anzumelden.

Tip:

Tipps:

- Authentifizierungstoken werden sicher gespeichert und automatisch aktualisiert
- Verwenden Sie „Clear authentication“ im `/mcp`-Menü, um den Zugriff zu widerrufen
- Wenn Ihr Browser nicht automatisch geöffnet wird, kopieren Sie die bereitgestellte URL und öffnen Sie sie manuell
- Wenn die Browser-Umleitung nach der Authentifizierung mit einem Verbindungsfehler fehlschlägt, fügen Sie die vollständige Callback-URL aus der Adressleiste Ihres Browsers in die URL-Eingabeaufforderung ein, die in Claude Code angezeigt wird
- OAuth-Authentifizierung funktioniert mit HTTP-Servern

Verwenden Sie einen festen OAuth-Callback-Port

Einige MCP-Server erfordern einen spezifischen Redirect-URI, der im Voraus registriert ist. Standardmäßig wählt Claude Code einen zufällig verfügbaren Port für den OAuth-Callback. Verwenden Sie `--callback-port`, um den Port zu fixieren, damit er einem vorregistrierten Redirect-URI der Form `http://localhost:PORT/callback` entspricht.

Sie können `--callback-port` allein (mit dynamischer Client-Registrierung) oder zusammen mit `--client-id` (mit vorkonfigurierten Anmeldedaten) verwenden.

```
## Fester Callback-Port mit dynamischer Client-Registrierung
claude mcp add --transport http \
  --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

Verwenden Sie vorkonfigurierte OAuth-Anmeldedaten

Einige MCP-Server unterstützen keine automatische OAuth-Einrichtung. Wenn Sie einen Fehler wie „Incompatible auth server: does not support dynamic client registration“ sehen, erfordert der Server vorkonfigurierte Anmeldedaten. Registrieren Sie zunächst eine OAuth-App über das Entwicklerportal des Servers und geben Sie dann die Anmeldedaten beim Hinzufügen des Servers an.

Step 1: Registrieren Sie eine OAuth-App beim Server

Erstellen Sie eine App über das Entwicklerportal des Servers und notieren Sie sich Ihre Client-ID und Ihren Client-Secret.

Viele Server erfordern auch einen Redirect-URI. Wenn ja, wählen Sie einen Port und registrieren Sie einen Redirect-URI im Format `http://localhost:PORT/callback`. Verwenden Sie denselben Port mit `--callback-port` im nächsten Schritt.

Step 2: Fügen Sie den Server mit Ihren Anmeldedaten hinzu

Wählen Sie eine der folgenden Methoden. Der für `--callback-port` verwendete Port kann ein beliebiger verfügbarer Port sein. Er muss nur dem Redirect-URI entsprechen, den Sie im vorherigen Schritt registriert haben.

claude mcp add

Verwenden Sie `--client-id`, um die Client-ID Ihrer App zu übergeben. Das Flag `--client-secret` fordert das Secret mit maskierter Eingabe an:

```
claude mcp add --transport http \
  --client-id your-client-id --client-secret --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

claude mcp add-json

Fügen Sie das Objekt `oauth` in die JSON-Konfiguration ein und übergeben Sie `--client-secret` als separates Flag:

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"clientId":"your-client-id","callbackPort":8080}}' \
  --client-secret
```

claude mcp add-json (nur Callback-Port)

Verwenden Sie `--callback-port` ohne Client-ID, um den Port zu fixieren und gleichzeitig die dynamische Client-Registrierung zu verwenden:

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"callbackPort":8080}}'
```

CI / Umgebungsvariable

Legen Sie das Secret über eine Umgebungsvariable fest, um die interaktive Eingabeaufforderung zu überspringen:

```
MCP_CLIENT_SECRET=your-secret claude mcp add --transport http \  
  --client-id your-client-id --client-secret --callback-port 8080 \  
  my-server https://mcp.example.com/mcp
```

Step 3: Authentifizieren Sie sich in Claude Code

Führen Sie `/mcp` in Claude Code aus und folgen Sie dem Browser-Login-Ablauf.

Tip:

Tipps:

- Das Client-Secret wird sicher in Ihrem System-Keychain (macOS) oder einer Anmeldedatei gespeichert, nicht in Ihrer Konfiguration
- Wenn der Server einen öffentlichen OAuth-Client ohne Secret verwendet, verwenden Sie nur `--client-id` ohne `--client-secret`
- `--callback-port` kann mit oder ohne `--client-id` verwendet werden
- Diese Flags gelten nur für HTTP- und SSE-Transporte. Sie haben keine Auswirkung auf Stdio-Server
- Verwenden Sie `claude mcp get <name>`, um zu überprüfen, dass OAuth-Anmeldedaten für einen Server konfiguriert sind

Überschreiben Sie die OAuth-Metadaten-Erkennung

Wenn Ihr MCP-Server Fehler auf dem Standard-OAuth-Metadaten-Endpunkt (`/.well-known/oauth-authorization-server`) zurückgibt, aber einen funktionierenden OIDC-Endpunkt verfügbar macht, können Sie Claude Code anweisen, OAuth-Metadaten direkt von einer URL zu abrufen, die Sie angeben, und die Standard-Erkennungskette zu umgehen.

Legen Sie `authServerMetadataUrl` im Objekt `oauth` der Konfiguration Ihres Servers in `.mcp.json` fest:

```
{
  "mcpServers": {
    "my-server": {
      "type": "http",
      "url": "https://mcp.example.com/mcp",
      "oauth": {
        "authServerMetadataUrl": "https://auth.example.com/.well-known/openid-configuration"
      }
    }
  }
}
```

Die URL muss `https://` verwenden. Diese Option erfordert Claude Code v2.1.64 oder später.

MCP-Server aus JSON-Konfiguration hinzufügen

Wenn Sie eine JSON-Konfiguration für einen MCP-Server haben, können Sie sie direkt hinzufügen:

Step 1: Fügen Sie einen MCP-Server aus JSON hinzu


```
## Grundlegende Syntax
claude mcp add-json <name> '<json>'

## Beispiel: Hinzufügen eines HTTP-Servers mit JSON-Konfiguration
claude mcp add-json weather-api '{"type":"http","url":"https://api.weather.com/mcp","headers":{"Authorization":"Bearer token"}}'

## Beispiel: Hinzufügen eines Stdio-Servers mit JSON-Konfiguration
claude mcp add-json local-weather '{"type":"stdio","command":"/path/to/weather-cli","args":["--api-key","abc123"],"env":{"CACHE_DIR":"/tmp"}}'

## Beispiel: Hinzufügen eines HTTP-Servers mit vorkonfigurierten OAuth-Anmeldedaten
claude mcp add-json my-server '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"clientId":"your-client-id","callbackPort":8080}}' --client-secret
```

Step 2: Überprüfen Sie, dass der Server hinzugefügt wurde

```
claude mcp get weather-api
```

Tip:

Tipps:

- Stellen Sie sicher, dass das JSON in Ihrer Shell ordnungsgemäß escaped ist
- Das JSON muss dem MCP-Server-Konfigurationsschema entsprechen
- Sie können `--scope user` verwenden, um den Server zu Ihrer Benutzerkonfiguration statt zur projektspezifischen hinzuzufügen

MCP-Server aus Claude Desktop importieren

Wenn Sie bereits MCP-Server in Claude Desktop konfiguriert haben, können Sie diese importieren:

Step 1: Importieren Sie Server aus Claude Desktop

```
## Grundlegende Syntax
claude mcp add-from-claude-desktop
```

Step 2: Wählen Sie aus, welche Server importiert werden sollen

Nach dem Ausführen des Befehls wird ein interaktives Dialogfeld angezeigt, in dem Sie auswählen können, welche Server Sie importieren möchten.

Step 3: Überprüfen Sie, dass die Server importiert wurden

```
claude mcp list
```

Tip:

Tipps:

- Diese Funktion funktioniert nur auf macOS und Windows Subsystem for Linux (WSL)
- Sie liest die Claude Desktop-Konfigurationsdatei von ihrem Standort auf diesen Plattformen
- Verwenden Sie das Flag `--scope user`, um Server zu Ihrer Benutzerkonfiguration hinzuzufügen
- Importierte Server haben die gleichen Namen wie in Claude Desktop
- Wenn Server mit den gleichen Namen bereits vorhanden sind, erhalten sie ein numerisches Suffix (zum Beispiel `server_1`)

MCP-Server von Claude.ai verwenden

Wenn Sie sich in Claude Code mit einem [Claude.ai](#)-Konto angemeldet haben, sind MCP-Server, die Sie in Claude.ai hinzugefügt haben, automatisch in Claude Code verfügbar:

Step 1: Konfigurieren Sie MCP-Server in Claude.ai

Fügen Sie Server unter [claude.ai/settings/connectors](#) hinzu. Bei Team- und Enterprise-Plänen können nur Administratoren Server hinzufügen.

Step 2: Authentifizieren Sie den MCP-Server

Führen Sie alle erforderlichen Authentifizierungsschritte in Claude.ai durch.

Step 3: Zeigen Sie Server in Claude Code an und verwalten Sie sie

In Claude Code verwenden Sie den Befehl:

```
/mcp
```

Claude.ai-Server erscheinen in der Liste mit Indikatoren, die zeigen, dass sie von Claude.ai stammen.

Um Claude.ai-MCP-Server in Claude Code zu deaktivieren, setzen Sie die Umgebungsvariable `ENABLE_CLAUDEAI_MCP_SERVERS` auf `false` :

```
ENABLE_CLAUDEAI_MCP_SERVERS=false claude
```

Claude Code als MCP-Server verwenden

Sie können Claude Code selbst als MCP-Server verwenden, mit dem sich andere Anwendungen verbinden können:

```
## Starten Sie Claude als Stdio-MCP-Server
claude mcp serve
```

Sie können dies in Claude Desktop verwenden, indem Sie diese Konfiguration zu `claude_desktop_config.json` hinzufügen:

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Warning:

Konfigurieren Sie den Pfad der ausführbaren Datei: Das Feld `command` muss auf die Claude Code-Ausführungsdatei verweisen. Wenn der Befehl `claude` nicht in Ihrem System-PATH vorhanden ist, müssen Sie den vollständigen Pfad zur ausführbaren Datei angeben.

Um den vollständigen Pfad zu finden:

```
which claude
```

Verwenden Sie dann den vollständigen Pfad in Ihrer Konfiguration:

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "/full/path/to/claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Ohne den korrekten Pfad der ausführbaren Datei treten Fehler wie `spawn claude ENOENT` auf.

Tip:

Tipps:

- Der Server bietet Zugriff auf Claudes Tools wie View, Edit, LS usw.
- In Claude Desktop versuchen Sie, Claude aufzufordern, Dateien in einem Verzeichnis zu lesen, Änderungen vorzunehmen und mehr.
- Beachten Sie, dass dieser MCP-Server nur Claudes Tools für Ihren MCP-Client verfügbar macht, daher ist Ihr eigener Client dafür verantwortlich, Benutzerbestätigung für einzelne Tool-Aufrufe zu implementieren.

MCP-Ausgabelimits und Warnungen

Wenn MCP-Tools große Ausgaben erzeugen, hilft Claude Code bei der Verwaltung der Token-Nutzung, um zu verhindern, dass Ihr Gesprächskontext überwältigt wird:

- **Ausgabe-Warnungsschwelle:** Claude Code zeigt eine Warnung an, wenn eine MCP-Tool-Ausgabe 10.000 Token überschreitet
- **Konfigurierbares Limit:** Sie können die maximale zulässige MCP-Ausgabe-Token mit der Umgebungsvariablen `MAX_MCP_OUTPUT_TOKENS` anpassen
- **Standardlimit:** Das Standardmaximum beträgt 25.000 Token

Um das Limit für Tools zu erhöhen, die große Ausgaben erzeugen:

```
## Legen Sie ein höheres Limit für MCP-Tool-Ausgaben fest
export MAX_MCP_OUTPUT_TOKENS=50000
claude
```

Dies ist besonders nützlich bei der Arbeit mit MCP-Servern, die:

- Große Datensätze oder Datenbanken abfragen
- Detaillierte Berichte oder Dokumentation generieren
- Umfangreiche Protokolldateien oder Debugging-Informationen verarbeiten

Warning:

Wenn Sie häufig Ausgabewarnungen bei bestimmten MCP-Servern erhalten, erwägen Sie, das Limit zu erhöhen oder den Server so zu konfigurieren, dass er seine Antworten paginiert oder filtert.

MCP-Ressourcen verwenden

MCP-Server können Ressourcen verfügbar machen, auf die Sie mit @-Erwähnungen verweisen können, ähnlich wie Sie auf Dateien verweisen.

Referenzieren Sie MCP-Ressourcen

Step 1: Verfügbare Ressourcen auflisten

Geben Sie @ in Ihre Eingabeaufforderung ein, um verfügbare Ressourcen von allen verbundenen MCP-Servern anzuzeigen. Ressourcen erscheinen neben Dateien im Autocomplete-Menü.

Step 2: Referenzieren Sie eine bestimmte Ressource

Verwenden Sie das Format `@server:protocol://resource/path`, um auf eine Ressource zu verweisen:

```
Can you analyze @github:issue://123 and suggest a fix?
```

```
Please review the API documentation at @docs:file://api/authentication
```

Step 3: Mehrere Ressourcenreferenzen

Sie können mehrere Ressourcen in einer einzelnen Eingabeaufforderung referenzieren:

```
Compare @postgres:schema://users with @docs:file://database/user-model
```

Tip:

Tipps:

- Ressourcen werden automatisch abgerufen und als Anhänge eingefügt, wenn sie referenziert werden
- Ressourcenpfade sind fuzzy-durchsuchbar im @-Erwähnungs-Autocomplete
- Claude Code stellt automatisch Tools zur Verfügung, um MCP-Ressourcen aufzulisten und zu lesen, wenn Server diese unterstützen
- Ressourcen können jeden Inhaltstyp enthalten, den der MCP-Server bereitstellt (Text, JSON, strukturierte Daten usw.)

Mit MCP-Tool-Suche skalieren

Wenn Sie viele MCP-Server konfiguriert haben, können Tool-Definitionen einen erheblichen Teil Ihres Kontextfensters verbrauchen. MCP-Tool-Suche löst dies, indem Tools bei Bedarf dynamisch geladen werden, anstatt sie alle vorab zu laden.

Wie es funktioniert

Claude Code aktiviert die Tool-Suche automatisch, wenn Ihre MCP-Tool-Beschreibungen mehr als 10 % des Kontextfensters verbrauchen würden. Sie können [diese Schwelle anpassen](#) oder die Tool-Suche vollständig deaktivieren. Wenn ausgelöst:

1. MCP-Tools werden aufgeschoben, anstatt sie vorab in den Kontext zu laden
2. Claude verwendet ein Such-Tool, um relevante MCP-Tools bei Bedarf zu entdecken
3. Nur die Tools, die Claude tatsächlich benötigt, werden in den Kontext geladen
4. MCP-Tools funktionieren aus Ihrer Perspektive genau wie zuvor

Für MCP-Server-Autoren

Wenn Sie einen MCP-Server erstellen, wird das Feld für Server-Anweisungen mit aktivierter Tool-Suche nützlicher. Server-Anweisungen helfen Claude zu verstehen, wann nach Ihren Tools gesucht werden soll, ähnlich wie [skills](#) funktionieren.

Fügen Sie klare, aussagekräftige Server-Anweisungen hinzu, die erklären:

- Welche Kategorie von Aufgaben Ihre Tools verarbeiten
- Wann Claude nach Ihren Tools suchen sollte
- Wichtige Funktionen, die Ihr Server bietet

Konfigurieren Sie die Tool-Suche

Die Tool-Suche wird standardmäßig im Auto-Modus ausgeführt, was bedeutet, dass sie nur aktiviert wird, wenn Ihre MCP-Tool-Definitionen die Kontextschwelle überschreiten. Wenn Sie nur wenige Tools haben, werden sie normal ohne Tool-Suche geladen. Diese Funktion erfordert Modelle, die `tool_reference`-Blöcke unterstützen: Sonnet 4 und später oder Opus 4 und später. Haiku-Modelle unterstützen die Tool-Suche nicht.

Steuern Sie das Verhalten der Tool-Suche mit der Umgebungsvariablen

`ENABLE_TOOL_SEARCH` :

Wert	Verhalten
<code>auto</code>	Aktiviert, wenn MCP-Tools 10 % des Kontexts überschreiten (Standard)
<code>auto:<N></code>	Aktiviert bei benutzerdefinierter Schwelle, wobei <code><N></code> ein Prozentsatz ist (z. B. <code>auto:5</code> für 5 %)
<code>true</code>	Immer aktiviert
<code>false</code>	Deaktiviert, alle MCP-Tools werden vorab geladen

```
## Verwenden Sie eine benutzerdefinierte 5%-Schwelle
ENABLE_TOOL_SEARCH=auto:5 claude
```

```
## Deaktivieren Sie die Tool-Suche vollständig
ENABLE_TOOL_SEARCH=false claude
```

Oder legen Sie den Wert im Feld `env` Ihrer [settings.json](#) fest.

Sie können das MCPSearch-Tool auch spezifisch mit der Einstellung `disallowedTools` deaktivieren:

```
{
  "permissions": {
    "deny": ["MCPSearch"]
  }
}
```

MCP-Prompts als Befehle verwenden

MCP-Server können Prompts verfügbar machen, die in Claude Code als Befehle verfügbar werden.

Führen Sie MCP-Prompts aus

Step 1: Entdecken Sie verfügbare Prompts

Geben Sie `/` ein, um alle verfügbaren Befehle anzuzeigen, einschließlich derer von MCP-Servern. MCP-Prompts erscheinen mit dem Format `/mcp__servername__promptname`.

Step 2: Führen Sie einen Prompt ohne Argumente aus

```
/mcp__github__list_prs
```

Step 3: Führen Sie einen Prompt mit Argumenten aus

Viele Prompts akzeptieren Argumente. Übergeben Sie sie durch Leerzeichen getrennt nach dem Befehl:

```
/mcp__github__pr_review 456
```

```
/mcp__jira__create_issue "Bug in login flow" high
```

Tip:

Tipps:

- MCP-Prompts werden dynamisch von verbundenen Servern entdeckt
- Argumente werden basierend auf den definierten Parametern des Prompts analysiert
- Prompt-Ergebnisse werden direkt in das Gespräch eingefügt
- Server- und Prompt-Namen werden normalisiert (Leerzeichen werden zu Unterstrichen)

Verwaltete MCP-Konfiguration

Für Organisationen, die eine zentralisierte Kontrolle über MCP-Server benötigen, unterstützt Claude Code zwei Konfigurationsoptionen:

1. **Exklusive Kontrolle mit `managed-mcp.json`**: Stellen Sie einen festen Satz von MCP-Servern bereit, die Benutzer nicht ändern oder erweitern können
2. **Richtlinienbasierte Kontrolle mit Allowlists/Denylists**: Ermöglichen Sie Benutzern, ihre eigenen Server hinzuzufügen, aber beschränken Sie, welche zulässig sind

Diese Optionen ermöglichen IT-Administratoren:

- **Kontrollieren Sie, auf welche MCP-Server Mitarbeiter zugreifen können**: Stellen Sie einen standardisierten Satz genehmigter MCP-Server in der gesamten Organisation bereit
- **Verhindern Sie nicht autorisierte MCP-Server**: Beschränken Sie Benutzer daran, nicht genehmigte MCP-Server hinzuzufügen
- **Deaktivieren Sie MCP vollständig**: Entfernen Sie die MCP-Funktionalität vollständig, falls erforderlich

Option 1: Exklusive Kontrolle mit `managed-mcp.json`

Wenn Sie eine `managed-mcp.json`-Datei bereitstellen, übernimmt sie die **exklusive Kontrolle** über alle MCP-Server. Benutzer können keine MCP-Server außer denen, die in dieser Datei definiert sind, hinzufügen, ändern oder verwenden. Dies ist der einfachste Ansatz für Organisationen, die vollständige Kontrolle wünschen.

Systemadministratoren stellen die Konfigurationsdatei in einem systemweiten Verzeichnis bereit:

- macOS: `/Library/Application Support/ClaudeCode/managed-mcp.json`
- Linux und WSL: `/etc/claude-code/managed-mcp.json`
- Windows: `C:\Program Files\ClaudeCode\managed-mcp.json`

Note:

Dies sind systemweite Pfade (nicht Benutzer-Home-Verzeichnisse wie `~/Library/...`), die Administratorrechte erfordern. Sie sind dazu bestimmt, von IT-Administratoren bereitgestellt zu werden.

Die `managed-mcp.json`-Datei verwendet das gleiche Format wie eine Standard-`.mcp.json`-Datei:

```
{
  "mcpServers": {
    "github": {
      "type": "http",
      "url": "https://api.githubcopilot.com/mcp/"
    },
    "sentry": {
      "type": "http",
      "url": "https://mcp.sentry.dev/mcp"
    },
    "company-internal": {
      "type": "stdio",
      "command": "/usr/local/bin/company-mcp-server",
      "args": ["--config", "/etc/company/mcp-config.json"],
      "env": {
        "COMPANY_API_URL": "https://internal.company.com"
      }
    }
  }
}
```

Option 2: Richtlinienbasierte Kontrolle mit Allowlists und Denylists

Anstatt exklusive Kontrolle zu übernehmen, können Administratoren Benutzern erlauben, ihre eigenen MCP-Server zu konfigurieren, während sie Einschränkungen durchsetzen, welche Server zulässig sind. Dieser Ansatz verwendet `allowedMcpServers` und `deniedMcpServers` in der [verwalteten Einstellungsdatei](#).

Note:

Wahl zwischen Optionen: Verwenden Sie Option 1 (`managed-mcp.json`), wenn Sie einen festen Satz von Servern ohne Benutzeranpassung bereitstellen möchten. Verwenden Sie Option 2 (Allowlists/Denylists), wenn Sie Benutzern erlauben möchten, ihre eigenen Server innerhalb von Richtlinienbeschränkungen hinzuzufügen.

Einschränkungsoptionen

Jeder Eintrag in der Allowlist oder Denylist kann Server auf drei Arten einschränken:

1. **Nach Servername** (`serverName`): Entspricht dem konfigurierten Namen des Servers
2. **Nach Befehl** (`serverCommand`): Entspricht dem genauen Befehl und den Argumenten, die zum Starten von Stdio-Servern verwendet werden
3. **Nach URL-Muster** (`serverUrl`): Entspricht Remote-Server-URLs mit Wildcard-Unterstützung

Wichtig: Jeder Eintrag muss genau einen von `serverName` , `serverCommand` oder `serverUrl` haben.

Beispielkonfiguration

```
{
  "allowedMcpServers": [
    // Nach Servername zulassen
    { "serverName": "github" },
    { "serverName": "sentry" },

    // Nach exaktem Befehl zulassen (für Stdio-Server)
    { "serverCommand": ["npx", "-y", "@modelcontextprotocol/server-filesystem"] },
    { "serverCommand": ["python", "/usr/local/bin/approved-server.py"] },

    // Nach URL-Muster zulassen (für Remote-Server)
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ],
  "deniedMcpServers": [
    // Nach Servername blockieren
    { "serverName": "dangerous-server" },

    // Nach exaktem Befehl blockieren (für Stdio-Server)
    { "serverCommand": ["npx", "-y", "unapproved-package"] },

    // Nach URL-Muster blockieren (für Remote-Server)
    { "serverUrl": "https://*.untrusted.com/*" }
  ]
}
```

Wie befehlsbasierte Einschränkungen funktionieren

Exakte Übereinstimmung:

- Befehlsarrays müssen **genau** übereinstimmen – sowohl der Befehl als auch alle Argumente in der richtigen Reihenfolge
- Beispiel: `["npx", "-y", "server"]` entspricht NICHT `["npx", "server"]` oder `["npx", "-y", "server", "--flag"]`

Stdio-Server-Verhalten:

- Wenn die Allowlist **irgendwelche** `serverCommand`-Einträge enthält, müssen Stdio-Server einem dieser Befehle entsprechen

- Stdio-Server können nicht allein nach Name bestehen, wenn Befehlsbeschränkungen vorhanden sind
- Dies stellt sicher, dass Administratoren erzwingen können, welche Befehle ausgeführt werden dürfen

Verhalten von Remote-Servern:

- Remote-Server (HTTP, SSE, WebSocket) verwenden URL-basierte Übereinstimmung, wenn `serverUrl`-Einträge in der Allowlist vorhanden sind
- Wenn keine URL-Einträge vorhanden sind, greifen Remote-Server auf namensbasierte Übereinstimmung zurück
- Befehlsbeschränkungen gelten nicht für Remote-Server

Wie URL-basierte Einschränkungen funktionieren

URL-Muster unterstützen Wildcards mit `*`, um eine beliebige Zeichenfolge zu entsprechen. Dies ist nützlich, um ganze Domänen oder Subdomänen zuzulassen.

Wildcard-Beispiele:

- `https://mcp.company.com/*` – Alle Pfade auf einer bestimmten Domäne zulassen
- `https://*.example.com/*` – Jede Subdomain von example.com zulassen
- `http://localhost:*/*` – Jeden Port auf localhost zulassen

Verhalten von Remote-Servern:

- Wenn die Allowlist **irgendwelche** `serverUrl`-Einträge enthält, müssen Remote-Server einem dieser URL-Muster entsprechen
- Remote-Server können nicht allein nach Name bestehen, wenn URL-Beschränkungen vorhanden sind
- Dies stellt sicher, dass Administratoren erzwingen können, welche Remote-Endpunkte zulässig sind

```
{
  "allowedMcpServers": [
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ]
}
```

Ergebnis:

- HTTP-Server unter `https://mcp.company.com/api` :  Zulässig (entspricht URL-Muster)
- HTTP-Server unter `https://api.internal.corp/mcp` :  Zulässig (entspricht Wildcard-Subdomain)
- HTTP-Server unter `https://external.com/mcp` :  Blockiert (entspricht keinem URL-Muster)
- Stdio-Server mit beliebigem Befehl:  Blockiert (keine Namen- oder Befehlseinträge zum Abgleichen)

```
{
  "allowedMcpServers": [
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

Ergebnis:

- Stdio-Server mit `["npx", "-y", "approved-package"]` :  Zulässig (entspricht Befehl)
- Stdio-Server mit `["node", "server.js"]` :  Blockiert (entspricht nicht dem Befehl)
- HTTP-Server mit Namen „my-api“:  Blockiert (keine Nameneinträge zum Abgleichen)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

Ergebnis:

- Stdio-Server mit Namen „local-tool“ und `["npx", "-y", "approved-package"]` :  Zulässig (entspricht Befehl)
- Stdio-Server mit Namen „local-tool“ und `["node", "server.js"]` :  Blockiert (Befehlseinträge vorhanden, aber entspricht nicht)

- Stdio-Server mit Namen „github“ und `[ "node", "server.js" ]`: ❌ Blockiert (Stdio-Server müssen Befehlen entsprechen, wenn Befehlseinträge vorhanden sind)
- HTTP-Server mit Namen „github“: ✅ Zulässig (entspricht Name)
- HTTP-Server mit Namen „other-api“: ❌ Blockiert (Name entspricht nicht)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverName": "internal-tool" }
  ]
}
```

Ergebnis:

- Stdio-Server mit Namen „github“ und beliebigem Befehl: ✅ Zulässig (keine Befehlsbeschränkungen)
- Stdio-Server mit Namen „internal-tool“ und beliebigem Befehl: ✅ Zulässig (keine Befehlsbeschränkungen)
- HTTP-Server mit Namen „github“: ✅ Zulässig (entspricht Name)
- Beliebiger Server mit Namen „other“: ❌ Blockiert (Name entspricht nicht)

Allowlist-Verhalten (`allowedMcpServers`)

- `undefined` (Standard): Keine Einschränkungen – Benutzer können jeden MCP-Server konfigurieren
- Leeres Array `[]`: Vollständige Sperrung – Benutzer können keinen MCP-Server konfigurieren
- Liste von Einträgen: Benutzer können nur Server konfigurieren, die nach Name, Befehl oder URL-Muster übereinstimmen

Denylist-Verhalten (`deniedMcpServers`)

- `undefined` (Standard): Keine Server werden blockiert
- Leeres Array `[]`: Keine Server werden blockiert
- Liste von Einträgen: Angegebene Server werden über alle Bereiche hinweg explizit blockiert

Wichtige Hinweise

- **Option 1 und Option 2 können kombiniert werden:** Wenn `managed-mcp.json` vorhanden ist, hat es exklusive Kontrolle und Benutzer können keine Server hinzufügen. Allowlists/Denylists gelten immer noch für die verwalteten Server selbst.
- **Denylist hat absolute Vorrang:** Wenn ein Server einem Denylist-Eintrag entspricht (nach Name, Befehl oder URL), wird er blockiert, auch wenn er auf der Allowlist ist
- Namensbasierte, befehlsbasierte und URL-basierte Einschränkungen funktionieren zusammen: Ein Server wird zugelassen, wenn er einem Namenseintrag, einem Befehlseintrag oder einem URL-Muster entspricht (es sei denn, er wird durch Denylist blockiert)

Note:

Bei Verwendung von `managed-mcp.json`: Benutzer können MCP-Server nicht über `claude mcp add` oder Konfigurationsdateien hinzufügen. Die Einstellungen `allowedMcpServers` und `deniedMcpServers` gelten immer noch, um zu filtern, welche verwalteten Server tatsächlich geladen werden.

Claude mit Skills erweitern

Erstellen, verwalten und teilen Sie Skills, um Claudes Funktionen in Claude Code zu erweitern. Umfasst benutzerdefinierte Befehle und gebündelte Skills.

Skills erweitern das, was Claude tun kann. Erstellen Sie eine `SKILL.md`-Datei mit Anweisungen, und Claude fügt sie zu seinem Toolkit hinzu. Claude verwendet Skills, wenn sie relevant sind, oder Sie können einen direkt mit `/skill-name` aufrufen.

Note:

Für integrierte Befehle wie `/help` und `/compact` siehe [interaktiver Modus](#).

Benutzerdefinierte Befehle wurden in Skills zusammengeführt. Eine Datei unter `.claude/commands/deploy.md` und ein Skill unter `.claude/skills/deploy/SKILL.md` erstellen beide `/deploy` und funktionieren auf die gleiche Weise. Ihre vorhandenen `.claude/commands/`-Dateien funktionieren weiterhin. Skills fügen optionale Funktionen hinzu: ein Verzeichnis für unterstützende Dateien, Frontmatter zum [Steuern, wer einen Skill aufruft](#), und die Möglichkeit für Claude, sie automatisch zu laden, wenn sie relevant sind.

Claude Code Skills folgen dem [Agent Skills](#) offenen Standard, der über mehrere KI-Tools funktioniert. Claude Code erweitert den Standard mit zusätzlichen Funktionen wie [Invocation Control](#), [Subagent-Ausführung](#) und [dynamischer Kontexteinspritzung](#).

Gebündelte Skills

Gebündelte Skills werden mit Claude Code ausgeliefert und sind in jeder Sitzung verfügbar. Im Gegensatz zu [integrierten Befehlen](#), die direkt feste Logik ausführen, sind gebündelte Skills prompt-basiert: Sie geben Claude ein detailliertes Playbook und lassen es die Arbeit mit seinen Tools orchestrieren. Das bedeutet, dass gebündelte Skills parallele Agenten spawnen, Dateien lesen und sich an Ihre Codebasis anpassen können.

Sie rufen gebündelte Skills auf die gleiche Weise auf wie jeden anderen Skill: Geben Sie `/` gefolgt vom Skill-Namen ein.

- `/simplify` : überprüft Ihre kürzlich geänderten Dateien auf Code-Wiederverwendung, Qualität und Effizienzprobleme und behebt sie dann. Führen Sie es nach der Implementierung einer Funktion oder Fehlerbehebung aus, um Ihre Arbeit zu bereinigen. Es spawnt drei Review-Agenten parallel (Code-Wiederverwendung, Code-Qualität, Effizienz), aggregiert ihre Erkenntnisse und wendet Fixes an. Übergeben Sie optionalen

Text, um sich auf spezifische Bedenken zu konzentrieren: `/simplify focus on memory efficiency`.

- `/batch <instruction>` : orchestriert großflächige Änderungen über eine Codebasis parallel. Geben Sie eine Beschreibung der Änderung an und `/batch` recherchiert die Codebasis, zerlegt die Arbeit in 5 bis 30 unabhängige Einheiten und präsentiert einen Plan zur Genehmigung. Nach der Genehmigung spawnt es einen Hintergrund-Agenten pro Einheit, jeweils in einem isolierten [git worktree](#). Jeder Agent implementiert seine Einheit, führt Tests aus und öffnet einen Pull Request. Erfordert ein Git-Repository. Beispiel: `/batch migrate src/ from Solid to React`.
- `/debug [description]` : behebt Probleme in Ihrer aktuellen Claude Code-Sitzung durch Lesen des Sitzungs-Debug-Logs. Beschreiben Sie optional das Problem, um die Analyse zu fokussieren.
- `/loop [interval] <prompt>` : führt einen Prompt wiederholt in einem Intervall aus, während die Sitzung offen bleibt. Claude analysiert das Intervall, plant eine wiederkehrende Cron-Aufgabe und bestätigt den Rhythmus. Nützlich zum Abfragen einer Bereitstellung, zum Betreuen eines PR oder zum periodischen Erneut-Ausführen eines anderen Skills. Beispiel: `/loop 5m check if the deploy finished`. Siehe [Prompts nach Zeitplan ausführen](#).
- `/claude-api` : lädt Claude API-Referenzmaterial für die Sprache Ihres Projekts (Python, TypeScript, Java, Go, Ruby, C#, PHP oder cURL) und Agent SDK-Referenz für Python und TypeScript. Behandelt Tool-Verwendung, Streaming, Batches, strukturierte Ausgaben und häufige Fallstricke. Wird auch automatisch aktiviert, wenn Ihr Code `anthropic`, `@anthropic-ai/sdk` oder `claude_agent_sdk` importiert.

Erste Schritte

Erstellen Sie Ihren ersten Skill

Dieses Beispiel erstellt einen Skill, der Claude lehrt, Code mit visuellen Diagrammen und Analogien zu erklären. Da er Standard-Frontmatter verwendet, kann Claude ihn automatisch laden, wenn Sie fragen, wie etwas funktioniert, oder Sie können ihn direkt mit `/explain-code` aufrufen.

Step 1: Erstellen Sie das Skill-Verzeichnis

Erstellen Sie ein Verzeichnis für den Skill in Ihrem persönlichen Skills-Ordner. Persönliche Skills sind über alle Ihre Projekte hinweg verfügbar.

```
mkdir -p ~/.claude/skills/explain-code
```

Step 2: Schreiben Sie SKILL.md

Jeder Skill benötigt eine `SKILL.md`-Datei mit zwei Teilen: YAML-Frontmatter (zwischen `---`-Markierungen), das Claude mitteilt, wann der Skill verwendet werden soll, und Markdown-Inhalt mit Anweisungen, die Claude befolgt, wenn der Skill aufgerufen wird. Das `name`-Feld wird zum `/slash-command`, und die `description` hilft Claude zu entscheiden, wann der Skill automatisch geladen werden soll.

Erstellen Sie `~/.claude/skills/explain-code/SKILL.md`:

```
---
name: explain-code
description: Explains code with visual diagrams and analogies. Use when explaining
how code works, teaching about a codebase, or when the user asks "how does this
work?"
---
```

When explaining code, always include:

- **Start with an analogy**:** Compare the code to something from everyday life
- **Draw a diagram**:** Use ASCII art to show the flow, structure, or relationships
- **Walk through the code**:** Explain step-by-step what happens
- **Highlight a gotcha**:** What's a common mistake or misconception?

Keep explanations conversational. For complex concepts, use multiple analogies.

Step 3: Testen Sie den Skill

Sie können ihn auf zwei Arten testen:

Lassen Sie Claude ihn automatisch aufrufen, indem Sie etwas eingeben, das der Beschreibung entspricht:

```
How does this code work?
```

Oder rufen Sie ihn direkt auf mit dem Skill-Namen:

```
/explain-code src/auth/login.ts
```

In beiden Fällen sollte Claude eine Analogie und ein ASCII-Diagramm in seiner Erklärung enthalten.

Wo Skills leben

Wo Sie einen Skill speichern, bestimmt, wer ihn verwenden kann:

Standort	Pfad	Gilt für
Unternehmen	Siehe verwaltete Einstellungen	Alle Benutzer in Ihrer Organisation
Persönlich	<code>~/.claude/skills/<skill-name>/SKILL.md</code>	Alle Ihre Projekte
Projekt	<code>.claude/skills/<skill-name>/SKILL.md</code>	Nur dieses Projekt
Plugin	<code><plugin>/skills/<skill-name>/SKILL.md</code>	Wo das Plugin aktiviert ist

Wenn Skills auf verschiedenen Ebenen denselben Namen haben, gewinnen Standorte mit höherer Priorität: Unternehmen > Persönlich > Projekt. Plugin-Skills verwenden einen `plugin-name:skill-name`-Namespace, sodass sie nicht mit anderen Ebenen in Konflikt geraten können. Wenn Sie Dateien in `.claude/commands/` haben, funktionieren diese auf die gleiche Weise, aber wenn ein Skill und ein Befehl denselben Namen haben, hat der Skill Vorrang.

Automatische Erkennung aus verschachtelten Verzeichnissen

Wenn Sie mit Dateien in Unterverzeichnissen arbeiten, erkennt Claude Code automatisch Skills aus verschachtelten `.claude/skills/`-Verzeichnissen. Wenn Sie beispielsweise eine Datei in `packages/frontend/` bearbeiten, sucht Claude Code auch nach Skills in `packages/frontend/.claude/skills/`. Dies unterstützt Monorepo-Setups, bei denen Pakete ihre eigenen Skills haben.

Jeder Skill ist ein Verzeichnis mit `SKILL.md` als Einstiegspunkt:

```

my-skill/
├─ SKILL.md           # Main instructions (required)
├─ template.md        # Template for Claude to fill in
├─ examples/
│   └─ sample.md      # Example output showing expected format
└─ scripts/
    └─ validate.sh    # Script Claude can execute

```

Die **SKILL.md** enthält die Hauptanweisungen und ist erforderlich. Andere Dateien sind optional und ermöglichen es Ihnen, leistungsfähigere Skills zu erstellen: Vorlagen für Claude zum Ausfüllen, Beispielausgaben, die das erwartete Format zeigen, Scripts, die Claude ausführen kann, oder detaillierte Referenzdokumentation. Verweisen Sie auf diese Dateien von Ihrer **SKILL.md** aus, damit Claude weiß, was sie enthalten und wann sie geladen werden sollen. Siehe [Unterstützende Dateien hinzufügen](#) für weitere Details.

Note:

Dateien in `.claude/commands/` funktionieren weiterhin und unterstützen das gleiche [Frontmatter](#). Skills werden empfohlen, da sie zusätzliche Funktionen wie unterstützende Dateien unterstützen.

Skills aus zusätzlichen Verzeichnissen

Skills, die in `.claude/skills/` in Verzeichnissen definiert sind, die über `--add-dir` hinzugefügt wurden, werden automatisch geladen und von der Live-Änderungserkennung aufgegriffen, sodass Sie sie während einer Sitzung bearbeiten können, ohne neu zu starten.

Note:

CLAUDE.md-Dateien aus `--add-dir`-Verzeichnissen werden standardmäßig nicht geladen. Um sie zu laden, setzen Sie `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1`. Siehe [Aus zusätzlichen Verzeichnissen laden](#).

Skills konfigurieren

Skills werden durch YAML-Frontmatter oben in **SKILL.md** und den folgenden Markdown-Inhalt konfiguriert.

Arten von Skill-Inhalten

Skill-Dateien können beliebige Anweisungen enthalten, aber das Nachdenken darüber, wie Sie sie aufrufen möchten, hilft zu leiten, was Sie einbeziehen:

Referenzinhalt fügt Wissen hinzu, das Claude auf Ihre aktuelle Arbeit anwendet. Konventionen, Muster, Stilhandbücher, Domänenwissen. Dieser Inhalt wird inline ausgeführt, sodass Claude ihn neben Ihrem Gesprächskontext verwenden kann.

```
---
name: api-conventions
description: API design patterns for this codebase
---
```

```
When writing API endpoints:
- Use RESTful naming conventions
- Return consistent error formats
- Include request validation
```

Task-Inhalt gibt Claude Schritt-für-Schritt-Anweisungen für eine bestimmte Aktion, wie Bereitstellungen, Commits oder Code-Generierung. Dies sind oft Aktionen, die Sie direkt mit `/skill-name` aufrufen möchten, anstatt Claude entscheiden zu lassen, wann sie ausgeführt werden. Fügen Sie `disable-model-invocation: true` hinzu, um zu verhindern, dass Claude sie automatisch auslöst.

```
---
name: deploy
description: Deploy the application to production
context: fork
disable-model-invocation: true
---
```

```
Deploy the application:
1. Run the test suite
2. Build the application
3. Push to the deployment target
```

Ihre `SKILL.md` kann alles enthalten, aber das Nachdenken darüber, wie Sie den Skill aufrufen möchten (von Ihnen, von Claude oder von beiden) und wo Sie ihn ausführen möchten (inline oder in einem Subagenten) hilft zu leiten, was Sie einbeziehen. Für komplexe Skills können Sie auch [unterstützende Dateien hinzufügen](#), um den Hauptskill fokussiert zu halten.

Frontmatter-Referenz

Über den Markdown-Inhalt hinaus können Sie das Skill-Verhalten mit YAML-Frontmatter-Feldern zwischen `---`-Markierungen oben in Ihrer `SKILL.md`-Datei konfigurieren:

```
---
name: my-skill
description: What this skill does
disable-model-invocation: true
allowed-tools: Read, Grep
---

Your skill instructions here...
```

Alle Felder sind optional. Nur `description` wird empfohlen, damit Claude weiß, wann der Skill verwendet werden soll.

Feld	Erforderlich	Beschreibung
<code>name</code>	Nein	Anzeigenname für den Skill. Falls weggelassen, wird der Verzeichnisname verwendet. Nur Kleinbuchstaben, Zahlen und Bindestriche (max. 64 Zeichen).
<code>description</code>	Empfohlen	Was der Skill tut und wann er verwendet werden soll. Claude verwendet dies, um zu entscheiden, wann der Skill angewendet werden soll. Falls weggelassen, wird der erste Absatz des Markdown-Inhalts verwendet.

Feld	Erforderlich	Beschreibung
<code>argument-hint</code>	Nein	Hinweis, der während der Autovervollständigung angezeigt wird, um erwartete Argumente anzuzeigen. Beispiel: <code>[issue-number]</code> oder <code>[filename] [format]</code> .
<code>disable-model-invocation</code>	Nein	Setzen Sie auf <code>true</code> , um zu verhindern, dass Claude diesen Skill automatisch lädt. Verwenden Sie für Workflows, die Sie manuell mit <code>/name</code> auslösen möchten. Standard: <code>false</code> .
<code>user-invocable</code>	Nein	Setzen Sie auf <code>false</code> , um aus dem <code>/</code> -Menü auszublenzen. Verwenden Sie für Hintergrundwissen, das Benutzer nicht direkt aufrufen sollten. Standard: <code>true</code> .
<code>allowed-tools</code>	Nein	Tools, die Claude ohne Genehmigung verwenden kann, wenn dieser Skill aktiv ist.
<code>model</code>	Nein	Modell, das verwendet werden soll, wenn dieser Skill aktiv ist.
<code>context</code>	Nein	Setzen Sie auf <code>fork</code> , um in einem verzweigten Subagenten-Kontext ausgeführt zu werden.
<code>agent</code>	Nein	Welcher Subagenten-Typ verwendet werden soll, wenn <code>context: fork</code> gesetzt ist.

Feld	Erforderlich	Beschreibung
<code>hooks</code>	Nein	Hooks, die auf den Lebenszyklus dieses Skills beschränkt sind. Siehe Hooks in Skills und Agenten für das Konfigurationsformat.

Verfügbare String-Substitutionen

Skills unterstützen String-Substitution für dynamische Werte im Skill-Inhalt:

Variable	Beschreibung
<code>\$ARGUMENTS</code>	Alle Argumente, die beim Aufrufen des Skills übergeben werden. Wenn <code>\$ARGUMENTS</code> nicht im Inhalt vorhanden ist, werden Argumente als <code>ARGUMENTS: <value></code> angehängt.
<code>\$ARGUMENTS[N]</code>	Greifen Sie auf ein bestimmtes Argument nach 0-basiertem Index zu, z. B. <code>\$ARGUMENTS[0]</code> für das erste Argument.
<code>\$N</code>	Kurzform für <code>\$ARGUMENTS[N]</code> , z. B. <code>\$0</code> für das erste Argument oder <code>\$1</code> für das zweite.
<code>\$ {CLAUDE_SESSION_ID }</code>	Die aktuelle Sitzungs-ID. Nützlich zum Protokollieren, Erstellen sitzungsspezifischer Dateien oder Korrelieren der Skill-Ausgabe mit Sitzungen.
<code>\$ {CLAUDE_SKILL_DIR}</code>	Das Verzeichnis, das die <code>SKILL.md</code> -Datei des Skills enthält. Für Plugin-Skills ist dies das Skill-Unterverzeichnis im Plugin, nicht das Plugin-Root. Verwenden Sie dies in Bash-Injektionsbefehlen, um auf Scripts oder Dateien zu verweisen, die mit dem Skill gebündelt sind, unabhängig vom aktuellen Arbeitsverzeichnis.

Beispiel mit Substitutionen:

```

---
name: session-logger
description: Log activity for this session
---

Log the following to logs/${CLAUDE_SESSION_ID}.log:

$ARGUMENTS

```

Unterstützende Dateien hinzufügen

Skills können mehrere Dateien in ihrem Verzeichnis enthalten. Dies hält **SKILL.md** auf das Wesentliche konzentriert, während Claude detailliertes Referenzmaterial nur bei Bedarf abrufen kann. Große Referenzdokumente, API-Spezifikationen oder Beispielsammlungen müssen nicht jedes Mal geladen werden, wenn der Skill ausgeführt wird.

```

my-skill/
├─ SKILL.md (required - overview and navigation)
├─ reference.md (detailed API docs - loaded when needed)
├─ examples.md (usage examples - loaded when needed)
└─ scripts/
    └─ helper.py (utility script - executed, not loaded)

```

Verweisen Sie auf unterstützende Dateien von **SKILL.md** aus, damit Claude weiß, was jede Datei enthält und wann sie geladen werden soll:

```

### Additional resources

- For complete API details, see [reference.md](reference.md)
- For usage examples, see [examples.md](examples.md)

```

Tip:

Halten Sie **SKILL.md** unter 500 Zeilen. Verschieben Sie detailliertes Referenzmaterial in separate Dateien.

Steuern Sie, wer einen Skill aufruft

Standardmäßig können sowohl Sie als auch Claude jeden Skill aufrufen. Sie können `/skill-name` eingeben, um ihn direkt aufzurufen, und Claude kann ihn automatisch laden, wenn er für Ihr Gespräch relevant ist. Zwei Frontmatter-Felder ermöglichen es Ihnen, dies einzuschränken:

- `disable-model-invocation: true` : Nur Sie können den Skill aufrufen. Verwenden Sie dies für Workflows mit Nebenwirkungen oder die Sie zeitlich steuern möchten, wie `/commit`, `/deploy` oder `/send-slack-message`. Sie möchten nicht, dass Claude bereitstellt, weil Ihr Code bereit aussieht.
- `user-invocable: false` : Nur Claude kann den Skill aufrufen. Verwenden Sie dies für Hintergrundwissen, das nicht als Befehl umsetzbar ist. Ein `legacy-system-context`-Skill erklärt, wie ein altes System funktioniert. Claude sollte dies kennen, wenn es relevant ist, aber `/legacy-system-context` ist keine aussagekräftige Aktion für Benutzer.

Dieses Beispiel erstellt einen Deploy-Skill, den nur Sie auslösen können. Das `disable-model-invocation: true`-Feld verhindert, dass Claude ihn automatisch ausführt:

```
---
name: deploy
description: Deploy the application to production
disable-model-invocation: true
---
```

Deploy \$ARGUMENTS to production:

1. Run the test suite
2. Build the application
3. Push to the deployment target
4. Verify the deployment succeeded

Hier ist, wie die beiden Felder Aufrufe und Kontextladung beeinflussen:

Front-matter	Sie können aufrufen	Claude kann aufrufen	Wann in Kontext geladen
(Standard)	Ja	Ja	Beschreibung immer im Kontext, vollständiger Skill wird beim Aufrufen geladen
<code>disable-model-invocation: true</code>	Ja	Nein	Beschreibung nicht im Kontext, vollständiger Skill wird geladen, wenn Sie aufrufen
<code>user-invocable: false</code>	Nein	Ja	Beschreibung immer im Kontext, vollständiger Skill wird beim Aufrufen geladen

Note:

In einer regulären Sitzung werden Skill-Beschreibungen in den Kontext geladen, damit Claude weiß, was verfügbar ist, aber der vollständige Skill-Inhalt wird nur beim Aufrufen geladen. [Sub-agenten mit vorgeladenen Skills](#) funktionieren anders: Der vollständige Skill-Inhalt wird beim Start eingespritzt.

Beschränken Sie den Tool-Zugriff

Verwenden Sie das `allowed-tools`-Feld, um zu begrenzen, welche Tools Claude verwenden kann, wenn ein Skill aktiv ist. Dieser Skill erstellt einen Nur-Lese-Modus, in dem Claude Dateien erkunden, aber nicht ändern kann:

```
---
name: safe-reader
description: Read files without making changes
allowed-tools: Read, Grep, Glob
---
```

Übergeben Sie Argumente an Skills

Sowohl Sie als auch Claude können Argumente beim Aufrufen eines Skills übergeben. Argumente sind über den `$ARGUMENTS`-Platzhalter verfügbar.

Dieser Skill behebt ein GitHub-Problem nach Nummer. Der `$ARGUMENTS`-Platzhalter wird durch alles ersetzt, was dem Skill-Namen folgt:

```
---
name: fix-issue
description: Fix a GitHub issue
disable-model-invocation: true
---

Fix GitHub issue $ARGUMENTS following our coding standards.

1. Read the issue description
2. Understand the requirements
3. Implement the fix
4. Write tests
5. Create a commit
```

Wenn Sie `/fix-issue 123` ausführen, erhält Claude `'Fix GitHub issue 123 following our coding standards...'`

Wenn Sie einen Skill mit Argumenten aufrufen, aber der Skill `$ARGUMENTS` nicht enthält, hängt Claude Code `ARGUMENTS: <your input>` am Ende des Skill-Inhalts an, damit Claude immer noch sieht, was Sie eingegeben haben.

Um auf einzelne Argumente nach Position zuzugreifen, verwenden Sie `$ARGUMENTS[N]` oder die kürzere Form `$N`:

```
---
name: migrate-component
description: Migrate a component from one framework to another
---

Migrate the $ARGUMENTS[0] component from $ARGUMENTS[1] to $ARGUMENTS[2].
Preserve all existing behavior and tests.
```

Wenn Sie `/migrate-component SearchBar React Vue` ausführen, wird `$ARGUMENTS[0]` durch `SearchBar`, `$ARGUMENTS[1]` durch `React` und `$ARGUMENTS[2]` durch `Vue` ersetzt. Der gleiche Skill mit der `$N`-Kurzform:

```
---
name: migrate-component
description: Migrate a component from one framework to another
---

Migrate the $0 component from $1 to $2.
Preserve all existing behavior and tests.
```

Fortgeschrittene Muster

Injizieren Sie dynamischen Kontext

Die `! command` ``-Syntax führt Shell-Befehle aus, bevor der Skill-Inhalt an Claude gesendet wird. Die Befehlsausgabe ersetzt den Platzhalter, sodass Claude tatsächliche Daten erhält, nicht den Befehl selbst.

Dieser Skill fasst einen Pull Request zusammen, indem er Live-PR-Daten mit der GitHub CLI abruft. Die `! gh pr diff` ``- und andere Befehle werden zuerst ausgeführt, und ihre Ausgabe wird in den Prompt eingefügt:

```

---
name: pr-summary
description: Summarize changes in a pull request
context: fork
agent: Explore
allowed-tools: Bash(gh *)
---

### Pull request context
- PR diff: !`gh pr diff`
- PR comments: !`gh pr view --comments`
- Changed files: !`gh pr diff --name-only`

### Your task
Summarize this pull request...

```

Wenn dieser Skill ausgeführt wird:

1. Jeder `!`command`` wird sofort ausgeführt (bevor Claude etwas sieht)
2. Die Ausgabe ersetzt den Platzhalter im Skill-Inhalt
3. Claude erhält den vollständig gerenderten Prompt mit tatsächlichen PR-Daten

Dies ist Vorverarbeitung, nicht etwas, das Claude ausführt. Claude sieht nur das Endergebnis.

Tip:

Um [erweitertes Denken](#) in einem Skill zu aktivieren, fügen Sie das Wort „ultrathink” irgendwo in Ihren Skill-Inhalt ein.

Führen Sie Skills in einem Subagenten aus

Fügen Sie `context: fork` zu Ihrem Frontmatter hinzu, wenn Sie möchten, dass ein Skill isoliert ausgeführt wird. Der Skill-Inhalt wird zum Prompt, der den Subagenten antreibt. Er hat keinen Zugriff auf Ihren Gesprächsverlauf.

Warning:

context: fork macht nur Sinn für Skills mit expliziten Anweisungen. Wenn Ihr Skill Richtlinien wie „verwenden Sie diese API-Konventionen“ ohne eine Aufgabe enthält, erhält der Subagent die Richtlinien, aber keinen umsetzbaren Prompt, und gibt ohne aussagekräftige Ausgabe zurück.

Skills und [Subagenten](#) funktionieren in zwei Richtungen zusammen:

Ansatz	System-Prompt	Aufgabe	Lädt auch
Skill mit context: fork	Vom Agenten-Typ (Explore , Plan , etc.)	SKILL.md-Inhalt	CLAUDE.md
Subagent mit skills -Feld	Markdown-Body des Subagenten	Claudes Delegationsnachricht	Vorgeladene Skills + CLAUDE.md

Mit **context: fork** schreiben Sie die Aufgabe in Ihren Skill und wählen einen Agenten-Typ aus, um sie auszuführen. Für das Inverse (Definieren eines benutzerdefinierten Subagenten, der Skills als Referenzmaterial verwendet), siehe [Subagenten](#).

Beispiel: Research-Skill mit Explore-Agent

Dieser Skill führt Recherchen in einem verzweigten Explore-Agenten aus. Der Skill-Inhalt wird zur Aufgabe, und der Agent bietet schreibgeschützte Tools, die für die Codebase-Erkundung optimiert sind:

```
---
name: deep-research
description: Research a topic thoroughly
context: fork
agent: Explore
---

Research $ARGUMENTS thoroughly:

1. Find relevant files using Glob and Grep
2. Read and analyze the code
3. Summarize findings with specific file references
```


Wenn dieser Skill ausgeführt wird:

1. Ein neuer isolierter Kontext wird erstellt
2. Der Subagent erhält den Skill-Inhalt als seinen Prompt („Research \$ARGUMENTS thoroughly...“)
3. Das `agent` -Feld bestimmt die Ausführungsumgebung (Modell, Tools und Berechtigungen)
4. Ergebnisse werden zusammengefasst und an Ihr Hauptgespräch zurückgegeben

Das `agent` -Feld gibt an, welche Subagenten-Konfiguration verwendet werden soll. Optionen umfassen integrierte Agenten (`Explore` , `Plan` , `general-purpose`) oder jeden benutzerdefinierten Subagenten aus `.claude/agents/`. Falls weggelassen, wird `general-purpose` verwendet.

Beschränken Sie Claudes Skill-Zugriff

Standardmäßig kann Claude jeden Skill aufrufen, der nicht `disable-model-invocation: true` gesetzt hat. Skills, die `allowed-tools` definieren, gewähren Claude Zugriff auf diese Tools ohne Genehmigung pro Verwendung, wenn der Skill aktiv ist. Ihre [Berechtigungseinstellungen](#) regeln weiterhin das Baseline-Genehmigungsverhalten für alle anderen Tools. Integrierte Befehle wie `/compact` und `/init` sind nicht über das Skill-Tool verfügbar.

Drei Möglichkeiten, um zu steuern, welche Skills Claude aufrufen kann:

Deaktivieren Sie alle Skills, indem Sie das Skill-Tool in `/permissions` ablehnen:

```
## Add to deny rules:  
Skill
```

Erlauben oder verweigern Sie bestimmte Skills mit [Berechtigungsregeln](#):

```
## Allow only specific skills  
Skill(commit)  
Skill(review-pr *)  
  
## Deny specific skills  
Skill(deploy *)
```

Berechtigungssyntax: `Skill(name)` für exakte Übereinstimmung, `Skill(name *)` für Präfix-Übereinstimmung mit beliebigen Argumenten.

Verstecken Sie einzelne Skills, indem Sie `disable-model-invocation: true` zu ihrem Frontmatter hinzufügen. Dies entfernt den Skill vollständig aus Claudes Kontext.

Note:

Das `user-invocable` -Feld steuert nur die Menüsichtbarkeit, nicht den Skill-Tool-Zugriff. Verwenden Sie `disable-model-invocation: true`, um die programmgesteuerte Aufrufe zu blockieren.

Teilen Sie Skills

Skills können je nach Ihrem Publikum in verschiedenen Bereichen verteilt werden:

- **Projekt-Skills:** Committen Sie `.claude/skills/` zur Versionskontrolle
- **Plugins:** Erstellen Sie ein `skills/` -Verzeichnis in Ihrem [Plugin](#)
- **Verwaltet:** Stellen Sie organisationsweit über [verwaltete Einstellungen](#) bereit

Generieren Sie visuelle Ausgabe

Skills können Scripts in jeder Sprache bündeln und ausführen, was Claude Funktionen gibt, die über das hinausgehen, was in einem einzelnen Prompt möglich ist. Ein leistungsstarkes Muster ist die Generierung visueller Ausgabe: interaktive HTML-Dateien, die in Ihrem Browser geöffnet werden, um Daten zu erkunden, zu debuggen oder Berichte zu erstellen.

Dieses Beispiel erstellt einen Codebase-Explorer: eine interaktive Baumansicht, in der Sie Verzeichnisse erweitern und reduzieren können, Dateigröße auf einen Blick sehen und Dateitypen nach Farbe identifizieren können.

Erstellen Sie das Skill-Verzeichnis:

```
mkdir -p ~/.claude/skills/codebase-visualizer/scripts
```

Erstellen Sie `~/.claude/skills/codebase-visualizer/SKILL.md`. Die Beschreibung teilt Claude mit, wann dieser Skill aktiviert werden soll, und die Anweisungen teilen Claude mit, das gebündelte Script auszuführen:

```

---
name: codebase-visualizer
description: Generate an interactive collapsible tree visualization of your
codebase. Use when exploring a new repo, understanding project structure, or
identifying large files.
allowed-tools: Bash(python *)
---

## Codebase Visualizer

Generate an interactive HTML tree view that shows your project's file structure
with collapsible directories.

### Usage

Run the visualization script from your project root:

```bash
python ~/.claude/skills/codebase-visualizer/scripts/visualize.py .
```text

This creates `codebase-map.html` in the current directory and opens it in your
default browser.

### What the visualization shows

- **Collapsible directories**: Click folders to expand/collapse
- **File sizes**: Displayed next to each file
- **Colors**: Different colors for different file types
- **Directory totals**: Shows aggregate size of each folder

```

Erstellen Sie `~/.claude/skills/codebase-visualizer/scripts/visualize.py`. Dieses Script scannt einen Verzeichnisbaum und generiert eine eigenständige HTML-Datei mit:

- Eine **Zusammenfassungs-Seitenleiste**, die Dateianzahl, Verzeichnisanzahl, Gesamtgröße und Anzahl der Dateitypen anzeigt
- Ein **Balkendiagramm**, das die Codebasis nach Dateityp aufschlüsselt (Top 8 nach Größe)

- Einen **zusammenklappbaren Baum**, in dem Sie Verzeichnisse erweitern und reduzieren können, mit farbcodierten Dateityp-Indikatoren

Das Script erfordert Python, verwendet aber nur integrierte Bibliotheken, daher müssen keine Pakete installiert werden:

```
#!/usr/bin/env python3
"""Generate an interactive collapsible tree visualization of a codebase."""

from pathlib import Path
from collections import Counter

IGNORE = {'.git', 'node_modules', '__pycache__', '.venv', 'venv', 'dist', 'build'}

def scan(path: Path, stats: dict) → dict:
    result = {"name": path.name, "children": [], "size": 0}
    try:
        for item in sorted(path.iterdir()):
            if item.name in IGNORE or item.name.startswith('.'):
                continue
            if item.is_file():
                size = item.stat().st_size
                ext = item.suffix.lower() or '(no ext)'
                result["children"].append({"name": item.name, "size": size,
"ext": ext})

                result["size"] += size
                stats["files"] += 1
                stats["extensions"][ext] += 1
                stats["ext_sizes"][ext] += size
            elif item.is_dir():
                stats["dirs"] += 1
                child = scan(item, stats)
                if child["children"]:
                    result["children"].append(child)
                    result["size"] += child["size"]
        except PermissionError:
            pass
    return result

def generate_html(data: dict, stats: dict, output: Path) → None:
    ext_sizes = stats["ext_sizes"]
    total_size = sum(ext_sizes.values()) or 1
    sorted_exts = sorted(ext_sizes.items(), key=lambda x: -x[1])[0:8]
    colors = {
```

```

        '.js': '#f7df1e', '.ts': '#3178c6', '.py': '#3776ab', '.go': '#00add8',
        '.rs': '#dea584', '.rb': '#cc342d', '.css': '#264de4', '.html': '#e34c26',
        '.json': '#6b7280', '.md': '#083fa1', '.yaml': '#cb171e', '.yml': '#cb171e
    ',

        '.mdx': '#083fa1', '.tsx': '#3178c6', '.jsx': '#61dafb', '.sh': '#4eaa25',
    }

    lang_bars = "".join(
        f'<div class="bar-row"><span class="bar-label">{ext}</span>'
        f'<div class="bar" style="width:{(size/total_size)*100}%;background:{color
s.get(ext,"#6b7280")}"></div>'
        f'<span class="bar-pct">{(size/total_size)*100:.1f}%</span></div>'
        for ext, size in sorted_exts
    )

    def fmt(b):
        if b < 1024: return f"{b} B"
        if b < 1048576: return f"{b/1024:.1f} KB"
        return f"{b/1048576:.1f} MB"

    html = f'''<!DOCTYPE html>
<html><head>
<meta charset="utf-8"><title>Codebase Explorer</title>
<style>
    body {{ font: 14px/1.5 system-ui, sans-serif; margin: 0; background: #1a1a2e;
color: #eee; }}
    .container {{ display: flex; height: 100vh; }}
    .sidebar {{ width: 280px; background: #252542; padding: 20px; border-right:
1px solid #3d3d5c; overflow-y: auto; flex-shrink: 0; }}
    .main {{ flex: 1; padding: 20px; overflow-y: auto; }}
    h1 {{ margin: 0 0 10px 0; font-size: 18px; }}
    h2 {{ margin: 20px 0 10px 0; font-size: 14px; color: #888; text-transform:
uppercase; }}
    .stat {{ display: flex; justify-content: space-between; padding: 8px 0;
border-bottom: 1px solid #3d3d5c; }}
    .stat-value {{ font-weight: bold; }}
    .bar-row {{ display: flex; align-items: center; margin: 6px 0; }}
    .bar-label {{ width: 55px; font-size: 12px; color: #aaa; }}
    .bar {{ height: 18px; border-radius: 3px; }}
    .bar-pct {{ margin-left: 8px; font-size: 12px; color: #666; }}
    .tree {{ list-style: none; padding-left: 20px; }}

```

```

    details {{ cursor: pointer; }}
    summary {{ padding: 4px 8px; border-radius: 4px; }}
    summary:hover {{ background: #2d2d44; }}
    .folder {{ color: #ffd700; }}
    .file {{ display: flex; align-items: center; padding: 4px 8px; border-radius:
4px; }}
    .file:hover {{ background: #2d2d44; }}
    .size {{ color: #888; margin-left: auto; font-size: 12px; }}
    .dot {{ width: 8px; height: 8px; border-radius: 50%; margin-right: 8px; }}
</style>
</head><body>
<div class="container">
  <div class="sidebar">
    <h1><img alt="Summary icon" data-bbox="211 356 228 371"/> Summary</h1>
    <div class="stat"><span>Files</span><span class="stat-
value">{stats["files"]:,}</span></div>
    <div class="stat"><span>Directories</span><span class="stat-value">{stats["d
irs"]:,}</span></div>
    <div class="stat"><span>Total size</span><span class="stat-
value">{fmt(data["size"])}</span></div>
    <div class="stat"><span>File types</span><span class="stat-value">{len(stats
["extensions"])}</span></div>
    <h2>By file type</h2>
    {lang_bars}
  </div>
  <div class="main">
    <h1><img alt="Folder icon" data-bbox="211 634 228 649"/> {data["name"]}</h1>
    <ul class="tree" id="root"></ul>
  </div>
</div>
<script>
  const data = {json.dumps(data)};
  const colors = {json.dumps(colors)};
  function fmt(b) {{ if (b < 1024) return b + ' B'; if (b < 1048576) return (b/
1024).toFixed(1) + ' KB'; return (b/1048576).toFixed(1) + ' MB'; }}
  function render(node, parent) {{
    if (node.children) {{
      const det = document.createElement('details');
      det.open = parent ≡ document.getElementById('root');

```

```

    det.innerHTML = `<summary><span class="folder">📁 ${node.name}</span><span class="size">${fmt(node.size)}</span></summary>`;
    const ul = document.createElement('ul'); ul.className = 'tree';
    node.children.sort((a,b) => (b.children?1:0)-(a.children?1:0) || a.name.localeCompare(b.name));
    node.children.forEach(c => render(c, ul));
    det.appendChild(ul);
    const li = document.createElement('li'); li.appendChild(det);
    parent.appendChild(li);
  } else {
    const li = document.createElement('li'); li.className = 'file';
    li.innerHTML = `<span class="dot" style="background:${colors[node.ext]} #6b7280"></span>${node.name}<span class="size">${fmt(node.size)}</span>`;
    parent.appendChild(li);
  }
}
data.children.forEach(c => render(c, document.getElementById('root')));
</script>
</body></html>'''
output.write_text(html)

if __name__ == '__main__':
    target = Path(sys.argv[1] if len(sys.argv) > 1 else '.').resolve()
    stats = {"files": 0, "dirs": 0, "extensions": Counter(), "ext_sizes": Counter()}
    data = scan(target, stats)
    out = Path('codebase-map.html')
    generate_html(data, stats, out)
    print(f'Generated {out.absolute()}')
    webbrowser.open(f'file://{out.absolute()}')
```

Um zu testen, öffnen Sie Claude Code in einem beliebigen Projekt und fragen Sie „Visualize this codebase.“ Claude führt das Script aus, generiert `codebase-map.html` und öffnet es in Ihrem Browser.

Dieses Muster funktioniert für jede visuelle Ausgabe: Abhängigkeitsgraphen, Test-Coverage-Berichte, API-Dokumentation oder Datenbankschema-Visualisierungen. Das gebündelte Script erledigt die schwere Arbeit, während Claude die Orchestrierung übernimmt.

Fehlerbehebung

Skill wird nicht ausgelöst

Wenn Claude Ihren Skill nicht verwendet, wenn erwartet:

1. Überprüfen Sie, ob die Beschreibung Schlüsselwörter enthält, die Benutzer natürlicherweise sagen würden
2. Überprüfen Sie, ob der Skill in „What skills are available?“ angezeigt wird
3. Versuchen Sie, Ihre Anfrage umzuformulieren, um die Beschreibung besser zu treffen
4. Rufen Sie ihn direkt mit `/skill-name` auf, wenn der Skill vom Benutzer aufgerufen werden kann

Skill wird zu oft ausgelöst

Wenn Claude Ihren Skill verwendet, wenn Sie das nicht möchten:

1. Machen Sie die Beschreibung spezifischer
2. Fügen Sie `disable-model-invocation: true` hinzu, wenn Sie nur manuelle Aufrufe möchten

Claude sieht nicht alle meine Skills

Skill-Beschreibungen werden in den Kontext geladen, damit Claude weiß, was verfügbar ist. Wenn Sie viele Skills haben, können sie das Zeichenbudget überschreiten. Das Budget skaliert dynamisch bei 2% des Kontextfensters, mit einem Fallback von 16.000 Zeichen. Führen Sie `/context` aus, um auf eine Warnung über ausgeschlossene Skills zu prüfen.

Um das Limit zu überschreiben, setzen Sie die Umgebungsvariable

`SLASH_COMMAND_TOOL_CHAR_BUDGET`.

Verwandte Ressourcen

- **Subagenten**: delegieren Sie Aufgaben an spezialisierte Agenten
- **Plugins**: packen und verteilen Sie Skills mit anderen Erweiterungen
- **Hooks**: automatisieren Sie Workflows um Tool-Ereignisse
- **Memory**: verwalten Sie CLAUDE.md-Dateien für persistenten Kontext
- **Interaktiver Modus**: integrierte Befehle und Shortcuts
- **Berechtigungen**: steuern Sie Tool- und Skill-Zugriff

Plugins erstellen

Erstellen Sie benutzerdefinierte Plugins, um Claude Code mit Skills, Agents, Hooks und MCP-Servern zu erweitern.

Plugins ermöglichen es Ihnen, Claude Code mit benutzerdefinierten Funktionen zu erweitern, die projektübergreifend und teamübergreifend freigegeben werden können. Diese Anleitung behandelt die Erstellung eigener Plugins mit Skills, Agents, Hooks und MCP-Servern.

Möchten Sie vorhandene Plugins installieren? Siehe [Plugins entdecken und installieren](#). Für vollständige technische Spezifikationen siehe [Plugins-Referenz](#).

Wann Plugins vs. eigenständige Konfiguration verwenden

Claude Code unterstützt zwei Möglichkeiten, um benutzerdefinierte Skills, Agents und Hooks hinzuzufügen:

Ansatz	Skill-Namen	Am besten für
Eigenständig (<code>.claude/</code> -Verzeichnis)	<code>/hello</code>	Persönliche Workflows, projektspezifische Anpassungen, schnelle Experimente
Plugins (Verzeichnisse mit <code>.claude-plugin/plugin.json</code>)	<code>/plugin-name:hello</code>	Freigabe für Teamkollegen, Verteilung an die Community, versionierte Releases, wiederverwendbar über Projekte hinweg

Verwenden Sie eigenständige Konfiguration, wenn:

- Sie Claude Code für ein einzelnes Projekt anpassen
- Die Konfiguration persönlich ist und nicht freigegeben werden muss
- Sie mit Skills oder Hooks experimentieren, bevor Sie diese verpacken
- Sie kurze Skill-Namen wie `/hello` oder `/deploy` möchten

Verwenden Sie Plugins, wenn:

- Sie Funktionen mit Ihrem Team oder der Community teilen möchten

- Sie die gleichen Skills/Agents über mehrere Projekte hinweg benötigen
- Sie Versionskontrolle und einfache Updates für Ihre Erweiterungen möchten
- Sie über einen Marketplace verteilen
- Sie mit Namespace-Skills wie `/my-plugin:hello` einverstanden sind (Namespacing verhindert Konflikte zwischen Plugins)

Tip:

Beginnen Sie mit eigenständiger Konfiguration in `.claude/` für schnelle Iteration, dann [konvertieren Sie zu einem Plugin](#), wenn Sie bereit sind zu teilen.

Schnellstart

Dieser Schnellstart führt Sie durch die Erstellung eines Plugins mit einem benutzerdefinierten Skill. Sie erstellen ein Manifest (die Konfigurationsdatei, die Ihr Plugin definiert), fügen einen Skill hinzu und testen ihn lokal mit dem Flag `--plugin-dir`.

Voraussetzungen

- Claude Code [installiert und authentifiziert](#)
- Claude Code Version 1.0.33 oder später (führen Sie `claude --version` aus, um zu überprüfen)

Note:

Wenn Sie den Befehl `/plugin` nicht sehen, aktualisieren Sie Claude Code auf die neueste Version. Siehe [Troubleshooting](#) für Upgrade-Anweisungen.

Erstellen Sie Ihr erstes Plugin

Step 1: Erstellen Sie das Plugin-Verzeichnis

Jedes Plugin befindet sich in seinem eigenen Verzeichnis, das ein Manifest und Ihre Skills, Agents oder Hooks enthält. Erstellen Sie jetzt eines:

```
mkdir my-first-plugin
```

Step 2: Erstellen Sie das Plugin-Manifest

Die Manifestdatei unter `.claude-plugin/plugin.json` definiert die Identität Ihres Plugins: seinen Namen, die Beschreibung und die Version. Claude Code verwendet diese Metadaten, um Ihr Plugin im Plugin-Manager anzuzeigen.

Erstellen Sie das `.claude-plugin`-Verzeichnis in Ihrem Plugin-Ordner:

```
mkdir my-first-plugin/.claude-plugin
```

Erstellen Sie dann `my-first-plugin/.claude-plugin/plugin.json` mit diesem Inhalt:

```
{
  "name": "my-first-plugin",
  "description": "A greeting plugin to learn the basics",
  "version": "1.0.0",
  "author": {
    "name": "Your Name"
  }
}
```

Feld	Zweck
<code>name</code>	Eindeutige Kennung und Skill-Namespace. Skills werden mit diesem Präfix versehen (z. B. <code>/my-first-plugin:hello</code>).
<code>description</code>	Wird im Plugin-Manager angezeigt, wenn Sie Plugins durchsuchen oder installieren.
<code>version</code>	Verfolgen Sie Releases mit semantischer Versionierung .
<code>author</code>	Optional. Hilfreich für die Zuordnung.

Für zusätzliche Felder wie `homepage`, `repository` und `license` siehe das [vollständige Manifest-Schema](#).

Step 3: Fügen Sie einen Skill hinzu

Skills befinden sich im Verzeichnis `skills/`. Jeder Skill ist ein Ordner, der eine Datei `SKILL.md` enthält. Der Ordnername wird zum Skill-Namen, mit dem Präfix des Plugin-Namespace (`hello/` in einem Plugin namens `my-first-plugin` erstellt `/my-first-plugin:hello`).

Erstellen Sie ein Skill-Verzeichnis in Ihrem Plugin-Ordner:

```
mkdir -p my-first-plugin/skills/hello
```

Erstellen Sie dann `my-first-plugin/skills/hello/SKILL.md` mit diesem Inhalt:

```
---
description: Greet the user with a friendly message
disable-model-invocation: true
---

Greet the user warmly and ask how you can help them today.
```

Step 4: Testen Sie Ihr Plugin

Führen Sie Claude Code mit dem Flag `--plugin-dir` aus, um Ihr Plugin zu laden:

```
claude --plugin-dir ./my-first-plugin
```

Sobald Claude Code startet, versuchen Sie Ihren neuen Skill:

```
/my-first-plugin:hello
```

Sie sehen Claude mit einer Begrüßung antworten. Führen Sie `/help` aus, um Ihren Skill unter dem Plugin-Namespace aufgelistet zu sehen.

Note:

Warum Namespacing? Plugin-Skills sind immer mit Namespace versehen (wie `/greet:hello`), um Konflikte zu vermeiden, wenn mehrere Plugins Skills mit demselben Namen haben.

Um das Namespace-Präfix zu ändern, aktualisieren Sie das Feld `name` in `plugin.json`.

Step 5: Fügen Sie Skill-Argumente hinzu

Machen Sie Ihren Skill dynamisch, indem Sie Benutzereingaben akzeptieren. Der Platzhalter `$ARGUMENTS` erfasst jeden Text, den der Benutzer nach dem Skill-Namen bereitstellt.

Aktualisieren Sie Ihre Datei `SKILL.md`:

```
---  
description: Greet the user with a personalized message  
---
```

```
## Hello Skill
```

```
Greet the user named "$ARGUMENTS" warmly and ask how you can help them today. Make  
the greeting personal and encouraging.
```

Führen Sie `/reload-plugins` aus, um die Änderungen zu übernehmen, und versuchen Sie dann den Skill mit Ihrem Namen:

```
/my-first-plugin:hello Alex
```

Claude wird Sie beim Namen begrüßen. Weitere Informationen zum Übergeben von Argumenten an Skills finden Sie unter [Skills](#).

Sie haben erfolgreich ein Plugin mit diesen Schlüsselkomponenten erstellt und getestet:

- **Plugin-Manifest** (`.claude-plugin/plugin.json`): beschreibt die Metadaten Ihres Plugins
- **Skills-Verzeichnis** (`skills/`): enthält Ihre benutzerdefinierten Skills
- **Skill-Argumente** (`$ARGUMENTS`): erfasst Benutzereingaben für dynamisches Verhalten

Tip:

Das Flag `--plugin-dir` ist nützlich für Entwicklung und Tests. Wenn Sie bereit sind, Ihr Plugin mit anderen zu teilen, siehe [Erstellen und verteilen Sie einen Plugin-Marketplace](#).

Übersicht über die Plugin-Struktur

Sie haben ein Plugin mit einem Skill erstellt, aber Plugins können viel mehr enthalten: benutzerdefinierte Agents, Hooks, MCP-Server und LSP-Server.

Warning:

Häufiger Fehler: Platzieren Sie `commands/`, `agents/`, `skills/` oder `hooks/` nicht im Verzeichnis `.claude-plugin/`. Nur `plugin.json` gehört in `.claude-plugin/`. Alle anderen Verzeichnisse müssen auf der Plugin-Root-Ebene sein.

Verzeichnis	Speicherort	Zweck
<code>.claude-plugin/</code>	Plugin-Root	Enthält <code>plugin.json</code> -Manifest (optional, wenn Komponenten Standard-speicherorte verwenden)
<code>commands/</code>	Plugin-Root	Skills als Markdown-Dateien
<code>agents/</code>	Plugin-Root	Benutzerdefinierte Agent-Definitionen
<code>skills/</code>	Plugin-Root	Agent-Skills mit <code>SKILL.md</code> -Dateien
<code>hooks/</code>	Plugin-Root	Event-Handler in <code>hooks.json</code>
<code>.mcp.json</code>	Plugin-Root	MCP-Server-Konfigurationen
<code>.lsp.json</code>	Plugin-Root	LSP-Server-Konfigurationen für Code-Intelligenz
<code>settings.json</code>	Plugin-Root	Standard-Einstellungen, die angewendet werden, wenn das Plugin aktiviert ist

Note:

Nächste Schritte: Bereit, weitere Funktionen hinzuzufügen? Springen Sie zu [Entwickeln Sie komplexere Plugins](#), um Agents, Hooks, MCP-Server und LSP-Server hinzuzufügen. Für vollständige technische Spezifikationen aller Plugin-Komponenten siehe [Plugins-Referenz](#).

Entwickeln Sie komplexere Plugins

Sobald Sie sich mit grundlegenden Plugins vertraut gemacht haben, können Sie anspruchsvollere Erweiterungen erstellen.

Fügen Sie Skills zu Ihrem Plugin hinzu

Plugins können [Agent-Skills](#) enthalten, um die Fähigkeiten von Claude zu erweitern. Skills werden vom Modell aufgerufen: Claude verwendet sie automatisch basierend auf dem Task-Kontext.

Fügen Sie ein Verzeichnis `skills/` auf Ihrer Plugin-Root mit Skill-Ordnern hinzu, die `SKILL.md`-Dateien enthalten:

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json
└── skills/
    └── code-review/
        └── SKILL.md
```

Jede `SKILL.md` benötigt Frontmatter mit den Feldern `name` und `description`, gefolgt von Anweisungen:

```
---
name: code-review
description: Reviews code for best practices and potential issues. Use when
reviewing code, checking PRs, or analyzing code quality.
---

When reviewing code, check for:
1. Code organization and structure
2. Error handling
3. Security concerns
4. Test coverage
```

Nach der Installation des Plugins führen Sie `/reload-plugins` aus, um die Skills zu laden. Für vollständige Anleitung zur Skill-Erstellung, einschließlich progressiver Offenlegung und Tool-Einschränkungen, siehe [Agent-Skills](#).

Fügen Sie LSP-Server zu Ihrem Plugin hinzu

Tip:

Für gängige Sprachen wie TypeScript, Python und Rust installieren Sie die vorgefertigten LSP-Plugins aus dem offiziellen Marketplace. Erstellen Sie benutzerdefinierte LSP-Plugins nur, wenn Sie Unterstützung für Sprachen benötigen, die noch nicht abgedeckt sind.

LSP-Plugins (Language Server Protocol) geben Claude Echtzeit-Code-Intelligenz. Wenn Sie eine Sprache unterstützen müssen, die kein offizielles LSP-Plugin hat, können Sie ein eigenes erstellen, indem Sie eine `.lsp.json`-Datei zu Ihrem Plugin hinzufügen:

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

Benutzer, die Ihr Plugin installieren, müssen die Language-Server-Binärdatei auf ihrem Computer installiert haben.

Für vollständige LSP-Konfigurationsoptionen siehe [LSP-Server](#).

Versenden Sie Standard-Einstellungen mit Ihrem Plugin

Plugins können eine Datei `settings.json` auf der Plugin-Root enthalten, um Standard-Konfiguration anzuwenden, wenn das Plugin aktiviert ist. Derzeit wird nur der Schlüssel `agent` unterstützt.

Das Setzen von `agent` aktiviert einen der [benutzerdefinierten Agents](#) des Plugins als Haupt-Thread und wendet seinen System-Prompt, Tool-Einschränkungen und Modell an. Dies ermöglicht es einem Plugin, das Standardverhalten von Claude Code zu ändern, wenn es aktiviert ist.

```
{
  "agent": "security-reviewer"
}
```

Dieses Beispiel aktiviert den Agent `security-reviewer`, der im Verzeichnis `agents/` des Plugins definiert ist. Einstellungen aus `settings.json` haben Vorrang vor `settings`, die in `plugin.json` deklariert sind. Unbekannte Schlüssel werden stillschweigend ignoriert.

Organisieren Sie komplexe Plugins

Für Plugins mit vielen Komponenten organisieren Sie Ihre Verzeichnisstruktur nach Funktionalität. Für vollständige Verzeichnislayouts und Organisationsmuster siehe [Plugin-Verzeichnisstruktur](#).

Testen Sie Ihre Plugins lokal

Verwenden Sie das Flag `--plugin-dir`, um Plugins während der Entwicklung zu testen. Dies lädt Ihr Plugin direkt, ohne dass eine Installation erforderlich ist.

```
claude --plugin-dir ./my-plugin
```

Wenn Sie Änderungen an Ihrem Plugin vornehmen, führen Sie `/reload-plugins` aus, um die Updates zu übernehmen, ohne neu zu starten. Änderungen an der LSP-Server-Konfiguration erfordern immer noch einen vollständigen Neustart. Testen Sie Ihre Plugin-Komponenten:

- Versuchen Sie Ihre Skills mit `/plugin-name:skill-name`
- Überprüfen Sie, dass Agents in `/agents` angezeigt werden
- Überprüfen Sie, dass Hooks wie erwartet funktionieren

Tip:

Sie können mehrere Plugins gleichzeitig laden, indem Sie das Flag mehrmals angeben:

```
claude --plugin-dir ./plugin-one --plugin-dir ./plugin-two
```

Debuggen Sie Plugin-Probleme

Wenn Ihr Plugin nicht wie erwartet funktioniert:

1. **Überprüfen Sie die Struktur:** Stellen Sie sicher, dass Ihre Verzeichnisse auf der Plugin-Root sind, nicht in `.claude-plugin/`
2. **Testen Sie Komponenten einzeln:** Überprüfen Sie jeden Befehl, Agent und Hook separat
3. **Verwenden Sie Validierungs- und Debugging-Tools:** Siehe [Debugging- und Entwicklungstools](#) für CLI-Befehle und Troubleshooting-Techniken

Teilen Sie Ihre Plugins

Wenn Ihr Plugin bereit zum Teilen ist:

1. **Fügen Sie Dokumentation hinzu:** Fügen Sie eine `README.md` mit Installations- und Verwendungsanweisungen ein
2. **Versionieren Sie Ihr Plugin:** Verwenden Sie [semantische Versionierung](#) in Ihrer `plugin.json`
3. **Erstellen oder verwenden Sie einen Marketplace:** Verteilen Sie über [Plugin-Marketplaces](#) zur Installation
4. **Testen Sie mit anderen:** Lassen Sie Teamkollegen das Plugin vor einer breiteren Verteilung testen

Sobald Ihr Plugin in einem Marketplace ist, können andere es mit den Anweisungen in [Plugins entdecken und installieren](#) installieren.

Reichen Sie Ihr Plugin beim offiziellen Marketplace ein

Um ein Plugin beim offiziellen Anthropic-Marketplace einzureichen, verwenden Sie eines der In-App-Einreichungsformulare:

- **Claude.ai:** claude.ai/settings/plugins/submit
- **Console:** platform.claude.com/plugins/submit

Note:

Für vollständige technische Spezifikationen, Debugging-Techniken und Verteilungsstrategien siehe [Plugins-Referenz](#).

Konvertieren Sie vorhandene Konfigurationen in Plugins

Wenn Sie bereits Skills oder Hooks in Ihrem Verzeichnis `.claude/` haben, können Sie diese in ein Plugin konvertieren, um die Freigabe und Verteilung zu vereinfachen.

Migrationschritte

Step 1: Erstellen Sie die Plugin-Struktur

Erstellen Sie ein neues Plugin-Verzeichnis:

```
mkdir -p my-plugin/.claude-plugin
```

Erstellen Sie die Manifestdatei unter `my-plugin/.claude-plugin/plugin.json` :

```
{
  "name": "my-plugin",
  "description": "Migrated from standalone configuration",
  "version": "1.0.0"
}
```

Step 2: Kopieren Sie Ihre vorhandenen Dateien

Kopieren Sie Ihre vorhandenen Konfigurationen in das Plugin-Verzeichnis:

```
## Copy commands
cp -r .claude/commands my-plugin/

## Copy agents (if any)
cp -r .claude/agents my-plugin/

## Copy skills (if any)
cp -r .claude/skills my-plugin/
```

Step 3: Migrieren Sie Hooks

Wenn Sie Hooks in Ihren Einstellungen haben, erstellen Sie ein Hooks-Verzeichnis:

```
mkdir my-plugin/hooks
```

Erstellen Sie `my-plugin/hooks/hooks.json` mit Ihrer Hooks-Konfiguration. Kopieren Sie das Objekt `hooks` aus Ihrer `.claude/settings.json` oder `settings.local.json`, da das Format gleich ist. Der Befehl empfängt Hook-Eingaben als JSON auf stdin, verwenden Sie also `jq`, um den Dateipfad zu extrahieren:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [{ "type": "command", "command":
          "jq -r '.tool_input.file_path' | xargs npm run lint:fix" }]
      }
    ]
  }
}
```

Step 4: Testen Sie Ihr migriertes Plugin

Laden Sie Ihr Plugin, um zu überprüfen, ob alles funktioniert:

```
claude --plugin-dir ./my-plugin
```

Testen Sie jede Komponente: Führen Sie Ihre Befehle aus, überprüfen Sie, dass Agents in `/agents` angezeigt werden, und überprüfen Sie, dass Hooks korrekt ausgelöst werden.

Was sich bei der Migration ändert

Eigenständig (<code>.claude/</code>)	Plugin
Nur in einem Projekt verfügbar	Kann über Marketplaces freigegeben werden
Dateien in <code>.claude/commands/</code>	Dateien in <code>plugin-name/commands/</code>
Hooks in <code>settings.json</code>	Hooks in <code>hooks/hooks.json</code>
Muss manuell kopiert werden, um zu teilen	Mit <code>/plugin install</code> installieren

Note:

Nach der Migration können Sie die ursprünglichen Dateien aus `.claude/` entfernen, um Duplikate zu vermeiden. Die Plugin-Version hat Vorrang, wenn sie geladen wird.

Nächste Schritte

Jetzt, da Sie das Plugin-System von Claude Code verstehen, finden Sie hier vorgeschlagene Pfade für verschiedene Ziele:

Für Plugin-Benutzer

- [Plugins entdecken und installieren](#): Durchsuchen Sie Marketplaces und installieren Sie Plugins
- [Konfigurieren Sie Team-Marketplaces](#): Richten Sie Repository-Level-Plugins für Ihr Team ein

Für Plugin-Entwickler

- [Erstellen und verteilen Sie einen Marketplace](#): Verpacken und teilen Sie Ihre Plugins
- [Plugins-Referenz](#): Vollständige technische Spezifikationen
- Tauchen Sie tiefer in spezifische Plugin-Komponenten ein:
 - [Skills](#): Details zur Skill-Entwicklung
 - [Subagents](#): Agent-Konfiguration und Fähigkeiten
 - [Hooks](#): Event-Handling und Automatisierung
 - [MCP](#): Integration externer Tools

Plugins-Referenz

Vollständige technische Referenz für das Claude Code Plugin-System, einschließlich Schemas, CLI-Befehle und Komponentenspezifikationen.

Tip:

Möchten Sie Plugins installieren? Siehe [Plugins entdecken und installieren](#). Zum Erstellen von Plugins siehe [Plugins](#). Zum Verteilen von Plugins siehe [Plugin-Marktplätze](#).

Diese Referenz bietet vollständige technische Spezifikationen für das Claude Code Plugin-System, einschließlich Komponentenschemas, CLI-Befehle und Entwicklungstools.

Ein **Plugin** ist ein eigenständiges Verzeichnis von Komponenten, das Claude Code mit benutzerdefinierten Funktionen erweitert. Plugin-Komponenten umfassen Skills, Agents, Hooks, MCP-Server und LSP-Server.

Plugin-Komponenten-Referenz

Skills

Plugins fügen Skills zu Claude Code hinzu und erstellen `/name` Verknüpfungen, die Sie oder Claude aufrufen können.

Speicherort: `skills/` oder `commands/` Verzeichnis im Plugin-Root

Dateiformat: Skills sind Verzeichnisse mit `SKILL.md`; Befehle sind einfache Markdown-Dateien

Skill-Struktur:

```
skills/
├── pdf-processor/
│   ├── SKILL.md
│   ├── reference.md (optional)
│   └── scripts/ (optional)
└── code-reviewer/
    └── SKILL.md
```

Integrationsverhalten:

- Skills und Befehle werden automatisch erkannt, wenn das Plugin installiert wird
- Claude kann sie automatisch basierend auf dem Task-Kontext aufrufen
- Skills können unterstützende Dateien neben SKILL.md enthalten

Vollständige Details finden Sie unter [Skills](#).

Agents

Plugins können spezialisierte Subagents für spezifische Aufgaben bereitstellen, die Claude automatisch aufrufen kann, wenn dies angemessen ist.

Speicherort: `agents/` Verzeichnis im Plugin-Root

Dateiformat: Markdown-Dateien, die Agent-Fähigkeiten beschreiben

Agent-Struktur:

```

---
name: agent-name
description: Worauf sich dieser Agent spezialisiert und wann Claude ihn aufrufen
sollte
---
```

Detailliertes System-Prompt für den Agent, das seine Rolle, Expertise und sein Verhalten beschreibt.

Integrationspunkte:

- Agents erscheinen in der `/agents` Schnittstelle
- Claude kann Agents automatisch basierend auf dem Task-Kontext aufrufen
- Agents können manuell von Benutzern aufgerufen werden
- Plugin-Agents funktionieren neben integrierten Claude-Agents

Vollständige Details finden Sie unter [Subagents](#).

Hooks

Plugins können Event-Handler bereitstellen, die automatisch auf Claude Code Events reagieren.

Speicherort: `hooks/hooks.json` im Plugin-Root oder inline in `plugin.json`

Format: JSON-Konfiguration mit Event-Matchern und Aktionen

Hook-Konfiguration:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format-code.sh"
          }
        ]
      }
    ]
  }
}
```

Verfügbare Events:

- **PreToolUse** : Bevor Claude ein beliebiges Tool verwendet
- **PostToolUse** : Nachdem Claude ein beliebiges Tool erfolgreich verwendet hat
- **PostToolUseFailure** : Nachdem die Tool-Ausführung von Claude fehlgeschlagen ist
- **PermissionRequest** : Wenn ein Berechtigungsdialog angezeigt wird
- **UserPromptSubmit** : Wenn der Benutzer einen Prompt einreicht
- **Notification** : Wenn Claude Code Benachrichtigungen sendet
- **Stop** : Wenn Claude versucht zu stoppen
- **SubagentStart** : Wenn ein Subagent gestartet wird
- **SubagentStop** : Wenn ein Subagent versucht zu stoppen
- **SessionStart** : Am Anfang von Sessions
- **SessionEnd** : Am Ende von Sessions
- **TeammateIdle** : Wenn ein Agent-Team-Teamkollege im Begriff ist, untätig zu werden
- **TaskCompleted** : Wenn eine Aufgabe als abgeschlossen markiert wird
- **PreCompact** : Bevor die Konversationshistorie komprimiert wird

Hook-Typen:

- **command** : Shell-Befehle oder Skripte ausführen
- **prompt** : Ein Prompt mit einem LLM evaluieren (verwendet `$ARGUMENTS` Platzhalter für Kontext)
- **agent** : Einen agentic Verifier mit Tools für komplexe Verifikationsaufgaben ausführen

MCP-Server

Plugins können Model Context Protocol (MCP) Server bündeln, um Claude Code mit externen Tools und Services zu verbinden.

Speicherort: `.mcp.json` im Plugin-Root oder inline in `plugin.json`

Format: Standard MCP-Server-Konfiguration

MCP-Server-Konfiguration:

```
{
  "mcpServers": {
    "plugin-database": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
      "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
      "env": {
        "DB_PATH": "${CLAUDE_PLUGIN_ROOT}/data"
      }
    },
    "plugin-api-client": {
      "command": "npx",
      "args": ["@company/mcp-server", "--plugin-mode"],
      "cwd": "${CLAUDE_PLUGIN_ROOT}"
    }
  }
}
```

Integrationsverhalten:

- Plugin MCP-Server starten automatisch, wenn das Plugin aktiviert wird
- Server erscheinen als Standard MCP-Tools in Claudes Toolkit
- Server-Fähigkeiten integrieren sich nahtlos mit Claudes vorhandenen Tools

- Plugin-Server können unabhängig von Benutzer MCP-Servern konfiguriert werden

LSP-Server

Tip:

Möchten Sie LSP-Plugins verwenden? Installieren Sie sie vom offiziellen Marktplatz: Suchen Sie nach „lsp“ im `/plugin` Discover-Tab. Dieser Abschnitt dokumentiert, wie Sie LSP-Plugins für Sprachen erstellen, die nicht vom offiziellen Marktplatz abgedeckt werden.

Plugins können [Language Server Protocol](#) (LSP) Server bereitstellen, um Claude Echtzeit-Code-Intelligenz beim Arbeiten an Ihrer Codebasis zu geben.

LSP-Integration bietet:

- **Sofortige Diagnose:** Claude sieht Fehler und Warnungen sofort nach jeder Bearbeitung
- **Code-Navigation:** Gehe zu Definition, finde Referenzen und Hover-Informationen
- **Sprachbewusstsein:** Typinformationen und Dokumentation für Code-Symbole

Speicherort: `.lsp.json` im Plugin-Root oder inline in `plugin.json`

Format: JSON-Konfiguration, die Language Server Namen ihren Konfigurationen zuordnet

`.lsp.json` Dateiformat:

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

Inline in `plugin.json`:

```
{
  "name": "my-plugin",
  "lspServers": {
    "go": {
      "command": "gopls",
      "args": ["serve"],
      "extensionToLanguage": {
        ".go": "go"
      }
    }
  }
}
```

Erforderliche Felder:

Feld	Beschreibung
<code>command</code>	Die auszuführende LSP-Binärdatei (muss in PATH sein)
<code>extensionToLanguage</code>	Ordnet Dateierweiterungen Sprachbezeichnern zu

Optionale Felder:

Feld	Beschreibung
<code>args</code>	Befehlszeilenargumente für den LSP-Server
<code>transport</code>	Kommunikationstransport: <code>stdio</code> (Standard) oder <code>socket</code>
<code>env</code>	Umgebungsvariablen, die beim Starten des Servers gesetzt werden
<code>initializationOptions</code>	Optionen, die während der Initialisierung an den Server übergeben werden
<code>settings</code>	Einstellungen, die über <code>workspace/didChangeConfiguration</code> übergeben werden
<code>workspaceFolder</code>	Workspace-Ordnerpfad für den Server
<code>startupTimeout</code>	Maximale Zeit zum Warten auf Server-Startup (Millisekunden)
<code>shutdownTimeout</code>	Maximale Zeit zum Warten auf ordnungsgemäßes Herunterfahren (Millisekunden)

Feld	Beschreibung
<code>restartOnCrash</code>	Ob der Server automatisch neu gestartet werden soll, wenn er abstürzt
<code>maxRestarts</code>	Maximale Anzahl von Neustartversuchen, bevor aufgegeben wird

Warning:

Sie müssen die Language Server Binärdatei separat installieren. LSP-Plugins konfigurieren, wie Claude Code sich mit einem Language Server verbindet, aber sie enthalten den Server selbst nicht. Wenn Sie `Executable not found in $PATH` im `/plugin` Errors-Tab sehen, installieren Sie die erforderliche Binärdatei für Ihre Sprache.

Verfügbare LSP-Plugins:

Plugin	Language Server	Installationsbefehl
<code>pyright-lsp</code>	Pyright (Python)	<code>pip install pyright</code> oder <code>npm install -g pyright</code>
<code>typescript-lsp</code>	TypeScript Language Server	<code>npm install -g typescript-language-server typescript</code>
<code>rust-lsp</code>	rust-analyzer	Siehe rust-analyzer Installation

Installieren Sie zuerst den Language Server, dann installieren Sie das Plugin vom Marktplatz.

Plugin-Installationsbereiche

Wenn Sie ein Plugin installieren, wählen Sie einen **Bereich**, der bestimmt, wo das Plugin verfügbar ist und wer es sonst noch verwenden kann:

Bereich	Einstellungsdatei	Anwendungsfall
<code>user</code>	<code>~/.claude/settings.json</code>	Persönliche Plugins, die in allen Projekten verfügbar sind (Standard)

Bereich	Einstellungsdatei	Anwendungsfall
project	<code>.claude/settings.json</code>	Team-Plugins, die über Versionskontrolle geteilt werden
local	<code>.claude/settings.local.json</code>	Projektspezifische Plugins, gitignoriert
managed	Verwaltete Einstellungen	Verwaltete Plugins (schreibgeschützt, nur aktualisierbar)

Plugins verwenden das gleiche Bereichssystem wie andere Claude Code Konfigurationen. Installationsanweisungen und Bereichs-Flags finden Sie unter [Plugins installieren](#). Eine vollständige Erklärung der Bereiche finden Sie unter [Konfigurationsbereiche](#).

Plugin-Manifest-Schema

Die `.claude-plugin/plugin.json` Datei definiert die Metadaten und Konfiguration Ihres Plugins. Dieser Abschnitt dokumentiert alle unterstützten Felder und Optionen.

Das Manifest ist optional. Wenn es weggelassen wird, erkennt Claude Code Komponenten automatisch in [Standardspeicherorten](#) und leitet den Plugin-Namen aus dem Verzeichnisnamen ab. Verwenden Sie ein Manifest, wenn Sie Metadaten oder benutzerdefinierte Komponentenfade bereitstellen müssen.

Vollständiges Schema

```
{
  "name": "plugin-name",
  "version": "1.2.0",
  "description": "Brief plugin description",
  "author": {
    "name": "Author Name",
    "email": "author@example.com",
    "url": "https://github.com/author"
  },
  "homepage": "https://docs.example.com/plugin",
  "repository": "https://github.com/author/plugin",
  "license": "MIT",
  "keywords": ["keyword1", "keyword2"],
  "commands": ["/custom/commands/special.md"],
  "agents": "/custom/agents/",
  "skills": "/custom/skills/",
  "hooks": "/config/hooks.json",
  "mcpServers": "/mcp-config.json",
  "outputStyles": "/styles/",
  "lspServers": "/.lsp.json"
}
```

Erforderliche Felder

Wenn Sie ein Manifest einschließen, ist `name` das einzige erforderliche Feld.

Feld	Typ	Beschreibung	Beispiel
<code>name</code>	string	Eindeutiger Bezeichner (kebab-case, keine Leerzeichen)	<code>"deployment-tools"</code>

Dieser Name wird für die Namensgebung von Komponenten verwendet. Beispielsweise wird der Agent `agent-creator` für das Plugin mit dem Namen `plugin-dev` in der Benutzeroberfläche als `plugin-dev:agent-creator` angezeigt.

Metadaten-Felder

Feld	Typ	Beschreibung	Beispiel
<code>version</code>	string	Semantische Version. Wenn auch im Marktplatz-Eintrag gesetzt, hat <code>plugin.json</code> Vorrang. Sie müssen es nur an einer Stelle setzen.	<code>"2.1.0"</code>
<code>description</code>	string	Kurze Erklärung des Plugin-Zwecks	<code>"Deployment automation tools"</code>
<code>author</code>	object	Autoreninformationen	<code>{"name": "Dev Team", "email": "dev@company.com"}</code>
<code>homepage</code>	string	Dokumentations-URL	<code>"https://docs.example.com"</code>
<code>repository</code>	string	Quellcode-URL	<code>"https://github.com/user/plugin"</code>
<code>license</code>	string	Lizenzbezeichner	<code>"MIT", "Apache-2.0"</code>
<code>keywords</code>	array	Discovery-Tags	<code>["deployment", "ci-cd"]</code>

Komponentenpfad-Felder

Feld	Typ	Beschreibung	Beispiel
<code>commands</code>	string array	Zusätzliche Befehlsdateien/Verzeichnis	<code>"./custom/cmd.md"</code> oder <code>["./cmd1.md"]</code>
<code>agents</code>	string array	Zusätzliche Agent-Dateien	<code>"./custom/agents/reviewer.md"</code>

Feld	Typ	Beschreibung	Beispiel
<code>skills</code>	string array	Zusätzliche Skill-Verzeichnisse	<code>"/custom/skills/"</code>
<code>hooks</code>	string array object	Hook-Konfigurationspfade oder Inline-Konfiguration	<code>"/my-extra-hooks.json"</code>
<code>mcpServers</code>	string array object	MCP-Konfigurationspfade oder Inline-Konfiguration	<code>"/my-extra-mcp-config.json"</code>
<code>outputStyles</code>	string array	Zusätzliche Output-Style-Dateien/Verzeichnisse	<code>"/styles/"</code>
<code>lspServers</code>	string array object	Language Server Protocol Konfigurationen für Code-Intelligenz (Gehe zu Definition, finde Referenzen, etc.)	<code>"/.lsp.json"</code>

Pfad-Verhaltensregeln

Wichtig: Benutzerdefinierte Pfade ergänzen Standardverzeichnisse - sie ersetzen sie nicht.

- Wenn `commands/` existiert, wird es zusätzlich zu benutzerdefinierten Befehlspfaden geladen
- Alle Pfade müssen relativ zum Plugin-Root sein und mit `./` beginnen
- Befehle aus benutzerdefinierten Pfaden verwenden die gleichen Benennungs- und Namensgebungsregeln
- Mehrere Pfade können als Arrays für Flexibilität angegeben werden

Pfad-Beispiele:

```
{
  "commands": [
    "./specialized/deploy.md",
    "./utilities/batch-process.md"
  ],
  "agents": [
    "./custom-agents/reviewer.md",
    "./custom-agents/tester.md"
  ]
}
```

Umgebungsvariablen

`${CLAUDE_PLUGIN_ROOT}` : Enthält den absoluten Pfad zu Ihrem Plugin-Verzeichnis. Verwenden Sie dies in Hooks, MCP-Servern und Skripten, um korrekte Pfade unabhängig vom Installationsort sicherzustellen.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/process.sh"
          }
        ]
      }
    ]
  }
}
```

Plugin-Caching und Dateiauflösung

Plugins werden auf eine von zwei Arten angegeben:

- Durch `claude --plugin-dir`, für die Dauer einer Session.
- Durch einen Marktplatz, installiert für zukünftige Sessions.

Aus Sicherheits- und Verifizierungsgründen kopiert Claude Code *Marktplatz*-Plugins in den lokalen **Plugin-Cache** des Benutzers (`~/.claude/plugins/cache`), anstatt sie an Ort und Stelle zu verwenden. Das Verständnis dieses Verhaltens ist wichtig, wenn Sie Plugins entwickeln, die auf externe Dateien verweisen.

Pfad-Traversal-Einschränkungen

Installierte Plugins können nicht auf Dateien außerhalb ihres Verzeichnisses verweisen. Pfade, die außerhalb des Plugin-Root traversieren (wie `../shared-utils`), funktionieren nach der Installation nicht, da diese externen Dateien nicht in den Cache kopiert werden.

Arbeiten mit externen Abhängigkeiten

Wenn Ihr Plugin auf Dateien außerhalb seines Verzeichnisses zugreifen muss, können Sie symbolische Links zu externen Dateien in Ihrem Plugin-Verzeichnis erstellen. Symlinks werden während des Kopiervorgangs berücksichtigt:

```
## Innerhalb Ihres Plugin-Verzeichnisses
ln -s /path/to/shared-utils ./shared-utils
```

Der verlinkte Inhalt wird in den Plugin-Cache kopiert. Dies bietet Flexibilität bei Beibehaltung der Sicherheitsvorteile des Caching-Systems.

Plugin-Verzeichnisstruktur

Standard-Plugin-Layout

Ein vollständiges Plugin folgt dieser Struktur:

```

enterprise-plugin/
├── .claude-plugin/          # Metadaten-Verzeichnis (optional)
│   ├── plugin.json         # Plugin-Manifest
│   └── commands/           # Standard-Befehlsspeicherort
│       ├── status.md
│       └── logs.md
├── agents/                  # Standard-Agent-Speicherort
│   ├── security-reviewer.md
│   ├── performance-tester.md
│   └── compliance-checker.md
├── skills/                  # Agent Skills
│   ├── code-reviewer/
│   │   └── SKILL.md
│   └── pdf-processor/
│       ├── SKILL.md
│       └── scripts/
├── hooks/                   # Hook-Konfigurationen
│   ├── hooks.json          # Haupt-Hook-Konfiguration
│   └── security-hooks.json  # Zusätzliche Hooks
├── settings.json            # Standardeinstellungen für das Plugin
├── .mcp.json                # MCP-Server-Definitionen
├── .lsp.json                # LSP-Server-Konfigurationen
├── scripts/                 # Hook- und Utility-Skripte
│   ├── security-scan.sh
│   ├── format-code.py
│   └── deploy.js
├── LICENSE                  # Lizenzdatei
└── CHANGELOG.md             # Versionsverlauf

```

Warning:

Das `.claude-plugin/` Verzeichnis enthält die `plugin.json` Datei. Alle anderen Verzeichnisse (`commands/`, `agents/`, `skills/`, `hooks/`) müssen sich im Plugin-Root befinden, nicht innerhalb von `.claude-plugin/`.

Datei-Speicherorte-Referenz

Komponente	Standard-Speicherort	Zweck
Manifest	<code>.claude-plugin/plugin.json</code>	Plugin-Metadaten und Konfiguration (optional)
Befehle	<code>commands/</code>	Skill Markdown-Dateien (veraltet; verwenden Sie <code>skills/</code> für neue Skills)
Agents	<code>agents/</code>	Subagent Markdown-Dateien
Skills	<code>skills/</code>	Skills mit <code><name>/SKILL.md</code> Struktur
Hooks	<code>hooks/hooks.json</code>	Hook-Konfiguration
MCP-Server	<code>.mcp.json</code>	MCP-Server-Definitionen
LSP-Server	<code>.lsp.json</code>	Language Server Konfigurationen
Einstellungen	<code>settings.json</code>	Standardkonfiguration, die angewendet wird, wenn das Plugin aktiviert wird. Derzeit werden nur <code>agent</code> Einstellungen unterstützt

CLI-Befehle-Referenz

Claude Code bietet CLI-Befehle für nicht-interaktive Plugin-Verwaltung, nützlich für Scripting und Automatisierung.

plugin install

Installieren Sie ein Plugin aus verfügbaren Marktplätzen.

```
claude plugin install <plugin> [options]
```

Argumente:

- `<plugin>` : Plugin-Name oder `plugin-name@marketplace-name` für einen bestimmten Marktplatz

Optionen:

Option	Beschreibung	Standard
<code>-s, --scope <scope></code>	Installationsbereich: <code>user</code> , <code>project</code> oder <code>local</code>	<code>user</code>
<code>-h, --help</code>	Hilfe für Befehl anzeigen	

Der Bereich bestimmt, welche Einstellungsdatei das installierte Plugin hinzugefügt wird. Beispielsweise schreibt `--scope project` zu `enabledPlugins` in `.claude/settings.json`, wodurch das Plugin für alle verfügbar wird, die das Projekt-Repository klonen.

Beispiele:

```
## Installieren Sie im Benutzerbereich (Standard)
claude plugin install formatter@my-marketplace

## Installieren Sie im Projektbereich (geteilt mit Team)
claude plugin install formatter@my-marketplace --scope project

## Installieren Sie im lokalen Bereich (gitignoriert)
claude plugin install formatter@my-marketplace --scope local
```

plugin uninstall

Entfernen Sie ein installiertes Plugin.

```
claude plugin uninstall <plugin> [options]
```

Argumente:

- `<plugin>` : Plugin-Name oder `plugin-name@marketplace-name`

Optionen:

Option	Beschreibung	Standard
<code>-s, --scope <scope></code>	Deinstallieren aus Bereich: <code>user</code> , <code>project</code> oder <code>local</code>	<code>user</code>
<code>-h, --help</code>	Hilfe für Befehl anzeigen	

Aliase: `remove`, `rm`

plugin enable

Aktivieren Sie ein deaktiviertes Plugin.

```
claude plugin enable <plugin> [options]
```

Argumente:

- `<plugin>` : Plugin-Name oder `plugin-name@marketplace-name`

Optionen:

Option	Beschreibung	Standard
<code>-s, --scope <scope></code>	Bereich zum Aktivieren: <code>user</code> , <code>project</code> oder <code>local</code>	<code>user</code>
<code>-h, --help</code>	Hilfe für Befehl anzeigen	

plugin disable

Deaktivieren Sie ein Plugin, ohne es zu deinstallieren.

```
claude plugin disable <plugin> [options]
```

Argumente:

- `<plugin>` : Plugin-Name oder `plugin-name@marketplace-name`

Optionen:

Option	Beschreibung	Standard
<code>-s, --scope <scope></code>	Bereich zum Deaktivieren: <code>user</code> , <code>project</code> oder <code>local</code>	<code>user</code>

Option	Beschreibung	Standard
<code>-h, --help</code>	Hilfe für Befehl anzeigen	

plugin update

Aktualisieren Sie ein Plugin auf die neueste Version.

```
claude plugin update <plugin> [options]
```

Argumente:

- `<plugin>` : Plugin-Name oder `plugin-name@marketplace-name`

Optionen:

Option	Beschreibung	Standard
<code>-s, --scope <scope></code>	Bereich zum Aktualisieren: <code>user</code> , <code>project</code> , <code>local</code> oder <code>managed</code>	<code>user</code>
<code>-h, --help</code>	Hilfe für Befehl anzeigen	

Debugging- und Entwicklungstools

Debugging-Befehle

Verwenden Sie `claude --debug` (oder `/debug` innerhalb der TUI), um Plugin-Lade-Details zu sehen:

Dies zeigt:

- Welche Plugins geladen werden
- Alle Fehler in Plugin-Manifesten
- Befehls-, Agent- und Hook-Registrierung
- MCP-Server-Initialisierung

Häufige Probleme

Problem	Ursache	Lösung
Plugin wird nicht geladen	Ungültige <code>plugin.json</code>	Validieren Sie JSON-Syntax mit <code>claude plugin validate</code> oder <code>/plugin validate</code>
Befehle erscheinen nicht	Falsche Verzeichnisstruktur	Stellen Sie sicher, dass <code>commands/</code> im Root ist, nicht in <code>.claude-plugin/</code>
Hooks werden nicht ausgelöst	Skript nicht ausführbar	Führen Sie <code>chmod +x script.sh</code> aus
MCP-Server schlägt fehl	Fehlender <code>\$ {CLAUDE_PLUGIN_ROOT}</code>	Verwenden Sie Variable für alle Plugin-Pfade
Pfadfehler	Absolute Pfade verwendet	Alle Pfade müssen relativ sein und mit <code>./</code> beginnen
LSP <code>Executable not found in \$PATH</code>	Language Server nicht installiert	Installieren Sie die Binärdatei (z.B. <code>npm install -g typescript-language-server typescript</code>)

Beispiel-Fehlermeldungen

Manifest-Validierungsfehler:

- `Invalid JSON syntax: Unexpected token } in JSON at position 142` : Überprüfen Sie auf fehlende Kommas, zusätzliche Kommas oder nicht zitierte Strings
- `Plugin has an invalid manifest file at .claude-plugin/plugin.json. Validation errors: name: Required` : Ein erforderliches Feld fehlt
- `Plugin has a corrupt manifest file at .claude-plugin/plugin.json. JSON parse error: ...` : JSON-Syntaxfehler

Plugin-Ladefehler:

- `Warning: No commands found in plugin my-plugin custom directory: ./cmds. Expected .md files or SKILL.md in subdirectories.` : Befehlspfad existiert, enthält aber keine gültigen Befehlsdateien

- **Plugin directory not found at path: ./plugins/my-plugin.** Check that the **marketplace entry has the correct path.** : Der **source** Pfad in marketplace.json verweist auf ein nicht existierendes Verzeichnis
- **Plugin my-plugin has conflicting manifests: both plugin.json and marketplace entry specify components.** : Entfernen Sie doppelte Komponentendefinitionen oder entfernen Sie **strict: false** im Marktplatz-Eintrag

Hook-Fehlerbehebung

Hook-Skript wird nicht ausgeführt:

1. Überprüfen Sie, dass das Skript ausführbar ist: `chmod +x ./scripts/your-script.sh`
2. Überprüfen Sie die Shebang-Zeile: Erste Zeile sollte `#!/bin/bash` oder `#!/usr/bin/env bash` sein
3. Überprüfen Sie, dass der Pfad `${CLAUDE_PLUGIN_ROOT}` verwendet: `"command": "${CLAUDE_PLUGIN_ROOT}/scripts/your-script.sh"`
4. Testen Sie das Skript manuell: `./scripts/your-script.sh`

Hook wird nicht bei erwarteten Events ausgelöst:

1. Überprüfen Sie, dass der Event-Name korrekt ist (Groß-/Kleinschreibung beachten): `PostToolUse`, nicht `postToolUse`
2. Überprüfen Sie, dass das Matcher-Muster Ihre Tools passt: `"matcher": "Write|Edit"` für Dateivorgänge
3. Bestätigen Sie, dass der Hook-Typ gültig ist: `command`, `prompt` oder `agent`

MCP-Server-Fehlerbehebung

Server wird nicht gestartet:

1. Überprüfen Sie, dass der Befehl existiert und ausführbar ist
2. Überprüfen Sie, dass alle Pfade die `${CLAUDE_PLUGIN_ROOT}` Variable verwenden
3. Überprüfen Sie die MCP-Server-Logs: `claude --debug` zeigt Initialisierungsfehler
4. Testen Sie den Server manuell außerhalb von Claude Code

Server-Tools erscheinen nicht:

1. Stellen Sie sicher, dass der Server ordnungsgemäß in `.mcp.json` oder `plugin.json` konfiguriert ist
2. Überprüfen Sie, dass der Server das MCP-Protokoll ordnungsgemäß implementiert
3. Überprüfen Sie auf Verbindungs-Timeouts in der Debug-Ausgabe

Verzeichnisstruktur-Fehler

Symptome: Plugin wird geladen, aber Komponenten (Befehle, Agents, Hooks) fehlen.

Korrekte Struktur: Komponenten müssen sich im Plugin-Root befinden, nicht innerhalb von `.claude-plugin/`. Nur `plugin.json` gehört in `.claude-plugin/`.

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json      ← Nur Manifest hier
├── commands/           ← Auf Root-Ebene
├── agents/             ← Auf Root-Ebene
└── hooks/              ← Auf Root-Ebene
```

Wenn sich Ihre Komponenten in `.claude-plugin/` befinden, verschieben Sie sie in den Plugin-Root.

Debug-Checkliste:

1. Führen Sie `claude --debug` aus und suchen Sie nach „loading plugin“ Meldungen
2. Überprüfen Sie, dass jedes Komponentenverzeichnis in der Debug-Ausgabe aufgelistet ist
3. Überprüfen Sie Dateiberechtigungen, die das Lesen der Plugin-Dateien ermöglichen

Verteilungs- und Versionierungs-Referenz

Versionsverwaltung

Folgen Sie semantischer Versionierung für Plugin-Releases:

```
{
  "name": "my-plugin",
  "version": "2.1.0"
}
```

Versionsformat: `MAJOR.MINOR.PATCH`

- **MAJOR:** Breaking Changes (inkompatible API-Änderungen)
- **MINOR:** Neue Features (abwärtskompatible Ergänzungen)
- **PATCH:** Bugfixes (abwärtskompatible Fixes)

Best Practices:

- Beginnen Sie mit `1.0.0` für Ihr erstes stabiles Release
- Aktualisieren Sie die Version in `plugin.json`, bevor Sie Änderungen verteilen
- Dokumentieren Sie Änderungen in einer `CHANGELOG.md` Datei
- Verwenden Sie Pre-Release-Versionen wie `2.0.0-beta.1` zum Testen

Warning:

Claude Code verwendet die Version, um zu bestimmen, ob Ihr Plugin aktualisiert werden soll. Wenn Sie den Code Ihres Plugins ändern, aber die Version in `plugin.json` nicht erhöhen, werden Ihre bestehenden Plugin-Benutzer Ihre Änderungen aufgrund von Caching nicht sehen.

Wenn sich Ihr Plugin in einem [Marktplatz](#) Verzeichnis befindet, können Sie die Version stattdessen über `marketplace.json` verwalten und das `version` Feld aus `plugin.json` weglassen.

Siehe auch

- [Plugins](#) - Tutorials und praktische Verwendung
- [Plugin-Marktplätze](#) - Erstellen und Verwalten von Marktplätzen
- [Skills](#) - Skill-Entwicklungsdetails
- [Subagents](#) - Agent-Konfiguration und Fähigkeiten
- [Hooks](#) - Event-Handling und Automatisierung
- [MCP](#) - Integration externer Tools
- [Einstellungen](#) - Konfigurationsoptionen für Plugins

Erstellen und Verteilen eines Plugin-Marktplatzes

Erstellen und hosten Sie Plugin-Marktplätze, um Claude Code-Erweiterungen in Teams und Communities zu verteilen.

Ein **Plugin-Marktplatz** ist ein Katalog, mit dem Sie Plugins an andere verteilen können. Marktplätze bieten zentrale Entdeckung, Versionsverfolgung, automatische Updates und Unterstützung für mehrere Quellentypen (Git-Repositories, lokale Pfade und mehr). Diese Anleitung zeigt Ihnen, wie Sie Ihren eigenen Marktplatz erstellen, um Plugins mit Ihrem Team oder Ihrer Community zu teilen.

Möchten Sie Plugins aus einem vorhandenen Marktplatz installieren? Siehe [Entdecken und Installieren vorgefertigter Plugins](#).

Übersicht

Das Erstellen und Verteilen eines Marktplatzes umfasst:

1. **Plugins erstellen:** Erstellen Sie ein oder mehrere Plugins mit Befehlen, Agents, hooks, MCP servers oder LSP servers. Diese Anleitung setzt voraus, dass Sie bereits Plugins zum Verteilen haben; siehe [Plugins erstellen](#) für Details zum Erstellen von Plugins.
2. **Marktplatzdatei erstellen:** Definieren Sie eine `marketplace.json`, die Ihre Plugins und deren Speicherorte auflistet (siehe [Marktplatzdatei erstellen](#)).
3. **Marktplatz hosten:** Pushen Sie zu GitHub, GitLab oder einem anderen Git-Host (siehe [Marktplätze hosten und verteilen](#)).
4. **Mit Benutzern teilen:** Benutzer fügen Ihren Marktplatz mit `/plugin marketplace add` hinzu und installieren einzelne Plugins (siehe [Plugins entdecken und installieren](#)).

Sobald Ihr Marktplatz live ist, können Sie ihn aktualisieren, indem Sie Änderungen in Ihr Repository pushen. Benutzer aktualisieren ihre lokale Kopie mit `/plugin marketplace update`.

Anleitung: Erstellen Sie einen lokalen Marktplatz

Dieses Beispiel erstellt einen Marktplatz mit einem Plugin: ein `/quality-review` skill für Code-Reviews. Sie erstellen die Verzeichnisstruktur, fügen ein skill hinzu, erstellen das Plugin-Manifest und den Marktplatzkatalog und installieren und testen ihn dann.

Step 1: Erstellen Sie die Verzeichnisstruktur

```
mkdir -p my-marketplace/.claude-plugin
mkdir -p my-marketplace/plugins/quality-review-plugin/.claude-plugin
mkdir -p my-marketplace/plugins/quality-review-plugin/skills/quality-review
```

Step 2: Erstellen Sie das skill

Erstellen Sie eine `SKILL.md`-Datei, die definiert, was das `/quality-review` skill tut.

```
---
description: Review code for bugs, security, and performance
disable-model-invocation: true
---
```

Review the code I've selected or the recent changes for:

- Potential bugs or edge cases
- Security concerns
- Performance issues
- Readability improvements

Be concise and actionable.

Step 3: Erstellen Sie das Plugin-Manifest

Erstellen Sie eine `plugin.json`-Datei, die das Plugin beschreibt. Das Manifest befindet sich im `.claude-plugin/`-Verzeichnis.

```
{  
  "name": "quality-review-plugin",  
  "description": "Adds a /quality-review skill for quick code reviews",  
  "version": "1.0.0"  
}
```

Step 4: Erstellen Sie die Marktplatzdatei

Erstellen Sie den Marktplatzkatalog, der Ihr Plugin auflistet.

```
{  
  "name": "my-plugins",  
  "owner": {  
    "name": "Your Name"  
  },  
  "plugins": [  
    {  
      "name": "quality-review-plugin",  
      "source": "./plugins/quality-review-plugin",  
      "description": "Adds a /quality-review skill for quick code reviews"  
    }  
  ]  
}
```

Step 5: Hinzufügen und Installieren

Fügen Sie den Marktplatz hinzu und installieren Sie das Plugin.

```
/plugin marketplace add ./my-marketplace  
/plugin install quality-review-plugin@my-plugins
```

Step 6: Probieren Sie es aus

Wählen Sie etwas Code in Ihrem Editor aus und führen Sie Ihren neuen Befehl aus.

```
/review
```

Um mehr über die Möglichkeiten von Plugins zu erfahren, einschließlich hooks, Agents, MCP servers und LSP servers, siehe [Plugins](#).

Note:

Wie Plugins installiert werden: Wenn Benutzer ein Plugin installieren, kopiert Claude Code das Plugin-Verzeichnis an einen Cache-Speicherort. Das bedeutet, dass Plugins keine Dateien außerhalb ihres Verzeichnisses mit Pfaden wie `../shared-utils` referenzieren können, da diese Dateien nicht kopiert werden.

Wenn Sie Dateien über Plugins hinweg teilen müssen, verwenden Sie Symlinks (die während des Kopierens verfolgt werden). Siehe [Plugin-Caching und Dateiauflösung](#) für Details.

Marktplatzdatei erstellen

Erstellen Sie `.claude-plugin/marketplace.json` im Stammverzeichnis Ihres Repositories. Diese Datei definiert den Namen Ihres Marktplatzes, Eigentümerinformationen und eine Liste von Plugins mit ihren Quellen.

Jeder Plugin-Eintrag benötigt mindestens einen `name` und eine `source` (wo man ihn abrufen kann). Siehe das [vollständige Schema](#) unten für alle verfügbaren Felder.


```
{
  "name": "company-tools",
  "owner": {
    "name": "DevTools Team",
    "email": "devtools@example.com"
  },
  "plugins": [
    {
      "name": "code-formatter",
      "source": "./plugins/formatter",
      "description": "Automatic code formatting on save",
      "version": "2.1.0",
      "author": {
        "name": "DevTools Team"
      }
    },
    {
      "name": "deployment-tools",
      "source": {
        "source": "github",
        "repo": "company/deploy-plugin"
      },
      "description": "Deployment automation tools"
    }
  ]
}
```

Marktplatz-Schema

Erforderliche Felder

Feld	Typ	Beschreibung	Beispiel
<code>name</code>	string	Marktplatz-Identifizier (Kebab-Case, keine Leerzeichen). Dies ist öffentlich sichtbar: Benutzer sehen es beim Installieren von Plugins (z. B. <code>/plugin install my-tool@your-marketplace</code>).	<code>"acme-tools"</code>
<code>owner</code>	object	Informationen zum Marktplatz-Betreuer (siehe Felder unten)	
<code>plugins</code>	array	Liste der verfügbaren Plugins	Siehe unten

Note:

Reservierte Namen: Die folgenden Marktplatznamen sind für die offizielle Nutzung durch Anthropic reserviert und können nicht von Drittanbieter-Marktplätzen verwendet werden: `claude-code-marketplace`, `claude-code-plugins`, `claude-plugins-official`, `anthropic-marketplace`, `anthropic-plugins`, `agent-skills`, `life-sciences`. Namen, die offizielle Marktplätze imitieren (wie `official-claude-plugins` oder `anthropic-tools-v2`), sind ebenfalls blockiert.

Eigentümer-Felder

Feld	Typ	Erforderlich	Beschreibung
<code>name</code>	string	Ja	Name des Betreuers oder Teams
<code>email</code>	string	Nein	Kontakt-E-Mail für den Betreuer

Optionale Metadaten

Feld	Typ	Beschreibung
<code>metadata.description</code>	string	Kurze Marktplatzbeschreibung
<code>metadata.version</code>	string	Marktplatz-Version
<code>metadata.pluginRoot</code>	string	Basisverzeichnis, das relativen Plugin-Quellpfaden vorangestellt wird (z. B. <code>./plugins</code> ermöglicht es Ihnen, <code>"source": "formatter"</code> statt <code>"source": "./plugins/formatter"</code> zu schreiben)

Plugin-Einträge

Jeder Plugin-Eintrag im `plugins`-Array beschreibt ein Plugin und wo man es findet. Sie können jedes Feld aus dem [Plugin-Manifest-Schema](#) einbeziehen (wie `description`, `version`, `author`, `commands`, `hooks` usw.), plus diese Marktplatz-spezifischen Felder: `source`, `category`, `tags` und `strict`.

Erforderliche Felder

Feld	Typ	Beschreibung
<code>name</code>	string	Plugin-Identifizier (Kebab-Case, keine Leerzeichen). Dies ist öffentlich sichtbar: Benutzer sehen es beim Installieren (z. B. <code>/plugin install my-plugin@marketplace</code>).
<code>source</code>	string object	Wo das Plugin abgerufen werden soll (siehe Plugin-Quellen unten)

Optionale Plugin-Felder

Standard-Metadatenfelder:

Feld	Typ	Beschreibung
<code>description</code>	string	Kurze Plugin-Beschreibung
<code>version</code>	string	Plugin-Version
<code>author</code>	object	Plugin-Autoreninformationen (<code>name</code> erforderlich, <code>email</code> optional)
<code>homepage</code>	string	Plugin-Homepage oder Dokumentations-URL
<code>repository</code>	string	Quellcode-Repository-URL
<code>license</code>	string	SPDX-Lizenz-Identifizier (z. B. MIT, Apache-2.0)
<code>keywords</code>	array	Tags für Plugin-Entdeckung und Kategorisierung
<code>category</code>	string	Plugin-Kategorie zur Organisation
<code>tags</code>	array	Tags für Suchbarkeit
<code>strict</code>	boolean	Steuert, ob <code>plugin.json</code> die Autorität für Komponentendefinitionen ist (Standard: true). Siehe Strict Mode unten.

Komponenten-Konfigurationsfelder:

Feld	Typ	Beschreibung
<code>commands</code>	string array	Benutzerdefinierte Pfade zu Befehlsdateien oder -verzeichnissen
<code>agents</code>	string array	Benutzerdefinierte Pfade zu Agent-Dateien

Feld	Typ	Beschreibung
<code>hooks</code>	string object	Benutzerdefinierte hooks-Konfiguration oder Pfad zu hooks-Datei
<code>mcpServers</code>	string object	MCP server-Konfigurationen oder Pfad zu MCP-Konfiguration
<code>lspServers</code>	string object	LSP server-Konfigurationen oder Pfad zu LSP-Konfiguration

Plugin-Quellen

Plugin-Quellen teilen Claude Code mit, wo jedes einzelne Plugin in Ihrem Marktplatz abgerufen werden soll. Diese werden im `source`-Feld jedes Plugin-Eintrags in `marketplace.json` festgelegt.

Sobald ein Plugin geklont oder auf den lokalen Computer kopiert wird, wird es in den lokalen versionierten Plugin-Cache unter `~/.claude/plugins/cache` kopiert.

Quelle	Typ	Felder	Notizen
Relativer Pfad	<code>string</code> (z. B. <code>"/my-plugin"</code>)	—	Lokales Verzeichnis im Marktplatz-Repo. Muss mit <code>./</code> beginnen
<code>github</code>	object	<code>repo</code> , <code>ref?</code> , <code>sha?</code>	
<code>url</code>	object	<code>url</code> (muss mit <code>.git</code> enden), <code>ref?</code> , <code>sha?</code>	Git-URL-Quelle
<code>git-subdir</code>	object	<code>url</code> , <code>path</code> , <code>ref?</code> , <code>sha?</code>	Unterverzeichnis in einem Git-Repo. Klont sparsam, um die Bandbreite für Monorepos zu minimieren

Quelle	Typ	Felder	Notizen
<code>npm</code>	object	<code>package</code> , <code>version?</code> , <code>registry?</code>	Installiert über <code>npm install</code>
<code>pip</code>	object	<code>package</code> , <code>version?</code> , <code>registry?</code>	Installiert über pip

Note:

Marktplatz-Quellen vs. Plugin-Quellen: Dies sind unterschiedliche Konzepte, die unterschiedliche Dinge steuern.

- **Marktplatz-Quelle** — wo der `marketplace.json` -Katalog selbst abgerufen werden soll. Wird festgelegt, wenn Benutzer `/plugin marketplace add` ausführen oder in `extraKnownMarketplaces` -Einstellungen. Unterstützt `ref` (Branch/Tag), aber nicht `sha`.
- **Plugin-Quelle** — wo ein einzelnes Plugin in der Marktplatz-Liste abgerufen werden soll. Wird im `source` -Feld jedes Plugin-Eintrags in `marketplace.json` festgelegt. Unterstützt sowohl `ref` (Branch/Tag) als auch `sha` (exakter Commit).

Beispielsweise kann ein Marktplatz, der unter `acme-corp/plugin-catalog` gehostet wird (Marktplatz-Quelle), ein Plugin auflisten, das von `acme-corp/code-formatter` abgerufen wird (Plugin-Quelle). Die Marktplatz-Quelle und die Plugin-Quelle verweisen auf unterschiedliche Repositories und werden unabhängig voneinander angeheftet.

Relative Pfade

Für Plugins im selben Repository:

```
{
  "name": "my-plugin",
  "source": "../plugins/my-plugin"
}
```

Note:

Relative Pfade funktionieren nur, wenn Benutzer Ihren Marktplatz über Git hinzufügen (GitHub, GitLab oder Git-URL). Wenn Benutzer Ihren Marktplatz über eine direkte URL zur `marketplace.json`-Datei hinzufügen, werden relative Pfade nicht korrekt aufgelöst. Verwenden Sie für URL-basierte Verteilung stattdessen GitHub-, npm- oder Git-URL-Quellen. Siehe [Fehlerbehebung](#) für Details.

GitHub-Repositories

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo"
  }
}
```

Sie können an einen bestimmten Branch, Tag oder Commit anheften:

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

Feld	Typ	Beschreibung
<code>repo</code>	string	Erforderlich. GitHub-Repository im Format <code>owner/repo</code>
<code>ref</code>	string	Optional. Git-Branch oder Tag (Standard: Standard-Branch des Repositories)

Feld	Typ	Beschreibung
<code>sha</code>	string	Optional. Vollständiger 40-stelliger Git-Commit-SHA zum Anheften an eine exakte Version

Git-Repositories

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git"
  }
}
```

Sie können an einen bestimmten Branch, Tag oder Commit anheften:

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git",
    "ref": "main",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

Feld	Typ	Beschreibung
<code>url</code>	string	Erforderlich. Vollständige Git-Repository-URL (muss mit <code>.git</code> enden)
<code>ref</code>	string	Optional. Git-Branch oder Tag (Standard: Standard-Branch des Repositories)

Feld	Typ	Beschreibung
<code>sha</code>	string	Optional. Vollständiger 40-stelliger Git-Commit-SHA zum Anheften an eine exakte Version

Git-Unterverzeichnisse

Verwenden Sie `git-subdir`, um auf ein Plugin zu verweisen, das sich in einem Unterverzeichnis eines Git-Repositories befindet. Claude Code verwendet einen sparsamen, teilweisen Klon, um nur das Unterverzeichnis abzurufen und die Bandbreite für große Monorepos zu minimieren.

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin"
  }
}
```

Sie können an einen bestimmten Branch, Tag oder Commit anheften:

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

Das `url`-Feld akzeptiert auch eine GitHub-Kurzform (`owner/repo`) oder SSH-URLs (`git@github.com:owner/repo.git`).

Feld	Typ	Beschreibung
<code>url</code>	string	Erforderlich. Git-Repository-URL, GitHub <code>owner/repo</code> -Kurzform oder SSH-URL
<code>path</code>	string	Erforderlich. Unterverzeichnispfad im Repo, das das Plugin enthält (z. B. <code>"tools/claude-plugin"</code>)
<code>ref</code>	string	Optional. Git-Branch oder Tag (Standard: Standard-Branch des Repositories)
<code>sha</code>	string	Optional. Vollständiger 40-stelliger Git-Commit-SHA zum Anheften an eine exakte Version

npm-Pakete

Plugins, die als npm-Pakete verteilt werden, werden mit `npm install` installiert. Dies funktioniert mit jedem Paket in der öffentlichen npm-Registry oder einer privaten Registry, die Ihr Team hostet.

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin"
  }
}
```

Um an eine bestimmte Version anzuhängen, fügen Sie das `version`-Feld hinzu:

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "2.1.0"
  }
}
```

Um von einer privaten oder internen Registry zu installieren, fügen Sie das `registry`-Feld hinzu:

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "^2.0.0",
    "registry": "https://npm.example.com"
  }
}
```

Feld	Typ	Beschreibung
<code>package</code>	string	Erforderlich. Paketname oder Scoped-Paket (z. B. <code>@org/plugin</code>)
<code>version</code>	string	Optional. Version oder Versionsspanne (z. B. <code>2.1.0</code> , <code>^2.0.0</code> , <code>~1.5.0</code>)
<code>registry</code>	string	Optional. Benutzerdefinierte npm-Registry-URL. Standard ist die System-npm-Registry (normalerweise <code>npmjs.org</code>)

Erweiterte Plugin-Einträge

Dieses Beispiel zeigt einen Plugin-Eintrag mit vielen optionalen Feldern, einschließlich benutzerdefinierter Pfade für Befehle, Agents, hooks und MCP servers:

```

{
  "name": "enterprise-tools",
  "source": {
    "source": "github",
    "repo": "company/enterprise-plugin"
  },
  "description": "Enterprise workflow automation tools",
  "version": "2.1.0",
  "author": {
    "name": "Enterprise Team",
    "email": "enterprise@example.com"
  },
  "homepage": "https://docs.example.com/plugins/enterprise-tools",
  "repository": "https://github.com/company/enterprise-plugin",
  "license": "MIT",
  "keywords": ["enterprise", "workflow", "automation"],
  "category": "productivity",
  "commands": [
    "./commands/core/",
    "./commands/enterprise/",
    "./commands/experimental/preview.md"
  ],
  "agents": ["/agents/security-reviewer.md", "/agents/compliance-checker.md"],
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/validate.sh"
          }
        ]
      }
    ]
  },
  "mcpServers": {
    "enterprise-db": {

```

```
    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"]
  }
},
"strict": false
}
```

Wichtige Dinge zu beachten:

- **commands** und **agents** : Sie können mehrere Verzeichnisse oder einzelne Dateien angeben. Pfade sind relativ zum Plugin-Root.
- **\${CLAUDE_PLUGIN_ROOT}** : Verwenden Sie diese Variable in hooks und MCP server-Konfigurationen, um auf Dateien im Installationsverzeichnis des Plugins zu verweisen. Dies ist notwendig, da Plugins beim Installieren an einen Cache-Speicherort kopiert werden.
- **strict: false** : Da dies auf false gesetzt ist, benötigt das Plugin keine eigene **plugin.json** . Der Marktplatz-Eintrag definiert alles. Siehe [Strict Mode](#) unten.

Strict Mode

Das **strict** -Feld steuert, ob **plugin.json** die Autorität für Komponentendefinitionen ist (Befehle, Agents, hooks, skills, MCP servers, Ausgabestile).

Wert	Verhalten
true (Standard)	plugin.json ist die Autorität. Der Marktplatz-Eintrag kann es mit zusätzlichen Komponenten ergänzen, und beide Quellen werden zusammengeführt.
false	Der Marktplatz-Eintrag ist die gesamte Definition. Wenn das Plugin auch eine plugin.json hat, die Komponenten deklariert, ist das ein Konflikt und das Plugin kann nicht geladen werden.

Wann jeder Modus verwendet werden sollte:

- **strict: true** : Das Plugin hat seine eigene **plugin.json** und verwaltet seine eigenen Komponenten. Der Marktplatz-Eintrag kann zusätzliche Befehle oder hooks hinzufügen. Dies ist der Standard und funktioniert für die meisten Plugins.
- **strict: false** : Der Marktplatz-Betreiber möchte vollständige Kontrolle. Das Plugin-Repo stellt Rohdateien bereit, und der Marktplatz-Eintrag definiert, welche dieser Dateien als Befehle, Agents, hooks usw. verfügbar gemacht werden. Nützlich, wenn

der Marktplatz die Komponenten eines Plugins anders strukturiert oder kuratiert als vom Plugin-Autor beabsichtigt.

Marktplätze hosten und verteilen

Auf GitHub hosten (empfohlen)

GitHub bietet die einfachste Verteilungsmethode:

- 1. **Repository erstellen:** Richten Sie ein neues Repository für Ihren Marktplatz ein
- 2. **Marktplatzdatei hinzufügen:** Erstellen Sie `.claude-plugin/marketplace.json` mit Ihren Plugin-Definitionen
- 3. **Mit Teams teilen:** Benutzer fügen Ihren Marktplatz mit `/plugin marketplace add owner/repo` hinzu

Vorteile: Integrierte Versionskontrolle, Issue-Tracking und Team-Zusammenarbeitsfunktionen.

Auf anderen Git-Services hosten

Jeder Git-Hosting-Service funktioniert, wie GitLab, Bitbucket und selbstgehostete Server. Benutzer fügen mit der vollständigen Repository-URL hinzu:

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

Private Repositories

Claude Code unterstützt die Installation von Plugins aus privaten Repositories. Für manuelle Installation und Updates verwendet Claude Code Ihre vorhandenen Git-Credential-Helper. Wenn `git clone` für ein privates Repository in Ihrem Terminal funktioniert, funktioniert es auch in Claude Code. Häufige Credential-Helper sind `gh auth login` für GitHub, macOS Keychain und `git-credential-store`.

Hintergrund-Auto-Updates werden beim Start ohne Credential-Helper ausgeführt, da interaktive Eingabeaufforderungen Claude Code am Start hindern würden. Um Auto-Updates für private Marktplätze zu aktivieren, legen Sie das entsprechende Authentifizierungstoken in Ihrer Umgebung fest:

Anbieter	Umgebungsvariablen	Notizen
GitHub	<code>GITHUB_TOKEN</code> oder <code>GH_TOKEN</code>	Persönliches Zugriffs-Token oder GitHub App-Token

Anbieter	Umgebungsvariablen	Notizen
GitLab	<code>GITLAB_TOKEN</code> oder <code>GL_TOKEN</code>	Persönliches Zugriffs-Token oder Projekt-Token
Bitbucket	<code>BITBUCKET_TOKEN</code>	App-Passwort oder Repository-Zugriffs-Token

Legen Sie das Token in Ihrer Shell-Konfiguration fest (z. B. `.bashrc`, `.zshrc`) oder übergeben Sie es beim Ausführen von Claude Code:

```
export GITHUB_TOKEN=ghp_XXXXXXXXXXXXXXXXXXXX
```

Note:

Konfigurieren Sie das Token für CI/CD-Umgebungen als geheime Umgebungsvariable. GitHub Actions stellt automatisch `GITHUB_TOKEN` für Repositories in derselben Organisation bereit.

Lokal vor der Verteilung testen

Testen Sie Ihren Marktplatz lokal, bevor Sie ihn teilen:

```
/plugin marketplace add ./my-local-marketplace  
/plugin install test-plugin@my-local-marketplace
```

Für die vollständige Palette von Add-Befehlen (GitHub, Git-URLs, lokale Pfade, Remote-URLs) siehe [Marktplätze hinzufügen](#).

Marktplätze für Ihr Team erforderlich machen

Sie können Ihr Repository so konfigurieren, dass Teammitglieder automatisch aufgefordert werden, Ihren Marktplatz zu installieren, wenn sie dem Projektordner vertrauen. Fügen Sie Ihren Marktplatz zu `.claude/settings.json` hinzu:


```
{
  "extraKnownMarketplaces": {
    "company-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

Sie können auch angeben, welche Plugins standardmäßig aktiviert sein sollen:

```
{
  "enabledPlugins": {
    "code-formatter@company-tools": true,
    "deployment-tools@company-tools": true
  }
}
```

Für vollständige Konfigurationsoptionen siehe [Plugin-Einstellungen](#).

Verwaltete Marktplatz-Einschränkungen

Für Organisationen, die strikte Kontrolle über Plugin-Quellen benötigen, können Administratoren einschränken, welche Plugin-Marktplätze Benutzer hinzufügen dürfen, indem sie die Einstellung `strictKnownMarketplaces` in verwalteten Einstellungen verwenden.

Wenn `strictKnownMarketplaces` in verwalteten Einstellungen konfiguriert ist, hängt das Einschränkungsverhalten vom Wert ab:

Wert	Verhalten
Nicht definiert (Standard)	Keine Einschränkungen. Benutzer können jeden Marktplatz hinzufügen
Leeres Array <code>[]</code>	Vollständige Sperrung. Benutzer können keine neuen Marktplätze hinzufügen

Wert	Verhalten
Liste von Quellen	Benutzer können nur Marktplätze hinzufügen, die genau mit der Zulassungsliste übereinstimmen

Häufige Konfigurationen

Deaktivieren Sie alle Marktplatz-Ergänzungen:

```
{  
  "strictKnownMarketplaces": []  
}
```

Nur bestimmte Marktplätze zulassen:

```
{  
  "strictKnownMarketplaces": [  
    {  
      "source": "github",  
      "repo": "acme-corp/approved-plugins"  
    },  
    {  
      "source": "github",  
      "repo": "acme-corp/security-tools",  
      "ref": "v2.0"  
    },  
    {  
      "source": "url",  
      "url": "https://plugins.example.com/marketplace.json"  
    }  
  ]  
}
```

Alle Marktplätze von einem internen Git-Server mit Regex-Musterabgleich auf dem Host zulassen:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

Dateisystem-basierte Marktplätze aus einem bestimmten Verzeichnis mit Regex-Musterabgleich auf dem Pfad zulassen:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "pathPattern",
      "pathPattern": "^/opt/approved/"
    }
  ]
}
```

Verwenden Sie `".*"` als `pathPattern`, um jeden Dateisystempfad zuzulassen und gleichzeitig Netzwerkquellen mit `hostPattern` zu steuern.

Wie Einschränkungen funktionieren

Einschränkungen werden früh im Plugin-Installationsprozess validiert, bevor Netzwerkanfragen oder Dateisystemoperationen auftreten. Dies verhindert unbefugte Marktplatz-Zugriffversuche.

Die Zulassungsliste verwendet exakten Abgleich für die meisten Quellentypen. Damit ein Marktplatz zulässig ist, müssen alle angegebenen Felder genau übereinstimmen:

- Für GitHub-Quellen: `repo` ist erforderlich, und `ref` oder `path` müssen auch übereinstimmen, wenn sie in der Zulassungsliste angegeben sind
- Für URL-Quellen: Die vollständige URL muss genau übereinstimmen
- Für `hostPattern`-Quellen: Der Marktplatz-Host wird gegen das Regex-Muster abgeglichen

- Für `pathPattern`-Quellen: Der Dateisystempfad des Marktplatzes wird gegen das Regex-Muster abgeglichen

Da `strictKnownMarketplaces` in [verwalteten Einstellungen](#) festgelegt ist, können einzelne Benutzer und Projektkonfigurationen diese Einschränkungen nicht überschreiben.

Für vollständige Konfigurationsdetails einschließlich aller unterstützten Quellentypen und Vergleich mit `extraKnownMarketplaces` siehe die [strictKnownMarketplaces-Referenz](#).

Versionsauflösung und Release-Kanäle

Plugin-Versionen bestimmen Cache-Pfade und Update-Erkennung. Sie können die Version im Plugin-Manifest (`plugin.json`) oder im Marktplatz-Eintrag (`marketplace.json`) angeben.

Warning:

Vermeiden Sie nach Möglichkeit, die Version an beiden Stellen festzulegen. Das Plugin-Manifest gewinnt immer stillschweigend, was dazu führen kann, dass die Marktplatz-Version ignoriert wird. Legen Sie für Plugins mit relativen Pfaden die Version im Marktplatz-Eintrag fest. Legen Sie für alle anderen Plugin-Quellen die Version im Plugin-Manifest fest.

Richten Sie Release-Kanäle ein

Um “stabile” und “neueste” Release-Kanäle für Ihre Plugins zu unterstützen, können Sie zwei Marktplätze einrichten, die auf verschiedene Refs oder SHAs desselben Repos verweisen. Sie können dann die beiden Marktplätze verschiedenen Benutzergruppen über [verwaltete Einstellungen](#) zuweisen.

Warning:

Die `plugin.json` des Plugins muss eine andere `version` bei jedem angehefteten Ref oder Commit deklarieren. Wenn zwei Refs oder Commits die gleiche Manifest-Version haben, behandelt Claude Code sie als identisch und überspringt das Update.

Beispiel

```
{
  "name": "stable-tools",
  "plugins": [
    {
      "name": "code-formatter",
      "source": {
        "source": "github",
        "repo": "acme-corp/code-formatter",
        "ref": "stable"
      }
    }
  ]
}
```

```
{
  "name": "latest-tools",
  "plugins": [
    {
      "name": "code-formatter",
      "source": {
        "source": "github",
        "repo": "acme-corp/code-formatter",
        "ref": "latest"
      }
    }
  ]
}
```

Kanäle Benutzergruppen zuweisen

Weisen Sie jeden Marktplatz der entsprechenden Benutzergruppe über verwaltete Einstellungen zu. Beispielsweise erhält die stabile Gruppe:

```
{
  "extraKnownMarketplaces": {
    "stable-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/stable-tools"
      }
    }
  }
}
```

Die Early-Access-Gruppe erhält stattdessen `latest-tools` :

```
{
  "extraKnownMarketplaces": {
    "latest-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/latest-tools"
      }
    }
  }
}
```

Validierung und Tests

Testen Sie Ihren Marktplatz vor dem Teilen.

Validieren Sie Ihre Marktplatz-JSON-Syntax:

```
claude plugin validate .
```

Oder von innerhalb von Claude Code:

```
/plugin validate .
```

Fügen Sie den Marktplatz zum Testen hinzu:

```
/plugin marketplace add ./path/to/marketplace
```

Installieren Sie ein Test-Plugin, um zu überprüfen, ob alles funktioniert:

```
/plugin install test-plugin@marketplace-name
```

Für vollständige Plugin-Test-Workflows siehe [Testen Sie Ihre Plugins lokal](#). Für technische Fehlerbehebung siehe [Plugins-Referenz](#).

Fehlerbehebung

Marktplatz wird nicht geladen

Symptome: Kann Marktplatz nicht hinzufügen oder Plugins von ihm nicht sehen

Lösungen:

- Überprüfen Sie, dass die Marktplatz-URL erreichbar ist
- Überprüfen Sie, dass `.claude-plugin/marketplace.json` im angegebenen Pfad vorhanden ist
- Stellen Sie sicher, dass die JSON-Syntax gültig ist, indem Sie `claude plugin validate` oder `/plugin validate` verwenden
- Bestätigen Sie für private Repositories, dass Sie Zugriffsberechtigung haben

Marktplatz-Validierungsfehler

Führen Sie `claude plugin validate .` oder `/plugin validate .` aus Ihrem Marktplatz-Verzeichnis aus, um auf Probleme zu überprüfen. Häufige Fehler:

Fehler	Ursache	Lösung
<code>File not found: .claude-plugin/marketplace.json</code>	Fehlendes Manifest	Erstellen Sie <code>.claude-plugin/marketplace.json</code> mit erforderlichen Feldern
<code>Invalid JSON syntax: Unexpected token...</code>	JSON-Syntaxfehler	Überprüfen Sie auf fehlende Kommas, zusätzliche Kommas oder nicht zitierte Strings

Fehler	Ursache	Lösung
<code>Duplicate plugin name "x" found in marketplace</code>	Zwei Plugins teilen denselben Namen	Geben Sie jedem Plugin einen eindeutigen <code>name</code> - Wert
<code>plugins[0].source: Path traversal not allowed</code>	Quellpfad enthält <code>..</code>	Verwenden Sie Pfade relativ zum Marktplatz-Root ohne <code>..</code>

Warnungen (nicht blockierend):

- `Marketplace has no plugins defined` : Fügen Sie mindestens ein Plugin zum `plugins` -Array hinzu
- `No marketplace description provided` : Fügen Sie `metadata.description` hinzu, um Benutzern zu helfen, Ihren Marktplatz zu verstehen

Plugin-Installationsfehler

Symptome: Marktplatz wird angezeigt, aber Plugin-Installation schlägt fehl

Lösungen:

- Überprüfen Sie, dass Plugin-Quell-URLs erreichbar sind
- Überprüfen Sie, dass Plugin-Verzeichnisse erforderliche Dateien enthalten
- Überprüfen Sie für GitHub-Quellen, dass Repositories öffentlich sind oder Sie Zugriff haben
- Testen Sie Plugin-Quellen manuell durch Klonen/Herunterladen

Authentifizierung für private Repositories schlägt fehl

Symptome: Authentifizierungsfehler beim Installieren von Plugins aus privaten Repositories

Lösungen:

Für manuelle Installation und Updates:

- Überprüfen Sie, dass Sie bei Ihrem Git-Anbieter authentifiziert sind (führen Sie z. B. `gh auth status` für GitHub aus)
- Überprüfen Sie, dass Ihr Credential-Helper korrekt konfiguriert ist: `git config --global credential.helper`
- Versuchen Sie, das Repository manuell zu klonen, um zu überprüfen, dass Ihre Anmeldedaten funktionieren

Für Hintergrund-Auto-Updates:

- Legen Sie das entsprechende Token in Ihrer Umgebung fest: `echo $GITHUB_TOKEN`
- Überprüfen Sie, dass das Token die erforderlichen Berechtigungen hat (Lesezugriff auf das Repository)
- Überprüfen Sie für GitHub, dass das Token den `repo`-Scope für private Repositories hat
- Überprüfen Sie für GitLab, dass das Token mindestens den `read_repository`-Scope hat
- Überprüfen Sie, dass das Token nicht abgelaufen ist

Git-Operationen zeitüberschreitung

Symptome: Plugin-Installation oder Marktplatz-Updates schlagen mit einem Timeout-Fehler fehl, wie “Git clone timed out after 120s” oder “Git pull timed out after 120s”.

Ursache: Claude Code verwendet ein 120-Sekunden-Timeout für alle Git-Operationen, einschließlich Klonen von Plugin-Repositories und Abrufen von Marktplatz-Updates. Große Repositories oder langsame Netzwerkverbindungen können dieses Limit überschreiten.

Lösung: Erhöhen Sie das Timeout mit der Umgebungsvariable

`CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS`. Der Wert ist in Millisekunden:

```
export CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS=300000 # 5 Minuten
```

Plugins mit relativen Pfaden schlagen in URL-basierten Marktplätzen fehl

Symptome: Einen Marktplatz über URL hinzugefügt (z. B. `https://example.com/marketplace.json`), aber Plugins mit relativen Pfadquellen wie `./plugins/my-plugin` schlagen mit “path not found”-Fehlern fehl.

Ursache: URL-basierte Marktplätze laden nur die `marketplace.json`-Datei selbst herunter. Sie laden keine Plugin-Dateien vom Server herunter. Relative Pfade im Marktplatz-Eintrag verweisen auf Dateien auf dem Remote-Server, die nicht heruntergeladen wurden.

Lösungen:

- **Verwenden Sie externe Quellen:** Ändern Sie Plugin-Einträge, um stattdessen GitHub-, npm- oder Git-URL-Quellen zu verwenden:

```
{ "name": "my-plugin", "source": { "source": "github", "repo": "owner/repo" } }
```

- **Verwenden Sie einen Git-basierten Marktplatz:** Hosten Sie Ihren Marktplatz in einem Git-Repository und fügen Sie ihn mit der Git-URL hinzu. Git-basierte Marktplätze klonen das gesamte Repository, wodurch relative Pfade funktionieren.

Dateien nicht gefunden nach Installation

Symptome: Plugin wird installiert, aber Verweise auf Dateien schlagen fehl, besonders Dateien außerhalb des Plugin-Verzeichnisses

Ursache: Plugins werden in ein Cache-Verzeichnis kopiert, anstatt an Ort und Stelle verwendet zu werden. Pfade, die auf Dateien außerhalb des Plugin-Verzeichnisses verweisen (wie `../shared-utils`), funktionieren nicht, da diese Dateien nicht kopiert werden.

Lösungen: Siehe [Plugin-Caching und Dateiauflösung](#) für Workarounds, einschließlich Symlinks und Verzeichnisumstrukturierung.

Für zusätzliche Debugging-Tools und häufige Probleme siehe [Debugging- und Entwicklungstools](#).

Siehe auch

- [Entdecken und Installieren vorgefertigter Plugins](#) - Installieren von Plugins aus vorhandenen Marktplätzen
- [Plugins](#) - Erstellen Ihrer eigenen Plugins
- [Plugins-Referenz](#) - Vollständige technische Spezifikationen und Schemas
- [Plugin-Einstellungen](#) - Plugin-Konfigurationsoptionen
- [strictKnownMarketplaces-Referenz](#) - Verwaltete Marktplatz-Einschränkungen

Entdecken und installieren Sie vorgefertigte Plugins über Marktplätze

Finden und installieren Sie Plugins aus Marktplätzen, um Claude Code mit neuen Befehlen, Agenten und Funktionen zu erweitern.

Plugins erweitern Claude Code mit skills, agents, hooks und MCP servers. Plugin-Marktplätze sind Kataloge, die Ihnen helfen, diese Erweiterungen zu entdecken und zu installieren, ohne sie selbst zu erstellen.

Möchten Sie Ihren eigenen Marktplatz erstellen und verteilen? Siehe [Erstellen und verteilen Sie einen Plugin-Marktplatz](#).

Wie Marktplätze funktionieren

Ein Marktplatz ist ein Katalog von Plugins, die jemand anderes erstellt und geteilt hat. Die Verwendung eines Marktplatzes ist ein zweistufiger Prozess:

Step 1: Fügen Sie den Marktplatz hinzu

Dies registriert den Katalog bei Claude Code, damit Sie durchsuchen können, was verfügbar ist. Es werden noch keine Plugins installiert.

Step 2: Installieren Sie einzelne Plugins

Durchsuchen Sie den Katalog und installieren Sie die Plugins, die Sie möchten.

Stellen Sie sich das vor wie das Hinzufügen eines App-Stores: Das Hinzufügen des Stores gibt Ihnen Zugriff zum Durchsuchen seiner Sammlung, aber Sie wählen immer noch aus, welche Apps Sie einzeln herunterladen möchten.

Offizieller Anthropic-Marktplatz

Der offizielle Anthropic-Marktplatz (`claude-plugins-official`) ist automatisch verfügbar, wenn Sie Claude Code starten. Führen Sie `/plugin` aus und gehen Sie zur Registerkarte **Discover**, um zu sehen, was verfügbar ist.

So installieren Sie ein Plugin aus dem offiziellen Marktplatz:

```
/plugin install plugin-name@claude-plugins-official
```

Note:

Der offizielle Marktplatz wird von Anthropic gepflegt. Um ein Plugin beim offiziellen Marktplatz einzureichen, verwenden Sie eines der In-App-Einreichungsformulare:

- **Claude.ai:** claude.ai/settings/plugins/submit
- **Console:** platform.claude.com/plugins/submit

Um Plugins unabhängig zu verteilen, [erstellen Sie Ihren eigenen Marktplatz](#) und teilen Sie ihn mit Benutzern.

Der offizielle Marktplatz umfasst mehrere Plugin-Kategorien:

Code-Intelligenz

Code-Intelligenz-Plugins aktivieren das integrierte LSP-Tool von Claude Code und geben Claude die Möglichkeit, zu Definitionen zu springen, Referenzen zu finden und Typfehler unmittelbar nach Änderungen zu sehen. Diese Plugins konfigurieren [Language Server Protocol](#)-Verbindungen, die gleiche Technologie, die die Code-Intelligenz von VS Code antreibt.

Diese Plugins erfordern, dass die Language-Server-Binärdatei auf Ihrem System installiert ist. Wenn Sie bereits einen Language Server installiert haben, kann Claude Sie möglicherweise auffordern, das entsprechende Plugin zu installieren, wenn Sie ein Projekt öffnen.

Sprache	Plugin	Erforderliche Binärdatei
C/C++	clangd-lsp	clangd
C#	csharp-lsp	csharp-ls
Go	gopls-lsp	gopls
Java	jdtls-lsp	jdtls
Kotlin	kotlin-lsp	kotlin-language-server
Lua	lua-lsp	lua-language-server
PHP	php-lsp	intelephense
Python	pyright-lsp	pyright-langserver

Sprache	Plugin	Erforderliche Binärdatei
Rust	<code>rust-analyzer-lsp</code>	<code>rust-analyzer</code>
Swift	<code>swift-lsp</code>	<code>sourcekit-lsp</code>
TypeScript	<code>typescript-lsp</code>	<code>typescript-language-server</code>

Sie können auch [Ihr eigenes LSP-Plugin erstellen](#) für andere Sprachen.

Note:

Wenn Sie nach der Installation eines Plugins `Executable not found in $PATH` in der Registerkarte `/plugin` Errors sehen, installieren Sie die erforderliche Binärdatei aus der obigen Tabelle.

Was Claude von Code-Intelligenz-Plugins gewinnt

Sobald ein Code-Intelligenz-Plugin installiert ist und seine Language-Server-Binärdatei verfügbar ist, gewinnt Claude zwei Funktionen:

- **Automatische Diagnose:** Nach jeder Dateiänderung, die Claude vornimmt, analysiert der Language Server die Änderungen und meldet Fehler und Warnungen automatisch zurück. Claude sieht Typfehler, fehlende Importe und Syntaxprobleme, ohne einen Compiler oder Linter ausführen zu müssen. Wenn Claude einen Fehler einführt, bemerkt es das Problem und behebt es in derselben Runde. Dies erfordert keine Konfiguration über die Installation des Plugins hinaus. Sie können Diagnosen inline anzeigen, indem Sie **Strg+O** drücken, wenn der Indikator „Diagnosen gefunden“ angezeigt wird.
- **Code-Navigation:** Claude kann den Language Server verwenden, um zu Definitionen zu springen, Referenzen zu finden, Typinformationen beim Hover zu erhalten, Symbole aufzulisten, Implementierungen zu finden und Call-Hierarchien zu verfolgen. Diese Operationen geben Claude eine präzisere Navigation als grep-basierte Suche, obwohl die Verfügbarkeit je nach Sprache und Umgebung variieren kann.

Wenn Sie auf Probleme stoßen, siehe [Code-Intelligenz-Fehlerbehebung](#).

Externe Integrationen

Diese Plugins bündeln vorkonfigurierte [MCP servers](#), damit Sie Claude mit externen Diensten verbinden können, ohne manuelle Einrichtung:

- **Quellcodeverwaltung:** `github`, `gitlab`

- **Projektmanagement:** `atlassian` (Jira/Confluence), `asana`, `linear`, `notion`
- **Design:** `figma`
- **Infrastruktur:** `vercel`, `firebase`, `supabase`
- **Kommunikation:** `slack`
- **Überwachung:** `sentry`

Entwicklungs-Workflows

Plugins, die Befehle und Agenten für häufige Entwicklungsaufgaben hinzufügen:

- **commit-commands:** Git-Commit-Workflows einschließlich Commit, Push und PR-Erstellung
- **pr-review-toolkit:** Spezialisierte Agenten für die Überprüfung von Pull Requests
- **agent-sdk-dev:** Tools zum Erstellen mit dem Claude Agent SDK
- **plugin-dev:** Toolkit zum Erstellen Ihrer eigenen Plugins

Ausgabestile

Passen Sie an, wie Claude antwortet:

- **explanatory-output-style:** Pädagogische Einblicke in Implementierungsentscheidungen
- **learning-output-style:** Interaktiver Lernmodus zum Aufbau von Fähigkeiten

Probieren Sie es aus: Fügen Sie den Demo-Marktplatz hinzu

Anthropic verwaltet auch einen [Demo-Plugins-Marktplatz](#) (`claude-code-plugins`) mit Beispiel-Plugins, die zeigen, was mit dem Plugin-System möglich ist. Im Gegensatz zum offiziellen Marktplatz müssen Sie diesen manuell hinzufügen.

Step 1: Fügen Sie den Marktplatz hinzu

Führen Sie in Claude Code den Befehl `plugin marketplace add` für den Marktplatz `anthropics/claude-code` aus:

```
/plugin marketplace add anthropics/claude-code
```

Dies lädt den Marktplatz-Katalog herunter und macht seine Plugins für Sie verfügbar.

Step 2: Durchsuchen Sie verfügbare Plugins

Führen Sie `/plugin` aus, um den Plugin-Manager zu öffnen. Dies öffnet eine Schnittstelle mit Registerkarten mit vier Registerkarten, die Sie mit **Tab** durchlaufen können (oder **Shift+Tab**, um rückwärts zu gehen):

- **Discover:** Durchsuchen Sie verfügbare Plugins aus allen Ihren Marktplätzen
- **Installed:** Zeigen Sie Ihre installierten Plugins an und verwalten Sie sie
- **Marketplaces:** Fügen Sie Marktplätze hinzu, entfernen Sie sie oder aktualisieren Sie sie
- **Errors:** Zeigen Sie alle Plugin-Ladefehler an

Gehen Sie zur Registerkarte **Discover**, um Plugins aus dem Marktplatz zu sehen, den Sie gerade hinzugefügt haben.

Step 3: Installieren Sie ein Plugin

Wählen Sie ein Plugin aus, um seine Details anzuzeigen, und wählen Sie dann einen Installationsbereich:

- **User scope:** Installieren Sie für sich selbst in allen Projekten
- **Project scope:** Installieren Sie für alle Mitarbeiter in diesem Repository
- **Local scope:** Installieren Sie für sich selbst nur in diesem Repository

Wählen Sie beispielsweise **commit-commands** (ein Plugin, das Git-Workflow-Befehle hinzufügt) und installieren Sie es in Ihrem Benutzerbereich.

Sie können auch direkt über die Befehlszeile installieren:

```
/plugin install commit-commands@anthropics-claude-code
```

Siehe [Konfigurationsbereiche](#), um mehr über Bereiche zu erfahren.

Step 4: Verwenden Sie Ihr neues Plugin

Nach der Installation sind die Befehle des Plugins sofort verfügbar. Plugin-Befehle werden nach dem Plugin-Namen benannt, daher bietet **commit-commands** Befehle wie `/commit-commands:commit`.

Probieren Sie es aus, indem Sie eine Änderung an einer Datei vornehmen und ausführen:

```
/commit-commands:commit
```

Dies stellt Ihre Änderungen bereit, generiert eine Commit-Nachricht und erstellt den Commit.

Jedes Plugin funktioniert anders. Überprüfen Sie die Beschreibung des Plugins in der Registerkarte **Discover** oder auf seiner Homepage, um zu erfahren, welche Befehle und Funktionen es bietet.

Der Rest dieses Leitfadens behandelt alle Möglichkeiten, wie Sie Marktplätze hinzufügen, Plugins installieren und Ihre Konfiguration verwalten können.

Marktplätze hinzufügen

Verwenden Sie den Befehl `/plugin marketplace add`, um Marktplätze aus verschiedenen Quellen hinzuzufügen.

Tip:

Verknüpfungen: Sie können `/plugin market` anstelle von `/plugin marketplace` verwenden und `rm` anstelle von `remove`.

- **GitHub-Repositories:** Format `owner/repo` (z. B. `anthropics/claude-code`)
- **Git-URLs:** Beliebige Git-Repository-URL (GitLab, Bitbucket, selbstgehostet)
- **Lokale Pfade:** Verzeichnisse oder direkte Pfade zu `marketplace.json`-Dateien
- **Remote-URLs:** Direkte URLs zu gehosteten `marketplace.json`-Dateien

Hinzufügen von GitHub

Fügen Sie ein GitHub-Repository hinzu, das eine `.claude-plugin/marketplace.json`-Datei enthält, indem Sie das Format `owner/repo` verwenden – wobei `owner` der GitHub-Benutzername oder die Organisation und `repo` der Repository-Name ist.

Beispielsweise bezieht sich `anthropics/claude-code` auf das Repository `claude-code`, das sich im Besitz von `anthropics` befindet:

```
/plugin marketplace add anthropics/claude-code
```

Hinzufügen von anderen Git-Hosts

Fügen Sie ein beliebiges Git-Repository hinzu, indem Sie die vollständige URL angeben. Dies funktioniert mit jedem Git-Host, einschließlich GitLab, Bitbucket und selbstgehosteten Servern:

Mit HTTPS:


```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

Mit SSH:

```
/plugin marketplace add git@gitlab.com:company/plugins.git
```

Um einen bestimmten Branch oder Tag hinzuzufügen, hängen Sie **#** gefolgt von der Referenz an:

```
/plugin marketplace add https://gitlab.com/company/plugins.git#v1.0.0
```

Hinzufügen von lokalen Pfaden

Fügen Sie ein lokales Verzeichnis hinzu, das eine **.claude-plugin/marketplace.json** -Datei enthält:

```
/plugin marketplace add ./my-marketplace
```

Sie können auch einen direkten Pfad zu einer **marketplace.json** -Datei hinzufügen:

```
/plugin marketplace add ./path/to/marketplace.json
```

Hinzufügen von Remote-URLs

Fügen Sie eine Remote-**marketplace.json** -Datei über URL hinzu:

```
/plugin marketplace add https://example.com/marketplace.json
```

Note:

URL-basierte Marktplätze haben einige Einschränkungen im Vergleich zu Git-basierten Marktplätzen. Wenn beim Installieren von Plugins Fehler „Pfad nicht gefunden“ auftreten, siehe [Fehlerbehebung](#).

Installieren Sie Plugins

Nachdem Sie Marktplätze hinzugefügt haben, können Sie Plugins direkt installieren (wird standardmäßig im Benutzerbereich installiert):

```
/plugin install plugin-name@marketplace-name
```

Um einen anderen [Installationsbereich](#) zu wählen, verwenden Sie die interaktive Benutzeroberfläche: Führen Sie `/plugin` aus, gehen Sie zur Registerkarte **Discover**, und drücken Sie **Enter** auf einem Plugin. Sie sehen Optionen für:

- **User scope** (Standard): Installieren Sie für sich selbst in allen Projekten
- **Project scope**: Installieren Sie für alle Mitarbeiter in diesem Repository (fügt zu `.claude/settings.json` hinzu)
- **Local scope**: Installieren Sie für sich selbst nur in diesem Repository (nicht mit Mitarbeitern geteilt)

Sie können auch Plugins mit **managed**-Bereich sehen – diese werden von Administratoren über [verwaltete Einstellungen](#) installiert und können nicht geändert werden.

Führen Sie `/plugin` aus und gehen Sie zur Registerkarte **Installed**, um Ihre Plugins nach Bereich gruppiert zu sehen.

Warning:

Stellen Sie sicher, dass Sie einem Plugin vertrauen, bevor Sie es installieren. Anthropic kontrolliert nicht, welche MCP servers, Dateien oder andere Software in Plugins enthalten sind, und kann nicht überprüfen, dass sie wie beabsichtigt funktionieren. Überprüfen Sie die Homepage jedes Plugins für weitere Informationen.

Verwalten Sie installierte Plugins

Führen Sie `/plugin` aus und gehen Sie zur Registerkarte **Installed**, um Ihre Plugins anzuzeigen, zu aktivieren, zu deaktivieren oder zu deinstallieren. Geben Sie ein, um die Liste nach Plugin-Name oder Beschreibung zu filtern.

Sie können Plugins auch mit direkten Befehlen verwalten.

Deaktivieren Sie ein Plugin, ohne es zu deinstallieren:

```
/plugin disable plugin-name@marketplace-name
```

Aktivieren Sie ein deaktiviertes Plugin erneut:

```
/plugin enable plugin-name@marketplace-name
```

Entfernen Sie ein Plugin vollständig:

```
/plugin uninstall plugin-name@marketplace-name
```

Die Option `--scope` ermöglicht es Ihnen, einen bestimmten Bereich mit CLI-Befehlen anzusteuern:

```
claude plugin install formatter@your-org --scope project
claude plugin uninstall formatter@your-org --scope project
```

Wenden Sie Plugin-Änderungen an, ohne neu zu starten

Wenn Sie während einer Sitzung Plugins installieren, aktivieren oder deaktivieren, werden einige Änderungen (wie neue Befehle und hooks) sofort wirksam. Andere, einschließlich LSP-Server-Updates, erfordern einen Neustart.

Um alle ausstehenden Plugin-Änderungen ohne Neustart zu aktivieren, führen Sie aus:

```
/reload-plugins
```

Claude Code lädt alle aktiven Plugins neu und meldet, was geladen wurde. Wenn LSP-Server hinzugefügt oder aktualisiert wurden, wird Ihnen mitgeteilt, dass diese einen Neustart erfordern, um wirksam zu werden.

Verwalten Sie Marktplätze

Sie können Marktplätze über die interaktive `/plugin` -Schnittstelle oder mit CLI-Befehlen verwalten.

Verwenden Sie die interaktive Schnittstelle

Führen Sie `/plugin` aus und gehen Sie zur Registerkarte **Marketplaces**, um:

- Alle Ihre hinzugefügten Marktplätze mit ihren Quellen und Status anzuzeigen
- Neue Marktplätze hinzuzufügen
- Marktplatz-Auflistungen aktualisieren, um die neuesten Plugins abzurufen
- Marktplätze zu entfernen, die Sie nicht mehr benötigen

Verwenden Sie CLI-Befehle

Sie können Marktplätze auch mit direkten Befehlen verwalten.

Listet alle konfigurierten Marktplätze auf:

```
/plugin marketplace list
```

Aktualisieren Sie Plugin-Auflistungen von einem Marktplatz:

```
/plugin marketplace update marketplace-name
```

Entfernen Sie einen Marktplatz:

```
/plugin marketplace remove marketplace-name
```

Warning:

Das Entfernen eines Marktplatzes deinstalliert alle Plugins, die Sie von ihm installiert haben.

Konfigurieren Sie automatische Updates

Claude Code kann Marktplätze und ihre installierten Plugins beim Start automatisch aktualisieren. Wenn die automatische Aktualisierung für einen Marktplatz aktiviert ist, aktualisiert Claude Code die Marktplatzdaten und aktualisiert installierte Plugins auf ihre neuesten Versionen. Wenn Plugins aktualisiert wurden, sehen Sie eine Benachrichtigung, die Sie auffordert, `/reload-plugins` auszuführen.

Schalten Sie die automatische Aktualisierung für einzelne Marktplätze über die Benutzeroberfläche um:

1. Führen Sie `/plugin` aus, um den Plugin-Manager zu öffnen

2. Wählen Sie **Marketplaces**

3. Wählen Sie einen Marktplatz aus der Liste

4. Wählen Sie **Enable auto-update** oder **Disable auto-update**

Offizielle Anthropic-Marktplätze haben die automatische Aktualisierung standardmäßig aktiviert. Marktplätze von Drittanbietern und lokale Entwicklungsmarktplätze haben die automatische Aktualisierung standardmäßig deaktiviert.

Um alle automatischen Updates vollständig für Claude Code und alle Plugins zu deaktivieren, setzen Sie die Umgebungsvariable `DISABLE_AUTOUPDATER`. Siehe [Automatische Updates](#) für Details.

Um Plugin-Auto-Updates aktiviert zu halten und gleichzeitig Claude Code-Auto-Updates zu deaktivieren, setzen Sie `FORCE_AUTOUPDATE_PLUGINS=true` zusammen mit `DISABLE_AUTOUPDATER`:

```
export DISABLE_AUTOUPDATER=true
export FORCE_AUTOUPDATE_PLUGINS=true
```

Dies ist nützlich, wenn Sie Claude Code-Updates manuell verwalten möchten, aber immer noch automatische Plugin-Updates erhalten möchten.

Konfigurieren Sie Team-Marktplätze

Team-Administratoren können die automatische Marktplatz-Installation für Projekte einrichten, indem sie Marktplatz-Konfiguration zu `.claude/settings.json` hinzufügen. Wenn Team-Mitglieder dem Repository-Ordner vertrauen, fordert Claude Code sie auf, diese Marktplätze und Plugins zu installieren.

Fügen Sie `extraKnownMarketplaces` zu Ihrer Projekt- `.claude/settings.json` hinzu:

```
{
  "extraKnownMarketplaces": {
    "my-team-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

Für vollständige Konfigurationsoptionen einschließlich `extraKnownMarketplaces` und `enabledPlugins` siehe [Plugin-Einstellungen](#).

Sicherheit

Plugins und Marktplätze sind hochgradig vertrauenswürdige Komponenten, die beliebigen Code auf Ihrem Computer mit Ihren Benutzerrechten ausführen können. Installieren Sie nur Plugins und fügen Sie Marktplätze aus Quellen hinzu, denen Sie vertrauen. Organisationen können einschränken, welche Marktplätze Benutzer hinzufügen dürfen, indem sie [verwaltete Marktplatz-Einschränkungen](#) verwenden.

Fehlerbehebung

/plugin-Befehl nicht erkannt

Wenn Sie „unknown command“ sehen oder der `/plugin`-Befehl nicht angezeigt wird:

- Überprüfen Sie Ihre Version:** Führen Sie `claude --version` aus. Plugins erfordern Version 1.0.33 oder später.
- Aktualisieren Sie Claude Code:**
 - **Homebrew:** `brew upgrade claude-code`
 - **npm:** `npm update -g @anthropic-ai/claude-code`
 - **Native Installer:** Führen Sie den Installationsbefehl von [Setup](#) erneut aus
- Starten Sie Claude Code neu:** Starten Sie nach dem Update Ihr Terminal neu und führen Sie `claude` erneut aus.

Häufige Probleme

- **Marktplatz wird nicht geladen:** Überprüfen Sie, dass die URL zugänglich ist und dass `.claude-plugin/marketplace.json` unter dem Pfad vorhanden ist
- **Plugin-Installationsfehler:** Überprüfen Sie, dass Plugin-Quell-URLs zugänglich sind und Repositories öffentlich sind (oder Sie haben Zugriff)
- **Dateien nach der Installation nicht gefunden:** Plugins werden in einen Cache kopiert, daher funktionieren Pfade, die auf Dateien außerhalb des Plugin-Verzeichnisses verweisen, nicht
- **Plugin-Skills werden nicht angezeigt:** Löschen Sie den Cache mit `rm -rf ~/.claude/plugins/cache`, starten Sie Claude Code neu und installieren Sie das Plugin erneut.

Für detaillierte Fehlerbehebung mit Lösungen siehe [Fehlerbehebung](#) im Marktplatz-Leitfaden. Für Debugging-Tools siehe [Debugging- und Entwicklungstools](#).

Code-Intelligenz-Probleme

- **Language Server startet nicht:** Überprüfen Sie, dass die Binärdatei installiert ist und in Ihrem `$PATH` verfügbar ist. Überprüfen Sie die Registerkarte `/plugin` Errors für Details.
- **Hohe Speichernutzung:** Language Server wie `rust-analyzer` und `pyright` können bei großen Projekten erhebliche Speichermengen verbrauchen. Wenn Sie Speicherprobleme haben, deaktivieren Sie das Plugin mit `/plugin disable <plugin-name>` und verlassen Sie sich stattdessen auf Claudes integrierte Suchtools.
- **Falsch positive Diagnosen in Monorepos:** Language Server können ungelöste Importfehler für interne Pakete melden, wenn der Arbeitsbereich nicht richtig konfiguriert ist. Diese beeinflussen nicht Claudes Fähigkeit, Code zu bearbeiten.

Nächste Schritte

- **Erstellen Sie Ihre eigenen Plugins:** Siehe [Plugins](#), um skills, agents und hooks zu erstellen
- **Erstellen Sie einen Marktplatz:** Siehe [Erstellen Sie einen Plugin-Marktplatz](#), um Plugins an Ihr Team oder Ihre Community zu verteilen
- **Technische Referenz:** Siehe [Plugins-Referenz](#) für vollständige Spezifikationen

Part 6: Advanced & Automation

Benutzerdefinierte Subagenten erstellen

Erstellen und verwenden Sie spezialisierte KI-Subagenten in Claude Code für aufgabenspezifische Workflows und verbesserte Kontextverwaltung.

Subagenten sind spezialisierte KI-Assistenten, die bestimmte Arten von Aufgaben bearbeiten. Jeder Subagent läuft in seinem eigenen Kontextfenster mit einem benutzerdefinierten Systemprompt, spezifischem Werkzeugzugriff und unabhängigen Berechtigungen. Wenn Claude auf eine Aufgabe trifft, die der Beschreibung eines Subagenten entspricht, delegiert es an diesen Subagenten, der unabhängig arbeitet und Ergebnisse zurückgibt.

Note:

Wenn Sie mehrere Agenten benötigen, die parallel arbeiten und miteinander kommunizieren, siehe stattdessen [Agent-Teams](#). Subagenten arbeiten innerhalb einer einzelnen Sitzung; Agent-Teams koordinieren über separate Sitzungen hinweg.

Subagenten helfen Ihnen:

- **Kontext bewahren**, indem Sie Exploration und Implementierung aus Ihrer Hauptkonversation heraushalten
- **Einschränkungen durchsetzen**, indem Sie begrenzen, welche Werkzeuge ein Subagent verwenden kann
- **Konfigurationen wiederverwenden** über Projekte hinweg mit Subagenten auf Benutzerebene
- **Verhalten spezialisieren** mit fokussierten Systemprompts für spezifische Domänen
- **Kosten kontrollieren**, indem Sie Aufgaben an schnellere, günstigere Modelle wie Haiku weiterleiten

Claude verwendet die Beschreibung jedes Subagenten, um zu entscheiden, wann Aufgaben delegiert werden. Wenn Sie einen Subagenten erstellen, schreiben Sie eine klare Beschreibung, damit Claude weiß, wann er ihn verwenden soll.

Claude Code enthält mehrere integrierte Subagenten wie **Explore**, **Plan** und **general-purpose**. Sie können auch benutzerdefinierte Subagenten erstellen, um spezifische Aufgaben zu bearbeiten. Diese Seite behandelt die [integrierten Subagenten](#), [wie Sie Ihre eigenen erstellen](#), [vollständige Konfigurationsoptionen](#), [Muster für die Arbeit mit Subagenten](#) und [Beispiel-Subagenten](#).

Integrierte Subagenten

Claude Code enthält integrierte Subagenten, die Claude automatisch bei Bedarf verwendet. Jeder erbt die Berechtigungen der übergeordneten Konversation mit zusätzlichen Werkzeugbeschränkungen.

Explore

Ein schneller, schreibgeschützter Agent, der für die Suche und Analyse von Codebases optimiert ist.

- **Modell:** Haiku (schnell, niedrige Latenz)
- **Werkzeuge:** Schreibgeschützte Werkzeuge (kein Zugriff auf Write- und Edit-Werkzeuge)
- **Zweck:** Dateiermittlung, Codesuche, Codebase-Exploration

Claude delegiert an Explore, wenn es eine Codebase durchsuchen oder verstehen muss, ohne Änderungen vorzunehmen. Dies hält Explorationsergebnisse aus Ihrem Hauptkonversationskontext heraus.

Beim Aufrufen von Explore gibt Claude eine Gründlichkeitsstufe an: **quick** für gezielte Lookups, **medium** für ausgewogene Exploration oder **very thorough** für umfassende Analyse.

Plan

Ein Forschungsagent, der während des [Plan-Modus](#) verwendet wird, um Kontext zu sammeln, bevor ein Plan präsentiert wird.

- **Modell:** Erbt von Hauptkonversation
- **Werkzeuge:** Schreibgeschützte Werkzeuge (kein Zugriff auf Write- und Edit-Werkzeuge)
- **Zweck:** Codebase-Forschung für Planung

Wenn Sie sich im Plan-Modus befinden und Claude Ihre Codebase verstehen muss, delegiert es die Forschung an den Plan-Subagenten. Dies verhindert unendliche Verschachtelung (Subagenten können keine anderen Subagenten spawnen), während dennoch notwendiger Kontext gesammelt wird.

General-purpose

Ein fähiger Agent für komplexe, mehrstufige Aufgaben, die sowohl Exploration als auch Aktion erfordern.

- **Modell:** Erbt von Hauptkonversation
- **Werkzeuge:** Alle Werkzeuge
- **Zweck:** Komplexe Forschung, mehrstufige Operationen, Codeänderungen

Claude delegiert an general-purpose, wenn die Aufgabe sowohl Exploration als auch Änderung, komplexes Denken zur Interpretation von Ergebnissen oder mehrere abhängige Schritte erfordert.

Other

Claude Code enthält zusätzliche Hilfigenten für spezifische Aufgaben. Diese werden normalerweise automatisch aufgerufen, daher müssen Sie sie nicht direkt verwenden.

Agent	Modell	Wann Claude ihn verwendet
Bash	Erbt	Ausführung von Terminalkommandos in einem separaten Kontext
statusline-setup	Sonnet	Wenn Sie <code>/statusline</code> ausführen, um Ihre Statuszeile zu konfigurieren
Claude Code Guide	Haiku	Wenn Sie Fragen zu Claude Code-Funktionen stellen

Über diese integrierten Subagenten hinaus können Sie Ihre eigenen mit benutzerdefinierten Prompts, Werkzeugbeschränkungen, Berechtigungsmodi, hooks und Skills erstellen. Die folgenden Abschnitte zeigen, wie Sie anfangen und Subagenten anpassen.

Schnellstart: Erstellen Sie Ihren ersten Subagenten

Subagenten werden in Markdown-Dateien mit YAML-Frontmatter definiert. Sie können sie [manuell erstellen](#) oder den `/agents`-Befehl verwenden.

Diese Anleitung führt Sie durch die Erstellung eines Subagenten auf Benutzerebene mit dem `/agent`-Befehl. Der Subagent überprüft Code und schlägt Verbesserungen für die Codebase vor.

Step 1: Öffnen Sie die Subagenten-Schnittstelle

In Claude Code führen Sie aus:

```
/agents
```

Step 2: Erstellen Sie einen neuen Agenten auf Benutzerebene

Wählen Sie **Create new agent**, dann wählen Sie **User-level**. Dies speichert den Subagenten in `~/.claude/agents/`, sodass er in allen Ihren Projekten verfügbar ist.

Step 3: Mit Claude generieren

Wählen Sie **Generate with Claude**. Wenn Sie dazu aufgefordert werden, beschreiben Sie den Subagenten:

```
A code improvement agent that scans files and suggests improvements  
for readability, performance, and best practices. It should explain  
each issue, show the current code, and provide an improved version.
```

Claude generiert den Systemprompt und die Konfiguration. Drücken Sie **e**, um ihn in Ihrem Editor zu öffnen, wenn Sie ihn anpassen möchten.

Step 4: Wählen Sie Werkzeuge

Für einen schreibgeschützten Reviewer deselektieren Sie alles außer **Read-only tools**. Wenn Sie alle Werkzeuge ausgewählt lassen, erbt der Subagent alle Werkzeuge, die der Hauptkonversation zur Verfügung stehen.

Step 5: Wählen Sie Modell

Wählen Sie, welches Modell der Subagent verwendet. Für diesen Beispielagenten wählen Sie **Sonnet**, das Fähigkeit und Geschwindigkeit für die Analyse von Codemustern ausgleicht.

Step 6: Wählen Sie eine Farbe

Wählen Sie eine Hintergrundfarbe für den Subagenten. Dies hilft Ihnen, zu identifizieren, welcher Subagent in der Benutzeroberfläche ausgeführt wird.

Step 7: Speichern und testen Sie

Speichern Sie den Subagenten. Er ist sofort verfügbar (kein Neustart erforderlich). Testen Sie ihn:

Use the code-improver agent to suggest improvements in this project

Claude delegiert an Ihren neuen Subagenten, der die Codebase durchsucht und Verbesserungsvorschläge zurückgibt.

Sie haben jetzt einen Subagenten, den Sie in jedem Projekt auf Ihrem Computer verwenden können, um Codebases zu analysieren und Verbesserungen vorzuschlagen.

Sie können Subagenten auch manuell als Markdown-Dateien erstellen, sie über CLI-Flags definieren oder sie über Plugins verteilen. Die folgenden Abschnitte behandeln alle Konfigurationsoptionen.

Konfigurieren Sie Subagenten

Verwenden Sie den `/agents`-Befehl

Der `/agents`-Befehl bietet eine interaktive Schnittstelle zur Verwaltung von Subagenten. Führen Sie `/agents` aus, um:

- Alle verfügbaren Subagenten anzuzeigen (integriert, Benutzer, Projekt und Plugin)
- Neue Subagenten mit geführtem Setup oder Claude-Generierung zu erstellen
- Vorhandene Subagenten-Konfiguration und Werkzeugzugriff zu bearbeiten
- Benutzerdefinierte Subagenten zu löschen
- Zu sehen, welche Subagenten aktiv sind, wenn Duplikate vorhanden sind

Dies ist die empfohlene Methode zum Erstellen und Verwalten von Subagenten. Für manuelle Erstellung oder Automatisierung können Sie auch Subagenten-Dateien direkt hinzufügen.

Um alle konfigurierten Subagenten von der Befehlszeile aus ohne Starten einer interaktiven Sitzung aufzulisten, führen Sie `claude agents` aus. Dies zeigt Agenten gruppiert nach Quelle an und gibt an, welche durch höherpriorisierte Definitionen überschrieben werden.

Wählen Sie den Subagenten-Umfang

Subagenten sind Markdown-Dateien mit YAML-Frontmatter. Speichern Sie sie an verschiedenen Orten je nach Umfang. Wenn mehrere Subagenten denselben Namen haben, gewinnt der Ort mit höherer Priorität.

Ort	Umfang	Priorität	Wie zu erstellen
<code>--agents</code> CLI-Flag	Aktuelle Sitzung	1 (höchste)	JSON beim Starten von Claude Code übergeben
<code>.claude/agents/</code>	Aktuelles Projekt	2	Interaktiv oder manuell
<code>~/.claude/agents/</code>	Alle Ihre Projekte	3	Interaktiv oder manuell
Plugin-Verzeichnis <code>agents/</code>	Wo Plugin aktiviert ist	4 (niedrigste)	Installiert mit Plugins

Projekt-Subagenten (`.claude/agents/`) sind ideal für Subagenten, die spezifisch für eine Codebase sind. Checken Sie sie in die Versionskontrolle ein, damit Ihr Team sie gemeinsam verwenden und verbessern kann.

Benutzer-Subagenten (`~/.claude/agents/`) sind persönliche Subagenten, die in allen Ihren Projekten verfügbar sind.

CLI-definierte Subagenten werden als JSON beim Starten von Claude Code übergeben. Sie existieren nur für diese Sitzung und werden nicht auf der Festplatte gespeichert, was sie für schnelle Tests oder Automatisierungsskripte nützlich macht:

```
claude --agents '{
  "code-reviewer": {
    "description": "Expert code reviewer. Use proactively after code changes.",
    "prompt": "You are a senior code reviewer. Focus on code quality, security,
and best practices.",
    "tools": ["Read", "Grep", "Glob", "Bash"],
    "model": "sonnet"
  }
}'
```

Das `--agents` -Flag akzeptiert JSON mit denselben [Frontmatter](#)-Feldern wie dateibasierte Subagenten: `description`, `prompt`, `tools`, `disallowedTools`, `model`, `permissionMode`, `mcpServers`, `hooks`, `maxTurns`, `skills` und `memory`. Verwenden Sie `prompt` für den Systemprompt, äquivalent zum Markdown-Body in dateibasierten Subagenten. Siehe die [CLI-Referenz](#) für das vollständige JSON-Format.

Plugin-Subagenten stammen von [Plugins](#), die Sie installiert haben. Sie erscheinen in `/agents` neben Ihren benutzerdefinierten Subagenten. Siehe die [Plugin-Komponenten-Referenz](#) für Details zum Erstellen von Plugin-Subagenten.

Schreiben Sie Subagenten-Dateien

Subagenten-Dateien verwenden YAML-Frontmatter für die Konfiguration, gefolgt vom Systemprompt in Markdown:

Note:

Subagenten werden beim Sitzungsstart geladen. Wenn Sie einen Subagenten durch manuelles Hinzufügen einer Datei erstellen, starten Sie Ihre Sitzung neu oder verwenden Sie `/agents`, um ihn sofort zu laden.

```
---
name: code-reviewer
description: Reviews code for quality and best practices
tools: Read, Glob, Grep
model: sonnet
---

You are a code reviewer. When invoked, analyze the code and provide
specific, actionable feedback on quality, security, and best practices.
```

Das Frontmatter definiert die Metadaten und Konfiguration des Subagenten. Der Body wird zum Systemprompt, der das Verhalten des Subagenten leitet. Subagenten erhalten nur diesen Systemprompt (plus grundlegende Umgebungsdetails wie Arbeitsverzeichnis), nicht den vollständigen Claude Code-Systemprompt.

Unterstützte Frontmatter-Felder

Die folgenden Felder können im YAML-Frontmatter verwendet werden. Nur `name` und `description` sind erforderlich.

Feld	Erforderlich	Beschreibung
<code>name</code>	Ja	Eindeutige Kennung mit Kleinbuchstaben und Bindestrichen

Feld	Erforderlich	Beschreibung
<code>description</code>	Ja	Wann Claude an diesen Subagenten delegieren sollte
<code>tools</code>	Nein	Werkzeuge , die der Subagent verwenden kann. Erbt alle Werkzeuge, wenn weggelassen
<code>disallowedTools</code>	Nein	Werkzeuge zu verweigern, entfernt aus geerbter oder angegebener Liste
<code>model</code>	Nein	Modell zu verwenden: <code>sonnet</code> , <code>opus</code> , <code>haiku</code> oder <code>inherit</code> . Standard ist <code>inherit</code>
<code>permissionMode</code>	Nein	Berechtigungsmodus : <code>default</code> , <code>acceptEdits</code> , <code>dontAsk</code> , <code>bypassPermissions</code> oder <code>plan</code>
<code>maxTurns</code>	Nein	Maximale Anzahl von agentengesteuerten Umdrehungen, bevor der Subagent stoppt
<code>skills</code>	Nein	Skills zum Laden in den Kontext des Subagenten beim Start. Der vollständige Skill-Inhalt wird eingespritzt, nicht nur zur Invokation verfügbar gemacht. Subagenten erben Skills nicht von der übergeordneten Konversation

Feld	Erforderlich	Beschreibung
<code>mcpServers</code>	Nein	MCP-Server verfügbar für diesen Subagenten. Jeder Eintrag ist entweder ein Servername, der auf einen bereits konfigurierten Server verweist (z. B. <code>"slack"</code>), oder eine Inline-Definition mit dem Servernamen als Schlüssel und einer vollständigen MCP-Server-Konfiguration als Wert
<code>hooks</code>	Nein	Lifecycle-Hooks mit Umfang auf diesen Subagenten
<code>memory</code>	Nein	Persistenter Speicherumfang : <code>user</code> , <code>project</code> oder <code>local</code> . Ermöglicht sitzungsübergreifendes Lernen
<code>background</code>	Nein	Auf <code>true</code> setzen, um diesen Subagenten immer als Hintergrundaufgabe auszuführen. Standard: <code>false</code>
<code>isolation</code>	Nein	Auf <code>worktree</code> setzen, um den Subagenten in einem temporären Git-Worktree auszuführen, was ihm eine isolierte Kopie des Repositoriums gibt. Der Worktree wird automatisch bereinigt, wenn der Subagent keine Änderungen vornimmt

Wählen Sie ein Modell

Das `model` -Feld steuert, welches [KI-Modell](#) der Subagent verwendet:

- **Modell-Alias:** Verwenden Sie einen der verfügbaren Aliase: `sonnet` , `opus` oder `haiku`
- **inherit:** Verwenden Sie dasselbe Modell wie die Hauptkonversation
- **Weggelassen:** Wenn nicht angegeben, wird standardmäßig `inherit` verwendet (verwendet dasselbe Modell wie die Hauptkonversation)

Kontrollieren Sie Subagenten-Fähigkeiten

Sie können kontrollieren, was Subagenten tun können, durch Werkzeugzugriff, Berechtigungsmodi und bedingte Regeln.

Verfügbare Werkzeuge

Subagenten können alle [internen Werkzeuge](#) von Claude Code verwenden. Standardmäßig erben Subagenten alle Werkzeuge von der Hauptkonversation, einschließlich MCP-Werkzeuge.

Um Werkzeuge einzuschränken, verwenden Sie das `tools` -Feld (Allowlist) oder `disallowedTools` -Feld (Denylist):

```
---
name: safe-researcher
description: Research agent with restricted capabilities
tools: Read, Grep, Glob, Bash
disallowedTools: Write, Edit
---
```

Beschränken Sie, welche Subagenten spawned werden können

Wenn ein Agent als Hauptthread mit `claude --agent` läuft, kann er Subagenten mit dem Agent-Werkzeug spawnen. Um zu beschränken, welche Subagenten-Typen er spawnen kann, verwenden Sie die `Agent(agent_type)` -Syntax im `tools` -Feld.

Note:

In Version 2.1.63 wurde das Task-Werkzeug in Agent umbenannt. Vorhandene `Task(...)` -Verweise in Einstellungen und Agent-Definitionen funktionieren weiterhin als Aliase.

```
---
name: coordinator
description: Coordinates work across specialized agents
tools: Agent(worker, researcher), Read, Bash
---
```

Dies ist eine Allowlist: Nur die `worker` - und `researcher` -Subagenten können gespawnt werden. Wenn der Agent versucht, einen anderen Typ zu spawnen, schlägt die Anfrage fehl und der Agent sieht nur die zulässigen Typen in seinem Prompt. Um bestimmte Agenten zu blockieren und alle anderen zuzulassen, verwenden Sie stattdessen `permissions.deny`.

Um das Spawnen eines beliebigen Subagenten ohne Einschränkungen zu ermöglichen, verwenden Sie `Agent` ohne Klammern:

```
tools: Agent, Read, Bash
```

Wenn `Agent` vollständig aus der `tools` -Liste weggelassen wird, kann der Agent keine Subagenten spawnen. Diese Einschränkung gilt nur für Agenten, die als Hauptthread mit `claude --agent` laufen. Subagenten können keine anderen Subagenten spawnen, daher hat `Agent(agent_type)` keine Auswirkung in Subagenten-Definitionen.

Berechtigungsmodi

Das `permissionMode` -Feld steuert, wie der Subagent Berechtigungsaufforderungen bearbeitet. Subagenten erben den Berechtigungskontext von der Hauptkonversation, können aber den Modus überschreiben.

Modus	Verhalten
<code>default</code>	Standardberechtigungsprüfung mit Aufforderungen
<code>acceptEdits</code>	Automatische Akzeptanz von Dateibearbeitungen
<code>dontAsk</code>	Automatische Ablehnung von Berechtigungsaufforderungen (explizit zulässige Werkzeuge funktionieren weiterhin)
<code>bypassPermissions</code>	Alle Berechtigungsprüfungen überspringen
<code>plan</code>	Plan-Modus (schreibgeschützte Exploration)

Warning:

Verwenden Sie `bypassPermissions` mit Vorsicht. Es überspringt alle Berechtigungsprüfungen und ermöglicht dem Subagenten, jeden Vorgang ohne Genehmigung auszuführen.

Wenn das übergeordnete Element `bypassPermissions` verwendet, hat dies Vorrang und kann nicht überschrieben werden.

Laden Sie Skills in Subagenten vor

Verwenden Sie das `skills`-Feld, um Skill-Inhalte beim Start in den Kontext eines Subagenten einzuspeisen. Dies gibt dem Subagenten Domänenwissen, ohne dass er Skills während der Ausführung entdecken und laden muss.

```
---
name: api-developer
description: Implement API endpoints following team conventions
skills:
  - api-conventions
  - error-handling-patterns
---

Implement API endpoints. Follow the conventions and patterns from the preloaded skills.
```

Der vollständige Inhalt jedes Skills wird in den Kontext des Subagenten eingespritzt, nicht nur zur Invokation verfügbar gemacht. Subagenten erben Skills nicht von der übergeordneten Konversation; Sie müssen sie explizit auflisten.

Note:

Dies ist das Gegenteil von [Ausführen eines Skills in einem Subagenten](#). Mit `skills` in einem Subagenten kontrolliert der Subagent den Systemprompt und lädt Skill-Inhalte. Mit `context: fork` in einem Skill wird der Skill-Inhalt in den von Ihnen angegebenen Agent eingespritzt. Beide verwenden das gleiche zugrunde liegende System.

Aktivieren Sie persistenten Speicher

Das `memory` -Feld gibt dem Subagenten ein persistentes Verzeichnis, das über Konversationen hinweg bestehen bleibt. Der Subagent verwendet dieses Verzeichnis, um im Laufe der Zeit Wissen aufzubauen, wie z. B. Codebase-Muster, Debugging-Erkenntnisse und architektonische Entscheidungen.

```
---
name: code-reviewer
description: Reviews code for quality and best practices
memory: user
---
```

You are a code reviewer. As you review code, update your agent memory with patterns, conventions, and recurring issues you discover.

Wählen Sie einen Umfang basierend darauf, wie breit der Speicher angewendet werden sollte:

Umfang	Ort	Verwenden Sie, wenn
user	<code>~/.claude/agent-memory/<name-of-agent>/</code>	der Subagent Erkenntnisse über alle Projekte hinweg merken sollte
project	<code>.claude/agent-memory/<name-of-agent>/</code>	das Wissen des Subagenten projektspezifisch ist und über Versionskontrolle teilbar ist
local	<code>.claude/agent-memory-local/<name-of-agent>/</code>	das Wissen des Subagenten projektspezifisch ist, aber nicht in die Versionskontrolle eingecheckt werden sollte

Wenn der Speicher aktiviert ist:

- Der Systemprompt des Subagenten enthält Anweisungen zum Lesen und Schreiben in das Speicherverzeichnis.

- Der Systemprompt des Subagenten enthält auch die ersten 200 Zeilen von `MEMORY.md` im Speicherverzeichnis, mit Anweisungen zur Verwaltung von `MEMORY.md`, wenn es 200 Zeilen überschreitet.
- Read-, Write- und Edit-Werkzeuge werden automatisch aktiviert, damit der Subagent seine Speicherdateien verwalten kann.

Tipps zum persistenten Speicher

- `user` ist der empfohlene Standard-Umfang. Verwenden Sie `project` oder `local`, wenn das Wissen des Subagenten nur für eine bestimmte Codebase relevant ist.
- Bitten Sie den Subagenten, seinen Speicher zu konsultieren, bevor er mit der Arbeit beginnt: “Review this PR, and check your memory for patterns you’ve seen before.”
- Bitten Sie den Subagenten, seinen Speicher nach Abschluss einer Aufgabe zu aktualisieren: “Now that you’re done, save what you learned to your memory.” Im Laufe der Zeit baut dies eine Wissensdatenbank auf, die den Subagenten effektiver macht.
- Fügen Sie Speicheranweisungen direkt in die Markdown-Datei des Subagenten ein, damit er proaktiv seine eigene Wissensdatenbank verwaltet:

```
Update your agent memory as you discover codepaths, patterns, library
locations, and key architectural decisions. This builds up institutional
knowledge across conversations. Write concise notes about what you found
and where.
```

Bedingte Regeln mit Hooks

Für dynamischere Kontrolle über die Werkzeugnutzung verwenden Sie

`PreToolUse`-Hooks, um Operationen vor ihrer Ausführung zu validieren. Dies ist nützlich, wenn Sie einige Operationen eines Werkzeugs zulassen müssen, während Sie andere blockieren.

Dieses Beispiel erstellt einen Subagenten, der nur schreibgeschützte Datenbankabfragen zulässt. Der `PreToolUse`-Hook führt das in `command` angegebene Skript vor jeder Bash-Befehlsausführung aus:

```

---
name: db-reader
description: Execute read-only database queries
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-readonly-query.sh"
---

```

Claude Code [übergibt Hook-Eingabe als JSON](#) über stdin an Hook-Befehle. Das Validierungsskript liest dieses JSON, extrahiert den Bash-Befehl und [beendet mit Code 2](#), um Schreiboperationen zu blockieren:

```

#!/bin/bash
## ./scripts/validate-readonly-query.sh

INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command // empty')

## Block SQL write operations (case-insensitive)
if echo "$COMMAND" | grep -iE '\b(INSERT|UPDATE|DELETE|DROP|CREATE|ALTER|TRUNCATE)\b' > /dev/null; then
    echo "Blocked: Only SELECT queries are allowed" >&2
    exit 2
fi

exit 0

```

Siehe [Hook-Eingabe](#) für das vollständige Eingabeschema und [Exit-Codes](#) für die Auswirkungen von Exit-Codes auf das Verhalten.

Deaktivieren Sie spezifische Subagenten

Sie können verhindern, dass Claude bestimmte Subagenten verwendet, indem Sie sie zum `deny`-Array in Ihren [Einstellungen](#) hinzufügen. Verwenden Sie das Format `Agent(subagent-name)`, wobei `subagent-name` dem `name`-Feld des Subagenten entspricht.

```
{
  "permissions": {
    "deny": ["Agent(Explore)", "Agent(my-custom-agent)"]
  }
}
```

Dies funktioniert für integrierte und benutzerdefinierte Subagenten. Sie können auch das `--disallowedTools` CLI-Flag verwenden:

```
claude --disallowedTools "Agent(Explore)"
```

Siehe [Berechtigungsdokumentation](#) für weitere Details zu Berechtigungsregeln.

Definieren Sie Hooks für Subagenten

Subagenten können [Hooks](#) definieren, die während des Lebenszyklus des Subagenten ausgeführt werden. Es gibt zwei Möglichkeiten, Hooks zu konfigurieren:

1. **Im Frontmatter des Subagenten:** Definieren Sie Hooks, die nur ausgeführt werden, während dieser Subagent aktiv ist
2. **In `settings.json`:** Definieren Sie Hooks, die in der Hauptsitzung ausgeführt werden, wenn Subagenten starten oder stoppen

Hooks im Subagenten-Frontmatter

Definieren Sie Hooks direkt in der Markdown-Datei des Subagenten. Diese Hooks werden nur ausgeführt, während dieser spezifische Subagent aktiv ist, und werden bereinigt, wenn er beendet wird.

Alle [Hook-Ereignisse](#) werden unterstützt. Die häufigsten Ereignisse für Subagenten sind:

Ereignis	Matcher-Eingabe	Wann es ausgelöst wird
PreToolUse	Werkzeugname	Bevor der Subagent ein Werkzeug verwendet
PostToolUse	Werkzeugname	Nachdem der Subagent ein Werkzeug verwendet hat
Stop	(keine)	Wenn der Subagent beendet wird (zur Laufzeit in SubagentStop konvertiert)

Dieses Beispiel validiert Bash-Befehle mit dem PreToolUse -Hook und führt einen Linter nach Dateibearbeitungen mit PostToolUse aus:

```
---
name: code-reviewer
description: Review code changes with automatic linting
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-command.sh $TOOL_INPUT"
  PostToolUse:
    - matcher: "Edit|Write"
      hooks:
        - type: command
          command: "./scripts/run-linter.sh"
---
```

Stop -Hooks im Frontmatter werden automatisch in SubagentStop -Ereignisse konvertiert.

Hooks auf Projektebene für Subagenten-Ereignisse

Konfigurieren Sie Hooks in settings.json , die auf Subagenten-Lebenszyklus-Ereignisse in der Hauptsitzung reagieren.

Ereignis	Matcher-Eingabe	Wann es ausgelöst wird
<code>SubagentStart</code>	Agent-Typname	Wenn ein Subagent mit der Ausführung beginnt
<code>SubagentStop</code>	Agent-Typname	Wenn ein Subagent abgeschlossen ist

Beide Ereignisse unterstützen Matcher, um bestimmte Agent-Typen nach Name zu adressieren. Dieses Beispiel führt ein Setup-Skript nur aus, wenn der `db-agent`-Subagent startet, und ein Cleanup-Skript, wenn ein beliebiger Subagent stoppt:

```
{
  "hooks": {
    "SubagentStart": [
      {
        "matcher": "db-agent",
        "hooks": [
          { "type": "command", "command": "./scripts/setup-db-connection.sh" }
        ]
      }
    ],
    "SubagentStop": [
      {
        "hooks": [
          { "type": "command", "command": "./scripts/cleanup-db-connection.sh" }
        ]
      }
    ]
  }
}
```

Siehe [Hooks](#) für das vollständige Hook-Konfigurationsformat.

Arbeiten Sie mit Subagenten

Verstehen Sie automatische Delegation

Claude delegiert automatisch Aufgaben basierend auf der Aufgabenbeschreibung in Ihrer Anfrage, dem `description`-Feld in Subagenten-Konfigurationen und dem aktuellen Kontext. Um proaktive Delegation zu fördern, fügen Sie Phrasen wie “use proactively” in das `description`-Feld Ihres Subagenten ein.

Sie können auch einen bestimmten Subagenten explizit anfordern:

```
Use the test-runner subagent to fix failing tests
Have the code-reviewer subagent look at my recent changes
```

Führen Sie Subagenten im Vordergrund oder Hintergrund aus

Subagenten können im Vordergrund (blockierend) oder Hintergrund (gleichzeitig) ausgeführt werden:

- **Vordergrund-Subagenten** blockieren die Hauptkonversation bis zur Fertigstellung. Berechtigungsaufforderungen und Klarstellungsfragen (wie [AskUserQuestion](#)) werden an Sie weitergeleitet.
- **Hintergrund-Subagenten** laufen gleichzeitig, während Sie weiterarbeiten. Vor dem Start fordert Claude Code alle Werkzeugberechtigungen an, die der Subagent benötigt, um sicherzustellen, dass er die erforderlichen Genehmigungen im Voraus hat. Nach dem Start erbt der Subagent diese Berechtigungen und lehnt automatisch alles ab, was nicht vorab genehmigt wurde. Wenn ein Hintergrund-Subagent Klarstellungsfragen stellen muss, schlägt dieser Werkzeugaufruf fehl, aber der Subagent setzt fort.

Wenn ein Hintergrund-Subagent aufgrund fehlender Berechtigungen fehlschlägt, können Sie ihn [fortsetzen](#) im Vordergrund, um es mit interaktiven Aufforderungen erneut zu versuchen.

Claude entscheidet, ob Subagenten im Vordergrund oder Hintergrund ausgeführt werden, basierend auf der Aufgabe. Sie können auch:

- Claude bitten, “run this in the background”
- **Ctrl+B** drücken, um eine laufende Aufgabe in den Hintergrund zu verschieben

Um alle Hintergrund-Aufgaben-Funktionalität zu deaktivieren, setzen Sie die Umgebungsvariable `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` auf `1`. Siehe [Umgebungsvariablen](#).

Häufige Muster

Isolieren Sie hochvolumige Operationen

Eine der effektivsten Verwendungen für Subagenten ist die Isolierung von Operationen, die große Mengen an Ausgaben erzeugen. Das Ausführen von Tests, das Abrufen von Dokumentation oder die Verarbeitung von Protokolldateien kann erheblichen Kontext verbrauchen. Durch die Delegierung an einen Subagenten bleibt die ausführliche Ausgabe im Kontext des Subagenten, während nur die relevante Zusammenfassung zu Ihrer Hauptkonversation zurückkehrt.

```
Use a subagent to run the test suite and report only the failing tests with their error messages
```

Führen Sie parallele Forschung durch

Für unabhängige Untersuchungen spawnen Sie mehrere Subagenten, um gleichzeitig zu arbeiten:

```
Research the authentication, database, and API modules in parallel using separate subagents
```

Jeder Subagent erkundet seinen Bereich unabhängig, dann synthetisiert Claude die Erkenntnisse. Dies funktioniert am besten, wenn die Forschungspfade nicht voneinander abhängen.

Warning:

Wenn Subagenten abgeschlossen sind, kehren ihre Ergebnisse zu Ihrer Hauptkonversation zurück. Das Ausführen vieler Subagenten, die jeweils detaillierte Ergebnisse zurückgeben, kann erheblichen Kontext verbrauchen.

Für Aufgaben, die anhaltende Parallelität benötigen oder Ihr Kontextfenster überschreiten, geben [Agent-Teams](#) jedem Worker seinen eigenen unabhängigen Kontext.

Verketteten Sie Subagenten

Für mehrstufige Workflows bitten Sie Claude, Subagenten nacheinander zu verwenden. Jeder Subagent vervollständigt seine Aufgabe und gibt Ergebnisse an Claude zurück, das dann relevanten Kontext an den nächsten Subagenten übergibt.

Use the code-reviewer subagent to find performance issues, then use the optimizer subagent to fix them

Wählen Sie zwischen Subagenten und Hauptkonversation

Verwenden Sie die **Hauptkonversation**, wenn:

- Die Aufgabe häufiges Hin und Her oder iterative Verfeinerung benötigt
- Mehrere Phasen teilen erheblichen Kontext (Planung → Implementierung → Testen)
- Sie eine schnelle, gezielte Änderung vornehmen
- Latenz wichtig ist. Subagenten starten von vorne und benötigen möglicherweise Zeit, um Kontext zu sammeln

Verwenden Sie **Subagenten**, wenn:

- Die Aufgabe ausführliche Ausgabe erzeugt, die Sie nicht in Ihrem Hauptkontext benötigen
- Sie spezifische Werkzeugbeschränkungen oder Berechtigungen durchsetzen möchten
- Die Arbeit in sich geschlossen ist und eine Zusammenfassung zurückgeben kann

Erwägen Sie stattdessen [Skills](#), wenn Sie wiederverwendbare Prompts oder Workflows möchten, die im Hauptkonversationskontext ausgeführt werden, anstatt in isoliertem Subagenten-Kontext.

Für eine schnelle Frage zu etwas, das bereits in Ihrer Konversation ist, verwenden Sie [/btw](#) anstelle eines Subagenten. Es sieht Ihren vollständigen Kontext, hat aber keinen Werkzeugzugriff, und die Antwort wird verworfen, anstatt zur Historie hinzugefügt zu werden.

Note:

Subagenten können keine anderen Subagenten spawnen. Wenn Ihr Workflow verschachtelte Delegation erfordert, verwenden Sie [Skills](#) oder [verketten Sie Subagenten](#) von der Hauptkonversation.

Verwalten Sie Subagenten-Kontext

Setzen Sie Subagenten fort

Jede Subagenten-Invokation erstellt eine neue Instanz mit frischem Kontext. Um die Arbeit eines vorhandenen Subagenten fortzusetzen, anstatt von vorne zu beginnen, bitten Sie Claude, ihn fortzusetzen.

Fortgesetzte Subagenten behalten ihre vollständige Konversationshistorie, einschließlich aller vorherigen Werkzeugaufrufe, Ergebnisse und Überlegungen. Der Subagent setzt genau dort an, wo er gestoppt hat, anstatt von vorne zu beginnen.

Wenn ein Subagent abgeschlossen ist, erhält Claude seine Agent-ID. Um einen Subagenten fortzusetzen, bitten Sie Claude, die vorherige Arbeit fortzusetzen:

```
Use the code-reviewer subagent to review the authentication module
[Agent completes]

Continue that code review and now analyze the authorization logic
[Claude resumes the subagent with full context from previous conversation]
```

Sie können Claude auch nach der Agent-ID fragen, wenn Sie sie explizit referenzieren möchten, oder IDs in den Transkriptdateien unter `~/.claude/projects/{project}/{sessionId}/subagents/` finden. Jedes Transkript wird als `agent-{agentId}.jsonl` gespeichert.

Subagenten-Transkripte bleiben unabhängig von der Hauptkonversation bestehen:

- **Hauptkonversations-Komprimierung:** Wenn die Hauptkonversation komprimiert wird, sind Subagenten-Transkripte nicht betroffen. Sie werden in separaten Dateien gespeichert.
- **Sitzungs-Persistenz:** Subagenten-Transkripte bleiben innerhalb ihrer Sitzung bestehen. Sie können [einen Subagenten fortsetzen](#), nachdem Sie Claude Code neu gestartet haben, indem Sie dieselbe Sitzung fortsetzen.
- **Automatische Bereinigung:** Transkripte werden basierend auf der `cleanupPeriodDays`-Einstellung bereinigt (Standard: 30 Tage).

Auto-Komprimierung

Subagenten unterstützen automatische Komprimierung mit der gleichen Logik wie die Hauptkonversation. Standardmäßig wird die Auto-Komprimierung bei ungefähr 95 % Kapazität ausgelöst. Um die Komprimierung früher auszulösen, setzen Sie `CLAUDE_AUTOCOMPACT_PCT_OVERRIDE` auf einen niedrigeren Prozentsatz (z. B. `50`). Siehe [Umgebungsvariablen](#) für Details.

Komprimierungsereignisse werden in Subagenten-Transkriptdateien protokolliert:

```
{
  "type": "system",
  "subtype": "compact_boundary",
  "compactMetadata": {
    "trigger": "auto",
    "preTokens": 167189
  }
}
```

Der `preTokens`-Wert zeigt, wie viele Token vor der Komprimierung verwendet wurden.

Beispiel-Subagenten

Diese Beispiele demonstrieren effektive Muster für die Erstellung von Subagenten. Verwenden Sie sie als Ausgangspunkte oder generieren Sie eine angepasste Version mit Claude.

Tip:

Best Practices:

- **Entwerfen Sie fokussierte Subagenten:** Jeder Subagent sollte bei einer spezifischen Aufgabe hervorragend sein
- **Schreiben Sie detaillierte Beschreibungen:** Claude verwendet die Beschreibung, um zu entscheiden, wann delegiert werden soll
- **Begrenzen Sie den Werkzeugzugriff:** Gewähren Sie nur notwendige Berechtigungen für Sicherheit und Fokus
- **Checken Sie in die Versionskontrolle ein:** Teilen Sie Projekt-Subagenten mit Ihrem Team

Code-Reviewer

Ein schreibgeschützter Subagent, der Code überprüft, ohne ihn zu ändern. Dieses Beispiel zeigt, wie man einen fokussierten Subagenten mit begrenztem Werkzeugzugriff (kein Edit oder Write) und einem detaillierten Prompt entwirft, der genau angibt, worauf zu achten ist und wie die Ausgabe formatiert wird.

```
---  
name: code-reviewer  
description: Expert code review specialist. Proactively reviews code for quality,  
security, and maintainability. Use immediately after writing or modifying code.  
tools: Read, Grep, Glob, Bash  
model: inherit  
---
```

You are a senior code reviewer ensuring high standards of code quality and security.

When invoked:

1. Run git diff to see recent changes
2. Focus on modified files
3. Begin review immediately

Review checklist:

- Code is clear and readable
- Functions and variables are well-named
- No duplicated code
- Proper error handling
- No exposed secrets or API keys
- Input validation implemented
- Good test coverage
- Performance considerations addressed

Provide feedback organized by priority:

- Critical issues (must fix)
- Warnings (should fix)
- Suggestions (consider improving)

Include specific examples of how to fix issues.

Debugger

Ein Subagent, der sowohl Probleme analysieren als auch beheben kann. Im Gegensatz zum Code-Reviewer enthält dieser Edit, da das Beheben von Bugs die Änderung von Code erfordert. Der Prompt bietet einen klaren Workflow von der Diagnose zur Verifizierung.

```
---  
name: debugger  
description: Debugging specialist for errors, test failures, and unexpected  
behavior. Use proactively when encountering any issues.  
tools: Read, Edit, Bash, Grep, Glob  
---
```

You are an expert debugger specializing in root cause analysis.

When invoked:

1. Capture error message and stack trace
2. Identify reproduction steps
3. Isolate the failure location
4. Implement minimal fix
5. Verify solution works

Debugging process:

- Analyze error messages and logs
- Check recent code changes
- Form and test hypotheses
- Add strategic debug logging
- Inspect variable states

For each issue, provide:

- Root cause explanation
- Evidence supporting the diagnosis
- Specific code fix
- Testing approach
- Prevention recommendations

Focus on fixing the underlying issue, not the symptoms.

Data Scientist

Ein domänenspezifischer Subagent für Datenanalyse-Arbeiten. Dieses Beispiel zeigt, wie man Subagenten für spezialisierte Workflows außerhalb typischer Coding-Aufgaben erstellt. Es setzt explizit `model: sonnet` für fähigere Analysen.


```
---  
name: data-scientist  
description: Data analysis expert for SQL queries, BigQuery operations, and data  
insights. Use proactively for data analysis tasks and queries.  
tools: Bash, Read, Write  
model: sonnet  
---
```

You are a data scientist specializing in SQL and BigQuery analysis.

When invoked:

1. Understand the data analysis requirement
2. Write efficient SQL queries
3. Use BigQuery command line tools (bq) when appropriate
4. Analyze and summarize results
5. Present findings clearly

Key practices:

- Write optimized SQL queries with proper filters
- Use appropriate aggregations and joins
- Include comments explaining complex logic
- Format results for readability
- Provide data-driven recommendations

For each analysis:

- Explain the query approach
- Document any assumptions
- Highlight key findings
- Suggest next steps based on data

Always ensure queries are efficient and cost-effective.

Datenbankabfrage-Validator

Ein Subagent, der Bash-Zugriff zulässt, aber Befehle validiert, um nur schreibgeschützte SQL-Abfragen zu ermöglichen. Dieses Beispiel zeigt, wie man `PreToolUse`-Hooks für bedingte Validierung verwendet, wenn Sie feinere Kontrolle benötigen, als das `tools`-Feld bietet.

```

---
name: db-reader
description: Execute read-only database queries. Use when analyzing data or
generating reports.
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-readonly-query.sh"
---

```

You are a database analyst with read-only access. Execute SELECT queries to answer questions about the data.

When asked to analyze data:

1. Identify which tables contain the relevant data
2. Write efficient SELECT queries with appropriate filters
3. Present results clearly with context

You cannot modify data. If asked to INSERT, UPDATE, DELETE, or modify schema, explain that you only have read access.

Claude Code [übergibt Hook-Eingabe als JSON](#) über stdin an Hook-Befehle. Das Validierungsskript liest dieses JSON, extrahiert den auszuführenden Befehl und prüft ihn gegen eine Liste von SQL-Schreiboperationen. Wenn eine Schreiboperation erkannt wird, beendet das Skript mit [Code 2](#), um die Ausführung zu blockieren, und gibt eine Fehlermeldung an Claude über stderr zurück.

Erstellen Sie das Validierungsskript überall in Ihrem Projekt. Der Pfad muss dem `command`-Feld in Ihrer Hook-Konfiguration entsprechen:

```
#!/bin/bash
## Blocks SQL write operations, allows SELECT queries

## Read JSON input from stdin
INPUT=$(cat)

## Extract the command field from tool_input using jq
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command // empty')

if [ -z "$COMMAND" ]; then
    exit 0
fi

## Block write operations (case-insensitive)
if echo "$COMMAND" | grep -iE '\b(INSERT|UPDATE|DELETE|DROP|CREATE|ALTER|TRUNCATE|REPLACE|MERGE)\b' > /dev/null; then
    echo "Blocked: Write operations not allowed. Use SELECT queries only." >&2
    exit 2
fi

exit 0
```

Machen Sie das Skript ausführbar:

```
chmod +x ./scripts/validate-readonly-query.sh
```

Der Hook empfängt JSON über stdin mit dem Bash-Befehl in `tool_input.command`. Exit-Code 2 blockiert die Operation und leitet die Fehlermeldung an Claude weiter. Siehe [Hooks](#) für Details zu Exit-Codes und [Hook-Eingabe](#) für das vollständige Eingabeschema.

Nächste Schritte

Jetzt, da Sie Subagenten verstehen, erkunden Sie diese verwandten Funktionen:

- [Verteilen Sie Subagenten mit Plugins](#), um Subagenten über Teams oder Projekte hinweg zu teilen
- [Führen Sie Claude Code programmgesteuert aus](#) mit dem Agent SDK für CI/CD und Automatisierung

- [Verwenden Sie MCP-Server](#), um Subagenten Zugriff auf externe Werkzeuge und Daten zu geben

Orchestrieren Sie Teams von Claude Code-Sitzungen

Koordinieren Sie mehrere Claude Code-Instanzen, die zusammen als Team arbeiten, mit gemeinsamen Aufgaben, Messaging zwischen Agenten und zentraler Verwaltung.

Warning:

Agent-Teams sind experimentell und standardmäßig deaktiviert. Aktivieren Sie sie, indem Sie `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` zu Ihrer `settings.json` oder Umgebung hinzufügen. Agent-Teams haben [bekannte Einschränkungen](#) bezüglich Sitzungswiederaufnahme, Aufgabenkoordination und Abschaltungsverhalten.

Agent-Teams ermöglichen es Ihnen, mehrere Claude Code-Instanzen zu koordinieren, die zusammenarbeiten. Eine Sitzung fungiert als Team-Lead und koordiniert die Arbeit, weist Aufgaben zu und synthetisiert Ergebnisse. Teammates arbeiten unabhängig, jeder in seinem eigenen Kontextfenster, und kommunizieren direkt miteinander.

Im Gegensatz zu [subagents](#), die innerhalb einer einzelnen Sitzung ausgeführt werden und nur an den Hauptagenten berichten können, können Sie auch direkt mit einzelnen Teammates interagieren, ohne den Lead einzubeziehen.

Note:

Agent-Teams erfordern Claude Code v2.1.32 oder später. Überprüfen Sie Ihre Version mit `claude --version`.

Diese Seite behandelt:

- [Wann Agent-Teams verwendet werden](#), einschließlich der besten Anwendungsfälle und wie sie sich mit subagents vergleichen
- [Starten eines Teams](#)
- [Kontrolle von Teammates](#), einschließlich Anzeigemodi, Aufgabenzuweisung und Delegation
- [Best Practices für parallele Arbeit](#)

Wann Agent-Teams verwendet werden

Agent-Teams sind am effektivsten für Aufgaben, bei denen parallele Exploration echten Wert bietet. Siehe [Anwendungsbeispiele](#) für vollständige Szenarien. Die stärksten Anwendungsfälle sind:

- **Recherche und Überprüfung:** mehrere Teammates können verschiedene Aspekte eines Problems gleichzeitig untersuchen und dann ihre Erkenntnisse austauschen und in Frage stellen
- **Neue Module oder Features:** Teammates können jeweils ein separates Stück besitzen, ohne sich gegenseitig zu behindern
- **Debugging mit konkurrierenden Hypothesen:** Teammates testen verschiedene Theorien parallel und konvergieren schneller zur Antwort
- **Schichtenübergreifende Koordination:** Änderungen, die Frontend, Backend und Tests umfassen, jeweils von einem anderen Teammate verwaltet

Agent-Teams fügen Koordinationsaufwand hinzu und verwenden deutlich mehr Tokens als eine einzelne Sitzung. Sie funktionieren am besten, wenn Teammates unabhängig arbeiten können. Für sequenzielle Aufgaben, Bearbeitungen in derselben Datei oder Arbeit mit vielen Abhängigkeiten sind eine einzelne Sitzung oder [subagents](#) effektiver.

Vergleich mit subagents

Sowohl Agent-Teams als auch [subagents](#) ermöglichen es Ihnen, Arbeit zu parallelisieren, aber sie funktionieren unterschiedlich. Wählen Sie basierend darauf, ob Ihre Worker miteinander kommunizieren müssen:

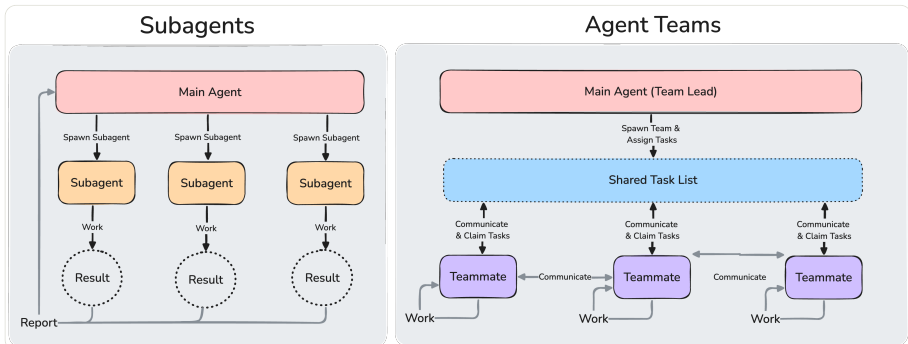


Diagramm zum Vergleich von Subagent- und Agent-Team-Architekturen. Subagents werden vom Hauptagenten erzeugt, führen Arbeit aus und berichten Ergebnisse zurück. Agent-Teams koordinieren sich über eine gemeinsame Aufgabenliste, wobei Teammates direkt miteinander kommunizieren.

	Subagents	Agent-Teams
Kontext	Eigenes Kontextfenster; Ergebnisse kehren zum Aufrufer zurück	Eigenes Kontextfenster; vollständig unabhängig
Kommunikation	Berichte Ergebnisse nur an den Hauptagenten zurück	Teammates senden sich gegenseitig direkt Nachrichten
Koordination	Hauptagent verwaltet alle Arbeiten	Gemeinsame Aufgabenliste mit Selbstkoordination
Am besten für	Fokussierte Aufgaben, bei denen nur das Ergebnis zählt	Komplexe Arbeit, die Diskussion und Zusammenarbeit erfordert
Token-Kosten	Niedriger: Ergebnisse werden zum Hauptkontext zusammengefasst	Höher: jeder Teammate ist eine separate Claude-Instanz

Verwenden Sie subagents, wenn Sie schnelle, fokussierte Worker benötigen, die berichten. Verwenden Sie Agent-Teams, wenn Teammates Erkenntnisse austauschen, sich gegenseitig in Frage stellen und selbst koordinieren müssen.

Agent-Teams aktivieren

Agent-Teams sind standardmäßig deaktiviert. Aktivieren Sie sie, indem Sie die Umgebungsvariable `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` auf `1` setzen, entweder in Ihrer Shell-Umgebung oder über [settings.json](#):

```
{
  "env": {
    "CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS": "1"
  }
}
```

Starten Sie Ihr erstes Agent-Team

Nach der Aktivierung von Agent-Teams teilen Sie Claude mit, dass Sie ein Agent-Team erstellen möchten, und beschreiben Sie die Aufgabe und die gewünschte Teamstruktur in natürlicher Sprache. Claude erstellt das Team, erzeugt Teammates und koordiniert die Arbeit basierend auf Ihrem Prompt.

Dieses Beispiel funktioniert gut, weil die drei Rollen unabhängig sind und das Problem erkunden können, ohne aufeinander zu warten:

```
I'm designing a CLI tool that helps developers track TODO comments across
their codebase. Create an agent team to explore this from different angles: one
teammate on UX, one on technical architecture, one playing devil's advocate.
```

Von dort aus erstellt Claude ein Team mit einer [gemeinsamen Aufgabenliste](#), erzeugt Teammates für jede Perspektive, lässt sie das Problem erkunden, synthetisiert Erkenntnisse und versucht, [das Team zu bereinigen](#), wenn es fertig ist.

Das Terminal des Leads listet alle Teammates und ihre aktuelle Arbeit auf. Verwenden Sie Shift+Down, um durch Teammates zu wechseln und ihnen direkt Nachrichten zu senden. Nach dem letzten Teammate wickelt Shift+Down zum Lead zurück.

Wenn Sie jeden Teammate in seinem eigenen Split-Pane haben möchten, siehe [Wählen Sie einen Anzeigemodus](#).

Kontrolle Ihres Agent-Teams

Teilen Sie dem Lead in natürlicher Sprache mit, was Sie möchten. Es kümmert sich um Teamkoordination, Aufgabenzuweisung und Delegation basierend auf Ihren Anweisungen.

Wählen Sie einen Anzeigemodus

Agent-Teams unterstützen zwei Anzeigemodi:

- **In-Process:** alle Teammates laufen in Ihrem Hauptterminal. Verwenden Sie Shift+Down, um durch Teammates zu wechseln und geben Sie ein, um ihnen direkt eine Nachricht zu senden. Funktioniert in jedem Terminal, keine zusätzliche Einrichtung erforderlich.
- **Split Panes:** jeder Teammate erhält seinen eigenen Pane. Sie können die Ausgabe aller gleichzeitig sehen und in einen Pane klicken, um direkt zu interagieren. Erfordert tmux oder iTerm2.

Note:

`tmux` hat bekannte Einschränkungen auf bestimmten Betriebssystemen und funktioniert traditionell am besten auf macOS. Die Verwendung von `tmux -CC` in iTerm2 ist der empfohlene Einstiegspunkt in `tmux`.

Der Standard ist `"auto"`, der Split Panes verwendet, wenn Sie bereits in einer tmux-Sitzung ausgeführt werden, und ansonsten In-Process. Die Einstellung `"tmux"` aktiviert den Split-Pane-Modus und erkennt automatisch, ob tmux oder iTerm2 basierend auf Ihrem Terminal verwendet werden soll. Um zu überschreiben, setzen Sie `teammateMode` in Ihrer [settings.json](#):

```
{
  "teammateMode": "in-process"
}
```

Um den In-Process-Modus für eine einzelne Sitzung zu erzwingen, übergeben Sie ihn als Flag:

```
claude --teammate-mode in-process
```

Der Split-Pane-Modus erfordert entweder [tmux](#) oder iTerm2 mit der [it2 CLI](#). Zur manuellen Installation:

- **tmux**: installieren Sie über den Paketmanager Ihres Systems. Siehe das [tmux Wiki](#) für plattformspezifische Anweisungen.
- **iTerm2**: installieren Sie die [it2 CLI](#), aktivieren Sie dann die Python-API in **iTerm2** → **Settings** → **General** → **Magic** → **Enable Python API**.

Geben Sie Teammates und Modelle an

Claude entscheidet die Anzahl der zu erzeugenden Teammates basierend auf Ihrer Aufgabe, oder Sie können genau angeben, was Sie möchten:

```
Create a team with 4 teammates to refactor these modules in parallel.
Use Sonnet for each teammate.
```

Genehmigung von Plänen für Teammates erforderlich

Für komplexe oder riskante Aufgaben können Sie verlangen, dass Teammates planen, bevor sie implementieren. Der Teammate arbeitet im schreibgeschützten Plan-Modus, bis der Lead seinen Ansatz genehmigt:

```
Spawn an architect teammate to refactor the authentication module.  
Require plan approval before they make any changes.
```

Wenn ein Teammate die Planung abgeschlossen hat, sendet er eine Genehmigungsanfrage an den Lead. Der Lead überprüft den Plan und genehmigt ihn entweder oder lehnt ihn mit Feedback ab. Bei Ablehnung bleibt der Teammate im Plan-Modus, überarbeitet basierend auf dem Feedback und reicht erneut ein. Nach der Genehmigung beendet der Teammate den Plan-Modus und beginnt mit der Implementierung.

Der Lead trifft Genehmigungsentscheidungen autonom. Um das Urteil des Leads zu beeinflussen, geben Sie ihm Kriterien in Ihrem Prompt, wie z. B. „genehmigen Sie nur Pläne, die Testabdeckung enthalten“ oder „lehnen Sie Pläne ab, die das Datenbankschema ändern“.

Sprechen Sie direkt mit Teammates

Jeder Teammate ist eine vollständige, unabhängige Claude Code-Sitzung. Sie können jedem Teammate direkt eine Nachricht senden, um zusätzliche Anweisungen zu geben, Folgefragen zu stellen oder seinen Ansatz umzuleiten.

- **In-Process-Modus:** Verwenden Sie Shift+Down, um durch Teammates zu wechseln, geben Sie dann ein, um ihnen eine Nachricht zu senden. Drücken Sie Enter, um die Sitzung eines Teammates anzuzeigen, dann Escape, um ihren aktuellen Turn zu unterbrechen. Drücken Sie Ctrl+T, um die Aufgabenliste umzuschalten.
- **Split-Pane-Modus:** klicken Sie in den Pane eines Teammates, um direkt mit seiner Sitzung zu interagieren. Jeder Teammate hat eine vollständige Ansicht seines eigenen Terminals.

Aufgaben zuweisen und beanspruchen

Die gemeinsame Aufgabenliste koordiniert die Arbeit im Team. Der Lead erstellt Aufgaben und Teammates arbeiten sie durch. Aufgaben haben drei Zustände: ausstehend, in Bearbeitung und abgeschlossen. Aufgaben können auch von anderen Aufgaben abhängen: eine ausstehende Aufgabe mit ungelösten Abhängigkeiten kann nicht beansprucht werden, bis diese Abhängigkeiten erfüllt sind.

Der Lead kann Aufgaben explizit zuweisen oder Teammates können selbst beanspruchen:

- **Lead weist zu:** teilen Sie dem Lead mit, welche Aufgabe welchem Teammate gegeben werden soll
- **Selbst beanspruchen:** nach Abschluss einer Aufgabe wählt ein Teammate die nächste nicht zugewiesene, nicht blockierte Aufgabe selbst aus

Das Beanspruchen von Aufgaben verwendet Dateisperren, um Race Conditions zu verhindern, wenn mehrere Teammates versuchen, gleichzeitig dieselbe Aufgabe zu beanspruchen.

Teammates herunterfahren

Um die Sitzung eines Teammates ordnungsgemäß zu beenden:

```
Ask the researcher teammate to shut down
```

Der Lead sendet eine Abschaltungsanfrage. Der Teammate kann zustimmen und ordnungsgemäß beenden oder mit einer Erklärung ablehnen.

Bereinigen Sie das Team

Wenn Sie fertig sind, bitten Sie den Lead zu bereinigen:

```
Clean up the team
```

Dies entfernt die gemeinsamen Teamressourcen. Wenn der Lead die Bereinigung ausführt, prüft er auf aktive Teammates und schlägt fehl, wenn noch welche laufen, also fahren Sie diese zuerst herunter.

Warning:

Verwenden Sie immer den Lead zum Bereinigen. Teammates sollten keine Bereinigung ausführen, da ihr Teamkontext möglicherweise nicht korrekt aufgelöst wird, was möglicherweise Ressourcen in einem inkonsistenten Zustand hinterlässt.

Erzwingen Sie Qualitätsgates mit hooks

Verwenden Sie [hooks](#), um Regeln durchzusetzen, wenn Teammates ihre Arbeit abschließen oder Aufgaben abgeschlossen werden:

- `TeammateIdle` : wird ausgeführt, wenn ein Teammate im Begriff ist, untätig zu werden. Beenden Sie mit Code 2, um Feedback zu senden und den Teammate weiterarbeiten zu lassen.
- `TaskCompleted` : wird ausgeführt, wenn eine Aufgabe als abgeschlossen markiert wird. Beenden Sie mit Code 2, um die Fertigstellung zu verhindern und Feedback zu senden.

Wie Agent-Teams funktionieren

Dieser Abschnitt behandelt die Architektur und Mechanik hinter Agent-Teams. Wenn Sie sie verwenden möchten, siehe [Kontrolle Ihres Agent-Teams](#) oben.

Wie Claude Agent-Teams startet

Es gibt zwei Möglichkeiten, wie Agent-Teams gestartet werden:

- **Sie fordern ein Team an:** geben Sie Claude eine Aufgabe, die von paralleler Arbeit profitiert, und fordern Sie explizit ein Agent-Team an. Claude erstellt eines basierend auf Ihren Anweisungen.
- **Claude schlägt ein Team vor:** wenn Claude feststellt, dass Ihre Aufgabe von paralleler Arbeit profitieren würde, kann es ein Team vorschlagen. Sie bestätigen, bevor es fortfährt.

In beiden Fällen behalten Sie die Kontrolle. Claude wird kein Team ohne Ihre Genehmigung erstellen.

Architektur

Ein Agent-Team besteht aus:

Komponente	Rolle
Team Lead	Die Haupt-Claude Code-Sitzung, die das Team erstellt, Teammates erzeugt und die Arbeit koordiniert
Teammates	Separate Claude Code-Instanzen, die jeweils an zugewiesenen Aufgaben arbeiten
Aufgabenliste	Gemeinsame Liste von Arbeitselementen, die Teammates beanspruchen und abschließen
Mailbox	Nachrichtensystem für Kommunikation zwischen Agenten

Siehe [Wählen Sie einen Anzeigemodus](#) für Anzeigeoptionen. Teammate-Nachrichten kommen automatisch beim Lead an.

Das System verwaltet Aufgabenabhängigkeiten automatisch. Wenn ein Teammate eine Aufgabe abschließt, von der andere Aufgaben abhängen, werden blockierte Aufgaben automatisch entsperrt.

Teams und Aufgaben werden lokal gespeichert:

- **Team-Konfiguration:** `~/.claude/teams/{team-name}/config.json`

- **Aufgabenliste:** `~/.claude/tasks/{team-name}/`

Die Team-Konfiguration enthält ein `members`-Array mit dem Namen, der Agent-ID und dem Agent-Typ jedes Teammates. Teammates können diese Datei lesen, um andere Teammitglieder zu entdecken.

Berechtigungen

Teammates starten mit den Berechtigungseinstellungen des Leads. Wenn der Lead mit `--dangerously-skip-permissions` ausgeführt wird, tun dies auch alle Teammates. Nach dem Erzeugen können Sie einzelne Teammate-Modi ändern, aber Sie können keine Pro-Teammate-Modi zum Zeitpunkt des Erzeugung setzen.

Kontext und Kommunikation

Jeder Teammate hat sein eigenes Kontextfenster. Beim Erzeugen lädt ein Teammate denselben Projektkontext wie eine reguläre Sitzung: CLAUDE.md, MCP servers und skills. Er erhält auch den Spawn-Prompt vom Lead. Die Gesprächshistorie des Leads wird nicht übertragen.

Wie Teammates Informationen teilen:

- **Automatische Nachrichtenlieferung:** wenn Teammates Nachrichten senden, werden sie automatisch an Empfänger geliefert. Der Lead muss nicht auf Updates abfragen.
- **Untätigkeitsbenachrichtigungen:** wenn ein Teammate fertig ist und stoppt, benachrichtigt er automatisch den Lead.
- **Gemeinsame Aufgabenliste:** alle Agenten können den Aufgabenstatus sehen und verfügbare Arbeit beanspruchen.

Teammate-Messaging:

- **message:** senden Sie eine Nachricht an einen bestimmten Teammate
- **broadcast:** senden Sie an alle Teammates gleichzeitig. Verwenden Sie sparsam, da die Kosten mit der Teamgröße skalieren.

Token-Nutzung

Agent-Teams verwenden deutlich mehr Tokens als eine einzelne Sitzung. Jeder Teammate hat sein eigenes Kontextfenster, und die Token-Nutzung skaliert mit der Anzahl der aktiven Teammates. Für Recherche, Überprüfung und neue Feature-Arbeit sind die zusätzlichen Tokens normalerweise lohnenswert. Für Routineaufgaben ist eine einzelne Sitzung kostengünstiger. Siehe [Agent-Team-Token-Kosten](#) für Nutzungsleitfäden.

Anwendungsbeispiele

Diese Beispiele zeigen, wie Agent-Teams Aufgaben handhaben, bei denen parallele Exploration Wert bietet.

Führen Sie eine parallele Code-Überprüfung durch

Ein einzelner Reviewer neigt dazu, sich jeweils auf eine Art von Problem zu konzentrieren. Das Aufteilen von Überprüfungskriterien in unabhängige Domänen bedeutet, dass Sicherheit, Leistung und Testabdeckung alle gleichzeitig gründlich beachtet werden. Der Prompt weist jedem Teammate eine unterschiedliche Perspektive zu, damit sie sich nicht überlappen:

```
Create an agent team to review PR #142. Spawn three reviewers:  
- One focused on security implications  
- One checking performance impact  
- One validating test coverage  
Have them each review and report findings.
```

Jeder Reviewer arbeitet vom selben PR aus, wendet aber einen anderen Filter an. Der Lead synthetisiert Erkenntnisse über alle drei nach Abschluss.

Untersuchen Sie mit konkurrierenden Hypothesen

Wenn die Grundursache unklar ist, neigt ein einzelner Agent dazu, eine plausible Erklärung zu finden und zu stoppen. Der Prompt bekämpft dies, indem er Teammates explizit gegnerisch macht: die Aufgabe jedes ist nicht nur, seine eigene Theorie zu untersuchen, sondern auch die anderen in Frage zu stellen.

```
Users report the app exits after one message instead of staying connected.  
Spawn 5 agent teammates to investigate different hypotheses. Have them talk to  
each other to try to disprove each other's theories, like a scientific  
debate. Update the findings doc with whatever consensus emerges.
```

Die Debattenstruktur ist der Schlüsselmechanismus hier. Sequenzielle Untersuchung leidet unter Verankerung: sobald eine Theorie untersucht wird, ist die nachfolgende Untersuchung zu ihr vorgespannt.

Mit mehreren unabhängigen Ermittlern, die aktiv versuchen, sich gegenseitig zu widerlegen, ist die Theorie, die überlebt, viel wahrscheinlicher die tatsächliche Grundursache.

Best Practices

Geben Sie Teammates genug Kontext

Teammates laden Projektkontext automatisch, einschließlich CLAUDE.md, MCP servers und skills, aber sie erben nicht die Gesprächshistorie des Leads. Siehe [Kontext und Kommunikation](#) für Details. Fügen Sie aufgabenspezifische Details in den Spawn-Prompt ein:

```
Spawn a security reviewer teammate with the prompt: "Review the authentication module at src/auth/ for security vulnerabilities. Focus on token handling, session management, and input validation. The app uses JWT tokens stored in httpOnly cookies. Report any issues with severity ratings."
```

Wählen Sie eine angemessene Teamgröße

Es gibt keine harte Grenze für die Anzahl der Teammates, aber praktische Einschränkungen gelten:

- **Token-Kosten skalieren linear:** jeder Teammate hat sein eigenes Kontextfenster und verbraucht Tokens unabhängig. Siehe [Agent-Team-Token-Kosten](#) für Details.
- **Koordinationsaufwand nimmt zu:** mehr Teammates bedeutet mehr Kommunikation, Aufgabenkoordination und Konfliktpotenzial
- **Sinkende Erträge:** über einen bestimmten Punkt hinaus beschleunigen zusätzliche Teammates die Arbeit nicht proportional

Beginnen Sie mit 3-5 Teammates für die meisten Workflows. Dies balanciert parallele Arbeit mit verwaltbarer Koordination. Die Beispiele in diesem Leitfaden verwenden 3-5 Teammates, weil dieser Bereich über verschiedene Aufgabentypen hinweg gut funktioniert.

Mit 5-6 [Aufgaben](#) pro Teammate bleibt jeder produktiv, ohne übermäßiges Kontextwechsel. Wenn Sie 15 unabhängige Aufgaben haben, sind 3 Teammates ein guter Ausgangspunkt.

Skalieren Sie nur auf, wenn die Arbeit wirklich davon profitiert, dass Teammates gleichzeitig arbeiten. Drei fokussierte Teammates übertreffen oft fünf verstreute.

Dimensionieren Sie Aufgaben angemessen

- **Zu klein:** Koordinationsaufwand übersteigt den Nutzen
- **Zu groß:** Teammates arbeiten zu lange ohne Check-ins, was das Risiko verschwendeter Anstrengungen erhöht

- **Genau richtig:** in sich geschlossene Einheiten, die ein klares Ergebnis liefern, wie eine Funktion, eine Testdatei oder eine Überprüfung

Tip:

Der Lead teilt Arbeit in Aufgaben auf und weist sie Teammates automatisch zu. Wenn er nicht genug Aufgaben erstellt, bitten Sie ihn, die Arbeit in kleinere Stücke aufzuteilen. Mit 5-6 Aufgaben pro Teammate bleibt jeder produktiv und der Lead kann Arbeit neu zuweisen, wenn jemand steckenbleibt.

Warten Sie, bis Teammates fertig sind

Manchmal beginnt der Lead, Aufgaben selbst zu implementieren, anstatt auf Teammates zu warten. Wenn Sie dies bemerken:

Wait for your teammates to complete their tasks before proceeding

Beginnen Sie mit Recherche und Überprüfung

Wenn Sie neu bei Agent-Teams sind, beginnen Sie mit Aufgaben, die klare Grenzen haben und nicht das Schreiben von Code erfordern: Überprüfung eines PR, Recherche einer Bibliothek oder Untersuchung eines Bugs. Diese Aufgaben zeigen den Wert paralleler Exploration ohne die Koordinationschallenges, die mit paralleler Implementierung einhergehen.

Vermeiden Sie Dateikonflikte

Zwei Teammates, die dieselbe Datei bearbeiten, führen zu Überschreibungen. Teilen Sie die Arbeit so auf, dass jeder Teammate einen anderen Satz von Dateien besitzt.

Überwachen und lenken Sie

Überprüfen Sie den Fortschritt der Teammates, leiten Sie Ansätze um, die nicht funktionieren, und synthetisieren Sie Erkenntnisse, wenn sie eintreffen. Ein Team zu lange unbeaufsichtigt laufen zu lassen, erhöht das Risiko verschwendeter Anstrengungen.

Fehlerbehebung

Teammates erscheinen nicht

Wenn Teammates nicht erscheinen, nachdem Sie Claude aufgefordert haben, ein Team zu erstellen:

- Im In-Process-Modus können Teammates bereits laufen, sind aber nicht sichtbar. Drücken Sie Shift+Down, um durch aktive Teammates zu wechseln.
- Überprüfen Sie, dass die Aufgabe, die Sie Claude gegeben haben, komplex genug war, um ein Team zu rechtfertigen. Claude entscheidet basierend auf der Aufgabe, ob Teammates erzeugt werden sollen.
- Wenn Sie explizit Split Panes angefordert haben, stellen Sie sicher, dass tmux installiert ist und in Ihrem PATH verfügbar ist:

```
which tmux
```

- Für iTerm2 überprüfen Sie, dass die `it2` CLI installiert ist und die Python-API in iTerm2-Einstellungen aktiviert ist.

Zu viele Berechtigungsaufforderungen

TeamMate-Berechtigungsanfragen sprudeln zum Lead auf, was zu Reibung führen kann. Genehmigen Sie häufige Operationen in Ihren [Berechtigungseinstellungen](#) vor dem Erzeugen von Teammates, um Unterbrechungen zu reduzieren.

Teammates stoppen bei Fehlern

Teammates können nach Fehlern stoppen, anstatt sich zu erholen. Überprüfen Sie ihre Ausgabe mit Shift+Down im In-Process-Modus oder durch Klicken auf den Pane im Split-Modus, dann entweder:

- Geben Sie ihnen zusätzliche Anweisungen direkt
- Erzeugen Sie einen Ersatz-Teammate, um die Arbeit fortzusetzen

Lead fährt herunter, bevor die Arbeit erledigt ist

Der Lead kann entscheiden, dass das Team fertig ist, bevor alle Aufgaben tatsächlich abgeschlossen sind. Wenn dies geschieht, teilen Sie ihm mit, dass er weitermachen soll. Sie können dem Lead auch mitteilen, auf Teammates zu warten, um zu beenden, bevor er fortfährt, wenn er anfängt, Arbeit zu erledigen, anstatt zu delegieren.

Verwaiste tmux-Sitzungen

Wenn eine tmux-Sitzung nach dem Ende des Teams bestehen bleibt, wurde sie möglicherweise nicht vollständig bereinigt. Listen Sie Sitzungen auf und beenden Sie die vom Team erstellte:

```
tmux ls  
tmux kill-session -t <session-name>
```

Einschränkungen

Agent-Teams sind experimentell. Aktuelle Einschränkungen, die Sie beachten sollten:

- **Keine Sitzungswiederaufnahme mit In-Process-Teammates:** `/resume` und `/rewind` stellen In-Process-Teammates nicht wieder her. Nach der Wiederaufnahme einer Sitzung kann der Lead versuchen, mit Teammates zu kommunizieren, die nicht mehr existieren. Wenn dies geschieht, teilen Sie dem Lead mit, neue Teammates zu erzeugen.
- **Aufgabenstatus kann verzögert sein:** Teammates markieren Aufgaben manchmal nicht als abgeschlossen, was abhängige Aufgaben blockiert. Wenn eine Aufgabe steckenbleibt, überprüfen Sie, ob die Arbeit tatsächlich erledigt ist, und aktualisieren Sie den Aufgabenstatus manuell oder teilen Sie dem Lead mit, den Teammate zu antoßen.
- **Abschaltung kann langsam sein:** Teammates beenden ihre aktuelle Anfrage oder ihren Werkzeugaufruf, bevor sie herunterfahren, was Zeit in Anspruch nehmen kann.
- **Ein Team pro Sitzung:** ein Lead kann jeweils nur ein Team verwalten. Bereinigen Sie das aktuelle Team, bevor Sie ein neues starten.
- **Keine verschachtelten Teams:** Teammates können ihre eigenen Teams oder Teammates nicht erzeugen. Nur der Lead kann das Team verwalten.
- **Lead ist fest:** die Sitzung, die das Team erstellt, ist der Lead für seine Lebensdauer. Sie können einen Teammate nicht zum Lead befördern oder die Führung übertragen.
- **Berechtigungen beim Erzeugen gesetzt:** alle Teammates starten mit dem Berechtigungsmodus des Leads. Sie können einzelne Teammate-Modi nach dem Erzeugen ändern, aber Sie können keine Pro-Teammate-Modi zum Zeitpunkt des Erzeugens setzen.
- **Split Panes erfordern tmux oder iTerm2:** der Standard-In-Process-Modus funktioniert in jedem Terminal. Der Split-Pane-Modus wird in VS Code's integriertem Terminal, Windows Terminal oder Ghostty nicht unterstützt.

Tip:

CLAUDE.md funktioniert normal: Teammates lesen **CLAUDE.md** -Dateien aus ihrem Arbeitsverzeichnis. Verwenden Sie dies, um projektspezifische Anleitung für alle Teammates bereitzustellen.

Nächste Schritte

Erkunden Sie verwandte Ansätze für parallele Arbeit und Delegation:

- **Leichte Delegation:** [subagents](#) erzeugen Helper-Agenten für Recherche oder Überprüfung innerhalb Ihrer Sitzung, besser für Aufgaben, die keine Inter-Agent-Koordination benötigen
- **Manuelle parallele Sitzungen:** [Git worktrees](#) ermöglichen es Ihnen, mehrere Claude Code-Sitzungen selbst ohne automatisierte Teamkoordination auszuführen
- **Vergleichen Sie Ansätze:** siehe den [Subagent vs Agent-Team](#) Vergleich für eine Seite-an-Seite-Aufschlüsselung

Claude Code programmgesteuert ausführen

Verwenden Sie das Agent SDK, um Claude Code programmgesteuert über die CLI, Python oder TypeScript auszuführen.

Das [Agent SDK](#) bietet Ihnen die gleichen Tools, die Agent-Schleife und das Kontextmanagement, die Claude Code antreiben. Es ist als CLI für Skripte und CI/CD verfügbar oder als [Python](#) und [TypeScript](#) Pakete für vollständige programmgesteuerte Kontrolle.

Note:

Die CLI hieß früher „Headless-Modus“. Das Flag `-p` und alle CLI-Optionen funktionieren auf die gleiche Weise.

Um Claude Code programmgesteuert über die CLI auszuführen, übergeben Sie `-p` mit Ihrer Eingabeaufforderung und allen [CLI-Optionen](#):

```
claude -p "Find and fix the bug in auth.py" --allowedTools "Read,Edit,Bash"
```

Diese Seite behandelt die Verwendung des Agent SDK über die CLI (`claude -p`). Für die Python- und TypeScript-SDK-Pakete mit strukturierten Ausgaben, Tool-Genehmigungsrückrufen und nativen Nachrichtenobjekten siehe die [vollständige Agent SDK-Dokumentation](#).

Grundlegende Verwendung

Fügen Sie das Flag `-p` (oder `--print`) zu jedem `claude`-Befehl hinzu, um ihn nicht interaktiv auszuführen. Alle [CLI-Optionen](#) funktionieren mit `-p`, einschließlich:

- `--continue` zum [Fortsetzen von Gesprächen](#)
- `--allowedTools` zum [automatischen Genehmigen von Tools](#)
- `--output-format` für [strukturierte Ausgabe](#)

Dieses Beispiel stellt Claude eine Frage zu Ihrer Codebasis und gibt die Antwort aus:

```
claude -p "What does the auth module do?"
```

Beispiele

Diese Beispiele zeigen häufige CLI-Muster.

Strukturierte Ausgabe abrufen

Verwenden Sie `--output-format`, um zu steuern, wie Antworten zurückgegeben werden:

- `text` (Standard): einfache Textausgabe
- `json`: strukturiertes JSON mit Ergebnis, Sitzungs-ID und Metadaten
- `stream-json`: zeilengetrennte JSON für Echtzeit-Streaming

Dieses Beispiel gibt eine Projektzusammenfassung als JSON mit Sitzungsmetadaten zurück, wobei sich das Textergebnis im Feld `result` befindet:

```
claude -p "Summarize this project" --output-format json
```

Um eine Ausgabe zu erhalten, die einem bestimmten Schema entspricht, verwenden Sie `--output-format json` mit `--json-schema` und einer [JSON Schema](#)-Definition. Die Antwort enthält Metadaten über die Anfrage (Sitzungs-ID, Nutzung usw.) mit der strukturierten Ausgabe im Feld `structured_output`.

Dieses Beispiel extrahiert Funktionsnamen und gibt sie als Array von Strings zurück:

```
claude -p "Extract the main function names from auth.py" \  
  --output-format json \  
  --json-schema '{"type":"object","properties":{"functions":  
{"type":"array","items":{"type":"string"}},"required":["functions"]}'
```

Tip:

Verwenden Sie ein Tool wie [jq](#), um die Antwort zu analysieren und bestimmte Felder zu extrahieren:

```
## Extract the text result
claude -p "Summarize this project" --output-format json | jq -r '.result'

## Extract structured output
claude -p "Extract function names from auth.py" \
  --output-format json \
  --json-schema '{"type":"object","properties":{"functions":\
{"type":"array","items":{"type":"string"}},"required":["functions"]}}' \
  | jq '.structured_output'
```

Antworten streamen

Verwenden Sie `--output-format stream-json` mit `--verbose` und `--include-partial-messages`, um Token zu erhalten, während sie generiert werden. Jede Zeile ist ein JSON-Objekt, das ein Ereignis darstellt:

```
claude -p "Explain recursion" --output-format stream-json --verbose --include-
partial-messages
```

Das folgende Beispiel verwendet `jq`, um nach Text-Deltas zu filtern und nur den Streaming-Text anzuzeigen. Das Flag `-r` gibt rohe Strings aus (keine Anführungszeichen) und `-j` verbindet ohne Zeilenumbrüche, sodass Token kontinuierlich streamen:

```
claude -p "Write a poem" --output-format stream-json --verbose --include-partial-
messages | \
  jq -rj 'select(.type == "stream_event" and .event.delta.type? == "text_delta")
| .event.delta.text'
```

Für programmgesteuertes Streaming mit Rückrufen und Nachrichtenobjekten siehe [Antworten in Echtzeit streamen](#) in der Agent SDK-Dokumentation.

Tools automatisch genehmigen

Verwenden Sie `--allowedTools`, um Claude bestimmte Tools ohne Aufforderung zu verwenden. Dieses Beispiel führt eine Test-Suite aus und behebt Fehler, wobei Claude Bash-Befehle ausführen und Dateien lesen/bearbeiten kann, ohne um Erlaubnis zu fragen:

```
claude -p "Run the test suite and fix any failures" \  
--allowedTools "Bash,Read,Edit"
```

Einen Commit erstellen

Dieses Beispiel überprüft bereitgestellte Änderungen und erstellt einen Commit mit einer angemessenen Nachricht:

```
claude -p "Look at my staged changes and create an appropriate commit" \  
--allowedTools "Bash(git diff *),Bash(git log *),Bash(git status *),Bash(git  
commit *)"
```

Das Flag `--allowedTools` verwendet [Berechtigungsregelsyntax](#). Das nachfolgende `*` aktiviert Präfix-Matching, sodass `Bash(git diff *)` jeden Befehl erlaubt, der mit `git diff` beginnt. Das Leerzeichen vor `*` ist wichtig; ohne es würde `Bash(git diff*)` auch `git diff-index` entsprechen.

Note:

Benutzer-aufgerufene [skills](#) wie `/commit` und [integrierte Befehle](#) sind nur im interaktiven Modus verfügbar. Im `-p`-Modus beschreiben Sie stattdessen die Aufgabe, die Sie ausführen möchten.

System-Eingabeaufforderung anpassen

Verwenden Sie `--append-system-prompt`, um Anweisungen hinzuzufügen und dabei das Standardverhalten von Claude Code beizubehalten. Dieses Beispiel leitet einen PR-Diff an Claude weiter und weist ihn an, auf Sicherheitslücken zu überprüfen:

```
gh pr diff "$1" | claude -p \  
--append-system-prompt "You are a security engineer. Review for  
vulnerabilities." \  
--output-format json
```

Siehe [System-Eingabeaufforderungs-Flags](#) für weitere Optionen, einschließlich `--system-prompt`, um die Standardeingabeaufforderung vollständig zu ersetzen.

Gespräche fortsetzen

Verwenden Sie `--continue`, um das neueste Gespräch fortzusetzen, oder `--resume` mit einer Sitzungs-ID, um ein bestimmtes Gespräch fortzusetzen. Dieses Beispiel führt eine Überprüfung durch und sendet dann Folgeeingabeaufforderungen:

```
## First request
claude -p "Review this codebase for performance issues"

## Continue the most recent conversation
claude -p "Now focus on the database queries" --continue
claude -p "Generate a summary of all issues found" --continue
```

Wenn Sie mehrere Gespräche führen, erfassen Sie die Sitzungs-ID, um ein bestimmtes Gespräch fortzusetzen:

```
session_id=$(claude -p "Start a review" --output-format json | jq -r
'.session_id')
claude -p "Continue that review" --resume "$session_id"
```

Nächste Schritte

- [Agent SDK Schnellstart](#): Erstellen Sie Ihren ersten Agent mit Python oder TypeScript
- [CLI-Referenz](#): alle CLI-Flags und Optionen
- [GitHub Actions](#): Verwenden Sie das Agent SDK in GitHub-Workflows
- [GitLab CI/CD](#): Verwenden Sie das Agent SDK in GitLab-Pipelines

Lokale Sitzungen von jedem Gerät aus mit Remote Control fortsetzen

Setzen Sie eine lokale Claude Code-Sitzung von Ihrem Telefon, Tablet oder einem beliebigen Browser aus mit Remote Control fort. Funktioniert mit claude.ai/code und der Claude-Mobile-App.

Note:

Remote Control ist in allen Plänen verfügbar. Team- und Enterprise-Administratoren müssen Claude Code zunächst in den [Admin-Einstellungen](#) aktivieren.

Remote Control verbindet claude.ai/code oder die Claude-App für [iOS](#) und [Android](#) mit einer Claude Code-Sitzung, die auf Ihrem Computer ausgeführt wird. Starten Sie eine Aufgabe an Ihrem Schreibtisch und setzen Sie sie dann von Ihrem Telefon auf der Couch oder einem Browser auf einem anderen Computer fort.

Wenn Sie eine Remote Control-Sitzung auf Ihrem Computer starten, wird Claude die ganze Zeit lokal ausgeführt, sodass nichts in die Cloud verschoben wird. Mit Remote Control können Sie:

- **Ihre vollständige lokale Umgebung remote nutzen:** Ihr Dateisystem, [MCP servers](#), Tools und Projektkonfiguration bleiben verfügbar
- **Von beiden Oberflächen gleichzeitig arbeiten:** Das Gespräch bleibt auf allen verbundenen Geräten synchronisiert, sodass Sie Nachrichten von Ihrem Terminal, Browser und Telefon austauschbar senden können
- **Unterbrechungen überstehen:** Wenn Ihr Laptop in den Ruhezustand wechselt oder Ihre Netzwerkverbindung unterbrochen wird, wird die Sitzung automatisch wiederhergestellt, wenn Ihr Computer wieder online ist

Im Gegensatz zu [Claude Code im Web](#), das auf Cloud-Infrastruktur ausgeführt wird, werden Remote Control-Sitzungen direkt auf Ihrem Computer ausgeführt und interagieren mit Ihrem lokalen Dateisystem. Die Web- und Mobile-Schnittstellen sind nur ein Fenster zu dieser lokalen Sitzung.

Note:

Remote Control erfordert Claude Code v2.1.51 oder später. Überprüfen Sie Ihre Version mit `claude --version`.

Diese Seite behandelt die Einrichtung, das Starten und Verbinden mit Sitzungen sowie den Vergleich von Remote Control mit Claude Code im Web.

Anforderungen

Bevor Sie Remote Control verwenden, bestätigen Sie, dass Ihre Umgebung diese Bedingungen erfüllt:

- **Abonnement:** verfügbar in Pro-, Max-, Team- und Enterprise-Plänen. Team- und Enterprise-Administratoren müssen Claude Code zunächst in den [Admin-Einstellungen](#) aktivieren. API-Schlüssel werden nicht unterstützt.
- **Authentifizierung:** Führen Sie `claude` aus und verwenden Sie `/login`, um sich über `claude.ai` anzumelden, falls Sie dies noch nicht getan haben.
- **Workspace-Vertrauen:** Führen Sie `claude` mindestens einmal in Ihrem Projektverzeichnis aus, um den Workspace-Vertrauensdialog zu akzeptieren.

Starten Sie eine Remote Control-Sitzung

Sie können eine neue Sitzung direkt in Remote Control starten oder eine bereits laufende Sitzung verbinden.

Neue Sitzung

Navigieren Sie zu Ihrem Projektverzeichnis und führen Sie aus:

```
claude remote-control
```

Der Prozess bleibt in Ihrem Terminal aktiv und wartet auf Remote-Verbindungen. Er zeigt eine Sitzungs-URL an, die Sie zum [Verbinden von einem anderen Gerät](#) verwenden können, und Sie können die Leertaste drücken, um einen QR-Code für schnellen Zugriff von Ihrem Telefon anzuzeigen. Während eine Remote-Sitzung aktiv ist, zeigt das Terminal den Verbindungsstatus und die Tool-Aktivität an.

Dieser Befehl unterstützt die folgenden Flags:

- `--name "My Project"`: Legen Sie einen benutzerdefinierten Sitzungstitel fest, der in der Sitzungsliste unter `claude.ai/code` sichtbar ist. Sie können den Namen auch als Positionsargument übergeben: `claude remote-control "My Project"`

- **--verbose** : Zeigen Sie detaillierte Verbindungs- und Sitzungsprotokolle an
- **--sandbox** / **--no-sandbox** : Aktivieren oder deaktivieren Sie [sandboxing](#) für Dateisystem- und Netzwerkisolation während der Sitzung. Sandboxing ist standardmäßig deaktiviert.

Aus einer bestehenden Sitzung

Wenn Sie bereits in einer Claude Code-Sitzung sind und diese remote fortsetzen möchten, verwenden Sie den Befehl `/remote-control` (oder `/rc`) :

```
/remote-control
```

Übergeben Sie einen Namen als Argument, um einen benutzerdefinierten Sitzungstitel festzulegen:

```
/remote-control My Project
```

Dies startet eine Remote Control-Sitzung, die Ihren aktuellen Gesprächsverlauf überträgt und eine Sitzungs-URL und einen QR-Code anzeigt, die Sie zum [Verbinden von einem anderen Gerät](#) verwenden können. Die Flags `--verbose` , `--sandbox` und `--no-sandbox` sind mit diesem Befehl nicht verfügbar.

Verbinden Sie sich von einem anderen Gerät

Sobald eine Remote Control-Sitzung aktiv ist, haben Sie mehrere Möglichkeiten, sich von einem anderen Gerät aus zu verbinden:

- **Öffnen Sie die Sitzungs-URL** in einem beliebigen Browser, um direkt zur Sitzung auf claude.ai/code zu gehen. Sowohl `claude remote-control` als auch `/remote-control` zeigen diese URL im Terminal an.
- **Scannen Sie den QR-Code**, der neben der Sitzungs-URL angezeigt wird, um ihn direkt in der Claude-App zu öffnen. Mit `claude remote-control` drücken Sie die Leertaste, um die QR-Code-Anzeige umzuschalten.
- **Öffnen Sie claude.ai/code oder die Claude-App** und suchen Sie die Sitzung nach Name in der Sitzungsliste. Remote Control-Sitzungen zeigen ein Computersymbol mit einem grünen Statusindikator an, wenn sie online sind.

Die Remote-Sitzung erhält ihren Namen vom Argument `--name` (oder dem Namen, der an `/remote-control` übergeben wird), Ihrer letzten Nachricht, Ihrem Wert `/rename` oder 'Remote Control-Sitzung', wenn es keinen Gesprächsverlauf gibt. Wenn die Umgebung bereits eine aktive Sitzung hat, werden Sie gefragt, ob Sie diese fortsetzen oder eine neue starten möchten.

Wenn Sie die Claude-App noch nicht haben, verwenden Sie den Befehl `/mobile` in Claude Code, um einen Download-QR-Code für [iOS](#) oder [Android](#) anzuzeigen.

Aktivieren Sie Remote Control für alle Sitzungen

Standardmäßig wird Remote Control nur aktiviert, wenn Sie explizit `claude remote-control` oder `/remote-control` ausführen. Um es automatisch für jede Sitzung zu aktivieren, führen Sie `/config` in Claude Code aus und setzen Sie **Enable Remote Control for all sessions** auf `true`. Setzen Sie es auf `false` zurück, um es zu deaktivieren.

Jede Claude Code-Instanz unterstützt jeweils eine Remote-Sitzung. Wenn Sie mehrere Instanzen ausführen, hat jede ihre eigene Umgebung und Sitzung.

Verbindung und Sicherheit

Ihre lokale Claude Code-Sitzung stellt nur ausgehende HTTPS-Anfragen und öffnet niemals eingehende Ports auf Ihrem Computer. Wenn Sie Remote Control starten, wird es bei der Anthropic API registriert und fragt nach Arbeit ab. Wenn Sie sich von einem anderen Gerät aus verbinden, leitet der Server Nachrichten zwischen dem Web- oder Mobile-Client und Ihrer lokalen Sitzung über eine Streaming-Verbindung weiter.

Der gesamte Datenverkehr verläuft über die Anthropic API über TLS, die gleiche Transportsicherheit wie jede Claude Code-Sitzung. Die Verbindung verwendet mehrere kurzlebige Anmeldeinformationen, die jeweils auf einen einzelnen Zweck beschränkt sind und unabhängig voneinander ablaufen.

Remote Control vs. Claude Code im Web

Remote Control und [Claude Code im Web](#) verwenden beide die Schnittstelle `claude.ai/code`. Der Hauptunterschied liegt darin, wo die Sitzung ausgeführt wird: Remote Control wird auf Ihrem Computer ausgeführt, sodass Ihre lokalen MCP servers, Tools und Projektkonfiguration verfügbar bleiben. Claude Code im Web wird in von Anthropic verwalteter Cloud-Infrastruktur ausgeführt.

Verwenden Sie Remote Control, wenn Sie sich mitten in lokaler Arbeit befinden und von einem anderen Gerät aus weitermachen möchten. Verwenden Sie Claude Code im Web, wenn Sie eine Aufgabe ohne lokale Einrichtung starten möchten, an einem Repository arbeiten, das Sie nicht geklont haben, oder mehrere Aufgaben parallel ausführen möchten.

Einschränkungen

- **Eine Remote-Sitzung gleichzeitig:** Jede Claude Code-Sitzung unterstützt eine Remote-Verbindung.
- **Terminal muss offen bleiben:** Remote Control wird als lokaler Prozess ausgeführt. Wenn Sie das Terminal schließen oder den `claude`-Prozess beenden, endet die Sitzung. Führen Sie `claude remote-control` erneut aus, um eine neue Sitzung zu starten.
- **Längerer Netzerkausfall:** Wenn Ihr Computer aktiv ist, aber länger als etwa 10 Minuten das Netzwerk nicht erreichen kann, wird die Sitzung beendet und der Prozess beendet. Führen Sie `claude remote-control` erneut aus, um eine neue Sitzung zu starten.

Verwandte Ressourcen

- [Claude Code im Web](#): Führen Sie Sitzungen in von Anthropic verwalteten Cloud-Umgebungen aus, anstatt auf Ihrem Computer
- [Authentifizierung](#): Richten Sie `/login` ein und verwalten Sie Anmeldeinformationen für `claude.ai`
- [CLI-Referenz](#): Vollständige Liste der Flags und Befehle einschließlich `claude remote-control`
- [Sicherheit](#): Wie Remote Control-Sitzungen in das Claude Code-Sicherheitsmodell passen
- [Datennutzung](#): Welche Daten während lokaler und Remote-Sitzungen durch die Anthropic API fließen

Prompts nach Zeitplan ausführen

Verwenden Sie `/loop` und die Cron-Planungstools, um Prompts wiederholt auszuführen, den Status abzurufen oder einmalige Erinnerungen innerhalb einer Claude Code-Sitzung zu setzen.

Note:

Geplante Aufgaben erfordern Claude Code v2.1.72 oder später. Überprüfen Sie Ihre Version mit `claude --version`.

Geplante Aufgaben ermöglichen es Claude, einen Prompt automatisch in regelmäßigen Abständen erneut auszuführen. Verwenden Sie sie, um eine Bereitstellung abzurufen, einen PR zu überwachen, einen langwierigen Build zu überprüfen oder sich später in der Sitzung an etwas zu erinnern.

Aufgaben sind sitzungsbezogen: Sie existieren im aktuellen Claude Code-Prozess und sind weg, wenn Sie beenden. Für dauerhafte Planung, die Neustarts übersteht und ohne aktive Terminalsitzung läuft, siehe [Desktop-geplante Aufgaben](#) oder [GitHub Actions](#).

Planen Sie einen wiederkehrenden Prompt mit `/loop`

Die `/loop` [bundled skill](#) ist der schnellste Weg, um einen wiederkehrenden Prompt zu planen. Übergeben Sie ein optionales Intervall und einen Prompt, und Claude richtet einen Cron-Job ein, der im Hintergrund läuft, während die Sitzung offen bleibt.

```
/loop 5m check if the deployment finished and tell me what happened
```

Claude analysiert das Intervall, konvertiert es in einen Cron-Ausdruck, plant den Job und bestätigt die Häufigkeit und die Job-ID.

Intervallsyntax

Intervalle sind optional. Sie können sie am Anfang, am Ende oder gar nicht verwenden.

Form	Beispiel	Analysiertes Intervall
Führendes Token	<code>/loop 30m check the build</code>	alle 30 Minuten
Nachfolgende <code>every</code> -Klausel	<code>/loop check the build every 2 hours</code>	alle 2 Stunden
Kein Intervall	<code>/loop check the build</code>	Standard: alle 10 Minuten

Unterstützte Einheiten sind `s` für Sekunden, `m` für Minuten, `h` für Stunden und `d` für Tage. Sekunden werden auf die nächste Minute aufgerundet, da Cron eine Granularität von einer Minute hat. Intervalle, die nicht gleichmäßig in ihre Einheit aufgehen, wie `7m` oder `90m`, werden auf das nächste saubere Intervall gerundet und Claude teilt Ihnen mit, was es gewählt hat.

Schleife über einen anderen Befehl

Der geplante Prompt kann selbst ein Befehl oder eine Skill-Invokation sein. Dies ist nützlich, um einen Workflow erneut auszuführen, den Sie bereits verpackt haben.

```
/loop 20m /review-pr 1234
```

Jedes Mal, wenn der Job läuft, führt Claude `/review-pr 1234` aus, als hätten Sie es eingegeben.

Setzen Sie eine einmalige Erinnerung

Für einmalige Erinnerungen beschreiben Sie, was Sie möchten, in natürlicher Sprache, anstatt `/loop` zu verwenden. Claude plant eine einmalige Aufgabe, die sich nach der Ausführung selbst löscht.

```
remind me at 3pm to push the release branch
```

```
in 45 minutes, check whether the integration tests passed
```

Claude heftet die Feuerzeit an eine bestimmte Minute und Stunde mit einem Cron-Ausdruck an und bestätigt, wann es läuft.

Verwalten Sie geplante Aufgaben

Bitten Sie Claude in natürlicher Sprache, Aufgaben aufzulisten oder zu stornieren, oder verweisen Sie direkt auf die zugrunde liegenden Tools.

what scheduled tasks do I have?

cancel the deploy check job

Unter der Haube verwendet Claude diese Tools:

Tool	Zweck
<code>CronCreate</code>	Planen Sie eine neue Aufgabe. Akzeptiert einen 5-Feld-Cron-Ausdruck, den auszuführenden Prompt und ob er wiederkehrend ist oder einmal läuft.
<code>CronList</code>	Listet alle geplanten Aufgaben mit ihren IDs, Zeitplänen und Prompts auf.
<code>CronDelete</code>	Stornieren Sie eine Aufgabe nach ID.

Jede geplante Aufgabe hat eine 8-stellige ID, die Sie an `CronDelete` übergeben können. Eine Sitzung kann gleichzeitig bis zu 50 geplante Aufgaben enthalten.

Wie geplante Aufgaben ausgeführt werden

Der Scheduler überprüft jede Sekunde auf fällige Aufgaben und reiht sie mit niedriger Priorität ein. Ein geplanter Prompt läuft zwischen Ihren Zügen, nicht während Claude mitten in einer Antwort ist. Wenn Claude beschäftigt ist, wenn eine Aufgabe fällig wird, wartet der Prompt, bis der aktuelle Zug endet.

Alle Zeiten werden in Ihrer lokalen Zeitzone interpretiert. Ein Cron-Ausdruck wie `0 9 * * *` bedeutet 9 Uhr, wo immer Sie Claude Code ausführen, nicht UTC.

Jitter

Um zu vermeiden, dass jede Sitzung die API zum gleichen Wanduhrzeitpunkt trifft, fügt der Scheduler einen kleinen deterministischen Offset zu Feuerzeiten hinzu:

- Wiederkehrende Aufgaben läuft bis zu 10% ihrer Periode zu spät, begrenzt auf 15 Minuten. Ein stündlicher Job könnte überall von `:00` bis `:06` laufen.
- Einmalige Aufgaben, die für die Ober- oder Unterseite der Stunde geplant sind, läuft bis zu 90 Sekunden früh.

Der Offset wird von der Task-ID abgeleitet, daher erhält die gleiche Aufgabe immer den gleichen Offset. Wenn genaue Zeitpunkte wichtig sind, wählen Sie eine Minute, die nicht `:00` oder `:30` ist, zum Beispiel `3 9 * * *` statt `0 9 * * *`, und der einmalige Jitter wird nicht angewendet.

Ablauf nach drei Tagen

Wiederkehrende Aufgaben verfallen automatisch 3 Tage nach der Erstellung. Die Aufgabe läuft ein letztes Mal, dann löscht sie sich selbst. Dies begrenzt, wie lange eine vergessene Schleife laufen kann. Wenn Sie benötigen, dass eine wiederkehrende Aufgabe länger dauert, stornieren und erstellen Sie sie neu, bevor sie abläuft, oder verwenden Sie [Desktop-geplante Aufgaben](#) für dauerhafte Planung.

Cron-Ausdrucksreferenz

`CronCreate` akzeptiert Standard-5-Feld-Cron-Ausdrücke: `minute hour day-of-month month day-of-week`. Alle Felder unterstützen Wildcards (`*`), einzelne Werte (`5`), Schritte (`*/15`), Bereiche (`1-5`) und kommagetrennte Listen (`1,15,30`).

Beispiel	Bedeutung
<code>* / 5 * * * *</code>	Alle 5 Minuten
<code>0 * * * *</code>	Jede Stunde zur vollen Stunde
<code>7 * * * *</code>	Jede Stunde um 7 Minuten nach
<code>0 9 * * *</code>	Jeden Tag um 9 Uhr lokal
<code>0 9 * * 1-5</code>	Wochentags um 9 Uhr lokal
<code>30 14 15 3 *</code>	15. März um 14:30 Uhr lokal

Der Wochentag verwendet `0` oder `7` für Sonntag bis `6` für Samstag. Erweiterte Syntax wie `L`, `W`, `?` und Name-Aliase wie `MON` oder `JAN` werden nicht unterstützt.

Wenn sowohl der Tag des Monats als auch der Wochentag eingeschränkt sind, stimmt ein Datum überein, wenn eines der Felder übereinstimmt. Dies folgt der Standard-Vixie-Cron-Semantik.

Deaktivieren Sie geplante Aufgaben

Setzen Sie `CLAUDE_CODE_DISABLE_CRON=1` in Ihrer Umgebung, um den Scheduler vollständig zu deaktivieren. Die Cron-Tools und `/loop` werden nicht verfügbar, und alle bereits geplanten Aufgaben stoppen das Läufen. Siehe [Umgebungsvariablen](#) für die vollständige Liste der Deaktivierungsflags.

Einschränkungen

Die sitzungsbezogene Planung hat inhärente Einschränkungen:

- Aufgaben läuft nur, während Claude Code läuft und untätig ist. Das Schließen des Terminals oder das Beenden der Sitzung storniert alles.
- Kein Aufholen für verpasste Läufe. Wenn die geplante Zeit einer Aufgabe verstreicht, während Claude mit einer langwierigen Anfrage beschäftigt ist, läuft sie einmal, wenn Claude untätig wird, nicht einmal pro verpasstem Intervall.
- Keine Persistenz über Neustarts hinweg. Das Neustarten von Claude Code löscht alle sitzungsbezogenen Aufgaben.

Für Cron-gesteuerte Automatisierung, die unbeaufsichtigt laufen muss, verwenden Sie einen [GitHub Actions-Workflow](#) mit einem `schedule`-Trigger, oder [Desktop-geplante Aufgaben](#), wenn Sie einen grafischen Setup-Ablauf möchten.

Code Review

Richten Sie automatisierte PR-Reviews ein, die Logikfehler, Sicherheitslücken und Regressionen durch Multi-Agent-Analyse Ihrer vollständigen Codebasis erkennen

Note:

Code Review befindet sich in der Forschungsvorschau und ist für [Teams](#) und [Enterprise](#) Abonnements verfügbar. Es ist nicht verfügbar für Organisationen mit aktiviertem [Zero Data Retention](#).

Code Review analysiert Ihre GitHub Pull Requests und veröffentlicht Erkenntnisse als Inline-Kommentare auf den Codezeilen, auf denen Probleme gefunden wurden. Eine Flotte spezialisierter Agenten untersucht die Codeänderungen im Kontext Ihrer vollständigen Codebasis und sucht nach Logikfehlern, Sicherheitslücken, fehlerhaften Grenzfällen und subtilen Regressionen.

Erkenntnisse werden nach Schweregrad gekennzeichnet und genehmigen oder blockieren Ihren PR nicht, sodass bestehende Review-Workflows intakt bleiben. Sie können anpassen, was Claude kennzeichnet, indem Sie eine `CLAUDE.md` oder `REVIEW.md` Datei zu Ihrem Repository hinzufügen.

Um Claude in Ihrer eigenen CI-Infrastruktur statt in diesem verwalteten Service auszuführen, siehe [GitHub Actions](#) oder [GitLab CI/CD](#).

Diese Seite behandelt:

- [Wie Reviews funktionieren](#)
- [Setup](#)
- [Anpassung von Reviews](#) mit `CLAUDE.md` und `REVIEW.md`
- [Preisgestaltung](#)

Wie Reviews funktionieren

Sobald ein Administrator Code Review für Ihre Organisation [aktiviert](#), werden Reviews automatisch ausgeführt, wenn ein Pull Request geöffnet oder aktualisiert wird. Mehrere Agenten analysieren parallel den Diff und den umgebenden Code auf Anthropic-Infrasstruktur. Jeder Agent sucht nach einer anderen Klasse von Problemen, dann überprüft ein

Verifizierungsschritt Kandidaten gegen das tatsächliche Codeverhalten, um falsch positive Ergebnisse herauszufiltern. Die Ergebnisse werden dedupliziert, nach Schweregrad eingestuft und als Inline-Kommentare auf den spezifischen Zeilen veröffentlicht, auf denen Probleme gefunden wurden. Wenn keine Probleme gefunden werden, veröffentlicht Claude einen kurzen Bestätigungskommentar auf dem PR.

Reviews skalieren in den Kosten mit PR-Größe und -Komplexität und werden im Durchschnitt in 20 Minuten abgeschlossen. Administratoren können Review-Aktivität und Ausgaben über das [Analytics-Dashboard](#) überwachen.

Schweregrad-Stufen

Jede Erkenntnis wird mit einer Schweregrad-Stufe gekennzeichnet:

Marker	Schweregrad	Bedeutung
	Normal	Ein Fehler, der vor dem Zusammenführen behoben werden sollte
	Nit	Ein kleineres Problem, das behoben werden sollte, aber nicht blockierend ist
	Bereits vorhanden	Ein Fehler, der in der Codebasis vorhanden ist, aber nicht durch diesen PR eingeführt wurde

Erkenntnisse enthalten einen ausklappbaren erweiterten Reasoning-Bereich, den Sie erweitern können, um zu verstehen, warum Claude das Problem gekennzeichnet hat und wie es das Problem überprüft hat.

Was Code Review überprüft

Standardmäßig konzentriert sich Code Review auf Korrektheit: Fehler, die die Produktion unterbrechen würden, nicht auf Formatierungspräferenzen oder fehlende Testabdeckung. Sie können erweitern, was es überprüft, indem Sie [Anleitungsdateien](#) zu Ihrem Repository hinzufügen.

Code Review einrichten

Ein Administrator aktiviert Code Review einmal für die Organisation und wählt aus, welche Repositories einbezogen werden sollen.

Step 1: Öffnen Sie die Claude Code Admin-Einstellungen

Gehen Sie zu claude.ai/admin-settings/claude-code und finden Sie den Code Review Bereich. Sie benötigen Admin-Zugriff auf Ihre Claude-Organisation und die Berechtigung, GitHub Apps in Ihrer GitHub-Organisation zu installieren.

Step 2: Setup starten

Klicken Sie auf **Setup**. Dies startet den GitHub App-Installationsablauf.

Step 3: Installieren Sie die Claude GitHub App

Folgen Sie den Aufforderungen, um die Claude GitHub App in Ihrer GitHub-Organisation zu installieren. Die App fordert diese Repository-Berechtigungen an:

- **Contents:** Lesen und Schreiben
- **Issues:** Lesen und Schreiben
- **Pull requests:** Lesen und Schreiben

Code Review verwendet Lesezugriff auf Inhalte und Schreibzugriff auf Pull Requests. Der breitere Berechtigungssatz unterstützt auch [GitHub Actions](#), wenn Sie diese später aktivieren.

Step 4: Wählen Sie Repositories aus

Wählen Sie aus, welche Repositories für Code Review aktiviert werden sollen. Wenn Sie ein Repository nicht sehen, stellen Sie sicher, dass Sie der Claude GitHub App während der Installation Zugriff darauf gewährt haben. Sie können später weitere Repositories hinzufügen.

Step 5: Legen Sie Review-Trigger pro Repo fest

Nach Abschluss des Setups zeigt der Code Review Bereich Ihre Repositories in einer Tabelle an. Verwenden Sie für jedes Repository das Dropdown-Menü, um auszuwählen, wann Reviews ausgeführt werden:

- **Nach PR-Erstellung nur:** Review wird einmal ausgeführt, wenn ein PR geöffnet oder als bereit zur Überprüfung markiert wird
- **Nach jedem Push zum PR-Branch:** Review wird bei jedem Push ausgeführt, erkennt neue Probleme, während sich der PR entwickelt, und löst Threads automatisch auf, wenn Sie gekennzeichnete Probleme beheben

Das Überprüfen bei jedem Push führt zu mehr Reviews und höheren Kosten. Beginnen Sie mit der PR-Erstellung nur und wechseln Sie zu On-Push für Repos, in denen Sie kontinuierliche Abdeckung und automatische Thread-Bereinigung wünschen.

Die Repositories-Tabelle zeigt auch die durchschnittlichen Kosten pro Review für jedes Repo basierend auf der letzten Aktivität. Verwenden Sie das Zeilenaktionsmenü, um Code Review pro Repository ein- oder auszuschalten, oder um ein Repository vollständig zu entfernen.

Um das Setup zu überprüfen, öffnen Sie einen Test-PR. Ein Check Run mit dem Namen **Claude Code Review** wird innerhalb weniger Minuten angezeigt. Wenn dies nicht der Fall ist, bestätigen Sie, dass das Repository in Ihren Admin-Einstellungen aufgeführt ist und die Claude GitHub App Zugriff darauf hat.

Reviews anpassen

Code Review liest zwei Dateien aus Ihrem Repository, um zu steuern, was es kennzeichnet. Beide sind zusätzlich zu den standardmäßigen Korrektheitsprüfungen:

- **CLAUDE.md** : gemeinsame Projektanweisungen, die Claude Code für alle Aufgaben verwendet, nicht nur für Reviews. Verwenden Sie es, wenn die Anleitung auch für interaktive Claude Code Sitzungen gilt.
- **REVIEW.md** : nur Review-Anleitung, die ausschließlich während Code Reviews gelesen wird. Verwenden Sie es für Regeln, die streng damit zu tun haben, was während der Überprüfung gekennzeichnet oder übersprungen werden soll, und würde Ihre allgemeine **CLAUDE.md** nicht überladen.

CLAUDE.md

Code Review liest Ihre Repository- **CLAUDE.md** Dateien und behandelt neu eingeführte Verstöße als Nit-Level-Erkenntnisse. Dies funktioniert bidirektional: Wenn Ihr PR Code so ändert, dass eine **CLAUDE.md** Aussage veraltet wird, kennzeichnet Claude, dass die Dokumentation aktualisiert werden muss.

Claude liest **CLAUDE.md** Dateien auf jeder Ebene Ihrer Verzeichnishierarchie, sodass Regeln in einer Unterverzeichnis- **CLAUDE.md** nur auf Dateien unter diesem Pfad angewendet werden. Weitere Informationen zur Funktionsweise von **CLAUDE.md** finden Sie in der [Memory-Dokumentation](#).

Für Review-spezifische Anleitung, die Sie nicht auf allgemeine Claude Code Sitzungen angewendet haben möchten, verwenden Sie stattdessen **REVIEW.md**.

REVIEW.md

Fügen Sie eine `REVIEW.md` Datei zu Ihrem Repository-Root für Review-spezifische Regeln hinzu. Verwenden Sie es zum Kodieren von:

- Unternehmens- oder Team-Stilrichtlinien: “frühe Returns gegenüber verschachtelten Bedingungen bevorzugen”
- Sprach- oder Framework-spezifische Konventionen, die nicht von Lintern abgedeckt werden
- Dinge, die Claude immer kennzeichnen sollte: “jede neue API-Route muss einen Integrationstest haben”
- Dinge, die Claude überspringen sollte: “keine Kommentare zur Formatierung in generiertem Code unter `/gen/`”

Beispiel `REVIEW.md` :

```
## Code Review Richtlinien

### Immer überprüfen
- Neue API-Endpunkte haben entsprechende Integrationstests
- Datenbankmigrationen sind rückwärtskompatibel
- Fehlermeldungen geben keine internen Details an Benutzer preis

### Stil
- `match` Anweisungen gegenüber verketteten `instance` Überprüfungen bevorzugen
- Strukturiertes Logging verwenden, nicht f-String-Interpolation in Log-Aufrufen

### Überspringen
- Generierte Dateien unter `src/gen/`
- Nur Formatierungsänderungen in `*.lock` Dateien
```

Claude erkennt `REVIEW.md` automatisch im Repository-Root. Keine Konfiguration erforderlich.

Nutzung anzeigen

Gehen Sie zu claude.ai/analytics/code-review, um Code Review Aktivität in Ihrer Organisation zu sehen. Das Dashboard zeigt:

Bereich	Was es zeigt
PRs überprüft	Tägliche Anzahl der überprüften Pull Requests über den ausgewählten Zeitraum
Kosten wöchentlich	Wöchentliche Ausgaben für Code Review
Feedback	Anzahl der Review-Kommentare, die automatisch aufgelöst wurden, weil ein Entwickler das Problem behoben hat
Repository-Aufschlüsselung	Pro-Repo-Anzahl der überprüften PRs und aufgelösten Kommentare

Die Repositories-Tabelle in Admin-Einstellungen zeigt auch durchschnittliche Kosten pro Review für jedes Repo.

Preisgestaltung

Code Review wird basierend auf der Token-Nutzung abgerechnet. Reviews kosten durchschnittlich \$15-25, skalierend mit PR-Größe, Codebasis-Komplexität und wie viele Probleme Verifizierung erfordern. Code Review Nutzung wird separat über [zusätzliche Nutzung](#) abgerechnet und zählt nicht gegen die in Ihrem Plan enthaltene Nutzung.

Der Review-Trigger, den Sie wählen, beeinflusst die Gesamtkosten:

- **Nach PR-Erstellung nur:** wird einmal pro PR ausgeführt
- **Nach jedem Push:** wird bei jedem Commit ausgeführt, multipliziert die Kosten mit der Anzahl der Pushes

Kosten erscheinen auf Ihrer Anthropic-Rechnung, unabhängig davon, ob Ihre Organisation AWS Bedrock oder Google Vertex AI für andere Claude Code Funktionen verwendet. Um eine monatliche Ausgabenbegrenzung für Code Review festzulegen, gehen Sie zu [claude.ai/admin-settings/usage](#) und konfigurieren Sie das Limit für den Claude Code Review Service.

Überwachen Sie Ausgaben über das wöchentliche Kostendiagramm in [Analytics](#) oder die durchschnittliche Kostenspalte pro Repo in Admin-Einstellungen.

Verwandte Ressourcen

Code Review ist so konzipiert, dass es neben dem Rest von Claude Code funktioniert. Wenn Sie Reviews lokal ausführen möchten, bevor Sie einen PR öffnen, eine selbst gehostete Einrichtung benötigen oder tiefer verstehen möchten, wie `CLAUDE.md` Claudes Verhalten über Tools hinweg prägt, sind diese Seiten gute nächste Schritte:

- [Plugins](#): Durchsuchen Sie den Plugin-Marktplatz, einschließlich eines `code-review` Plugins zum Ausführen von On-Demand-Reviews lokal vor dem Pushen
- [GitHub Actions](#): Führen Sie Claude in Ihren eigenen GitHub Actions Workflows für benutzerdefinierte Automatisierung über Code Review hinaus aus
- [GitLab CI/CD](#): Selbst gehostete Claude-Integration für GitLab-Pipelines
- [Memory](#): Wie `CLAUDE.md` Dateien über Claude Code funktionieren
- [Analytics](#): Verfolgen Sie Claude Code Nutzung über Code Review hinaus

Part 7: CI/CD & Integrations

Claude Code GitHub Actions

Erfahren Sie, wie Sie Claude Code in Ihren Entwicklungs-Workflow mit Claude Code GitHub Actions integrieren

Claude Code GitHub Actions bringt KI-gestützte Automatisierung in Ihren GitHub-Workflow. Mit einer einfachen `@Claude`-Erwähnung in einem beliebigen PR oder Issue kann Claude Ihren Code analysieren, Pull Requests erstellen, Features implementieren und Bugs beheben – alles während er die Standards Ihres Projekts befolgt. Für automatische Reviews, die auf jedem PR ohne Trigger gepostet werden, siehe [GitHub Code Review](#).

Note:

Claude Code GitHub Actions basiert auf dem [Claude Agent SDK](#), das die programmgesteuerte Integration von Claude Code in Ihre Anwendungen ermöglicht. Sie können das SDK verwenden, um benutzerdefinierte Automatisierungs-Workflows über GitHub Actions hinaus zu erstellen.

Info:

Claude Opus 4.6 ist jetzt verfügbar. Claude Code GitHub Actions verwenden standardmäßig Sonnet. Um Opus 4.6 zu verwenden, konfigurieren Sie den [Model-Parameter](#) auf `claude-opus-4-6`.

Warum Claude Code GitHub Actions verwenden?

- **Sofortige PR-Erstellung:** Beschreiben Sie, was Sie benötigen, und Claude erstellt einen vollständigen PR mit allen notwendigen Änderungen
- **Automatisierte Code-Implementierung:** Verwandeln Sie Issues in funktionierenden Code mit einem einzigen Befehl
- **Befolgt Ihre Standards:** Claude respektiert Ihre `CLAUDE.md`-Richtlinien und vorhandene Code-Muster
- **Einfaches Setup:** Beginnen Sie in Minuten mit unserem Installer und API-Schlüssel
- **Sicher von Anfang an:** Ihr Code bleibt auf Githubs Runnern

Was kann Claude tun?

Claude Code bietet eine leistungsstarke GitHub Action, die verändert, wie Sie mit Code arbeiten:

Claude Code Action

Diese GitHub Action ermöglicht es Ihnen, Claude Code in Ihren GitHub Actions-Workflows auszuführen. Sie können dies verwenden, um jeden benutzerdefinierten Workflow auf Basis von Claude Code zu erstellen.

[Repository anzeigen](#) →

Setup

Schnelles Setup

Der einfachste Weg, diese Action einzurichten, ist über Claude Code im Terminal. Öffnen Sie einfach Claude und führen Sie `/install-github-app` aus.

Dieser Befehl führt Sie durch die Einrichtung der GitHub-App und erforderlichen Secrets.

Note:

- Sie müssen ein Repository-Admin sein, um die GitHub-App zu installieren und Secrets hinzuzufügen
- Die GitHub-App fordert Lese- und Schreibberechtigungen für Contents, Issues und Pull Requests an
- Diese Schnellstart-Methode ist nur für direkte Claude API-Benutzer verfügbar. Wenn Sie AWS Bedrock oder Google Vertex AI verwenden, siehe bitte den Abschnitt [Verwendung mit AWS Bedrock & Google Vertex AI](#).

Manuelles Setup

Wenn der Befehl `/install-github-app` fehlschlägt oder Sie manuelles Setup bevorzugen, folgen Sie bitte diesen manuellen Setup-Anweisungen:

1. **Installieren Sie die Claude GitHub-App** in Ihrem Repository: <https://github.com/apps/claude>

Die Claude GitHub-App erfordert die folgenden Repository-Berechtigungen:

- **Contents:** Lesen & Schreiben (zum Ändern von Repository-Dateien)
- **Issues:** Lesen & Schreiben (zum Antworten auf Issues)

- **Pull requests:** Lesen & Schreiben (zum Erstellen von PRs und Pushen von Änderungen)

Weitere Details zu Sicherheit und Berechtigungen finden Sie in der [Sicherheitsdokumentation](#).

2. **Fügen Sie ANTHROPIC_API_KEY** zu Ihren Repository-Secrets hinzu ([Erfahren Sie, wie Sie Secrets in GitHub Actions verwenden](#))
3. **Kopieren Sie die Workflow-Datei** von [examples/claude.yml](#) in das Verzeichnis `.github/workflows/` Ihres Repositories

Tip:

Nach Abschluss des Schnellstarts oder manuellen Setups testen Sie die Action, indem Sie `@claude` in einem Issue- oder PR-Kommentar markieren.

Upgrade von Beta

Warning:

Claude Code GitHub Actions v1.0 führt Breaking Changes ein, die ein Update Ihrer Workflow-Dateien erfordern, um von der Beta-Version auf v1.0 zu aktualisieren.

Wenn Sie derzeit die Beta-Version von Claude Code GitHub Actions verwenden, empfehlen wir Ihnen, Ihre Workflows auf die GA-Version zu aktualisieren. Die neue Version vereinfacht die Konfiguration und fügt leistungsstarke neue Funktionen wie automatische Modusterkennung hinzu.

Wesentliche Änderungen

Alle Beta-Benutzer müssen diese Änderungen an ihren Workflow-Dateien vornehmen, um zu aktualisieren:

1. **Aktualisieren Sie die Action-Version:** Ändern Sie `@beta` zu `@v1`
2. **Entfernen Sie die Moduskonfiguration:** Löschen Sie `mode: "tag"` oder `mode: "agent"` (wird jetzt automatisch erkannt)
3. **Aktualisieren Sie Prompt-Eingaben:** Ersetzen Sie `direct_prompt` durch `prompt`
4. **Verschieben Sie CLI-Optionen:** Konvertieren Sie `max_turns`, `model`, `custom_instructions` usw. zu `claude_args`

Breaking Changes Referenz

Alte Beta-Eingabe	Neue v1.0-Eingabe
mode	(Entfernt - automatisch erkannt)
direct_prompt	prompt
override_prompt	prompt mit GitHub-Variablen
custom_instructions	claude_args: --append-system-prompt
max_turns	claude_args: --max-turns
model	claude_args: --model
allowed_tools	claude_args: --allowedTools
disallowed_tools	claude_args: --disallowedTools
claude_env	settings JSON-Format

Vorher- und Nachher-Beispiel

Beta-Version:

```
- uses: anthropics/claude-code-action@beta
  with:
    mode: "tag"
    direct_prompt: "Review this PR for security issues"
    anthropic_api_key: "${ secrets.ANTHROPIC_API_KEY }"
    custom_instructions: "Follow our coding standards"
    max_turns: "10"
    model: "claude-sonnet-4-6"
```

GA-Version (v1.0):

```
- uses: anthropics/claude-code-action@v1
  with:
    prompt: "Review this PR for security issues"
    anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
    claude_args: |
      --append-system-prompt "Follow our coding standards"
      --max-turns 10
      --model claude-sonnet-4-6
```

Tip:

Die Action erkennt jetzt automatisch, ob sie im interaktiven Modus (antwortet auf **@claude** -Erwähnungen) oder im Automatisierungsmodus (wird sofort mit einem Prompt ausgeführt) ausgeführt werden soll, basierend auf Ihrer Konfiguration.

Beispiel-Anwendungsfälle

Claude Code GitHub Actions kann Ihnen bei einer Vielzahl von Aufgaben helfen. Das [Beispielverzeichnis](#) enthält einsatzbereite Workflows für verschiedene Szenarien.

Basis-Workflow

```
name: Claude Code
on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
jobs:
  claude:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          # Responds to @claude mentions in comments
```

Verwendung von skills

```
name: Code Review
on:
  pull_request:
    types: [opened, synchronize]
jobs:
  review:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          prompt: "Review this pull request for code quality, correctness, and
security. Analyze the diff, then post your findings as review comments."
          claude_args: "--max-turns 5"
```

Benutzerdefinierte Automatisierung mit Prompts

```
name: Daily Report
on:
  schedule:
    - cron: "0 9 * * *"
jobs:
  report:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          prompt: "Generate a summary of yesterday's commits and open issues"
          claude_args: "--model opus"
```

Häufige Anwendungsfälle

In Issue- oder PR-Kommentaren:

```
@claude implement this feature based on the issue description
@claude how should I implement user authentication for this endpoint?
@claude fix the TypeError in the user dashboard component
```

Claude wird automatisch den Kontext analysieren und angemessen antworten.

Best Practices

CLAUDE.md-Konfiguration

Erstellen Sie eine **CLAUDE.md**-Datei im Root-Verzeichnis Ihres Repositories, um Code-Style-Richtlinien, Review-Kriterien, projektspezifische Regeln und bevorzugte Muster zu definieren. Diese Datei leitet Claudes Verständnis Ihrer Projektstandards.

Sicherheitsüberlegungen

Warning:

Committieren Sie API-Schlüssel niemals direkt in Ihr Repository.

Umfassende Sicherheitsleitlinien einschließlich Berechtigungen, Authentifizierung und Best Practices finden Sie in der [Claude Code Action-Sicherheitsdokumentation](#).

Verwenden Sie immer GitHub Secrets für API-Schlüssel:

- Fügen Sie Ihren API-Schlüssel als Repository-Secret namens **ANTHROPIC_API_KEY** hinzu
- Referenzieren Sie ihn in Workflows: **anthropic_api_key: \${{ secrets.ANTHROPIC_API_KEY }}**
- Begrenzen Sie Action-Berechtigungen auf nur das Notwendigste
- Überprüfen Sie Claudes Vorschläge vor dem Mergen

Verwenden Sie immer GitHub Secrets (zum Beispiel **\${{ secrets.ANTHROPIC_API_KEY }}**), anstatt API-Schlüssel direkt in Ihren Workflow-Dateien zu hardcodieren.

Optimierung der Leistung

Verwenden Sie Issue-Templates, um Kontext bereitzustellen, halten Sie Ihre **CLAUDE.md** prägnant und fokussiert, und konfigurieren Sie angemessene Timeouts für Ihre Workflows.

CI-Kosten

Bei der Verwendung von Claude Code GitHub Actions sollten Sie sich der damit verbundenen Kosten bewusst sein:

GitHub Actions-Kosten:

- Claude Code wird auf GitHub-gehosteten Runnern ausgeführt, die Ihre GitHub Actions-Minuten verbrauchen
- Siehe [Abrechnungsdokumentation von GitHub](#) für detaillierte Preise und Minuten-limits

API-Kosten:

- Jede Claude-Interaktion verbraucht API-Token basierend auf der Länge von Prompts und Antworten
- Die Token-Nutzung variiert je nach Aufgabenkomplexität und Codebase-Größe
- Siehe [Claudes Preisseite](#) für aktuelle Token-Raten

Tipps zur Kostenoptimierung:

- Verwenden Sie spezifische `@claude` -Befehle, um unnötige API-Aufrufe zu reduzieren
- Konfigurieren Sie angemessene `--max-turns` in `claude_args`, um übermäßige Iterationen zu verhindern
- Legen Sie Workflow-Level-Timeouts fest, um unkontrollierte Jobs zu vermeiden
- Erwägen Sie die Verwendung von Githubs Concurrency-Kontrollen, um parallele Ausführungen zu begrenzen

Konfigurationsbeispiele

Die Claude Code Action v1 vereinfacht die Konfiguration mit einheitlichen Parametern:

```
- uses: anthropics/claude-code-action@v1
  with:
    anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
    prompt: "Your instructions here" # Optional
    claude_args: "--max-turns 5" # Optional CLI arguments
```

Wichtige Funktionen:

- **Einheitliche Prompt-Schnittstelle** - Verwenden Sie `prompt` für alle Anweisungen
- **Skills** - Rufen Sie installierte [skills](#) direkt aus dem Prompt auf

- **CLI-Passthrough** - Jedes Claude Code CLI-Argument über `claude_args`
- **Flexible Trigger** - Funktioniert mit jedem GitHub-Event

Besuchen Sie das [Beispielverzeichnis](#) für vollständige Workflow-Dateien.

Tip:

Wenn Claude auf Issue- oder PR-Kommentare antwortet, antwortet er automatisch auf @claude-Erwähnungen. Für andere Events verwenden Sie den `prompt`-Parameter, um Anweisungen bereitzustellen.

Verwendung mit AWS Bedrock & Google Vertex AI

Für Unternehmensumgebungen können Sie Claude Code GitHub Actions mit Ihrer eigenen Cloud-Infrastruktur verwenden. Dieser Ansatz gibt Ihnen Kontrolle über Datenresidenz und Abrechnung, während Sie die gleiche Funktionalität beibehalten.

Voraussetzungen

Bevor Sie Claude Code GitHub Actions mit Cloud-Providern einrichten, benötigen Sie:

Für Google Cloud Vertex AI:

1. Ein Google Cloud-Projekt mit aktiviertem Vertex AI
2. Workload Identity Federation für GitHub Actions konfiguriert
3. Ein Service-Konto mit erforderlichen Berechtigungen
4. Eine GitHub-App (empfohlen) oder verwenden Sie das Standard-GITHUB_TOKEN

Für AWS Bedrock:

1. Ein AWS-Konto mit aktiviertem Amazon Bedrock
2. GitHub OIDC Identity Provider in AWS konfiguriert
3. Eine IAM-Rolle mit Bedrock-Berechtigungen
4. Eine GitHub-App (empfohlen) oder verwenden Sie das Standard-GITHUB_TOKEN

Step 1: Erstellen Sie eine benutzerdefinierte GitHub-App (empfohlen für 3P-Provider)

Für beste Kontrolle und Sicherheit bei der Verwendung von 3P-Providern wie Vertex AI oder Bedrock empfehlen wir, Ihre eigene GitHub-App zu erstellen:

1. Gehen Sie zu <https://github.com/settings/apps/new>

2. Füllen Sie die grundlegenden Informationen aus:

- **GitHub App-Name:** Wählen Sie einen eindeutigen Namen (z. B. "YourOrg Claude Assistant")
- **Homepage URL:** Website Ihrer Organisation oder die Repository-URL

1. Konfigurieren Sie die App-Einstellungen:

- **Webhooks:** Deaktivieren Sie „Active“ (nicht erforderlich für diese Integration)

1. Legen Sie die erforderlichen Berechtigungen fest:

• **Repository-Berechtigungen:**

- Contents: Lesen & Schreiben
- Issues: Lesen & Schreiben
- Pull requests: Lesen & Schreiben

1. Klicken Sie auf „Create GitHub App“

2. Nach der Erstellung klicken Sie auf „Generate a private key“ und speichern Sie die heruntergeladene `.pem`-Datei

3. Notieren Sie sich Ihre App-ID von der App-Einstellungsseite

4. Installieren Sie die App in Ihrem Repository:

- Klicken Sie auf der Einstellungsseite Ihrer App auf „Install App“ in der linken Seitenleiste
- Wählen Sie Ihr Konto oder Ihre Organisation
- Wählen Sie „Only select repositories“ und wählen Sie das spezifische Repository
- Klicken Sie auf „Install“

1. Fügen Sie den privaten Schlüssel als Secret zu Ihrem Repository hinzu:

- Gehen Sie zu den Einstellungen Ihres Repositories → Secrets and variables → Actions
- Erstellen Sie ein neues Secret namens `APP_PRIVATE_KEY` mit dem Inhalt der `.pem`-Datei

1. Fügen Sie die App-ID als Secret hinzu:

- Erstellen Sie ein neues Secret namens `APP_ID` mit der ID Ihrer GitHub-App

Note:

Diese App wird mit der [actions/create-github-app-token](#)-Action verwendet, um Authentifizierungstoken in Ihren Workflows zu generieren.

Alternative für Claude API oder wenn Sie keine eigene Github-App einrichten möchten: Verwenden Sie die offizielle Anthropic-App:

1. Installieren Sie von: <https://github.com/apps/claude>
2. Keine zusätzliche Konfiguration erforderlich für die Authentifizierung

Step 2: Konfigurieren Sie die Cloud-Provider-Authentifizierung

Wählen Sie Ihren Cloud-Provider und richten Sie sichere Authentifizierung ein:

Konfigurieren Sie AWS, um GitHub Actions die sichere Authentifizierung ohne Speicherung von Anmeldedaten zu ermöglichen.

***Sicherheitshinweis:** Verwenden Sie Repository-spezifische Konfigurationen und gewähren Sie nur die minimal erforderlichen Berechtigungen.*

Erforderliches Setup:

1. Aktivieren Sie Amazon Bedrock:

- Fordern Sie Zugriff auf Claude-Modelle in Amazon Bedrock an
- Für regionsübergreifende Modelle fordern Sie Zugriff in allen erforderlichen Regionen an

1. Richten Sie GitHub OIDC Identity Provider ein:

- Provider-URL: `https://token.actions.githubusercontent.com`
- Audience: `sts.amazonaws.com`

1. Erstellen Sie eine IAM-Rolle für GitHub Actions:

- Vertrauenswürdiger Entity-Typ: Web Identity
- Identity Provider: `token.actions.githubusercontent.com`
- Berechtigungen: `AmazonBedrockFullAccess` -Richtlinie
- Konfigurieren Sie die Trust-Richtlinie für Ihr spezifisches Repository

Erforderliche Werte:

Nach dem Setup benötigen Sie:

- **AWS_ROLE_TO_ASSUME:** Das ARN der IAM-Rolle, die Sie erstellt haben

Tip:

OIDC ist sicherer als die Verwendung statischer AWS-Zugriffstasten, da Anmeldedaten temporär sind und automatisch rotiert werden.

Siehe [AWS-Dokumentation](#) für detaillierte OIDC-Setup-Anweisungen.

Konfigurieren Sie Google Cloud, um GitHub Actions die sichere Authentifizierung ohne Speicherung von Anmeldedaten zu ermöglichen.

***Sicherheitshinweis:** Verwenden Sie Repository-spezifische Konfigurationen und gewähren Sie nur die minimal erforderlichen Berechtigungen.*

Erforderliches Setup:

1. Aktivieren Sie APIs in Ihrem Google Cloud-Projekt:

- IAM Credentials API
- Security Token Service (STS) API
- Vertex AI API

1. Erstellen Sie Workload Identity Federation-Ressourcen:

- Erstellen Sie einen Workload Identity Pool
- Fügen Sie einen GitHub OIDC-Provider mit hinzu:
- Issuer: `https://token.actions.githubusercontent.com`
- Attribut-Mappings für Repository und Owner
- **Sicherheitsempfehlung:** Verwenden Sie Repository-spezifische Attribut-Bedingungen

1. Erstellen Sie ein Service-Konto:

- Gewähren Sie nur die Rolle `Vertex AI User`
- **Sicherheitsempfehlung:** Erstellen Sie ein dediziertes Service-Konto pro Repository

1. Konfigurieren Sie IAM-Bindungen:

- Erlauben Sie dem Workload Identity Pool, das Service-Konto zu imitieren
- **Sicherheitsempfehlung:** Verwenden Sie Repository-spezifische Principal-Sets

Erforderliche Werte:

Nach dem Setup benötigen Sie:

- **GCP_WORKLOAD_IDENTITY_PROVIDER:** Der vollständige Provider-Resourcename

- **GCP_SERVICE_ACCOUNT**: Die Service-Konto-E-Mail-Adresse

Tip:

Workload Identity Federation eliminiert die Notwendigkeit herunterladbarer Service-Konto-Schlüssel und verbessert die Sicherheit.

Für detaillierte Setup-Anweisungen konsultieren Sie die [Google Cloud Workload Identity Federation-Dokumentation](#).

Step 3: Fügen Sie erforderliche Secrets hinzu

Fügen Sie die folgenden Secrets zu Ihrem Repository hinzu (Settings → Secrets and variables → Actions):

Für Claude API (direkt):

1. Für API-Authentifizierung:

- **ANTHROPIC_API_KEY**: Ihr Claude API-Schlüssel von console.anthropic.com

1. Für GitHub-App (wenn Sie Ihre eigene App verwenden):

- **APP_ID**: Die ID Ihrer GitHub-App
- **APP_PRIVATE_KEY**: Der Inhalt des privaten Schlüssels (.pem)

Für Google Cloud Vertex AI

1. Für GCP-Authentifizierung:

- **GCP_WORKLOAD_IDENTITY_PROVIDER**
- **GCP_SERVICE_ACCOUNT**

1. Für GitHub-App (wenn Sie Ihre eigene App verwenden):

- **APP_ID**: Die ID Ihrer GitHub-App
- **APP_PRIVATE_KEY**: Der Inhalt des privaten Schlüssels (.pem)

Für AWS Bedrock

1. Für AWS-Authentifizierung:

- **AWS_ROLE_TO_ASSUME**

1. Für GitHub-App (wenn Sie Ihre eigene App verwenden):

- **APP_ID**: Die ID Ihrer GitHub-App

- `APP_PRIVATE_KEY` : Der Inhalt des privaten Schlüssels (.pem)

Step 4: Erstellen Sie Workflow-Dateien

Erstellen Sie GitHub Actions-Workflow-Dateien, die sich in Ihren Cloud-Provider integrieren. Die folgenden Beispiele zeigen vollständige Konfigurationen für AWS Bedrock und Google Vertex AI:

Voraussetzungen:

- AWS Bedrock-Zugriff aktiviert mit Claude-Modell-Berechtigungen
- GitHub als OIDC-Identity-Provider in AWS konfiguriert
- IAM-Rolle mit Bedrock-Berechtigungen, die GitHub Actions vertraut

Erforderliche GitHub-Secrets:

Secret-Name	Beschreibung
<code>AWS_ROLE_TO_ASSUME</code>	ARN der IAM-Rolle für Bedrock-Zugriff
<code>APP_ID</code>	Ihre GitHub-App-ID (aus App-Einstellungen)
<code>APP_PRIVATE_KEY</code>	Der private Schlüssel, den Sie für Ihre GitHub-App generiert haben

```

name: Claude PR Action

permissions:
  contents: write
  pull-requests: write
  issues: write
  id-token: write

on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
  issues:
    types: [opened, assigned]

jobs:
  claude-pr:
    if: |
      (github.event_name == 'issue_comment' && contains(github.event.comment.body,
        '@claude')) ||
      (github.event_name == 'pull_request_review_comment' &&
        contains(github.event.comment.body, '@claude')) ||
      (github.event_name == 'issues' && contains(github.event.issue.body,
        '@claude'))
    runs-on: ubuntu-latest
    env:
      AWS_REGION: us-west-2
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Generate GitHub App token
        id: app-token
        uses: actions/create-github-app-token@v2
        with:
          app-id: ${ secrets.APP_ID }
          private-key: ${ secrets.APP_PRIVATE_KEY }

```



```
- name: Configure AWS Credentials (OIDC)
  uses: aws-actions/configure-aws-credentials@v4
  with:
    role-to-assume: ${{ secrets.AWS_ROLE_TO_ASSUME }}
    aws-region: us-west-2

- uses: anthropics/claude-code-action@v1
  with:
    github_token: ${{ steps.app-token.outputs.token }}
    use_bedrock: "true"
    claude_args: '--model us.anthropic.claude-sonnet-4-6 --max-turns 10'
```

Tip:
Das Modell-ID-Format für Bedrock enthält ein Regions-Präfix (zum Beispiel `us.anthropic.claude-sonnet-4-6`).

Voraussetzungen:

- Vertex AI API in Ihrem GCP-Projekt aktiviert
- Workload Identity Federation für GitHub konfiguriert
- Service-Konto mit Vertex AI-Berechtigungen

Erforderliche GitHub-Secrets:

Secret-Name	Beschreibung
GCP_WORKLOAD_IDENTITY_PROVIDER	Workload Identity Provider-Ressourcenname
GCP_SERVICE_ACCOUNT	Service-Konto-E-Mail mit Vertex AI-Zugriff
APP_ID	Ihre GitHub-App-ID (aus App-Einstellungen)
APP_PRIVATE_KEY	Der private Schlüssel, den Sie für Ihre GitHub-App generiert haben

```

name: Claude PR Action

permissions:
  contents: write
  pull-requests: write
  issues: write
  id-token: write

on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
  issues:
    types: [opened, assigned]

jobs:
  claude-pr:
    if: |
      (github.event_name == 'issue_comment' && contains(github.event.comment.body,
        '@claude')) ||
      (github.event_name == 'pull_request_review_comment' &&
        contains(github.event.comment.body, '@claude')) ||
      (github.event_name == 'issues' && contains(github.event.issue.body,
        '@claude'))
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Generate GitHub App token
        id: app-token
        uses: actions/create-github-app-token@v2
        with:
          app-id: ${ secrets.APP_ID }
          private-key: ${ secrets.APP_PRIVATE_KEY }

      - name: Authenticate to Google Cloud

```

```

    id: auth
    uses: google-github-actions/auth@v2
    with:
      workload_identity_provider: $
    {{ secrets.GCP_WORKLOAD_IDENTITY_PROVIDER }}
      service_account: ${{ secrets.GCP_SERVICE_ACCOUNT }}

- uses: anthropics/claude-code-action@v1
  with:
    github_token: ${{ steps.app-token.outputs.token }}
    trigger_phrase: "@claude"
    use_vertex: "true"
    claude_args: '--model claude-sonnet-4@20250514 --max-turns 10'
  env:
    ANTHROPIC_VERTEX_PROJECT_ID: ${{ steps.auth.outputs.project_id }}
    CLOUD_ML_REGION: us-east5
    VERTEX_REGION_CLAUDE_3_7_SONNET: us-east5

```

Tip:

Die Projekt-ID wird automatisch aus dem Google Cloud-Authentifizierungsschritt abgerufen, daher müssen Sie sie nicht hardcodieren.

Fehlerbehebung

Claude antwortet nicht auf @claude-Befehle

Überprüfen Sie, dass die GitHub-App korrekt installiert ist, stellen Sie sicher, dass Workflows aktiviert sind, überprüfen Sie, dass der API-Schlüssel in Repository-Secrets gesetzt ist, und bestätigen Sie, dass der Kommentar `@claude` enthält (nicht `/claude`).

CI wird nicht auf Claudes Commits ausgeführt

Stellen Sie sicher, dass Sie die GitHub-App oder benutzerdefinierte App verwenden (nicht Actions-Benutzer), überprüfen Sie, dass Workflow-Trigger die erforderlichen Events enthalten, und überprüfen Sie, dass App-Berechtigungen CI-Trigger enthalten.

Authentifizierungsfehler

Bestätigen Sie, dass der API-Schlüssel gültig ist und ausreichende Berechtigungen hat. Für Bedrock/Vertex überprüfen Sie die Anmeldedaten-Konfiguration und stellen Sie sicher, dass Secrets in Workflows korrekt benannt sind.

Erweiterte Konfiguration

Action-Parameter

Die Claude Code Action v1 verwendet eine vereinfachte Konfiguration:

Parameter	Beschreibung	Erforderlich
<code>prompt</code>	Anweisungen für Claude (Klartext oder ein <code>skill</code> -Name)	Nein*
<code>claude_args</code>	CLI-Argumente, die an Claude Code übergeben werden	Nein
<code>anthropic_api_key</code>	Claude API-Schlüssel	Ja**
<code>github_token</code>	GitHub-Token für API-Zugriff	Nein
<code>trigger_phrase</code>	Benutzerdefinierte Trigger-Phrase (Standard: "@claude")	Nein
<code>use_bedrock</code>	Verwenden Sie AWS Bedrock statt Claude API	Nein
<code>use_vertex</code>	Verwenden Sie Google Vertex AI statt Claude API	Nein

*Prompt ist optional – wenn für Issue/PR-Kommentare weggelassen, antwortet Claude auf Trigger-Phrase

**Erforderlich für direkte Claude API, nicht für Bedrock/Vertex

Übergeben Sie CLI-Argumente

Der Parameter `claude_args` akzeptiert alle Claude Code CLI-Argumente:

```
claude_args: "--max-turns 5 --model claude-sonnet-4-6 --mcp-config /path/to/config.json"
```

Häufige Argumente:

- `--max-turns` : Maximale Gesprächs-Turns (Standard: 10)
- `--model` : Zu verwendendes Modell (zum Beispiel `claude-sonnet-4-6`)
- `--mcp-config` : Pfad zur MCP-Konfiguration
- `--allowed-tools` : Komma-getrennte Liste zulässiger Tools
- `--debug` : Debug-Ausgabe aktivieren

Alternative Integrationsmethoden

Während der Befehl `/install-github-app` der empfohlene Ansatz ist, können Sie auch:

- **Benutzerdefinierte GitHub-App:** Für Organisationen, die Branded-Benutzernamen oder benutzerdefinierte Authentifizierungs-Flows benötigen. Erstellen Sie Ihre eigene GitHub-App mit erforderlichen Berechtigungen (Contents, Issues, Pull Requests) und verwenden Sie die `actions/create-github-app-token`-Action, um Token in Ihren Workflows zu generieren.
- **Manuelle GitHub Actions:** Direkte Workflow-Konfiguration für maximale Flexibilität
- **MCP-Konfiguration:** Dynamisches Laden von Model Context Protocol-Servern

Siehe die [Claude Code Action-Dokumentation](#) für detaillierte Leitfäden zu Authentifizierung, Sicherheit und erweiterte Konfiguration.

Anpassung von Claudes Verhalten

Sie können Claudes Verhalten auf zwei Arten konfigurieren:

1. **CLAUDE.md:** Definieren Sie Coding-Standards, Review-Kriterien und projektspezifische Regeln in einer `CLAUDE.md` -Datei im Root-Verzeichnis Ihres Repositories. Claude wird diese Richtlinien beim Erstellen von PRs und Antworten auf Anfragen befolgen. Weitere Details finden Sie in unserer [Memory-Dokumentation](#).
2. **Benutzerdefinierte Prompts:** Verwenden Sie den Parameter `prompt` in der Workflow-Datei, um Workflow-spezifische Anweisungen bereitzustellen. Dies ermöglicht es Ihnen, Claudes Verhalten für verschiedene Workflows oder Aufgaben anzupassen.

Claude wird diese Richtlinien beim Erstellen von PRs und Antworten auf Anfragen befolgen.

Claude Code GitLab CI/CD

Erfahren Sie, wie Sie Claude Code in Ihren Entwicklungs-Workflow mit GitLab CI/CD integrieren

Info:

Claude Code für GitLab CI/CD befindet sich derzeit in der Beta-Phase. Funktionen und Funktionalität können sich weiterentwickeln, während wir die Erfahrung verfeinern.

Diese Integration wird von GitLab gepflegt. Für Support siehe das folgende [GitLab-Problem](#).

Note:

Diese Integration basiert auf der [Claude Code CLI](#) und [Agent SDK](#) und ermöglicht die programmgesteuerte Nutzung von Claude in Ihren CI/CD-Jobs und benutzerdefinierten Automatisierungs-Workflows.

Warum Claude Code mit GitLab verwenden?

- **Sofortige MR-Erstellung:** Beschreiben Sie, was Sie benötigen, und Claude schlägt einen vollständigen MR mit Änderungen und Erklärung vor
- **Automatisierte Implementierung:** Verwandeln Sie Probleme mit einem einzigen Befehl oder einer Erwähnung in funktionierenden Code
- **Projektbewusst:** Claude folgt Ihren `CLAUDE.md`-Richtlinien und vorhandenen Code-Mustern
- **Einfaches Setup:** Fügen Sie einen Job zu `.gitlab-ci.yml` und eine maskierte CI/CD-Variable hinzu
- **Enterprise-ready:** Wählen Sie Claude API, AWS Bedrock oder Google Vertex AI, um Anforderungen an Datenresidenz und Beschaffung zu erfüllen
- **Standardmäßig sicher:** Läuft in Ihren GitLab-Runnern mit Ihrem Branch-Schutz und Genehmigungen

Wie es funktioniert

Claude Code verwendet GitLab CI/CD, um KI-Aufgaben in isolierten Jobs auszuführen und Ergebnisse über MRs zurückzukommen:

1. **Ereignisgesteuerte Orchestrierung:** GitLab lauscht auf Ihre gewählten Trigger (zum Beispiel ein Kommentar, der `@claude` in einem Problem, MR oder Review-Thread erwähnt). Der Job sammelt Kontext aus dem Thread und Repository, erstellt Prompts aus dieser Eingabe und führt Claude Code aus.
2. **Provider-Abstraktion:** Verwenden Sie den Provider, der zu Ihrer Umgebung passt:
 - Claude API (SaaS)
 - AWS Bedrock (IAM-basierter Zugriff, regionsübergreifende Optionen)
 - Google Vertex AI (GCP-nativ, Workload Identity Federation)
3. **Sandboxed-Ausführung:** Jede Interaktion läuft in einem Container mit strikten Netzwerk- und Dateisystem-Regeln. Claude Code erzwingt Workspace-bezogene Berechtigungen, um Schreibvorgänge einzuschränken. Jede Änderung fließt durch einen MR, damit Reviewer den Diff sehen und Genehmigungen weiterhin gelten.

Wählen Sie regionale Endpunkte, um die Latenz zu reduzieren und Anforderungen an die Datensouveränität zu erfüllen, während Sie vorhandene Cloud-Vereinbarungen nutzen.

Was kann Claude tun?

Claude Code ermöglicht leistungsstarke CI/CD-Workflows, die verändern, wie Sie mit Code arbeiten:

- Erstellen und aktualisieren Sie MRs aus Problembeschreibungen oder Kommentaren
- Analysieren Sie Performance-Regressionstests und schlagen Sie Optimierungen vor
- Implementieren Sie Features direkt in einem Branch und öffnen Sie dann einen MR
- Beheben Sie Bugs und Regressionstests, die durch Tests oder Kommentare identifiziert wurden
- Antworten Sie auf Folgekommentare, um auf angeforderte Änderungen zu iterieren

Setup

Schnelles Setup

Der schnellste Weg zum Einstieg ist, einen minimalen Job zu Ihrer `.gitlab-ci.yml` hinzuzufügen und Ihren API-Schlüssel als maskierte Variable festzulegen.

1. Fügen Sie eine maskierte CI/CD-Variable hinzu

- Gehen Sie zu **Einstellungen** → **CI/CD** → **Variablen**
- Fügen Sie `ANTHROPIC_API_KEY` hinzu (maskiert, bei Bedarf geschützt)

2. Fügen Sie einen Claude-Job zu `.gitlab-ci.yml` hinzu


```

stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  # Passen Sie die Regeln an, um zu passen, wie Sie den Job auslösen möchten:
  # - manuelle Ausführungen
  # - Merge-Request-Ereignisse
  # - Web/API-Trigger, wenn ein Kommentar '@claude' enthält
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
  variables:
    GIT_STRATEGY: fetch
  before_script:
    - apk update
    - apk add --no-cache git curl bash
    - curl -fsSL https://claude.ai/install.sh | bash
  script:
    # Optional: Starten Sie einen GitLab MCP-Server, wenn Ihr Setup einen
bereitstellt
    - /bin/gitlab-mcp-server || true
    # Verwenden Sie AI_FLOW_*-Variablen beim Aufrufen über Web/API-Trigger mit
Kontext-Payloads
    - echo "$AI_FLOW_INPUT for $AI_FLOW_CONTEXT on $AI_FLOW_EVENT"
    - >
      claude
      -p "${AI_FLOW_INPUT:-'Review this MR and implement the requested changes'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp_gitlab"
      --debug

```

Nach dem Hinzufügen des Jobs und Ihrer `ANTHROPIC_API_KEY`-Variable testen Sie, indem Sie den Job manuell von **CI/CD** → **Pipelines** ausführen, oder lösen Sie ihn von einem MR aus, um Claude zu ermöglichen, Updates in einem Branch vorzuschlagen und bei Bedarf einen MR zu öffnen.

Note:

Um stattdessen auf AWS Bedrock oder Google Vertex AI auszuführen, siehe den Abschnitt [Verwendung mit AWS Bedrock & Google Vertex AI](#) unten für Authentifizierung und Umgebungss-
etup.

Manuelles Setup (empfohlen für Produktion)

Wenn Sie ein kontrolliertes Setup bevorzugen oder Enterprise-Provider benötigen:

1. Konfigurieren Sie Provider-Zugriff:

- **Claude API:** Erstellen und speichern Sie `ANTHROPIC_API_KEY` als maskierte CI/CD-Variable
- **AWS Bedrock: Konfigurieren Sie GitLab → AWS OIDC** und erstellen Sie eine IAM-Rolle für Bedrock
- **Google Vertex AI: Konfigurieren Sie Workload Identity Federation für GitLab → GCP**

2. Fügen Sie Projekt-Anmeldedaten für GitLab API-Operationen hinzu:

- Verwenden Sie `CI_JOB_TOKEN` standardmäßig, oder erstellen Sie ein Project Access Token mit `api`-Bereich
- Speichern Sie als `GITLAB_ACCESS_TOKEN` (maskiert), wenn Sie ein PAT verwenden

3. Fügen Sie den Claude-Job zu `.gitlab-ci.yml` hinzu (siehe Beispiele unten)**4. (Optional) Aktivieren Sie Mention-gesteuerte Trigger:**

- Fügen Sie einen Projekt-Webhook für “Kommentare (Notizen)” zu Ihrem Event-Listener hinzu (falls Sie einen verwenden)
- Lassen Sie den Listener die Pipeline-Trigger-API mit Variablen wie `AI_FLOW_INPUT` und `AI_FLOW_CONTEXT` aufrufen, wenn ein Kommentar `@claude` enthält

Beispiel-Anwendungsfälle**Verwandeln Sie Probleme in MRs**

In einem Problemkommentar:

```
@claude implement this feature based on the issue description
```

Claude analysiert das Problem und die Codebasis, schreibt Änderungen in einem Branch und öffnet einen MR zur Überprüfung.

Erhalten Sie Implementierungshilfe

In einer MR-Diskussion:

```
@claude suggest a concrete approach to cache the results of this API call
```

Claude schlägt Änderungen vor, fügt Code mit angemessenem Caching hinzu und aktualisiert den MR.

Beheben Sie Bugs schnell

In einem Problem- oder MR-Kommentar:

```
@claude fix the TypeError in the user dashboard component
```

Claude lokalisiert den Bug, implementiert eine Korrektur und aktualisiert den Branch oder öffnet einen neuen MR.

Verwendung mit AWS Bedrock & Google Vertex AI

Für Enterprise-Umgebungen können Sie Claude Code vollständig auf Ihrer Cloud-Infrastruktur mit der gleichen Entwicklererfahrung ausführen.

AWS Bedrock

Voraussetzungen

Bevor Sie Claude Code mit AWS Bedrock einrichten, benötigen Sie:

1. Ein AWS-Konto mit Amazon Bedrock-Zugriff auf die gewünschten Claude-Modelle
2. GitLab als OIDC-Identitätsanbieter in AWS IAM konfiguriert
3. Eine IAM-Rolle mit Bedrock-Berechtigungen und einer Vertrauensrichtlinie, die auf Ihr GitLab-Projekt/Refs beschränkt ist
4. GitLab CI/CD-Variablen für Rollenübernahme:
 - `AWS_ROLE_TO_ASSUME` (Rollen-ARN)
 - `AWS_REGION` (Bedrock-Region)

Setup-Anweisungen

Konfigurieren Sie AWS, um GitLab CI-Jobs zu ermöglichen, eine IAM-Rolle über OIDC anzunehmen (keine statischen Schlüssel).

Erforderliches Setup:

1. Aktivieren Sie Amazon Bedrock und fordern Sie Zugriff auf Ihre Ziel-Claude-Modelle an
2. Erstellen Sie einen IAM OIDC-Provider für GitLab, falls nicht bereits vorhanden
3. Erstellen Sie eine IAM-Rolle, der der GitLab OIDC-Provider vertraut, beschränkt auf Ihr Projekt und geschützte Refs
4. Fügen Sie Least-Privilege-Berechtigungen für Bedrock-Invoke-APIs an

Erforderliche Werte zum Speichern in CI/CD-Variablen:

- `AWS_ROLE_TO_ASSUME`
- `AWS_REGION`

Fügen Sie Variablen in Einstellungen → CI/CD → Variablen hinzu:

```
## Für AWS Bedrock:
- AWS_ROLE_TO_ASSUME
- AWS_REGION
```

Verwenden Sie das AWS Bedrock-Job-Beispiel oben, um das GitLab-Job-Token gegen temporäre AWS-Anmeldedaten zur Laufzeit auszutauschen.

Google Vertex AI

Voraussetzungen

Bevor Sie Claude Code mit Google Vertex AI einrichten, benötigen Sie:

1. Ein Google Cloud-Projekt mit:
 - Aktivierter Vertex AI API
 - Workload Identity Federation konfiguriert, um GitLab OIDC zu vertrauen
1. Ein dediziertes Service-Konto mit nur den erforderlichen Vertex AI-Rollen
2. GitLab CI/CD-Variablen für WIF:
 - `GCP_WORKLOAD_IDENTITY_PROVIDER` (vollständiger Ressourcenname)
 - `GCP_SERVICE_ACCOUNT` (Service-Konto-E-Mail)

Setup-Anweisungen

Konfigurieren Sie Google Cloud, um GitLab CI-Jobs zu ermöglichen, ein Service-Konto über Workload Identity Federation zu imitieren.

Erforderliches Setup:

1. Aktivieren Sie IAM Credentials API, STS API und Vertex AI API
2. Erstellen Sie einen Workload Identity Pool und Provider für GitLab OIDC
3. Erstellen Sie ein dediziertes Service-Konto mit Vertex AI-Rollen
4. Gewähren Sie dem WIF-Principal die Berechtigung, das Service-Konto zu imitieren

Erforderliche Werte zum Speichern in CI/CD-Variablen:

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

Fügen Sie Variablen in Einstellungen → CI/CD → Variablen hinzu:

```
## Für Google Vertex AI:  
- GCP_WORKLOAD_IDENTITY_PROVIDER  
- GCP_SERVICE_ACCOUNT  
- CLOUD_ML_REGION (zum Beispiel us-east5)
```

Verwenden Sie das Google Vertex AI-Job-Beispiel oben, um sich ohne Speicherung von Schlüsseln zu authentifizieren.

Konfigurationsbeispiele

Nachfolgend finden Sie einsatzbereite Snippets, die Sie an Ihre Pipeline anpassen können.

Basis .gitlab-ci.yml (Claude API)

```

stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
  variables:
    GIT_STRATEGY: fetch
  before_script:
    - apk update
    - apk add --no-cache git curl bash
    - curl -fsSL https://claude.ai/install.sh | bash
  script:
    - /bin/gitlab-mcp-server || true
    - >

    claude
    -p "${AI_FLOW_INPUT:-'Summarize recent changes and suggest improvements'}"
    --permission-mode acceptEdits
    --allowedTools "Bash Read Edit Write mcp__gitlab"
    --debug

# Claude Code wird ANTHROPIC_API_KEY aus CI/CD-Variablen verwenden

```

AWS Bedrock-Job-Beispiel (OIDC)

Voraussetzungen:

- Amazon Bedrock aktiviert mit Zugriff auf Ihr gewähltes Claude-Modell(e)
- GitLab OIDC in AWS konfiguriert mit einer Rolle, die Ihr GitLab-Projekt und Refs vertraut
- IAM-Rolle mit Bedrock-Berechtigungen (Least Privilege empfohlen)

Erforderliche CI/CD-Variablen:

- **AWS_ROLE_TO_ASSUME** : ARN der IAM-Rolle für Bedrock-Zugriff
- **AWS_REGION** : Bedrock-Region (zum Beispiel **us-west-2**)

```

claude-bedrock:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apk add --no-cache bash curl jq git python3 py3-pip
    - pip install --no-cache-dir awscli
    - curl -fsSL https://claude.ai/install.sh | bash
    # Tauschen Sie GitLab OIDC-Token gegen AWS-Anmeldedaten aus
    - export AWS_WEB_IDENTITY_TOKEN_FILE="${CI_JOB_JWT_FILE:-/tmp/oidc_token}"
    - if [ -n "${CI_JOB_JWT_V2}" ]; then printf "%s" "${CI_JOB_JWT_V2}" >
"$AWS_WEB_IDENTITY_TOKEN_FILE"; fi
    - >
      aws sts assume-role-with-web-identity
      --role-arn "${AWS_ROLE_TO_ASSUME}"
      --role-session-name "gitlab-claude-$(date +%s)"
      --web-identity-token "file://$AWS_WEB_IDENTITY_TOKEN_FILE"
      --duration-seconds 3600 > /tmp/aws_creds.json
    - export AWS_ACCESS_KEY_ID="$(jq -r .Credentials.AccessKeyId /tmp/
aws_creds.json)"
    - export AWS_SECRET_ACCESS_KEY="$(jq -r .Credentials.SecretAccessKey /tmp/
aws_creds.json)"
    - export AWS_SESSION_TOKEN="$(jq -r .Credentials.SessionToken /tmp/
aws_creds.json)"
  script:
    - /bin/gitlab-mcp-server || true
    - >
      claude
      -p "${AI_FLOW_INPUT:-'Implement the requested changes and open an MR'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
  variables:
    AWS_REGION: "us-west-2"

```

Note:

Modell-IDs für Bedrock enthalten regionsspezifische Präfixe (zum Beispiel `us.anthropic.claude-sonnet-4-6`). Übergeben Sie das gewünschte Modell über Ihre Job-Konfiguration oder den Prompt, wenn Ihr Workflow dies unterstützt.

Google Vertex AI-Job-Beispiel (Workload Identity Federation)

Voraussetzungen:

- Vertex AI API in Ihrem GCP-Projekt aktiviert
- Workload Identity Federation konfiguriert, um GitLab OIDC zu vertrauen
- Ein Service-Konto mit Vertex AI-Berechtigungen

Erforderliche CI/CD-Variablen:

- `GCP_WORKLOAD_IDENTITY_PROVIDER` : Vollständiger Provider-Ressourcenname
- `GCP_SERVICE_ACCOUNT` : Service-Konto-E-Mail
- `CLOUD_ML_REGION` : Vertex-Region (zum Beispiel `us-east5`)


```

claude-vertex:
  stage: ai
  image: gcr.io/google.com/cloudsdktool/google-cloud-cli:slim
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apt-get update && apt-get install -y git && apt-get clean
    - curl -fsSL https://claude.ai/install.sh | bash
    # Authentifizieren Sie sich bei Google Cloud über WIF (keine heruntergeladenen
    Schlüssel)
    - >
      gcloud auth login --cred-file=<(cat <<EOF
      {
        "type": "external_account",
        "audience": "${GCP_WORKLOAD_IDENTITY_PROVIDER}",
        "subject_token_type": "urn:ietf:params:oauth:token-type:jwt",
        "service_account_impersonation_url": "https://
iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/${
GCP_SERVICE_ACCOUNT}:generateAccessToken",
        "token_url": "https://sts.googleapis.com/v1/token"
      }
      EOF
    )
    - gcloud config set project "$(gcloud projects list --
format='value(projectId)' --filter="name:${CI_PROJECT_NAMESPACE}" | head -n1)" ||
true
  script:
    - /bin/gitlab-mcp-server || true
    - >
      CLOUD_ML_REGION="${CLOUD_ML_REGION:-us-east5}"
      claude
      -p "${AI_FLOW_INPUT:-'Review and update code as requested'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
  variables:
    CLOUD_ML_REGION: "us-east5"

```

Note:

Mit Workload Identity Federation müssen Sie keine Service-Konto-Schlüssel speichern. Verwenden Sie Repository-spezifische Vertrauensbedingungen und Least-Privilege-Service-Konten.

Best Practices

CLAUDE.md-Konfiguration

Erstellen Sie eine `CLAUDE.md`-Datei im Repository-Root, um Coding-Standards, Review-Kriterien und projektspezifische Regeln zu definieren. Claude liest diese Datei während der Ausführung und folgt Ihren Konventionen bei der Vorschlag von Änderungen.

Sicherheitsüberlegungen

Commiten Sie niemals API-Schlüssel oder Cloud-Anmeldedaten in Ihr Repository. Verwenden Sie immer GitLab CI/CD-Variablen:

- Fügen Sie `ANTHROPIC_API_KEY` als maskierte Variable hinzu (und schützen Sie sie bei Bedarf)
- Verwenden Sie Provider-spezifisches OIDC, wo möglich (keine langlebigen Schlüssel)
- Begrenzen Sie Job-Berechtigungen und Netzwerk-Egress
- Überprüfen Sie Claudes MRs wie jeden anderen Beitrag

Optimierung der Leistung

- Halten Sie `CLAUDE.md` fokussiert und prägnant
- Geben Sie klare Problem-/MR-Beschreibungen an, um Iterationen zu reduzieren
- Konfigurieren Sie angemessene Job-Timeouts, um unkontrollierte Ausführungen zu vermeiden
- Cachen Sie npm und Paketinstallationen in Runnern, wo möglich

CI-Kosten

Bei der Verwendung von Claude Code mit GitLab CI/CD sollten Sie sich der damit verbundenen Kosten bewusst sein:

- **GitLab Runner-Zeit:**
 - Claude läuft auf Ihren GitLab-Runnern und verbraucht Compute-Minuten
 - Siehe die Runner-Abrechnung Ihres GitLab-Plans für Details

- **API-Kosten:**

- Jede Claude-Interaktion verbraucht Token basierend auf Prompt- und Antwortgröße
- Die Token-Nutzung variiert je nach Aufgabenkomplexität und Codebasis-Größe
- Siehe [Anthropic-Preisgestaltung](#) für Details

- **Tipps zur Kostenoptimierung:**

- Verwenden Sie spezifische `@claude` -Befehle, um unnötige Durchläufe zu reduzieren
- Legen Sie angemessene `max_turns` - und Job-Timeout-Werte fest
- Begrenzen Sie die Parallelität, um parallele Ausführungen zu kontrollieren

Sicherheit und Governance

- Jeder Job läuft in einem isolierten Container mit eingeschränktem Netzwerkzugriff
- Claudes Änderungen fließen durch MRs, damit Reviewer jeden Diff sehen
- Branch-Schutz und Genehmigungsregeln gelten für KI-generierte Code
- Claude Code verwendet Workspace-bezogene Berechtigungen, um Schreibvorgänge einzuschränken
- Kosten bleiben unter Ihrer Kontrolle, da Sie Ihre eigenen Provider-Anmeldedaten mitbringen

Fehlerbehebung

Claude antwortet nicht auf `@claude`-Befehle

- Überprüfen Sie, ob Ihre Pipeline ausgelöst wird (manuell, MR-Ereignis oder über einen Note-Event-Listener/Webhook)
- Stellen Sie sicher, dass CI/CD-Variablen (`ANTHROPIC_API_KEY` oder Cloud-Provider-Einstellungen) vorhanden und unmaskiert sind
- Überprüfen Sie, dass der Kommentar `@claude` enthält (nicht `/claude`) und dass Ihr Mention-Trigger konfiguriert ist

Job kann keine Kommentare schreiben oder MRs öffnen

- Stellen Sie sicher, dass `CI_JOB_TOKEN` ausreichende Berechtigungen für das Projekt hat, oder verwenden Sie ein Project Access Token mit `api` -Bereich
- Überprüfen Sie, dass das `mcp_gitlab` -Tool in `--allowedTools` aktiviert ist

- Bestätigen Sie, dass der Job im Kontext des MR ausgeführt wird oder über `AI_FLOW_*`-Variablen genug Kontext hat

Authentifizierungsfehler

- **Für Claude API:** Bestätigen Sie, dass `ANTHROPIC_API_KEY` gültig und nicht abgelaufen ist
- **Für Bedrock/Vertex:** Überprüfen Sie OIDC/WIF-Konfiguration, Rollenimitierung und Geheimnisnamen; bestätigen Sie Region und Modellverfügbarkeit

Erweiterte Konfiguration

Häufig verwendete Parameter und Variablen

Claude Code unterstützt diese häufig verwendeten Eingaben:

- `prompt` / `prompt_file`: Geben Sie Anweisungen inline (`-p`) oder über eine Datei an
- `max_turns`: Begrenzen Sie die Anzahl der Hin- und Herbewegungen
- `timeout_minutes`: Begrenzen Sie die Gesamtausführungszeit
- `ANTHROPIC_API_KEY`: Erforderlich für die Claude API (nicht für Bedrock/Vertex verwendet)
- Provider-spezifische Umgebung: `AWS_REGION`, Projekt-/Regionsvariablen für Vertex

Note:

Genaue Flags und Parameter können je nach Version von `@anthropic-ai/claude-code` variieren. Führen Sie `claude --help` in Ihrem Job aus, um unterstützte Optionen zu sehen.

Anpassung von Claudes Verhalten

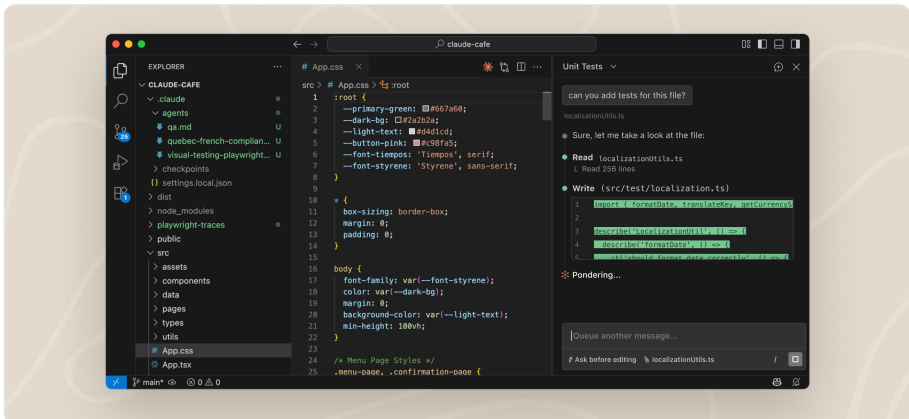
Sie können Claude auf zwei primäre Arten lenken:

1. **CLAUDE.md:** Definieren Sie Coding-Standards, Sicherheitsanforderungen und Projektkonventionen. Claude liest dies während der Ausführung und folgt Ihren Regeln.
2. **Benutzerdefinierte Prompts:** Übergeben Sie aufgabenspezifische Anweisungen über `prompt` / `prompt_file` im Job. Verwenden Sie unterschiedliche Prompts für verschiedene Jobs (zum Beispiel Review, Implementierung, Refactoring).

Part 8: IDE & Platform Integration

Claude Code in VS Code verwenden

Installieren und konfigurieren Sie die Claude Code-Erweiterung für VS Code. Erhalten Sie KI-Codierungshilfe mit Inline-Diffs, @-Erwähnungen, Planüberprüfung und Tastaturkürzeln.



VS Code-Editor mit dem geöffneten Claude Code-Erweiterungspanel auf der rechten Seite, das ein Gespräch mit Claude zeigt

Die VS Code-Erweiterung bietet eine native grafische Benutzeroberfläche für Claude Code, die direkt in Ihre IDE integriert ist. Dies ist die empfohlene Methode zur Verwendung von Claude Code in VS Code.

Mit der Erweiterung können Sie Claudes Pläne überprüfen und bearbeiten, bevor Sie sie akzeptieren, Bearbeitungen automatisch akzeptieren, während sie vorgenommen werden, @-Erwähnungen von Dateien mit bestimmten Zeilenbereichen aus Ihrer Auswahl hinzufügen, auf Gesprächsverlauf zugreifen und mehrere Gespräche in separaten Registerkarten oder Fenstern öffnen.

Voraussetzungen

Stellen Sie vor der Installation sicher, dass Sie Folgendes haben:

- VS Code 1.98.0 oder höher

- Ein Anthropic-Konto (Sie melden sich an, wenn Sie die Erweiterung zum ersten Mal öffnen). Wenn Sie einen Drittanbieter wie Amazon Bedrock oder Google Vertex AI verwenden, siehe stattdessen [Drittanbieter verwenden](#).

Tip:

Die Erweiterung enthält die CLI (Befehlszeilenschnittstelle), auf die Sie über das integrierte Terminal von VS Code für erweiterte Funktionen zugreifen können. Siehe [VS Code-Erweiterung vs. Claude Code CLI](#) für Details.

Installieren Sie die Erweiterung

Klicken Sie auf den Link für Ihre IDE, um direkt zu installieren:

- [Für VS Code installieren](#)
- [Für Cursor installieren](#)

Oder drücken Sie in VS Code **Cmd+Shift+X** (Mac) oder **Ctrl+Shift+X** (Windows/Linux), um die Ansicht „Erweiterungen“ zu öffnen, suchen Sie nach „Claude Code“ und klicken Sie auf **Installieren**.

Note:

Wenn die Erweiterung nach der Installation nicht angezeigt wird, starten Sie VS Code neu oder führen Sie „Developer: Reload Window“ aus der Befehlspalette aus.

Erste Schritte

Nach der Installation können Sie Claude Code über die VS Code-Benutzeroberfläche verwenden:

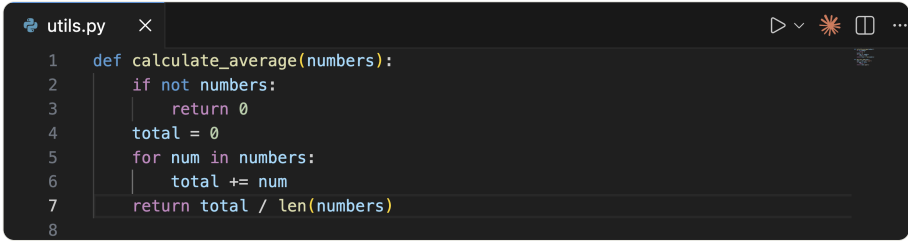
Step 1: Öffnen Sie das Claude Code-Panel

In VS Code zeigt das Spark-Symbol Claude Code an:



Spark-Symbol

Die schnellste Möglichkeit, Claude zu öffnen, besteht darin, auf das Spark-Symbol in der **Editor-Symbolleiste** (obere rechte Ecke des Editors) zu klicken. Das Symbol wird nur angezeigt, wenn Sie eine Datei geöffnet haben.



VS Code-Editor mit dem Spark-Symbol in der Editor-Symbolleiste

Weitere Möglichkeiten zum Öffnen von Claude Code:

- **Aktivitätsleiste:** Klicken Sie auf das Spark-Symbol in der linken Seitenleiste, um die Sitzungsliste zu öffnen. Klicken Sie auf eine beliebige Sitzung, um sie als vollständige Editor-Registerkarte zu öffnen, oder starten Sie eine neue. Dieses Symbol ist immer in der Aktivitätsleiste sichtbar.
- **Befehlspalette:** `Cmd+Shift+P` (Mac) oder `Ctrl+Shift+P` (Windows/Linux), geben Sie „Claude Code“ ein und wählen Sie eine Option wie „In neuer Registerkarte öffnen“
- **Statusleiste:** Klicken Sie auf **Claude Code** in der unteren rechten Ecke des Fensters. Dies funktioniert auch, wenn keine Datei geöffnet ist.

Wenn Sie das Panel zum ersten Mal öffnen, wird eine **Learn Claude Code**-Checkliste angezeigt. Arbeiten Sie jedes Element durch, indem Sie auf **Show me** klicken, oder schließen Sie es mit dem X. Um es später erneut zu öffnen, deaktivieren Sie **Hide Onboarding** in den VS Code-Einstellungen unter Extensions → Claude Code.

Sie können das Claude-Panel ziehen, um es überall in VS Code zu repositionieren. Siehe [Passen Sie Ihren Workflow an](#) für Details.

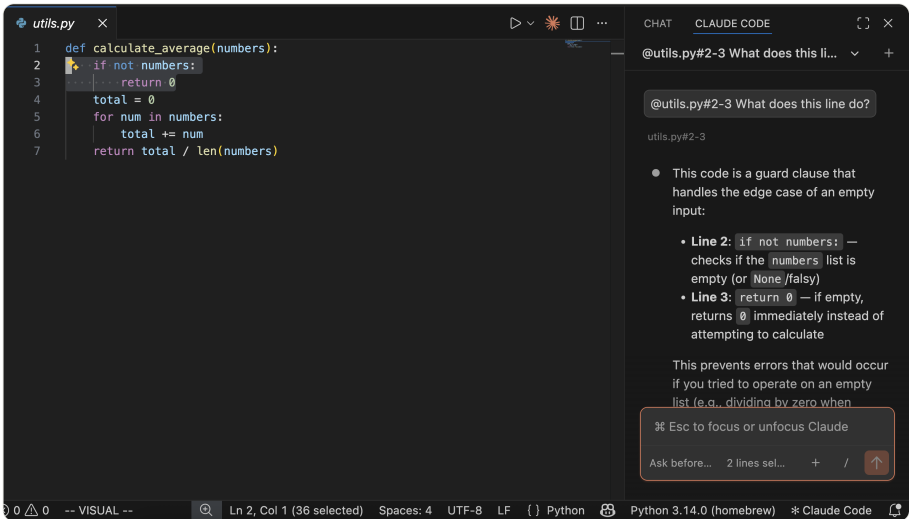
Step 2: Senden Sie eine Eingabeaufforderung

Bitte Sie Claude, Ihnen bei Ihrem Code oder Ihren Dateien zu helfen, sei es zum Erklären, wie etwas funktioniert, zum Debuggen eines Problems oder zum Vornehmen von Änderungen.

Tip:

Claude sieht Ihren ausgewählten Text automatisch. Drücken Sie `Option+K` (Mac) / `Alt+K` (Windows/Linux), um auch eine @-Erwähnungsreferenz (wie `@file.ts#5-10`) in Ihre Eingabeaufforderung einzufügen.

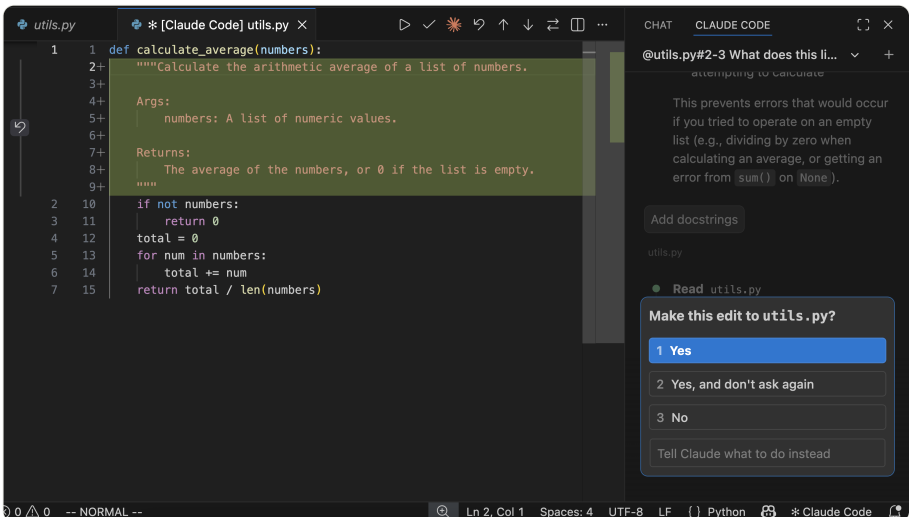
Hier ist ein Beispiel für eine Frage zu einer bestimmten Zeile in einer Datei:



VS Code-Editor mit den Zeilen 2-3, die in einer Python-Datei ausgewählt sind, und das Claude Code-Panel zeigt eine Frage zu diesen Zeilen mit einer @-Erwähnungsreferenz

Step 3: Überprüfen Sie Änderungen

Wenn Claude eine Datei bearbeiten möchte, zeigt es einen Vergleich nebeneinander zwischen dem Original und den vorgeschlagenen Änderungen an und fordert dann die Genehmigung an. Sie können akzeptieren, ablehnen oder Claude sagen, was stattdessen zu tun ist.



VS Code zeigt einen Diff von Claudes vorgeschlagenen Änderungen mit einer Genehmigungsaufforderung, die fragt, ob die Bearbeitung vorgenommen werden soll

Weitere Ideen, was Sie mit Claude Code tun können, finden Sie unter [Häufige Workflows](#).

Tip:

Führen Sie „Claude Code: Open Walkthrough“ aus der Befehlspalette aus, um eine geführte Tour durch die Grundlagen zu erhalten.

Verwenden Sie das Eingabefeld

Das Eingabefeld unterstützt mehrere Funktionen:

- **Genehmigungsmodi:** Klicken Sie auf den Modusindikator am unteren Rand des Eingabefelds, um Modi zu wechseln. Im normalen Modus fragt Claude vor jeder Aktion um Genehmigung. Im Plan Mode beschreibt Claude, was es tun wird, und wartet auf Genehmigung, bevor es Änderungen vornimmt. VS Code öffnet den Plan automatisch als vollständiges Markdown-Dokument, in dem Sie Inline-Kommentare hinzufügen können, um Feedback zu geben, bevor Claude beginnt. Im Auto-Accept-Modus nimmt Claude Bearbeitungen vor, ohne zu fragen. Legen Sie den Standard in den VS Code-Einstellungen unter `claudeCode.initialPermissionMode` fest.
- **Befehlsmenü:** Klicken Sie auf `/` oder geben Sie `/` ein, um das Befehlsmenü zu öffnen. Zu den Optionen gehören das Anhängen von Dateien, das Wechseln von Modellen, das Umschalten von Extended Thinking und das Anzeigen der Plannutzung (`/usage`). Der Abschnitt „Customize“ bietet Zugriff auf MCP servers, hooks, memory, permissions und plugins. Elemente mit einem Terminal-Symbol werden im integrierten Terminal geöffnet.
- **Kontextindikator:** Das Eingabefeld zeigt, wie viel von Claudes context window Sie verwenden. Claude komprimiert automatisch bei Bedarf, oder Sie können `/compact` manuell ausführen.
- **Extended Thinking:** Ermöglicht Claude, mehr Zeit für die Überlegung komplexer Probleme aufzuwenden. Aktivieren Sie es über das Befehlsmenü (`/`). Siehe [Extended Thinking](#) für Details.
- **Mehrzeilige Eingabe:** Drücken Sie `Shift+Enter`, um eine neue Zeile hinzuzufügen, ohne zu senden. Dies funktioniert auch in der „Other“-Freitexteingabe von Frage-Dialogen.

Referenzdateien und -ordner

Verwenden Sie `@`-Erwähnungen, um Claude Kontext über bestimmte Dateien oder Ordner zu geben. Wenn Sie `@` gefolgt von einem Datei- oder Ordnernamen eingeben, liest Claude diesen Inhalt und kann Fragen dazu beantworten oder Änderungen daran vornehmen. Claude Code unterstützt Fuzzy Matching, sodass Sie Teilnamen eingeben können, um das zu finden, was Sie benötigen:

```
> Explain the logic in @auth (fuzzy matches auth.js, AuthService.ts, etc.)
> What's in @src/components/ (include a trailing slash for folders)
```

Für große PDFs können Sie Claude bitten, bestimmte Seiten statt der gesamten Datei zu lesen: eine einzelne Seite, einen Bereich wie Seiten 1-10 oder einen offenen Bereich wie Seite 3 und darüber hinaus.

Wenn Sie Text im Editor auswählen, kann Claude Ihren hervorgehobenen Code automatisch sehen. Die Fußzeile des Eingabefelds zeigt, wie viele Zeilen ausgewählt sind. Drücken Sie **Option+K** (Mac) / **Alt+K** (Windows/Linux), um eine `@`-Erwähnung mit dem Dateipfad und den Zeilennummern einzufügen (z. B. `@app.ts#5-10`). Klicken Sie auf den Auswahlindikator, um umzuschalten, ob Claude Ihren hervorgehobenen Text sehen kann – das Symbol mit durchgestrichenem Auge bedeutet, dass die Auswahl vor Claude verborgen ist.

Sie können auch **Shift** gedrückt halten, während Sie Dateien in das Eingabefeld ziehen, um sie als Anhänge hinzuzufügen. Klicken Sie auf das X bei einem Anhang, um ihn aus dem Kontext zu entfernen.

Frühere Gespräche fortsetzen

Klicken Sie auf das Dropdown-Menü oben im Claude Code-Panel, um auf Ihren Gesprächsverlauf zuzugreifen. Sie können nach Schlüsselwort suchen oder nach Zeit durchsuchen (Heute, Gestern, Letzte 7 Tage usw.). Klicken Sie auf ein beliebiges Gespräch, um es mit dem vollständigen Nachrichtenverlauf fortzusetzen. Bewegen Sie den Mauszeiger über eine Sitzung, um Umbenennungs- und Entfernungsaktionen anzuzeigen: Benennen Sie um, um ihr einen beschreibenden Titel zu geben, oder entfernen Sie sie, um sie aus der Liste zu löschen. Weitere Informationen zum Fortsetzen von Sitzungen finden Sie unter [Häufige Workflows](#).

Fortsetzen von Remote-Sitzungen von Claude.ai

Wenn Sie [Claude Code im Web](#) verwenden, können Sie diese Remote-Sitzungen direkt in VS Code fortsetzen. Dies erfordert die Anmeldung mit **Claude.ai Subscription**, nicht Anthropic Console.

Step 1: Öffnen Sie Frühere Gespräche

Klicken Sie auf das Dropdown-Menü **Past Conversations** oben im Claude Code-Panel.

Step 2: Wählen Sie die Remote-Registerkarte

Der Dialog zeigt zwei Registerkarten: Local und Remote. Klicken Sie auf **Remote**, um Sitzungen von claude.ai anzuzeigen.

Step 3: Wählen Sie eine Sitzung zum Fortsetzen

Durchsuchen oder suchen Sie Ihre Remote-Sitzungen. Klicken Sie auf eine beliebige Sitzung, um sie herunterzuladen und das Gespräch lokal fortzusetzen.

Note:

Nur Web-Sitzungen, die mit einem GitHub-Repository gestartet wurden, werden auf der Remote-Registerkarte angezeigt. Das Fortsetzen lädt den Gesprächsverlauf lokal; Änderungen werden nicht mit claude.ai synchronisiert.

Passen Sie Ihren Workflow an

Sobald Sie einsatzbereit sind, können Sie das Claude-Panel repositionieren, mehrere Sitzungen ausführen oder zum Terminal-Modus wechseln.

Wählen Sie, wo Claude lebt

Sie können das Claude-Panel ziehen, um es überall in VS Code zu repositionieren. Greifen Sie die Registerkarte oder Titelleiste des Panels und ziehen Sie es zu:

- **Sekundäre Seitenleiste:** die rechte Seite des Fensters. Hält Claude sichtbar, während Sie codieren.
- **Primäre Seitenleiste:** die linke Seitenleiste mit Symbolen für Explorer, Search usw.
- **Editor-Bereich:** öffnet Claude als Registerkarte neben Ihren Dateien. Nützlich für Nebenaufgaben.

Tip:

Verwenden Sie die Seitenleiste für Ihre Haupt-Claude-Sitzung und öffnen Sie zusätzliche Registerkarten für Nebenaufgaben. Claude merkt sich Ihren bevorzugten Ort. Das Symbol der Sitzungsliste in der Aktivitätsleiste ist separat vom Claude-Panel: Die Sitzungsliste ist immer in der Aktivitätsleiste sichtbar, während das Claude-Panel-Symbol nur dort angezeigt wird, wenn das Panel an der linken Seitenleiste angedockt ist.

Führen Sie mehrere Gespräche aus

Verwenden Sie **Open in New Tab** oder **Open in New Window** aus der Befehlspalette, um zusätzliche Gespräche zu starten. Jedes Gespräch behält seinen eigenen Verlauf und Kontext bei, sodass Sie parallel an verschiedenen Aufgaben arbeiten können.

Bei Verwendung von Registerkarten zeigt ein kleiner farbiger Punkt auf dem Spark-Symbol den Status an: Blau bedeutet, dass eine Genehmigungsanfrage ausstehend ist, Orange bedeutet, dass Claude fertig ist, während die Registerkarte verborgen war.

Wechseln Sie zum Terminal-Modus

Standardmäßig öffnet die Erweiterung ein grafisches Chat-Panel. Wenn Sie die CLI-ähnliche Benutzeroberfläche bevorzugen, öffnen Sie die [Use Terminal-Einstellung](#) und aktivieren Sie das Kontrollkästchen.

Sie können auch VS Code-Einstellungen öffnen (**Cmd+**, auf Mac oder **Ctrl+**, auf Windows/Linux), zu Extensions → Claude Code gehen und **Use Terminal** aktivieren.

Verwalten Sie Plugins

Die VS Code-Erweiterung enthält eine grafische Benutzeroberfläche zum Installieren und Verwalten von [plugins](#). Geben Sie **/plugins** in das Eingabefeld ein, um die Benutzeroberfläche **Manage plugins** zu öffnen.

Installieren Sie Plugins

Der Plugin-Dialog zeigt zwei Registerkarten: **Plugins** und **Marketplaces**.

Auf der Registerkarte 'Plugins':

- **Installed plugins** werden oben mit Umschaltern angezeigt, um sie zu aktivieren oder zu deaktivieren
- **Available plugins** aus Ihren konfigurierten Marketplaces werden unten angezeigt
- Suchen Sie, um Plugins nach Name oder Beschreibung zu filtern
- Klicken Sie auf **Install** bei einem beliebigen verfügbaren Plugin

Wenn Sie ein Plugin installieren, wählen Sie den Installationsumfang:

- **Install for you:** verfügbar in allen Ihren Projekten (Benutzerumfang)
- **Install for this project:** geteilt mit Projektmitarbeitern (Projektumfang)
- **Install locally:** nur für Sie, nur in diesem Repository (lokaler Umfang)

Verwalten Sie Marketplaces

Wechseln Sie zur Registerkarte **Marketplaces**, um Plugin-Quellen hinzuzufügen oder zu entfernen:

- Geben Sie ein GitHub-Repo, eine URL oder einen lokalen Pfad ein, um einen neuen Marketplace hinzuzufügen
- Klicken Sie auf das Aktualisierungssymbol, um die Plugin-Liste eines Marketplace zu aktualisieren
- Klicken Sie auf das Papierkorbsymbol, um einen Marketplace zu entfernen

Nach dem Vornehmen von Änderungen fordert ein Banner Sie auf, Claude Code neu zu starten, um die Aktualisierungen anzuwenden.

Note:

Die Plugin-Verwaltung in VS Code verwendet unter der Haube die gleichen CLI-Befehle. Plugins und Marketplaces, die Sie in der Erweiterung konfigurieren, sind auch in der CLI verfügbar, und umgekehrt.

Weitere Informationen zum Plugin-System finden Sie unter [Plugins](#) und [Plugin marketplaces](#).

Automatisieren Sie Browser-Aufgaben mit Chrome

Verbinden Sie Claude mit Ihrem Chrome-Browser, um Web-Apps zu testen, mit Konsolenprotokollen zu debuggen und Browser-Workflows zu automatisieren, ohne VS Code zu verlassen. Dies erfordert die [Claude in Chrome-Erweiterung](#) Version 1.0.36 oder höher.

Geben Sie `@browser` in das Eingabefeld ein, gefolgt von dem, was Claude tun soll:

```
@browser go to localhost:3000 and check the console for errors
```

Sie können auch das Anhang-Menü öffnen, um bestimmte Browser-Tools wie das Öffnen einer neuen Registerkarte oder das Lesen von Seiteninhalten auszuwählen.

Claude öffnet neue Registerkarten für Browser-Aufgaben und teilt den Anmeldestatus Ihres Browsers, sodass es auf jede Website zugreifen kann, bei der Sie bereits angemeldet sind.

Anweisungen zum Einrichten, die vollständige Liste der Funktionen und Fehlerbehebung finden Sie unter [Claude Code mit Chrome verwenden](#).

VS Code-Befehle und Tastaturkürzel

Öffnen Sie die Befehlspalette (`Cmd+Shift+P` auf Mac oder `Ctrl+Shift+P` auf Windows/Linux) und geben Sie „Claude Code“ ein, um alle verfügbaren VS Code-Befehle für die Claude Code-Erweiterung anzuzeigen.

Einige Tastaturkürzel hängen davon ab, welches Panel „fokussiert“ ist (Tastatureingabe empfängt). Wenn sich Ihr Cursor in einer Codedatei befindet, ist der Editor fokussiert. Wenn sich Ihr Cursor im Eingabefeld von Claude befindet, ist Claude fokussiert. Verwenden Sie `Cmd+Esc` / `Ctrl+Esc` , um zwischen ihnen zu wechseln.

Note:

Dies sind VS Code-Befehle zum Steuern der Erweiterung. Nicht alle integrierten Claude Code-Befehle sind in der Erweiterung verfügbar. Siehe [VS Code-Erweiterung vs. Claude Code CLI](#) für Details.

Befehl	Tastaturkürzel	Beschreibung
Focus Input	<code>Cmd+Esc</code> (Mac) / <code>Ctrl+Esc</code> (Windows/Linux)	Fokus zwischen Editor und Claude umschalten
Open in Side Bar	-	Öffnen Sie Claude in der linken Seitenleiste
Open in Terminal	-	Öffnen Sie Claude im Terminal-Modus
Open in New Tab	<code>Cmd+Shift+Esc</code> (Mac) / <code>Ctrl+Shift+Esc</code> (Windows/Linux)	Öffnen Sie ein neues Gespräch als Editor-Registerkarte
Open in New Window	-	Öffnen Sie ein neues Gespräch in einem separaten Fenster

Befehl	Tastaturkürzel	Beschreibung
New Conversation	Cmd+N (Mac) / Ctrl+N (Windows/Linux)	Starten Sie ein neues Gespräch (erfordert, dass Claude fokussiert ist)
Insert @-Mention Reference	Option+K (Mac) / Alt+K (Windows/Linux)	Fügen Sie eine Referenz zur aktuellen Datei und Auswahl ein (erfordert, dass der Editor fokussiert ist)
Show Logs	-	Zeigen Sie Erweiterungs-Debug-Protokolle an
Logout	-	Melden Sie sich von Ihrem Anthropic-Konto ab

Konfigurieren Sie Einstellungen

Die Erweiterung hat zwei Arten von Einstellungen:

- **Extension settings** in VS Code: Steuern Sie das Verhalten der Erweiterung in VS Code. Öffnen Sie mit **Cmd+,** (Mac) oder **Ctrl+,** (Windows/Linux), dann gehen Sie zu Extensions → Claude Code. Sie können auch **/** eingeben und **General Config** auswählen, um Einstellungen zu öffnen.
- **Claude Code settings** in `~/.claude/settings.json` : Geteilt zwischen der Erweiterung und CLI. Verwenden Sie für zulässige Befehle, Umgebungsvariablen, hooks und MCP servers. Siehe [Settings](#) für Details.

Tip:

Fügen Sie `"$schema": "https://json.schemastore.org/claude-code-settings.json"` zu Ihrer `settings.json` hinzu, um Autovervollständigung und Inline-Validierung für alle verfügbaren Einstellungen direkt in VS Code zu erhalten.

Extension-Einstellungen

Einstellung	Standard	Beschreibung
<code>selectedModel</code>	<code>default</code>	Modell für neue Gespräche. Ändern Sie pro Sitzung mit <code>/model</code> .

Einstellung	Standard	Beschreibung
<code>useTerminal</code>	<code>false</code>	Starten Sie Claude im Terminal-Modus statt im grafischen Panel
<code>initialPermissionMode</code>	<code>default</code>	Steuert Genehmigungsaufforderungen: <code>default</code> (jedes Mal fragen), <code>plan</code> , <code>acceptEdits</code> oder <code>bypassPermissions</code>
<code>preferredLocation</code>	<code>panel</code>	Wo Claude öffnet: <code>sidebar</code> (rechts) oder <code>panel</code> (neue Registerkarte)
<code>autosave</code>	<code>true</code>	Speichern Sie Dateien automatisch, bevor Claude sie liest oder schreibt
<code>useCtrlEnterToSend</code>	<code>false</code>	Verwenden Sie Ctrl/Cmd+Enter statt Enter, um Eingabeaufforderungen zu senden
<code>enableNewConversationShortcut</code>	<code>true</code>	Aktivieren Sie Cmd/Ctrl+N, um ein neues Gespräch zu starten
<code>hideOnboarding</code>	<code>false</code>	Blenden Sie die Onboarding-Checkliste aus (Abschlusskappe-Symbol)
<code>respectGitIgnore</code>	<code>true</code>	Schließen Sie .gitignore-Muster aus Dateisuchvorgängen aus
<code>environmentVariables</code>	<code>[]</code>	Legen Sie Umgebungsvariablen für den Claude-Prozess fest. Verwenden Sie stattdessen Claude Code-Einstellungen für gemeinsame Konfiguration.

Einstellung	Standard	Beschreibung
<code>disableLoginPrompt</code>	<code>false</code>	Überspringen Sie Authentifizierungsaufforderungen (für Setups von Drittanbietern)
<code>allowDangerouslySkipPermissions</code>	<code>false</code>	Umgehen Sie alle Genehmigungsaufforderungen. Verwenden Sie mit äußerster Vorsicht.
<code>claudeProcessWrapper</code>	-	Ausführbarer Pfad, der zum Starten des Claude-Prozesses verwendet wird

VS Code-Erweiterung vs. Claude Code CLI

Claude Code ist sowohl als VS Code-Erweiterung (grafisches Panel) als auch als CLI (Befehlszeilenschnittstelle im Terminal) verfügbar. Einige Funktionen sind nur in der CLI verfügbar. Wenn Sie eine CLI-only-Funktion benötigen, führen Sie `claude` im integrierten Terminal von VS Code aus.

Funktion	CLI	VS Code-Erweiterung
Commands and skills	Alle	Teilmenge (geben Sie <code>/</code> ein, um verfügbare anzuzeigen)
MCP server config	Ja	Teilweise (fügen Sie Server über CLI hinzu; verwalten Sie vorhandene Server mit <code>/mcp</code> im Chat-Panel)
Checkpoints	Ja	Ja
<code>! bash</code> shortcut	Ja	Nein
Tab completion	Ja	Nein

Zurückspulen mit Checkpoints

Die VS Code-Erweiterung unterstützt Checkpoints, die Claudes Dateibearbeitungen verfolgen und es Ihnen ermöglichen, zu einem vorherigen Zustand zurückzuspringen. Bewegen Sie den Mauszeiger über eine beliebige Nachricht, um die Zurückspul-Schaltfläche anzuzeigen, und wählen Sie dann aus drei Optionen:

- **Fork conversation from here:** Starten Sie einen neuen Gesprächszweig von dieser Nachricht aus, während Sie alle Codeänderungen intakt halten
- **Rewind code to here:** Setzen Sie Dateiänderungen auf diesen Punkt im Gespräch zurück, während Sie den vollständigen Gesprächsverlauf behalten
- **Fork conversation and rewind code:** Starten Sie einen neuen Gesprächszweig und setzen Sie Dateiänderungen auf diesen Punkt zurück

Vollständige Details zur Funktionsweise von Checkpoints und deren Einschränkungen finden Sie unter [Checkpointing](#).

Führen Sie CLI in VS Code aus

Um die CLI zu verwenden und in VS Code zu bleiben, öffnen Sie das integrierte Terminal (`Ctrl+`` auf Windows/Linux oder `Cmd+`` auf Mac) und führen Sie `claude` aus. Die CLI wird automatisch mit Ihrer IDE für Funktionen wie Diff-Anzeige und Diagnosefreigabe integriert.

Wenn Sie ein externes Terminal verwenden, führen Sie `/ide` in Claude Code aus, um es mit VS Code zu verbinden.

Wechseln Sie zwischen Erweiterung und CLI

Die Erweiterung und CLI teilen den gleichen Gesprächsverlauf. Um ein Erweiterungsgepräch in der CLI fortzusetzen, führen Sie `claude --resume` im Terminal aus. Dies öffnet eine interaktive Auswahl, in der Sie Ihr Gespräch suchen und auswählen können.

Beziehen Sie Terminal-Ausgabe in Eingabeaufforderungen ein

Referenzieren Sie Terminal-Ausgabe in Ihren Eingabeaufforderungen mit `@terminal:name`, wobei `name` der Titel des Terminals ist. Dies ermöglicht Claude, Befehlsausgabe, Fehlermeldungen oder Protokolle zu sehen, ohne zu kopieren und einzufügen.

Überwachen Sie Hintergrundprozesse

Wenn Claude lange laufende Befehle ausführt, zeigt die Erweiterung den Fortschritt in der Statusleiste an. Die Sichtbarkeit für Hintergrundaufgaben ist jedoch im Vergleich zur CLI begrenzt. Für bessere Sichtbarkeit lassen Sie Claude den Befehl ausgeben, damit Sie ihn im integrierten Terminal von VS Code ausführen können.

Verbinden Sie sich mit externen Tools mit MCP

MCP (Model Context Protocol) servers geben Claude Zugriff auf externe Tools, Datenbanken und APIs.

Um einen MCP server hinzuzufügen, öffnen Sie das integrierte Terminal (**Ctrl+`** oder **Cmd+`**) und führen Sie aus:

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

Nach der Konfiguration bitten Sie Claude, die Tools zu verwenden (z. B. „Review PR #456“).

Um MCP servers zu verwalten, ohne VS Code zu verlassen, geben Sie **/mcp** im Chat-Panel ein. Der MCP-Verwaltungsdialog ermöglicht es Ihnen, Server zu aktivieren oder zu deaktivieren, sich erneut mit einem Server zu verbinden und OAuth-Authentifizierung zu verwalten. Siehe die [MCP-Dokumentation](#) für verfügbare servers.

Arbeiten Sie mit Git

Claude Code wird mit Git integriert, um Versionskontroll-Workflows direkt in VS Code zu unterstützen. Bitten Sie Claude, Änderungen zu committen, Pull Requests zu erstellen oder über Branches zu arbeiten.

Erstellen Sie Commits und Pull Requests

Claude kann Änderungen bereitstellen, Commit-Nachrichten schreiben und Pull Requests basierend auf Ihrer Arbeit erstellen:

```
> commit my changes with a descriptive message
> create a pr for this feature
> summarize the changes I've made to the auth module
```

Beim Erstellen von Pull Requests generiert Claude Beschreibungen basierend auf den tatsächlichen Codeänderungen und kann Kontext über Tests oder Implementierungsentscheidungen hinzufügen.

Verwenden Sie Git Worktrees für parallele Aufgaben

Verwenden Sie das Flag `--worktree` (`-w`), um Claude in einem isolierten Worktree mit seinen eigenen Dateien und Branch zu starten:

```
claude --worktree feature-auth
```

Jeder Worktree behält einen unabhängigen Dateizustand bei, während er die Git-Historie teilt. Dies verhindert, dass Claude-Instanzen sich gegenseitig beeinflussen, wenn sie an verschiedenen Aufgaben arbeiten. Weitere Details finden Sie unter [Führen Sie parallele Claude Code-Sitzungen mit Git Worktrees aus](#).

Verwenden Sie Drittanbieter

Standardmäßig verbindet sich Claude Code direkt mit der API von Anthropic. Wenn Ihre Organisation Amazon Bedrock, Google Vertex AI oder Microsoft Foundry verwendet, um auf Claude zuzugreifen, konfigurieren Sie die Erweiterung, um stattdessen Ihren Anbieter zu verwenden:

Step 1: Deaktivieren Sie die Anmeldungsaufforderung

Öffnen Sie die [Disable Login Prompt-Einstellung](#) und aktivieren Sie das Kontrollkästchen.

Sie können auch VS Code-Einstellungen öffnen (`Cmd+`, auf Mac oder `Ctrl+`, auf Windows/Linux), nach "Claude Code login" suchen und **Disable Login Prompt** aktivieren.

Step 2: Konfigurieren Sie Ihren Anbieter

Folgen Sie dem Setup-Leitfaden für Ihren Anbieter:

- [Claude Code auf Amazon Bedrock](#)
- [Claude Code auf Google Vertex AI](#)
- [Claude Code auf Microsoft Foundry](#)

Diese Leitfäden behandeln die Konfiguration Ihres Anbieters in `~/.claude/settings.json`, was sicherstellt, dass Ihre Einstellungen zwischen der VS Code-Erweiterung und der CLI geteilt werden.

Sicherheit und Datenschutz

Ihr Code bleibt privat. Claude Code verarbeitet Ihren Code, um Unterstützung zu bieten, verwendet ihn aber nicht zum Trainieren von Modellen. Weitere Informationen zur Datenbehandlung und zum Deaktivieren der Protokollierung finden Sie unter [Data and privacy](#).

Mit aktivierten Auto-Edit-Berechtigungen kann Claude Code VS Code-Konfigurationsdateien (wie `settings.json` oder `tasks.json`) ändern, die VS Code möglicherweise automatisch ausführt. Um das Risiko bei der Arbeit mit nicht vertrauenswürdigen Code zu verringern:

- Aktivieren Sie [VS Code Restricted Mode](#) für nicht vertrauenswürdige Workspaces
- Verwenden Sie den manuellen Genehmigungsmodus statt Auto-Accept für Bearbeitungen
- Überprüfen Sie Änderungen sorgfältig, bevor Sie sie akzeptieren

Beheben Sie häufige Probleme

Erweiterung wird nicht installiert

- Stellen Sie sicher, dass Sie eine kompatible Version von VS Code haben (1.98.0 oder später)
- Überprüfen Sie, dass VS Code die Berechtigung zum Installieren von Erweiterungen hat
- Versuchen Sie, direkt vom [VS Code Marketplace](#) zu installieren

Spark-Symbol nicht sichtbar

Das Spark-Symbol wird in der **Editor-Symboleiste** (oben rechts des Editors) angezeigt, wenn Sie eine Datei geöffnet haben. Wenn Sie es nicht sehen:

1. **Öffnen Sie eine Datei:** Das Symbol erfordert, dass eine Datei geöffnet ist. Nur einen Ordner zu öffnen reicht nicht aus.
2. **Überprüfen Sie die VS Code-Version:** Erfordert 1.98.0 oder höher (Help → About)
3. **Starten Sie VS Code neu:** Führen Sie "Developer: Reload Window" aus der Befehlspalette aus
4. **Deaktivieren Sie konfliktverursachende Erweiterungen:** Deaktivieren Sie vorübergehend andere KI-Erweiterungen (Cline, Continue usw.)

5. **Überprüfen Sie die Workspace-Vertrauenswürdigkeit:** Die Erweiterung funktioniert nicht im Restricted Mode

Alternativ klicken Sie auf „[Claude Code](#)“ in der **Statusleiste** (untere rechte Ecke). Dies funktioniert auch ohne eine geöffnete Datei. Sie können auch die **Befehlspalette** (`Cmd+Shift+P` / `Ctrl+Shift+P`) verwenden und „Claude Code“ eingeben.

Claude Code antwortet nie

Wenn Claude Code nicht auf Ihre Eingabeaufforderungen antwortet:

1. **Überprüfen Sie Ihre Internetverbindung:** Stellen Sie sicher, dass Sie eine stabile Internetverbindung haben
2. **Starten Sie ein neues Gespräch:** Versuchen Sie, ein neues Gespräch zu starten, um zu sehen, ob das Problem weiterhin besteht
3. **Versuchen Sie die CLI:** Führen Sie `claude` vom Terminal aus, um detailliertere Fehlermeldungen zu erhalten

Wenn Probleme weiterhin bestehen, [melden Sie ein Problem auf GitHub](#) mit Details zum Fehler.

Deinstallieren Sie die Erweiterung

So deinstallieren Sie die Claude Code-Erweiterung:

1. Öffnen Sie die Ansicht „Erweiterungen“ (`Cmd+Shift+X` auf Mac oder `Ctrl+Shift+X` auf Windows/Linux)
2. Suchen Sie nach „Claude Code“
3. Klicken Sie auf **Uninstall**

Um auch Erweiterungsdaten zu entfernen und alle Einstellungen zurückzusetzen:

```
rm -rf ~/.vscode/globalStorage/anthropic.claude-code
```

Weitere Hilfe finden Sie im [Troubleshooting-Leitfaden](#).

Nächste Schritte

Jetzt, da Sie Claude Code in VS Code eingerichtet haben:

- [Erkunden Sie häufige Workflows](#), um das Beste aus Claude Code herauszuholen

- [Richten Sie MCP servers ein](#), um Claudes Funktionen mit externen Tools zu erweitern. Fügen Sie Server über die CLI hinzu und verwalten Sie sie dann mit `/mcp` im Chat-Panel.
- [Konfigurieren Sie Claude Code-Einstellungen](#), um zulässige Befehle, hooks und mehr anzupassen. Diese Einstellungen werden zwischen der Erweiterung und CLI geteilt.

JetBrains IDEs

Verwenden Sie Claude Code mit JetBrains IDEs einschließlich IntelliJ, PyCharm, WebStorm und mehr

Claude Code integriert sich mit JetBrains IDEs durch ein dediziertes Plugin und bietet Funktionen wie interaktive Diff-Anzeige, Freigabe von Auswahlkontext und mehr.

Unterstützte IDEs

Das Claude Code Plugin funktioniert mit den meisten JetBrains IDEs, einschließlich:

- IntelliJ IDEA
- PyCharm
- Android Studio
- WebStorm
- PhpStorm
- GoLand

Funktionen

- **Schnellstart:** Verwenden Sie **Cmd+Esc** (Mac) oder **Ctrl+Esc** (Windows/Linux), um Claude Code direkt aus Ihrem Editor zu öffnen, oder klicken Sie auf die Claude Code Schaltfläche in der Benutzeroberfläche
- **Diff-Anzeige:** Code-Änderungen können direkt im IDE Diff-Viewer anstelle des Terminals angezeigt werden
- **Auswahlkontext:** Die aktuelle Auswahl/der aktuelle Tab in der IDE wird automatisch mit Claude Code geteilt
- **Dateireferenz-Verknüpfungen:** Verwenden Sie **Cmd+Option+K** (Mac) oder **Alt+Ctrl+K** (Linux/Windows), um Dateireferenzen einzufügen (zum Beispiel @File#L1-99)
- **Diagnose-Freigabe:** Diagnosefehler (Lint, Syntax usw.) aus der IDE werden automatisch mit Claude Code geteilt, während Sie arbeiten

Installation

Marketplace-Installation

Suchen Sie das [Claude Code Plugin](#) im JetBrains Marketplace und installieren Sie es, dann starten Sie Ihre IDE neu.

Falls Sie Claude Code noch nicht installiert haben, lesen Sie [unseren Schnellstart-Leitfaden](#) für Installationsanweisungen.

Note:

Nach der Installation des Plugins müssen Sie Ihre IDE möglicherweise vollständig neu starten, damit es wirksam wird.

Verwendung

Aus Ihrer IDE

Führen Sie `claude` aus dem integrierten Terminal Ihrer IDE aus, und alle Integrationsfunktionen sind aktiv.

Aus externen Terminals

Verwenden Sie den `/ide` Befehl in einem beliebigen externen Terminal, um Claude Code mit Ihrer JetBrains IDE zu verbinden und alle Funktionen zu aktivieren:

```
claude
```

```
/ide
```

Wenn Sie möchten, dass Claude Zugriff auf die gleichen Dateien wie Ihre IDE hat, starten Sie Claude Code aus dem gleichen Verzeichnis wie Ihr IDE-Projektstammverzeichnis.

Konfiguration

Claude Code Einstellungen

Konfigurieren Sie die IDE-Integration durch Claude Code Einstellungen:

1. Führen Sie `claude` aus
2. Geben Sie den `/config` Befehl ein

3. Stellen Sie das Diff-Tool auf **auto** für automatische IDE-Erkennung ein

Plugin-Einstellungen

Konfigurieren Sie das Claude Code Plugin, indem Sie zu **Einstellungen** → **Tools** → **Claude Code [Beta]** gehen:

Allgemeine Einstellungen

- **Claude Befehl:** Geben Sie einen benutzerdefinierten Befehl an, um Claude auszuführen (zum Beispiel `claude`, `/usr/local/bin/claude` oder `npx @anthropic/claude`)
- **Benachrichtigung für Claude-Befehl nicht gefunden unterdrücken:** Überspringen Sie Benachrichtigungen über das Nichtfinden des Claude-Befehls
- **Option+Enter für mehrzeilige Eingabeaufforderungen aktivieren** (nur macOS): Wenn aktiviert, fügt Option+Enter neue Zeilen in Claude Code Eingabeaufforderungen ein. Deaktivieren Sie dies, wenn Sie Probleme mit der Option-Taste haben, die unerwartet erfasst wird (erfordert Terminal-Neustart)
- **Automatische Updates aktivieren:** Automatisch nach Plugin-Updates suchen und diese installieren (wird beim Neustart angewendet)

Tip:

Für WSL-Benutzer: Stellen Sie `wsl -d Ubuntu -- bash -lic "claude"` als Ihren Claude-Befehl ein (ersetzen Sie `Ubuntu` durch Ihren WSL-Distributionsnamen)

ESC-Taste Konfiguration

Wenn die ESC-Taste Claude Code Operationen in JetBrains Terminals nicht unterbricht:

1. Gehen Sie zu **Einstellungen** → **Tools** → **Terminal**
2. Entweder:
 - Deaktivieren Sie „Fokus mit Escape zum Editor verschieben“, oder
 - Klicken Sie auf „Terminal-Tastenkombinationen konfigurieren“ und löschen Sie die Verknüpfung „Fokus zum Editor wechseln“
3. Wenden Sie die Änderungen an

Dies ermöglicht es der ESC-Taste, Claude Code Operationen ordnungsgemäß zu unterbrechen.

Spezielle Konfigurationen

Remote-Entwicklung

Warning:

Bei Verwendung von JetBrains Remote Development müssen Sie das Plugin auf dem Remote-Host über **Einstellungen** → **Plugin (Host)** installieren.

Das Plugin muss auf dem Remote-Host installiert werden, nicht auf Ihrem lokalen Client-Computer.

WSL-Konfiguration

Warning:

WSL-Benutzer benötigen möglicherweise zusätzliche Konfiguration, damit die IDE-Erkennung ordnungsgemäß funktioniert. Lesen Sie unseren [WSL-Troubleshooting-Leitfaden](#) für detaillierte Setupanweisungen.

Die WSL-Konfiguration kann Folgendes erfordern:

- Ordnungsgemäße Terminal-Konfiguration
- Anpassungen des Netzwerkmodus
- Aktualisierungen der Firewall-Einstellungen

Troubleshooting

Plugin funktioniert nicht

- Stellen Sie sicher, dass Sie Claude Code aus dem Projektstammverzeichnis ausführen
- Überprüfen Sie, dass das JetBrains Plugin in den IDE-Einstellungen aktiviert ist
- Starten Sie die IDE vollständig neu (möglicherweise müssen Sie dies mehrmals tun)
- Stellen Sie für Remote Development sicher, dass das Plugin auf dem Remote-Host installiert ist

IDE nicht erkannt

- Überprüfen Sie, dass das Plugin installiert und aktiviert ist
- Starten Sie die IDE vollständig neu
- Überprüfen Sie, dass Sie Claude Code aus dem integrierten Terminal ausführen

- Für WSL-Benutzer lesen Sie den [WSL-Troubleshooting-Leitfaden](#)

Befehl nicht gefunden

Wenn das Klicken auf das Claude-Symbol „Befehl nicht gefunden“ anzeigt:

1. Überprüfen Sie, dass Claude Code installiert ist: `npm list -g @anthropic-ai/claude-code`
2. Konfigurieren Sie den Claude-Befehlspfad in den Plugin-Einstellungen
3. Für WSL-Benutzer verwenden Sie das WSL-Befehlsformat, das im Konfigurationsabschnitt erwähnt wird

Sicherheitsaspekte

Wenn Claude Code in einer JetBrains IDE mit aktivierten Auto-Edit-Berechtigungen ausgeführt wird, kann es möglicherweise IDE-Konfigurationsdateien ändern, die automatisch von Ihrer IDE ausgeführt werden können. Dies kann das Risiko der Ausführung von Claude Code im Auto-Edit-Modus erhöhen und es ermöglichen, Claude Code Berechtigungsaufforderungen für die Bash-Ausführung zu umgehen.

Bei der Ausführung in JetBrains IDEs sollten Sie Folgendes beachten:

- Verwenden Sie den manuellen Genehmigungsmodus für Bearbeitungen
- Achten Sie besonders darauf, dass Claude nur mit vertrauenswürdigen Eingabeaufforderungen verwendet wird
- Seien Sie sich bewusst, welche Dateien Claude Code ändern kann

Weitere Hilfe finden Sie in unserem [Troubleshooting-Leitfaden](#).

Claude Code Desktop verwenden

Nutzen Sie Claude Code Desktop optimal: parallele Sitzungen mit Git-Isolation, visuelle Diff-Überprüfung, App-Vorschau, PR-Überwachung, Berechtigungsmodi, Konnektoren und Unternehmenskonfiguration.

Die Registerkarte „Code“ in der Claude Desktop-App ermöglicht es Ihnen, Claude Code über eine grafische Benutzeroberfläche statt über das Terminal zu verwenden.

Desktop bietet diese Funktionen zusätzlich zur Standard-Claude-Code-Erfahrung:

- [Visuelle Diff-Überprüfung](#) mit Inline-Kommentaren
- [Live-App-Vorschau](#) mit Dev-Servern
- [GitHub-PR-Überwachung](#) mit automatischer Fehlerbehebung und automatischem Merge
- [Parallele Sitzungen](#) mit automatischer Git-Worktree-Isolation
- [Geplante Aufgaben](#), die Claude nach einem wiederkehrenden Zeitplan ausführen
- [Konnektoren](#) für GitHub, Slack, Linear und mehr
- Lokale, [SSH](#)- und [Cloud](#)-Umgebungen

Tip:

Neu bei Desktop? Beginnen Sie mit [Erste Schritte](#), um die App zu installieren und Ihre erste Bearbeitung vorzunehmen.

Diese Seite behandelt [Arbeiten mit Code](#), [Verwalten von Sitzungen](#), [Erweitern von Claude Code](#), [Geplante Aufgaben](#) und [Konfiguration](#). Sie enthält auch einen [CLI-Vergleich](#) und [Fehlerbehebung](#).

Sitzung starten

Bevor Sie Ihre erste Nachricht senden, konfigurieren Sie vier Dinge im Eingabebereich:

- **Umgebung:** Wählen Sie, wo Claude ausgeführt wird. Wählen Sie **Lokal** für Ihren Computer, **Remote** für von Anthropic gehostete Cloud-Sitzungen oder eine [SSH-Verbindung](#) für einen von Ihnen verwalteten Remote-Computer. Siehe [Umgebungs-konfiguration](#).

- **Projektordner:** Wählen Sie den Ordner oder das Repository aus, in dem Claude arbeitet. Für Remote-Sitzungen können Sie [mehrere Repositories](#) hinzufügen.
- **Modell:** Wählen Sie ein [Modell](#) aus dem Dropdown neben der Schaltfläche „Senden“. Das Modell wird gesperrt, sobald die Sitzung startet.
- **Berechtigungsmodus:** Wählen Sie, wie viel Autonomie Claude aus dem [Moduswahlschalter](#) hat. Sie können dies während der Sitzung ändern.

Geben Sie Ihre Aufgabe ein und drücken Sie **Eingabe**, um zu starten. Jede Sitzung verfolgt ihren eigenen Kontext und Änderungen unabhängig.

Arbeiten mit Code

Geben Sie Claude den richtigen Kontext, kontrollieren Sie, wie viel es eigenständig tut, und überprüfen Sie, was es geändert hat.

Verwenden Sie das Eingabefeld

Geben Sie ein, was Claude tun soll, und drücken Sie **Eingabe**, um zu senden. Claude liest Ihre Projektdateien, nimmt Änderungen vor und führt Befehle basierend auf Ihrem [Berechtigungsmodus](#) aus. Sie können Claude jederzeit unterbrechen: Klicken Sie auf die Stoppschaltfläche oder geben Sie Ihre Korrektur ein und drücken Sie **Eingabe**. Claude stoppt, was es tut, und passt sich basierend auf Ihrer Eingabe an.

Die Schaltfläche + neben dem Eingabefeld gibt Ihnen Zugriff auf Dateianhänge, [Skills](#), [Konnektoren](#) und [Plugins](#).

Fügen Sie Dateien und Kontext zu Eingaben hinzu

Das Eingabefeld unterstützt zwei Möglichkeiten, um externen Kontext einzubinden:

- **@mention-Dateien:** Geben Sie  gefolgt von einem Dateinamen ein, um eine Datei zum Gesprächskontext hinzuzufügen. Claude kann diese Datei dann lesen und referenzieren.
- **Dateien anhängen:** Hängen Sie Bilder, PDFs und andere Dateien an Ihre Eingabe an, indem Sie die Schaltfläche „Anhängen“ verwenden, oder ziehen Sie Dateien direkt in die Eingabe. Dies ist nützlich zum Teilen von Screenshots von Fehlern, Design-Mockups oder Referenzdokumenten.

Wählen Sie einen Berechtigungsmodus

Berechtigungsmodi kontrollieren, wie viel Autonomie Claude während einer Sitzung hat: ob es vor dem Bearbeiten von Dateien, dem Ausführen von Befehlen oder beidem fragt. Sie können Modi jederzeit mit dem Moduswahlschalter neben der Schaltfläche „Senden“

wechseln. Beginnen Sie mit „Berechtigungen erfragen“, um genau zu sehen, was Claude tut, und wechseln Sie dann zu „Bearbeitungen automatisch akzeptieren“ oder „Plan Mode“, wenn Sie sich wohler fühlen.

Modus	Einstellungsschlüssel	Verhalten
Berechtigungen erfragen	default	Claude fragt vor dem Bearbeiten von Dateien oder dem Ausführen von Befehlen. Sie sehen einen Diff und können jede Änderung akzeptieren oder ablehnen. Empfohlen für neue Benutzer.
Bearbeitungen automatisch akzeptieren	acceptEdits	Claude akzeptiert Datei-bearbeitungen automatisch, fragt aber immer noch vor dem Ausführen von Terminal-Befehlen. Verwenden Sie dies, wenn Sie Dateiänderungen vertrauen und schnellere Iterationen wünschen.
Plan Mode	plan	Claude analysiert Ihren Code und erstellt einen Plan, ohne Dateien zu ändern oder Befehle auszuführen. Gut für komplexe Aufgaben, bei denen Sie den Ansatz zuerst überprüfen möchten.

Modus	Einstellungsschlüssel	Verhalten
Berechtigungen umgehen	<code>bypassPermissions</code>	Claude läuft ohne Berechtigungsaufforderungen, äquivalent zu <code>--dangerously-skip-permissions</code> in der CLI. Aktivieren Sie dies in Ihren Einstellungen → Claude Code unter „Bypass-Berechtigungsmodus zulassen“. Verwenden Sie dies nur in sandboxierten Containern oder VMs. Enterprise-Administratoren können diese Option deaktivieren.

Der Berechtigungsmodus `dontAsk` ist nur in der [CLI](#) verfügbar.

Tip:

Beginnen Sie komplexe Aufgaben im Plan Mode, damit Claude einen Ansatz abbildet, bevor Änderungen vorgenommen werden. Sobald Sie den Plan genehmigen, wechseln Sie zu "Bearbeitungen automatisch akzeptieren" oder „Berechtigungen erfragen“, um ihn auszuführen. Siehe [Zuerst erkunden, dann planen, dann codieren](#) für mehr zu diesem Workflow.

Remote-Sitzungen unterstützen „Bearbeitungen automatisch akzeptieren“ und „Plan Mode“. „Berechtigungen erfragen“ ist nicht verfügbar, da Remote-Sitzungen Dateibearbeitungen standardmäßig automatisch akzeptieren, und „Berechtigungen umgehen“ ist nicht verfügbar, da die Remote-Umgebung bereits sandboxed ist.

Enterprise-Administratoren können einschränken, welche Berechtigungsmodi verfügbar sind. Siehe [Unternehmenskonfiguration](#) für Details.

Vorschau Ihrer App

Claude kann einen Dev-Server starten und einen eingebetteten Browser öffnen, um seine Änderungen zu überprüfen. Dies funktioniert sowohl für Frontend-Web-Apps als auch für Backend-Server: Claude kann API-Endpunkte testen, Server-Protokolle anzeigen und Probleme, die er findet, iterieren. In den meisten Fällen startet Claude den Server automa-

tisch nach dem Bearbeiten von Projektdateien. Sie können Claude auch jederzeit bitten, eine Vorschau anzuzeigen. Standardmäßig [überprüft Claude automatisch](#) Änderungen nach jeder Bearbeitung.

Aus dem Vorschau-Panel können Sie:

- Direkt im eingebetteten Browser mit Ihrer laufenden App interagieren
- Beobachten, wie Claude seine eigenen Änderungen automatisch überprüft: Es macht Screenshots, inspiziert das DOM, klickt auf Elemente, füllt Formulare aus und behebt Probleme, die es findet
- Server aus dem Dropdown **Vorschau** in der Sitzungs-Symboleiste starten oder stoppen
- Cookies und lokalen Speicher über Server-Neustarts hinweg beibehalten, indem Sie **Sitzungen beibehalten** im Dropdown auswählen, damit Sie sich während der Entwicklung nicht erneut anmelden müssen
- Die Server-Konfiguration bearbeiten oder alle Server auf einmal stoppen

Claude erstellt die anfängliche Server-Konfiguration basierend auf Ihrem Projekt. Wenn Ihre App einen benutzerdefinierten Dev-Befehl verwendet, bearbeiten Sie `.claude/launch.json`, um Ihr Setup zu entsprechen. Siehe [Vorschau-Server konfigurieren](#) für die vollständige Referenz.

Um gespeicherte Sitzungsdaten zu löschen, schalten Sie **Vorschau-Sitzungen beibehalten** in Einstellungen → Claude Code aus. Um die Vorschau vollständig zu deaktivieren, schalten Sie **Vorschau** in Einstellungen → Claude Code aus.

Überprüfen Sie Änderungen mit der Diff-Ansicht

Nachdem Claude Änderungen an Ihrem Code vorgenommen hat, können Sie mit der Diff-Ansicht Änderungen dateiweise überprüfen, bevor Sie einen Pull Request erstellen.

Wenn Claude Dateien ändert, wird ein Diff-Statistik-Indikator angezeigt, der die Anzahl der hinzugefügten und entfernten Zeilen anzeigt, z. B. `+12 -1`. Klicken Sie auf diesen Indikator, um den Diff-Viewer zu öffnen, der eine Dateiliste auf der linken Seite und die Änderungen für jede Datei auf der rechten Seite anzeigt.

Um Kommentare zu bestimmten Zeilen hinzuzufügen, klicken Sie auf eine beliebige Zeile im Diff, um ein Kommentarfeld zu öffnen. Geben Sie Ihr Feedback ein und drücken Sie **Eingabe**, um den Kommentar hinzuzufügen. Nach dem Hinzufügen von Kommentaren zu mehreren Zeilen senden Sie alle Kommentare auf einmal:

- **macOS:** drücken Sie **Cmd+Eingabe**
- **Windows:** drücken Sie **Strg+Eingabe**

Claude liest Ihre Kommentare und nimmt die angeforderten Änderungen vor, die als neuer Diff angezeigt werden, den Sie überprüfen können.

Überprüfen Sie Ihren Code

Klicken Sie in der Diff-Ansicht auf **Code überprüfen** in der oberen rechten Symbolleiste, um Claude zu bitten, die Änderungen vor dem Commit zu bewerten. Claude untersucht die aktuellen Diffs und hinterlässt Kommentare direkt in der Diff-Ansicht. Sie können auf jeden Kommentar antworten oder Claude bitten, zu überarbeiten.

Die Überprüfung konzentriert sich auf hochwertige Probleme: Kompilierungsfehler, definitive Logikfehler, Sicherheitslücken und offensichtliche Fehler. Sie kennzeichnet keine Stil-, Formatierungs-, bereits vorhandenen Probleme oder etwas, das ein Linter erfassen würde.

Überwachen Sie den Pull-Request-Status

Nachdem Sie einen Pull Request öffnen, wird eine CI-Statusleiste in der Sitzung angezeigt. Claude Code verwendet die GitHub CLI, um Prüfergebnisse abzurufen und Fehler anzuzeigen.

- **Automatische Fehlerbehebung:** Wenn aktiviert, versucht Claude automatisch, fehlgeschlagene CI-Prüfungen zu beheben, indem die Fehlerausgabe gelesen und iteriert wird.
- **Automatisches Merge:** Wenn aktiviert, führt Claude den PR zusammen, sobald alle Prüfungen bestanden sind. Die Merge-Methode ist Squash. Das automatische Merge muss [in Ihren GitHub-Repository-Einstellungen aktiviert sein](#), damit dies funktioniert.

Verwenden Sie die Umschalter **Automatische Fehlerbehebung** und **Automatisches Merge** in der CI-Statusleiste, um eine der beiden Optionen zu aktivieren. Claude Code sendet auch eine Desktop-Benachrichtigung, wenn CI abgeschlossen ist.

Note:

Die PR-Überwachung erfordert, dass die **GitHub CLI (gh)** auf Ihrem Computer installiert und authentifiziert ist. Wenn **gh** nicht installiert ist, fordert Desktop Sie auf, es beim ersten Versuch, einen PR zu erstellen, zu installieren.

Verwalten Sie Sitzungen

Jede Sitzung ist ein unabhängiges Gespräch mit eigenem Kontext und Änderungen. Sie können mehrere Sitzungen parallel ausführen oder Arbeit in die Cloud senden.

Arbeiten Sie parallel mit Sitzungen

Klicken Sie auf + **Neue Sitzung** in der Seitenleiste, um an mehreren Aufgaben parallel zu arbeiten. Für Git-Repositories erhält jede Sitzung ihre eigene isolierte Kopie Ihres Projekts mit [Git Worktrees](#), sodass Änderungen in einer Sitzung andere Sitzungen nicht beeinflussen, bis Sie sie committen.

Worktrees werden standardmäßig in `<project-root>/.claude/worktrees/` gespeichert. Sie können dies in Einstellungen → Claude Code unter „Worktree-Speicherort“ in ein benutzerdefiniertes Verzeichnis ändern. Sie können auch ein Branch-Präfix festlegen, das jedem von Claude erstellten Worktree-Branch-Namen vorangestellt wird, was nützlich ist, um von Claude erstellte Branches organisiert zu halten. Um einen Worktree zu entfernen, wenn Sie fertig sind, bewegen Sie den Mauszeiger über die Sitzung in der Seitenleiste und klicken Sie auf das Archiv-Symbol.

Note:

Die Sitzungsisolierung erfordert [Git](#). Die meisten Macs enthalten Git standardmäßig. Führen Sie `git --version` im Terminal aus, um zu überprüfen. Unter Windows ist Git erforderlich, damit die Registerkarte „Code“ funktioniert: [Laden Sie Git für Windows herunter](#), installieren Sie es und starten Sie die App neu. Wenn Sie auf Git-Fehler stoßen, versuchen Sie eine Cowork-Sitzung, um Ihr Setup zu beheben.

Verwenden Sie das Filtersymbol oben in der Seitenleiste, um Sitzungen nach Status (Aktiv, Archiviert) und Umgebung (Lokal, Cloud) zu filtern. Um eine Sitzung umzubenennen oder die Kontextnutzung zu überprüfen, klicken Sie auf den Sitzungstitel in der Symbolleiste oben in der aktiven Sitzung. Wenn der Kontext voll wird, fasst Claude das Gespräch automatisch zusammen und arbeitet weiter. Sie können auch `/compact` eingeben, um die Zusammenfassung früher auszulösen und Kontextraum freizugeben. Siehe [das Kontextfenster](#) für Details, wie die Komprimierung funktioniert.

Führen Sie lange laufende Aufgaben remote aus

Für große Refaktorisierungen, Test-Suites, Migrationen oder andere lange laufende Aufgaben wählen Sie **Remote** statt **Lokal**, wenn Sie eine Sitzung starten. Remote-Sitzungen laufen auf Anthropic Cloud-Infrastruktur und werden fortgesetzt, auch wenn Sie die App schließen oder Ihren Computer herunterfahren. Überprüfen Sie jederzeit den Fortschritt oder lenken Sie Claude in eine andere Richtung. Sie können Remote-Sitzungen auch von [claude.ai/code](#) oder der Claude iOS-App aus überwachen.

Remote-Sitzungen unterstützen auch mehrere Repositories. Nach Auswahl einer Cloud-Umgebung klicken Sie auf die Schaltfläche + neben dem Repo-Pill, um zusätzliche Repositories zur Sitzung hinzuzufügen. Jedes Repo erhält seinen eigenen Branch-Wahlschalter. Dies ist nützlich für Aufgaben, die mehrere Codebases umfassen, z. B. das Aktualisieren einer gemeinsamen Bibliothek und ihrer Consumer.

Siehe [Claude Code im Web](#) für mehr darüber, wie Remote-Sitzungen funktionieren.

Fortsetzen auf einer anderen Oberfläche

Das Menü **Fortsetzen in**, das über das VS Code-Symbol unten rechts in der Sitzungs-Symbolleiste zugänglich ist, ermöglicht es Ihnen, Ihre Sitzung auf eine andere Oberfläche zu verschieben:

- **Claude Code im Web:** sendet Ihre lokale Sitzung, um remote weiter zu laufen. Desktop pusht Ihren Branch, generiert eine Zusammenfassung des Gesprächs und erstellt eine neue Remote-Sitzung mit dem vollständigen Kontext. Sie können dann wählen, die lokale Sitzung zu archivieren oder zu behalten. Dies erfordert einen sauberen Arbeitsbaum und ist nicht für SSH-Sitzungen verfügbar.
- **Ihre IDE:** öffnet Ihr Projekt in einer unterstützten IDE im aktuellen Arbeitsverzeichnis.

Erweitern Sie Claude Code

Verbinden Sie externe Dienste, fügen Sie wiederverwendbare Workflows hinzu, passen Sie Claudes Verhalten an und konfigurieren Sie Vorschau-Server.

Verbinden Sie externe Tools

Für lokale und [SSH](#)-Sitzungen klicken Sie auf die Schaltfläche + neben dem Eingabefeld und wählen Sie **Konnektoren**, um Integrationen wie Google Calendar, Slack, GitHub, Linear, Notion und mehr hinzuzufügen. Sie können Konnektoren vor oder während einer Sitzung hinzufügen. Konnektoren sind nicht für Remote-Sitzungen verfügbar.

Um Konnektoren zu verwalten oder zu trennen, gehen Sie zu Einstellungen → Konnektoren in der Desktop-App oder wählen Sie **Konnektoren verwalten** aus dem Konnektoren-Menü im Eingabefeld.

Nach der Verbindung kann Claude Ihren Kalender lesen, Nachrichten senden, Probleme erstellen und direkt mit Ihren Tools interagieren. Sie können Claude fragen, welche Konnektoren in Ihrer Sitzung konfiguriert sind.

Konnektoren sind [MCP-Server](#) mit einem grafischen Setup-Ablauf. Verwenden Sie sie für schnelle Integration mit unterstützten Diensten. Für Integrationen, die nicht in Konnektoren aufgelistet sind, fügen Sie MCP-Server manuell über [Einstellungsdateien](#) hinzu. Sie können auch [benutzerdefinierte Konnektoren erstellen](#).

Verwenden Sie Skills

[Skills](#) erweitern, was Claude tun kann. Claude lädt sie automatisch, wenn relevant, oder Sie können eine direkt aufrufen: Geben Sie **/** im Eingabefeld ein oder klicken Sie auf die Schaltfläche **+** und wählen Sie **Slash-Befehle**, um zu sehen, was verfügbar ist. Dies umfasst [integrierte Befehle](#), Ihre [benutzerdefinierten Skills](#), Projekt-Skills aus Ihrer Codebasis und Skills aus allen [installierten Plugins](#). Wählen Sie einen aus und er wird im Eingabefeld hervorgehoben angezeigt. Geben Sie Ihre Aufgabe danach ein und senden Sie wie gewohnt.

Installieren Sie Plugins

[Plugins](#) sind wiederverwendbare Pakete, die Skills, Agents, hooks, MCP-Server und LSP-Konfigurationen zu Claude Code hinzufügen. Sie können Plugins aus der Desktop-App installieren, ohne das Terminal zu verwenden.

Für lokale und [SSH](#)-Sitzungen klicken Sie auf die Schaltfläche **+** neben dem Eingabefeld und wählen Sie **Plugins**, um Ihre installierten Plugins und ihre Befehle zu sehen. Um ein Plugin hinzuzufügen, wählen Sie **Plugin hinzufügen** aus dem Untermenü, um den Plugin-Browser zu öffnen, der verfügbare Plugins aus Ihren konfigurierten [Marketplaces](#) einschließlich des offiziellen Anthropic-Marketplace anzeigt. Wählen Sie **Plugins verwalten**, um Plugins zu aktivieren, zu deaktivieren oder zu deinstallieren.

Plugins können auf Ihr Benutzerkonto, ein bestimmtes Projekt oder nur lokal beschränkt sein. Plugins sind nicht für Remote-Sitzungen verfügbar. Für die vollständige Plugin-Referenz einschließlich der Erstellung eigener Plugins siehe [Plugins](#).

Konfigurieren Sie Vorschau-Server

Claude erkennt automatisch Ihr Dev-Server-Setup und speichert die Konfiguration in `.claude/launch.json` im Stammverzeichnis des Ordners, den Sie beim Starten der Sitzung ausgewählt haben. Die Vorschau verwendet diesen Ordner als Arbeitsverzeichnis. Wenn Sie also einen übergeordneten Ordner ausgewählt haben, werden Unterordner mit ihren eigenen Dev-Servern nicht automatisch erkannt. Um mit dem Server eines Unterordners zu arbeiten, starten Sie entweder eine Sitzung direkt in diesem Ordner oder fügen Sie eine Konfiguration manuell hinzu.

Um anzupassen, wie Ihr Server startet, z. B. um `yarn dev` statt `npm run dev` zu verwenden oder den Port zu ändern, bearbeiten Sie die Datei manuell oder klicken Sie auf **Konfiguration bearbeiten** im Dropdown „Vorschau“, um sie in Ihrem Code-Editor zu öffnen. Die Datei unterstützt JSON mit Kommentaren.

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000
    }
  ]
}
```

Sie können mehrere Konfigurationen definieren, um verschiedene Server aus demselben Projekt auszuführen, z. B. ein Frontend und eine API. Siehe die [Beispiele](#) unten.

Automatische Überprüfung von Änderungen

Wenn `autoVerify` aktiviert ist, überprüft Claude automatisch Code-Änderungen nach dem Bearbeiten von Dateien. Es macht Screenshots, prüft auf Fehler und bestätigt, dass Änderungen funktionieren, bevor es seine Antwort abschließt.

Die automatische Überprüfung ist standardmäßig aktiviert. Deaktivieren Sie sie pro Projekt, indem Sie `"autoVerify": false` zu `.claude/launch.json` hinzufügen, oder schalten Sie sie aus dem Dropdown **Vorschau** um.

```
{
  "version": "0.0.1",
  "autoVerify": false,
  "configurations": [...]
}
```

Wenn deaktiviert, sind Vorschau-Tools immer noch verfügbar und Sie können Claude jederzeit bitten, zu überprüfen. Die automatische Überprüfung macht es automatisch nach jeder Bearbeitung.

Konfigurationsfelder

Jeder Eintrag im Array `configurations` akzeptiert die folgenden Felder:

Feld	Typ	Beschreibung
<code>name</code>	string	Ein eindeutiger Bezeichner für diesen Server
<code>runtimeExecutable</code>	string	Der auszuführende Befehl, z. B. <code>npm</code> , <code>yarn</code> oder <code>node</code>
<code>runtimeArgs</code>	string[]	An <code>runtimeExecutable</code> übergebene Argumente, z. B. <code>["run", "dev"]</code>
<code>port</code>	number	Der Port, auf dem Ihr Server lauscht. Standardmäßig 3000
<code>cwd</code>	string	Arbeitsverzeichnis relativ zu Ihrem Projektstamverzeichnis. Standardmäßig das Projektstamverzeichnis. Verwenden Sie <code>\${workspaceFolder}</code> , um das Projektstamverzeichnis explizit zu referenzieren
<code>env</code>	object	Zusätzliche Umgebungsvariablen als Schlüssel-Wert-Paare, z. B. <code>{ "NODE_ENV": "development" }</code> . Legen Sie hier keine Geheimnisse ab, da diese Datei in Ihr Repo committed wird. Geheimnisse, die in Ihrem Shell-Profil festgelegt sind, werden automatisch geerbt.
<code>autoPort</code>	boolean	Wie Port-Konflikte behandelt werden. Siehe unten

Feld	Typ	Beschreibung
<code>program</code>	<code>string</code>	Ein mit <code>node</code> auszuführendes Skript. Siehe wann <code>program</code> vs <code>runtimeExecutable</code> verwendet werden
<code>args</code>	<code>string[]</code>	An <code>program</code> übergebene Argumente. Wird nur verwendet, wenn <code>program</code> gesetzt ist

Wann `program` vs `runtimeExecutable` verwendet werden

Verwenden Sie `runtimeExecutable` mit `runtimeArgs` , um einen Dev-Server über einen Package Manager zu starten. Zum Beispiel `"runtimeExecutable": "npm"` mit `"runtimeArgs": ["run", "dev"]` führt `npm run dev` aus.

Verwenden Sie `program` , wenn Sie ein eigenständiges Skript haben, das Sie direkt mit `node` ausführen möchten. Zum Beispiel `"program": "server.js"` führt `node server.js` aus. Übergeben Sie zusätzliche Flags mit `args` .

Port-Konflikte

Das Feld `autoPort` kontrolliert, was passiert, wenn Ihr bevorzugter Port bereits verwendet wird:

- **true** : Claude findet und verwendet automatisch einen freien Port. Geeignet für die meisten Dev-Server.
- **false** : Claude schlägt mit einem Fehler fehl. Verwenden Sie dies, wenn Ihr Server einen bestimmten Port verwenden muss, z. B. für OAuth-Callbacks oder CORS-Allow-lists.
- **Nicht gesetzt (Standard)**: Claude fragt, ob der Server diesen genauen Port benötigt, und speichert dann Ihre Antwort.

Wenn Claude einen anderen Port wählt, übergibt es den zugewiesenen Port an Ihren Server über die Umgebungsvariable `PORT` .

Beispiele

Diese Konfigurationen zeigen häufige Setups für verschiedene Projekttypen:

Next.js

Diese Konfiguration führt eine Next.js-App mit Yarn auf Port 3000 aus:

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "web",
      "runtimeExecutable": "yarn",
      "runtimeArgs": ["dev"],
      "port": 3000
    }
  ]
}
```

Multiple servers

Für ein Monorepo mit einem Frontend und einem API-Server definieren Sie mehrere Konfigurationen. Das Frontend verwendet `autoPort: true`, sodass es einen freien Port wählt, wenn 3000 belegt ist, während der API-Server Port 8080 genau benötigt:

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "frontend",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "cwd": "apps/web",
      "port": 3000,
      "autoPort": true
    },
    {
      "name": "api",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "start"],
      "cwd": "server",
      "port": 8080,
      "env": { "NODE_ENV": "development" },
      "autoPort": false
    }
  ]
}
```

Node.js script

Um ein Node.js-Skript direkt auszuführen, statt einen Package-Manager-Befehl zu verwenden, verwenden Sie das Feld `program` :

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "server",
      "program": "server.js",
      "args": ["--verbose"],
      "port": 4000
    }
  ]
}
```

Planen Sie wiederkehrende Aufgaben

Geplante Aufgaben starten automatisch zu einer von Ihnen gewählten Zeit und Häufigkeit eine neue lokale Sitzung. Verwenden Sie sie für wiederkehrende Arbeiten wie tägliche Code-Überprüfungen, Abhängigkeitsaktualisierungsprüfungen oder morgendliche Briefings, die aus Ihrem Kalender und Posteingang abrufen.

Aufgaben laufen auf Ihrem Computer, daher muss die Desktop-App offen und Ihr Computer wach sein, damit sie ausgelöst werden. Siehe [Wie geplante Aufgaben ausgeführt werden](#) für Details zu verpassten Läufen und Aufholverhalten.

Note:

Standardmäßig laufen geplante Aufgaben gegen den aktuellen Zustand Ihres Arbeitsverzeichnisses, einschließlich nicht committeter Änderungen. Aktivieren Sie den Worktree-Umschalter in der Eingabe, um jedem Lauf seinen eigenen isolierten Git-Worktree zu geben, genauso wie [parallele Sitzungen](#) funktionieren.

Um eine geplante Aufgabe zu erstellen, klicken Sie auf **Zeitplan** in der Seitenleiste und dann auf + **Neue Aufgabe**. Konfigurieren Sie diese Felder:

Feld	Beschreibung
Name	Bezeichner für die Aufgabe. Wird in Kleinbuchstaben Kebab-Case konvertiert und als Ordnername auf der Festplatte verwendet. Muss eindeutig über Ihre Aufgaben sein.

Feld	Beschreibung
Be-schrei-bung	Kurze Zusammenfassung, die in der Aufgabenliste angezeigt wird.
Einga-be	Die Anweisungen, die an Claude gesendet werden, wenn die Aufgabe ausgeführt wird. Schreiben Sie dies genauso, wie Sie jede Nachricht im Eingabefeld schreiben würden. Die Eingabe enthält auch Steuerelemente für Modell, Berechtigungsmodus, Arbeitsordner und Worktree.
Häu-figkeit	Wie oft die Aufgabe ausgeführt wird. Siehe Häufigkeitsoptionen unten.

Sie können auch eine Aufgabe erstellen, indem Sie in einer beliebigen Sitzung beschreiben, was Sie möchten. Zum Beispiel: „Richten Sie eine tägliche Code-Überprüfung ein, die jeden Morgen um 9 Uhr ausgeführt wird.“

Häufigkeitsoptionen

- **Manuell:** kein Zeitplan, läuft nur, wenn Sie auf **Jetzt ausführen** klicken. Nützlich zum Speichern einer Eingabe, die Sie bei Bedarf auslösen
- **Stündlich:** läuft jede Stunde. Jede Aufgabe erhält einen festen Versatz von bis zu 10 Minuten von der vollen Stunde, um API-Verkehr zu staffeln
- **Täglich:** zeigt einen Zeitwahlschalter, Standard ist 9:00 Uhr Ortszeit
- **Wochentage:** wie täglich, aber überspringt Samstag und Sonntag
- **Wöchentlich:** zeigt einen Zeitwahlschalter und einen Tageswahlschalter

Für Intervalle, die der Wahlschalter nicht anbietet (alle 15 Minuten, erste des Monats usw.), bitten Sie Claude in einer beliebigen Desktop-Sitzung, den Zeitplan festzulegen. Verwenden Sie einfache Sprache; zum Beispiel: „Planen Sie eine Aufgabe, um alle Tests alle 6 Stunden auszuführen.“

Wie geplante Aufgaben ausgeführt werden

Geplante Aufgaben laufen lokal auf Ihrem Computer. Desktop überprüft den Zeitplan jede Minute, während die App offen ist, und startet eine neue Sitzung, wenn eine Aufgabe fällig ist, unabhängig von manuellen Sitzungen, die Sie offen haben. Jede Aufgabe erhält eine feste Verzögerung von bis zu 10 Minuten nach der geplanten Zeit, um API-Verkehr zu staffeln. Die Verzögerung ist deterministisch: dieselbe Aufgabe startet immer mit demselben Versatz.

Wenn eine Aufgabe ausgelöst wird, erhalten Sie eine Desktop-Benachrichtigung und eine neue Sitzung wird unter einem Abschnitt **Geplant** in der Seitenleiste angezeigt. Öffnen Sie sie, um zu sehen, was Claude getan hat, Änderungen zu überprüfen oder auf Berechtigungsaufforderungen zu antworten. Die Sitzung funktioniert wie jede andere: Claude kann Dateien bearbeiten, Befehle ausführen, Commits erstellen und Pull Requests öffnen.

Aufgaben laufen nur, während die Desktop-App läuft und Ihr Computer wach ist. Wenn Ihr Computer durch eine geplante Zeit schläft, wird der Lauf übersprungen. Um Leerlauf-Schlaf zu verhindern, aktivieren Sie **Computer wach halten** in Einstellungen unter **Desktop-App** → **Allgemein**. Das Schließen des Laptop-Deckels setzt ihn immer noch in den Schlaf.

Verpasste Läufe

Wenn die App startet oder Ihr Computer aufwacht, überprüft Desktop, ob jede Aufgabe in den letzten sieben Tagen Läufe verpasst hat. Wenn ja, startet Desktop genau einen Aufhol-Lauf für die zuletzt verpasste Zeit und verwirft alles Ältere. Eine täglich geplante Aufgabe, die sechs Tage verpasst hat, läuft einmal beim Aufwachen. Desktop zeigt eine Benachrichtigung, wenn ein Aufhol-Lauf startet.

Beachten Sie dies beim Schreiben von Eingaben. Eine für 9 Uhr geplante Aufgabe könnte um 23 Uhr ausgeführt werden, wenn Ihr Computer den ganzen Tag schläft. Wenn das Timing wichtig ist, fügen Sie Schutzmaßnahmen zur Eingabe selbst hinzu, zum Beispiel: „Überprüfen Sie nur die heutigen Commits. Wenn es nach 17 Uhr ist, überspringen Sie die Überprüfung und posten Sie einfach eine Zusammenfassung dessen, was verpasst wurde.“

Berechtigungen für geplante Aufgaben

Jede Aufgabe hat ihren eigenen Berechtigungsmodus, den Sie beim Erstellen oder Bearbeiten der Aufgabe festlegen. Erlauben-Regeln aus `~/.claude/settings.json` gelten auch für geplante Aufgaben-Sitzungen. Wenn eine Aufgabe im Ask-Modus läuft und ein Tool ausführen muss, für das sie keine Berechtigung hat, stellt sich der Lauf hin, bis Sie ihn genehmigen. Die Sitzung bleibt offen in der Seitenleiste, damit Sie später antworten können.

Um Stalls zu vermeiden, klicken Sie auf **Jetzt ausführen**, nachdem Sie eine Aufgabe erstellt haben, achten Sie auf Berechtigungsaufforderungen und wählen Sie „Immer zulassen“ für jede aus. Zukünftige Läufe dieser Aufgabe genehmigen automatisch dieselben Tools ohne Aufforderung. Sie können diese Genehmigungen auf der Detailseite der Aufgabe überprüfen und widerrufen.

Verwalten Sie geplante Aufgaben

Klicken Sie auf eine Aufgabe in der Liste **Zeitplan**, um ihre Detailseite zu öffnen. Von hier aus können Sie:

- **Jetzt ausführen:** Starten Sie die Aufgabe sofort, ohne auf die nächste geplante Zeit zu warten
- **Wiederholungen umschalten:** Pausieren oder fortsetzen Sie geplante Läufe, ohne die Aufgabe zu löschen
- **Bearbeiten:** Ändern Sie die Eingabe, Häufigkeit, den Ordner oder andere Einstellungen
- **Verlauf überprüfen:** Sehen Sie jeden vergangenen Lauf, einschließlich solcher, die übersprungen wurden, weil Ihr Computer schläft
- **Erlaubte Berechtigungen überprüfen:** Sehen Sie und widerrufen Sie gespeicherte Tool-Genehmigungen für diese Aufgabe aus dem Panel **Immer zulassen**
- **Löschen:** Entfernen Sie die Aufgabe und archivieren Sie alle Sitzungen, die sie erstellt hat

Sie können Aufgaben auch verwalten, indem Sie Claude in einer beliebigen Desktop-Sitzung bitten. Zum Beispiel: „Pausieren Sie meine Aufgabe zur Abhängigkeitsprüfung“, „Löschen Sie die Aufgabe zur Standup-Vorbereitung“ oder „Zeigen Sie mir meine geplanten Aufgaben.“

Um die Eingabe einer Aufgabe auf der Festplatte zu bearbeiten, öffnen Sie `~/.claude/scheduled-tasks/<task-name>/SKILL.md` (oder unter `CLAUDE_CONFIG_DIR`, falls gesetzt). Die Datei verwendet YAML-Frontmatter für `name` und `description`, mit der Eingabe als Body. Änderungen treten beim nächsten Lauf in Kraft. Zeitplan, Ordner, Modell und aktiver Zustand sind nicht in dieser Datei: Ändern Sie sie über das Bearbeitungsformular oder bitten Sie Claude.

Umgebungskonfiguration

Die Umgebung, die Sie beim [Starten einer Sitzung](#) wählen, bestimmt, wo Claude ausgeführt wird und wie Sie sich verbinden:

- **Lokal:** läuft auf Ihrem Computer mit direktem Zugriff auf Ihre Dateien
- **Remote:** läuft auf Anthropic's Cloud-Infrastruktur. Sitzungen werden fortgesetzt, auch wenn Sie die App schließen.
- **SSH:** läuft auf einem Remote-Computer, mit dem Sie sich über SSH verbinden, z. B. Ihre eigenen Server, Cloud-VMs oder Dev-Container

Lokale Sitzungen

Lokale Sitzungen erben Umgebungsvariablen aus Ihrer Shell. Wenn Sie zusätzliche Variablen benötigen, legen Sie sie in Ihrem Shell-Profil fest, z. B. `~/.zshrc` oder `~/.bashrc`, und starten Sie die Desktop-App neu. Siehe [Umgebungsvariablen](#) für die vollständige Liste der unterstützten Variablen.

[Erweitertes Denken](#) ist standardmäßig aktiviert, was die Leistung bei komplexen Denkaufgaben verbessert, aber zusätzliche Token verwendet. Um das Denken vollständig zu deaktivieren, setzen Sie `MAX_THINKING_TOKENS=0` in Ihrem Shell-Profil. Bei Opus wird `MAX_THINKING_TOKENS` ignoriert, außer für `0`, da adaptive Argumentation die Denktiefe steuert.

Remote-Sitzungen

Remote-Sitzungen werden im Hintergrund fortgesetzt, auch wenn Sie die App schließen. Die Nutzung wird auf Ihre [Abonnementplanlimits](#) angerechnet, ohne separate Compute-Gebühren.

Sie können benutzerdefinierte Cloud-Umgebungen mit verschiedenen Netzwerkzugriffsstufen und Umgebungsvariablen erstellen. Wählen Sie das Umgebungs-Dropdown beim Starten einer Remote-Sitzung und wählen Sie **Umgebung hinzufügen**. Siehe [Cloud-Umgebungen](#) für Details zur Konfiguration von Netzwerkzugriff und Umgebungsvariablen.

SSH-Sitzungen

SSH-Sitzungen ermöglichen es Ihnen, Claude Code auf einem Remote-Computer auszuführen, während Sie die Desktop-App als Ihre Schnittstelle verwenden. Dies ist nützlich für die Arbeit mit Codebases, die auf Cloud-VMs, Dev-Containern oder Servern mit spezifischer Hardware oder Abhängigkeiten leben.

Um eine SSH-Verbindung hinzuzufügen, klicken Sie auf das Umgebungs-Dropdown vor dem Starten einer Sitzung und wählen Sie **+ SSH-Verbindung hinzufügen**. Der Dialog fragt nach:

- **Name:** ein freundlicher Bezeichner für diese Verbindung
- **SSH-Host:** `user@hostname` oder ein in `~/.ssh/config` definierter Host
- **SSH-Port:** Standard ist 22, wenn leer gelassen, oder verwendet den Port aus Ihrer SSH-Konfiguration
- **Identity-Datei:** Pfad zu Ihrem privaten Schlüssel, z. B. `~/.ssh/id_rsa`. Lassen Sie leer, um den Standardschlüssel oder Ihre SSH-Konfiguration zu verwenden.

Nach dem Hinzufügen wird die Verbindung im Umgebungs-Dropdown angezeigt. Wählen Sie sie aus, um eine Sitzung auf diesem Computer zu starten. Claude läuft auf dem Remote-Computer mit Zugriff auf seine Dateien und Tools.

Claude Code muss auf dem Remote-Computer installiert sein. Nach der Verbindung unterstützen SSH-Sitzungen Berechtigungsmodi, Konnektoren, Plugins und MCP-Server.

Unternehmenskonfiguration

Organisationen in Teams- oder Enterprise-Plänen können das Verhalten der Desktop-App durch Admin-Konsolen-Steuerelemente, verwaltete Einstellungsdateien und Geräteverwaltungsrichtlinien verwalten.

Admin-Konsolen-Steuerelemente

Diese Einstellungen werden über die [Admin-Einstellungskonsole](#) konfiguriert:

- **Aktivieren oder deaktivieren Sie die Registerkarte „Code“:** Kontrollieren Sie, ob Benutzer in Ihrer Organisation auf Claude Code in der Desktop-App zugreifen können
- **Deaktivieren Sie den Bypass-Berechtigungsmodus:** Verhindern Sie, dass Benutzer in Ihrer Organisation den Bypass-Berechtigungsmodus aktivieren
- **Deaktivieren Sie Claude Code im Web:** Aktivieren oder deaktivieren Sie Remote-Sitzungen für Ihre Organisation

Verwaltete Einstellungen

Verwaltete Einstellungen überschreiben Projekt- und Benutzereinstellungen und gelten, wenn Desktop CLI-Sitzungen startet. Sie können diese Schlüssel in der [verwalteten Einstellungsdatei](#) Ihrer Organisation festlegen oder sie remote über die Admin-Konsole pushen.

Schlüssel	Beschreibung
<code>disableBypassPermissionsMode</code>	auf <code>"disable"</code> setzen, um Benutzer daran zu hindern, den Bypass-Berechtigungsmodus zu aktivieren. Siehe verwaltete Einstellungen .

Für die vollständige Liste der verwalteten Einstellungen einschließlich `allowManagedPermissionRulesOnly` und `allowManagedHooksOnly` siehe [verwaltete Einstellungen](#).

Remote-verwaltete Einstellungen, die über die Admin-Konsole hochgeladen werden, gelten derzeit nur für CLI- und IDE-Sitzungen. Für Desktop-spezifische Einschränkungen verwenden Sie die Admin-Konsolen-Steuerelemente oben.

Geräteverwaltungsrichtlinien

IT-Teams können die Desktop-App über MDM auf macOS oder Gruppenrichtlinie unter Windows verwalten. Verfügbare Richtlinien umfassen das Aktivieren oder Deaktivieren der Claude-Code-Funktion, das Steuern von Auto-Updates und das Festlegen einer benutzerdefinierten Bereitstellungs-URL.

- **macOS:** Konfigurieren Sie über die Präferenzdomäne `com.anthropic.Claude` mit Tools wie Jamf oder Kandji
- **Windows:** Konfigurieren Sie über die Registrierung unter `SOFTWARE\Policies\Claude`

Authentifizierung und SSO

Enterprise-Organisationen können SSO für alle Benutzer verlangen. Siehe [Authentifizierung](#) für Details auf Ebene und [Einrichten von SSO](#) für SAML- und OIDC-Konfiguration.

Datenbehandlung

Claude Code verarbeitet Ihren Code lokal in lokalen Sitzungen oder auf Anthropic Cloud-Infrastruktur in Remote-Sitzungen. Gespräche und Code-Kontext werden an Anthropic API zur Verarbeitung gesendet. Siehe [Datenbehandlung](#) für Details zu Datenspeicherung, Datenschutz und Compliance.

Bereitstellung

Desktop kann über Enterprise-Bereitstellungstools verteilt werden:

- **macOS:** Verteilen Sie über MDM wie Jamf oder Kandji mit dem `.dmg` -Installer
- **Windows:** Stellen Sie über MSIX-Paket oder `.exe` -Installer bereit. Siehe [Claude Desktop für Windows bereitstellen](#) für Enterprise-Bereitstellungsoptionen einschließlich stiller Installation

Für Netzwerkkonfiguration wie Proxy-Einstellungen, Firewall-Allowlisting und LLM-Gateways siehe [Netzwerkkonfiguration](#).

Für die vollständige Unternehmenskonfigurationsreferenz siehe das [Unternehmenskonfigurationshandbuch](#).

Kommen Sie von der CLI?

Wenn Sie bereits die Claude Code CLI verwenden, führt Desktop dieselbe zugrunde liegende Engine mit einer grafischen Benutzeroberfläche aus. Sie können beide gleichzeitig auf demselben Computer ausführen, sogar auf demselben Projekt. Jede behält separate Sitzungsverlauf, aber sie teilen Konfiguration und Projektgedächtnis über CLAUDE.md-Dateien.

Um eine CLI-Sitzung in Desktop zu verschieben, führen Sie `/desktop` im Terminal aus. Claude speichert Ihre Sitzung und öffnet sie in der Desktop-App, dann beendet die CLI. Dieser Befehl ist nur auf macOS und Windows verfügbar.

Tip:

Wann Desktop vs CLI verwendet werden: Verwenden Sie Desktop, wenn Sie visuelle Diff-Überprüfung, Dateianhänge oder Sitzungsverwaltung in einer Seitenleiste möchten. Verwenden Sie die CLI, wenn Sie Scripting, Automatisierung, Drittanbieter oder ein Terminal-Workflow bevorzugen.

CLI-Flag-Äquivalente

Diese Tabelle zeigt das Desktop-App-Äquivalent für häufige CLI-Flags. Flags, die nicht aufgelistet sind, haben kein Desktop-Äquivalent, da sie für Scripting oder Automatisierung konzipiert sind.

CLI	Desktop-Äquivalent
<code>--model sonnet</code>	Modell-Dropdown neben der Schaltfläche „Senden“, vor dem Starten einer Sitzung
<code>--resume</code> , <code>--continue</code>	Klicken Sie auf eine Sitzung in der Seitenleiste
<code>--permission-mode</code>	Moduswahlschalter neben der Schaltfläche „Senden“
<code>--dangerously-skip-permissions</code>	Bypass-Berechtigungsmodus. Aktivieren Sie in Einstellungen → Claude Code → „Bypass-Berechtigungsmodus zulassen“. Enterprise-Administratoren können diese Einstellung deaktivieren.
<code>--add-dir</code>	Fügen Sie mehrere Repos mit der Schaltfläche + in Remote-Sitzungen hinzu
<code>--allowedTools</code> , <code>--disallowedTools</code>	nicht in Desktop verfügbar

CLI	Desktop-Äquivalent
<code>--verbose</code>	nicht verfügbar. Überprüfen Sie Systemprotokolle: Console.app auf macOS, Event Viewer → Windows-Protokolle → Anwendung unter Windows
<code>--print</code> , <code>--output-format</code>	nicht verfügbar. Desktop ist nur interaktiv.
<code>ANTHROPIC_MODEL</code> Umgebungsvariable	Modell-Dropdown neben der Schaltfläche „Senden“
<code>MAX_THINKING_TOKENS</code> Umgebungsvariable	in Shell-Profil festlegen; gilt für lokale Sitzungen. Siehe Umgebungs-konfiguration .

Gemeinsame Konfiguration

Desktop und CLI lesen dieselben Konfigurationsdateien, daher wird Ihr Setup übertragen:

- **CLAUDE.md-Dateien** in Ihrem Projekt werden von beiden verwendet
- **MCP-Server**, die in `~/.claude.json` oder `.mcp.json` konfiguriert sind, funktionieren in beiden
- **Hooks** und **Skills**, die in Einstellungen definiert sind, gelten für beide
- **Einstellungen** in `~/.claude.json` und `~/.claude/settings.json` werden geteilt. Berechtigungsregeln, erlaubte Tools und andere Einstellungen in `settings.json` gelten für Desktop-Sitzungen.
- **Modelle**: Sonnet, Opus und Haiku sind in beiden verfügbar. Wählen Sie in Desktop das Modell aus dem Dropdown neben der Schaltfläche „Senden“ vor dem Starten einer Sitzung. Sie können das Modell während einer aktiven Sitzung nicht ändern.

Note:

MCP-Server: Desktop-Chat-App vs Claude Code: MCP-Server, die für die Claude Desktop-Chat-App in `claude_desktop_config.json` konfiguriert sind, sind getrennt von Claude Code und werden nicht auf der Registerkarte „Code“ angezeigt. Um MCP-Server in Claude Code zu verwenden, konfigurieren Sie sie in `~/.claude.json` oder der `.mcp.json`-Datei Ihres Projekts. Siehe [MCP-Konfiguration](#) für Details.

Funktionsvergleich

Diese Tabelle vergleicht Kernfunktionen zwischen CLI und Desktop. Für eine vollständige Liste der CLI-Flags siehe die [CLI-Referenz](#).

Funktion	CLI	Desktop
Berechtigungsmodi	alle Modi einschließlich <code>dontAsk</code>	Berechtigungen erfragen, Bearbeitungen automatisch akzeptieren, Plan Mode und Bypass-Berechtigungen über Einstellungen
<code>--dangerously-skip-permissions</code>	CLI-Flag	Bypass-Berechtigungsmodus. Aktivieren Sie in Einstellungen → Claude Code → „Bypass-Berechtigungsmodus zulassen“
Drittanbieter-Provider	Bedrock, Vertex, Foundry	nicht verfügbar. Desktop verbindet sich direkt mit Anthropic API.
MCP-Server	in Einstellungsdateien konfigurieren	Konnektoren-UI für lokale und SSH-Sitzungen oder Einstellungsdateien
Plugins	<code>/plugin</code> -Befehl	Plugin-Manager-UI
@mention-Dateien	textbasiert	mit Autovervollständigung
Dateianhänge	nicht verfügbar	Bilder, PDFs
Sitzungsisoliation	<code>--worktree</code> -Flag	automatische Worktrees
Mehrere Sitzungen	separate Terminals	Seitenleisten-Tabs
Wiederkehrende Aufgaben	Cron-Jobs, CI-Pipelines	geplante Aufgaben
Scripting und Automatisierung	<code>--print</code> , Agent SDK	nicht verfügbar

Was ist nicht in Desktop verfügbar

Die folgenden Funktionen sind nur in der CLI oder VS Code-Erweiterung verfügbar:

- **Drittanbieter-Provider:** Desktop verbindet sich direkt mit Anthropic API. Verwenden Sie stattdessen die [CLI](#) mit Bedrock, Vertex oder Foundry.
- **Linux:** Die Desktop-App ist nur auf macOS und Windows verfügbar.

- **Inline-Code-Vorschläge:** Desktop bietet keine Autovervollständigungsstil-Vorschläge. Es funktioniert durch Gesprächseingaben und explizite Code-Änderungen.
- **Agent-Teams:** Multi-Agent-Orchestrierung ist über die [CLI](#) und [Agent SDK](#) verfügbar, nicht in Desktop.

Fehlerbehebung

Überprüfen Sie Ihre Version

Um zu sehen, welche Version der Desktop-App Sie ausführen:

- **macOS:** Klicken Sie auf **Claude** in der Menüleiste und dann auf **Über Claude**
- **Windows:** Klicken Sie auf **Hilfe** und dann auf **Über**

Klicken Sie auf die Versionsnummer, um sie in Ihre Zwischenablage zu kopieren.

403 oder Authentifizierungsfehler auf der Registerkarte „Code“

Wenn Sie **Error 403: Forbidden** oder andere Authentifizierungsfehler bei der Verwendung der Registerkarte „Code“ sehen:

1. Melden Sie sich aus dem App-Menü ab und wieder an. Dies ist die häufigste Lösung.
2. Überprüfen Sie, ob Sie ein aktives bezahltes Abonnement haben: Pro, Max, Teams oder Enterprise.
3. Wenn die CLI funktioniert, aber Desktop nicht, beenden Sie die Desktop-App vollständig, nicht nur das Fenster schließen, und öffnen Sie sie dann erneut und melden Sie sich an.
4. Überprüfen Sie Ihre Internetverbindung und Proxy-Einstellungen.

Leerer oder hängender Bildschirm beim Start

Wenn die App öffnet, aber einen leeren oder nicht reagierenden Bildschirm anzeigt:

1. Starten Sie die App neu.
2. Überprüfen Sie auf ausstehende Updates. Die App wird beim Start automatisch aktualisiert.
3. Überprüfen Sie unter Windows die Ereignisanzeige auf Absturzprotokolle unter **Windows-Protokolle** → **Anwendung**.

„Fehler beim Laden der Sitzung“

Wenn Sie `Failed to load session` sehen, existiert der ausgewählte Ordner möglicherweise nicht mehr, ein Git-Repository benötigt möglicherweise Git LFS, das nicht installiert ist, oder Dateiberechtigungen verhindern möglicherweise den Zugriff. Versuchen Sie, einen anderen Ordner auszuwählen oder die App neu zu starten.

Sitzung findet installierte Tools nicht

Wenn Claude Tools wie `npm`, `node` oder andere CLI-Befehle nicht finden kann, überprüfen Sie, dass die Tools in Ihrem regulären Terminal funktionieren, überprüfen Sie, dass Ihr Shell-Profil PATH ordnungsgemäß einrichtet, und starten Sie die Desktop-App neu, um Umgebungsvariablen neu zu laden.

Git- und Git LFS-Fehler

Unter Windows ist Git erforderlich, damit die Registerkarte „Code“ lokale Sitzungen startet. Wenn Sie „Git is required“ sehen, installieren Sie [Git für Windows](#) und starten Sie die App neu.

Wenn Sie „Git LFS is required by this repository but is not installed“ sehen, installieren Sie Git LFS von git-lfs.com, führen Sie `git lfs install` aus und starten Sie die App neu.

MCP-Server funktionieren nicht unter Windows

Wenn MCP-Server-Umschalter nicht reagieren oder Server unter Windows keine Verbindung herstellen, überprüfen Sie, dass der Server ordnungsgemäß in Ihren Einstellungen konfiguriert ist, starten Sie die App neu, überprüfen Sie, dass der Server-Prozess im Task Manager läuft, und überprüfen Sie Server-Protokolle auf Verbindungsfehler.

App wird nicht beendet

- **macOS:** drücken Sie `Cmd+Q`. Wenn die App nicht reagiert, verwenden Sie Force Quit mit `Cmd+Option+Esc`, wählen Sie Claude und klicken Sie auf Force Quit.
- **Windows:** verwenden Sie Task Manager mit `Strg+Umschalt+Esc`, um den Claude-Prozess zu beenden.

Windows-spezifische Probleme

- **PATH nicht aktualisiert nach Installation:** Öffnen Sie ein neues Terminal-Fenster. PATH-Updates gelten nur für neue Terminal-Sitzungen.
- **Fehler bei gleichzeitiger Installation:** Wenn Sie einen Fehler über eine andere Installation sehen, die läuft, aber es gibt keine, versuchen Sie, das Installationsprogramm als Administrator auszuführen.
- **ARM64:** Windows ARM64-Geräte werden vollständig unterstützt.

Registerkarte „Cowork“ nicht verfügbar auf Intel Macs

Die Registerkarte „Cowork“ erfordert Apple Silicon (M1 oder später) auf macOS. Unter Windows ist Cowork auf allen unterstützten Hardware verfügbar. Die Registerkarten „Chat“ und „Code“ funktionieren normalerweise auf Intel Macs.

„Branch existiert noch nicht“ beim Öffnen in CLI

Remote-Sitzungen können Branches erstellen, die auf Ihrem lokalen Computer nicht existieren. Klicken Sie auf den Branch-Namen in der Sitzungs-Symbolleiste, um ihn zu kopieren, und rufen Sie ihn dann lokal ab:

```
git fetch origin <branch-name>
git checkout <branch-name>
```

Immer noch stecken?

- Suchen Sie oder melden Sie einen Fehler auf [GitHub Issues](#)
- Besuchen Sie das [Claude Support Center](#)

Wenn Sie einen Fehler melden, geben Sie Ihre Desktop-App-Version, Ihr Betriebssystem, die genaue Fehlermeldung und relevante Protokolle an. Überprüfen Sie auf macOS Console.app. Überprüfen Sie unter Windows Event Viewer → Windows-Protokolle → Anwendung.

Claude Code mit Chrome verwenden (Beta)

Verbinden Sie Claude Code mit Ihrem Chrome-Browser, um Web-Apps zu testen, mit Konsolenprotokollen zu debuggen, Formularausfüllungen zu automatisieren und Daten von Webseiten zu extrahieren.

Claude Code integriert sich mit der Claude in Chrome Browser-Erweiterung, um Ihnen Browser-Automatisierungsfunktionen über die CLI oder die [VS Code-Erweiterung](#) bereitzustellen. Erstellen Sie Ihren Code und testen und debuggen Sie ihn dann im Browser, ohne den Kontext zu wechseln.

Claude öffnet neue Registerkarten für Browser-Aufgaben und teilt den Anmeldestatus Ihres Browsers, sodass er auf alle Websites zugreifen kann, bei denen Sie bereits angemeldet sind. Browser-Aktionen werden in Echtzeit in einem sichtbaren Chrome-Fenster ausgeführt. Wenn Claude auf eine Anmeldeseite oder ein CAPTCHA trifft, wird es angehalten und fordert Sie auf, es manuell zu bearbeiten.

Note:

Die Chrome-Integration befindet sich in der Beta-Phase und funktioniert derzeit nur mit Google Chrome. Sie wird noch nicht auf Brave, Arc oder anderen Chromium-basierten Browsern unterstützt. WSL (Windows Subsystem for Linux) wird ebenfalls nicht unterstützt.

Funktionen

Mit verbundenem Chrome können Sie Browser-Aktionen mit Codierungsaufgaben in einem einzigen Workflow verketten:

- **Live-Debugging:** Lesen Sie Konsolenfehler und DOM-Status direkt aus und beheben Sie dann den Code, der sie verursacht hat
- **Design-Verifizierung:** Erstellen Sie eine Benutzeroberfläche aus einem Figma-Mock und öffnen Sie sie dann im Browser, um zu überprüfen, ob sie übereinstimmt
- **Web-App-Tests:** Testen Sie die Formularvalidierung, überprüfen Sie auf visuelle Regressionen oder überprüfen Sie Benutzerflüsse
- **Authentifizierte Web-Apps:** Interagieren Sie mit Google Docs, Gmail, Notion oder einer beliebigen App, bei der Sie angemeldet sind, ohne API-Konnektoren

- **Datenextraktion:** Extrahieren Sie strukturierte Informationen von Webseiten und speichern Sie sie lokal
- **Task-Automatisierung:** Automatisieren Sie wiederholte Browser-Aufgaben wie Dateneingabe, Formularausfüllung oder Multi-Site-Workflows
- **Sitzungsaufzeichnung:** Zeichnen Sie Browser-Interaktionen als GIFs auf, um zu dokumentieren oder zu teilen, was passiert ist

Voraussetzungen

Bevor Sie Claude Code mit Chrome verwenden, benötigen Sie:

- [Google Chrome](#) Browser
- [Claude in Chrome-Erweiterung](#) Version 1.0.36 oder höher
- [Claude Code](#) Version 2.0.73 oder höher
- Einen direkten Anthropic-Plan (Pro, Max, Team oder Enterprise)

Note:

Die Chrome-Integration ist nicht über Drittanbieter wie Amazon Bedrock, Google Cloud Vertex AI oder Microsoft Foundry verfügbar. Wenn Sie Claude ausschließlich über einen Drittanbieter nutzen, benötigen Sie ein separates claude.ai-Konto, um diese Funktion zu verwenden.

Erste Schritte in der CLI

Step 1: Claude Code mit Chrome starten

Starten Sie Claude Code mit dem Flag `--chrome`:

```
claude --chrome
```

Sie können Chrome auch innerhalb einer bestehenden Sitzung aktivieren, indem Sie `/chrome` ausführen.

Step 2: Bitten Sie Claude, den Browser zu verwenden

Dieses Beispiel navigiert zu einer Seite, interagiert mit ihr und meldet, was es findet, alles von Ihrem Terminal oder Editor aus:

Go to `code.claude.com/docs`, click on the search box, type "hooks", and tell me what results appear

Führen Sie `/chrome` jederzeit aus, um den Verbindungsstatus zu überprüfen, Berechtigungen zu verwalten oder die Erweiterung erneut zu verbinden.

Für VS Code siehe [Browser-Automatisierung in VS Code](#).

Chrome standardmäßig aktivieren

Um zu vermeiden, dass Sie `--chrome` jede Sitzung übergeben müssen, führen Sie `/chrome` aus und wählen Sie „Standardmäßig aktiviert“.

In der [VS Code-Erweiterung](#) ist Chrome verfügbar, wenn die Chrome-Erweiterung installiert ist. Kein zusätzliches Flag ist erforderlich.

Note:

Das standardmäßige Aktivieren von Chrome in der CLI erhöht die Kontextnutzung, da Browser-Tools immer geladen werden. Wenn Sie eine erhöhte Kontextnutzung bemerken, deaktivieren Sie diese Einstellung und verwenden Sie `--chrome` nur bei Bedarf.

Verwalten Sie Website-Berechtigungen

Website-Berechtigungen werden von der Chrome-Erweiterung geerbt. Verwalten Sie Berechtigungen in den Einstellungen der Chrome-Erweiterung, um zu steuern, welche Websites Claude durchsuchen, anklicken und eingeben kann.

Beispiel-Workflows

Diese Beispiele zeigen häufige Möglichkeiten, Browser-Aktionen mit Codierungsaufgaben zu kombinieren. Führen Sie `/mcp` aus und wählen Sie `claude-in-chrome`, um die vollständige Liste der verfügbaren Browser-Tools anzuzeigen.

Testen Sie eine lokale Web-Anwendung

Wenn Sie eine Web-App entwickeln, bitten Sie Claude, zu überprüfen, ob Ihre Änderungen ordnungsgemäß funktionieren:

```
I just updated the login form validation. Can you open localhost:3000,
try submitting the form with invalid data, and check if the error
messages appear correctly?
```

Claude navigiert zu Ihrem lokalen Server, interagiert mit dem Formular und meldet, was es beobachtet.

Debuggen mit Konsolenprotokollen

Claude kann Konsolenausgaben lesen, um Probleme zu diagnostizieren. Teilen Sie Claude mit, welche Muster zu suchen sind, anstatt alle Konsolenausgaben anzufordern, da Protokolle ausführlich sein können:

```
Open the dashboard page and check the console for any errors when
the page loads.
```

Claude liest die Konsolenmeldungen und kann nach bestimmten Mustern oder Fehlertypen filtern.

Automatisieren Sie die Formularausfüllung

Beschleunigen Sie wiederholte Dateneingabeaufgaben:

```
I have a spreadsheet of customer contacts in contacts.csv. For each row,
go to the CRM at crm.example.com, click "Add Contact", and fill in the
name, email, and phone fields.
```

Claude liest Ihre lokale Datei, navigiert die Web-Schnittstelle und gibt die Daten für jeden Datensatz ein.

Entwurf von Inhalten in Google Docs

Verwenden Sie Claude, um direkt in Ihren Dokumenten zu schreiben, ohne API-Setup:

```
Draft a project update based on the recent commits and add it to my
Google Doc at docs.google.com/document/d/abc123
```

Claude öffnet das Dokument, klickt in den Editor und gibt den Inhalt ein. Dies funktioniert mit jeder Web-App, bei der Sie angemeldet sind: Gmail, Notion, Sheets und mehr.

Extrahieren Sie Daten von Webseiten

Extrahieren Sie strukturierte Informationen von Websites:

Go to the product listings page and extract the name, price, and availability for each item. Save the results as a CSV file.

Claude navigiert zur Seite, liest den Inhalt und kompiliert die Daten in ein strukturiertes Format.

Führen Sie Multi-Site-Workflows aus

Koordinieren Sie Aufgaben über mehrere Websites hinweg:

Check my calendar for meetings tomorrow, then for each meeting with an external attendee, look up their company website and add a note about what they do.

Claude arbeitet über Registerkarten hinweg, um Informationen zu sammeln und den Workflow abzuschließen.

Zeichnen Sie eine Demo-GIF auf

Erstellen Sie teilbare Aufzeichnungen von Browser-Interaktionen:

Record a GIF showing how to complete the checkout flow, from adding an item to the cart through to the confirmation page.

Claude zeichnet die Interaktionssequenz auf und speichert sie als GIF-Datei.

Fehlerbehebung

Erweiterung nicht erkannt

Wenn Claude Code „Chrome-Erweiterung nicht erkannt“ anzeigt:

1. Überprüfen Sie, ob die Chrome-Erweiterung in `chrome://extensions` installiert und aktiviert ist
2. Überprüfen Sie, ob Claude Code aktuell ist, indem Sie `claude --version` ausführen
3. Überprüfen Sie, ob Chrome ausgeführt wird

4. Führen Sie `/chrome` aus und wählen Sie „Erweiterung erneut verbinden“, um die Verbindung wiederherzustellen
5. Wenn das Problem weiterhin besteht, starten Sie sowohl Claude Code als auch Chrome neu

Wenn Sie die Chrome-Integration zum ersten Mal aktivieren, installiert Claude Code eine Konfigurationsdatei für den nativen Messaging-Host. Chrome liest diese Datei beim Start, daher sollten Sie Chrome neu starten, um die neue Konfiguration zu übernehmen, wenn die Erweiterung beim ersten Versuch nicht erkannt wird.

Wenn die Verbindung weiterhin fehlschlägt, überprüfen Sie, ob die Host-Konfigurationsdatei vorhanden ist unter:

- **macOS:** `~/Library/Application Support/Google/Chrome/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`
- **Linux:** `~/.config/google-chrome/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`
- **Windows:** Überprüfen Sie `HKCU\Software\Google\Chrome\NativeMessagingHosts\` in der Windows-Registrierung

Browser antwortet nicht

Wenn Claudes Browser-Befehle nicht mehr funktionieren:

1. Überprüfen Sie, ob ein modales Dialogfeld (Warnung, Bestätigung, Eingabeaufforderung) die Seite blockiert. JavaScript-Dialoge blockieren Browser-Ereignisse und verhindern, dass Claude Befehle empfängt. Schließen Sie das Dialogfeld manuell und teilen Sie Claude mit, dass es fortfahren soll.
2. Bitten Sie Claude, eine neue Registerkarte zu erstellen und es erneut zu versuchen
3. Starten Sie die Chrome-Erweiterung neu, indem Sie sie in `chrome://extensions` deaktivieren und erneut aktivieren

Verbindung wird während langer Sitzungen unterbrochen

Der Service Worker der Chrome-Erweiterung kann während längerer Sitzungen in den Leerlauf gehen, was die Verbindung unterbricht. Wenn Browser-Tools nach einer Inaktivitätsphase nicht mehr funktionieren, führen Sie `/chrome` aus und wählen Sie „Erweiterung erneut verbinden“.

Windows-spezifische Probleme

Unter Windows können folgende Probleme auftreten:

- **Named Pipe-Konflikte (EADDRINUSE):** Wenn ein anderer Prozess das gleiche Named Pipe verwendet, starten Sie Claude Code neu. Schließen Sie alle anderen Claude Code-Sitzungen, die möglicherweise Chrome verwenden.
- **Fehler beim nativen Messaging-Host:** Wenn der native Messaging-Host beim Start abstürzt, versuchen Sie, Claude Code neu zu installieren, um die Host-Konfiguration zu regenerieren.

Häufige Fehlermeldungen

Dies sind die am häufigsten auftretenden Fehler und wie man sie behebt:

Fehler	Ursache	Behebung
„Browser-Erweiterung ist nicht verbunden“	Der native Messaging-Host kann die Erweiterung nicht erreichen	Starten Sie Chrome und Claude Code neu und führen Sie dann <code>/chrome</code> aus, um die Verbindung wiederherzustellen
„Erweiterung nicht erkannt“	Chrome-Erweiterung ist nicht installiert oder deaktiviert	Installieren oder aktivieren Sie die Erweiterung in <code>chrome://extensions</code>
„Keine Registerkarte verfügbar“	Claude versuchte zu handeln, bevor eine Registerkarte bereit war	Bitten Sie Claude, eine neue Registerkarte zu erstellen und es erneut zu versuchen
„Empfänger existiert nicht“	Der Service Worker der Erweiterung ist in den Leerlauf gegangen	Führen Sie <code>/chrome</code> aus und wählen Sie „Erweiterung erneut verbinden“

Siehe auch

- [Claude Code in VS Code verwenden](#): Browser-Automatisierung in der VS Code-Erweiterung
- [CLI-Referenz](#): Befehlszeilenflags einschließlich `--chrome`
- [Häufige Workflows](#): Weitere Möglichkeiten zur Verwendung von Claude Code
- [Daten und Datenschutz](#): Wie Claude Code Ihre Daten verarbeitet

- [Erste Schritte mit Claude in Chrome](#): Vollständige Dokumentation für die Chrome-Erweiterung, einschließlich Verknüpfungen, Planung und Berechtigungen

Claude Code in Slack

Delegieren Sie Codierungsaufgaben direkt aus Ihrem Slack-Arbeitsbereich

Claude Code in Slack bringt die Leistung von Claude Code direkt in Ihren Slack-Arbeitsbereich. Wenn Sie `@Claude` mit einer Codierungsaufgabe erwähnen, erkennt Claude automatisch die Absicht und erstellt eine Claude Code-Sitzung im Web, sodass Sie Entwicklungsarbeiten delegieren können, ohne Ihre Team-Gespräche zu verlassen.

Diese Integration basiert auf der bestehenden Claude for Slack-App, fügt aber intelligentes Routing zu Claude Code im Web für codierungsbezogene Anfragen hinzu.

Anwendungsfälle

- **Fehleruntersuchung und -behebung:** Bitten Sie Claude, Fehler zu untersuchen und zu beheben, sobald sie in Slack-Kanälen gemeldet werden.
- **Schnelle Code-Reviews und Änderungen:** Lassen Sie Claude kleine Funktionen implementieren oder Code basierend auf Team-Feedback umgestalten.
- **Kollaboratives Debugging:** Wenn Team-Diskussionen wichtigen Kontext bieten (z. B. Fehlerreproduzierungen oder Benutzerberichte), kann Claude diese Informationen nutzen, um seinen Debugging-Ansatz zu informieren.
- **Parallele Aufgabenausführung:** Starten Sie Codierungsaufgaben in Slack, während Sie andere Arbeiten fortsetzen, und erhalten Sie Benachrichtigungen nach Abschluss.

Voraussetzungen

Bevor Sie Claude Code in Slack verwenden, stellen Sie sicher, dass Sie über Folgendes verfügen:

Anforderung	Details
Claude Plan	Pro, Max, Team oder Enterprise mit Claude Code-Zugriff (Premium-Plätze)
Claude Code im Web	Der Zugriff auf Claude Code im Web muss aktiviert sein
GitHub-Konto	Mit Claude Code im Web verbunden mit mindestens einem authentifizierten Repository

Anforderung	Details
Slack-Authentifizierung	Ihr Slack-Konto ist über die Claude-App mit Ihrem Claude-Konto verknüpft

Einrichten von Claude Code in Slack

Step 1: Installieren Sie die Claude-App in Slack

Ein Workspace-Administrator muss die Claude-App aus dem Slack App Marketplace installieren. Besuchen Sie den [Slack App Marketplace](#) und klicken Sie auf „Zu Slack hinzufügen“, um den Installationsprozess zu starten.

Step 2: Verbinden Sie Ihr Claude-Konto

Nach der Installation der App authentifizieren Sie Ihr individuelles Claude-Konto:

1. Öffnen Sie die Claude-App in Slack, indem Sie auf „Claude“ in Ihrem Apps-Bereich klicken
2. Navigieren Sie zur Registerkarte „App Home“
3. Klicken Sie auf „Verbinden“, um Ihr Slack-Konto mit Ihrem Claude-Konto zu verknüpfen
4. Schließen Sie den Authentifizierungsfluss in Ihrem Browser ab

Step 3: Konfigurieren Sie Claude Code im Web

Stellen Sie sicher, dass Ihr Claude Code im Web ordnungsgemäß konfiguriert ist:

- Besuchen Sie [claude.ai/code](#) und melden Sie sich mit dem gleichen Konto an, das Sie mit Slack verbunden haben
- Verbinden Sie Ihr GitHub-Konto, falls noch nicht geschehen
- Authentifizieren Sie mindestens ein Repository, mit dem Claude arbeiten soll

Step 4: Wählen Sie Ihren Routing-Modus

Nach dem Verbinden Ihrer Konten konfigurieren Sie, wie Claude Ihre Nachrichten in Slack verarbeitet. Navigieren Sie zur Claude App Home in Slack, um die Einstellung **Routing Mode** zu finden.

Modus	Verhalten
Nur Code	Claude leitet alle @mentions zu Claude Code-Sitzungen weiter. Am besten für Teams, die Claude in Slack ausschließlich für Entwicklungsaufgaben verwenden.

Mo- dus	Verhalten
Code + Chat	Claude analysiert jede Nachricht und leitet intelligent zwischen Claude Code (für Codierungsaufgaben) und Claude Chat (für Schreiben, Analyse und allgemeine Fragen) weiter. Am besten für Teams, die einen einzigen @Claude-Einstiegspunkt für alle Arten von Arbeiten möchten.

Note:

Im Code + Chat-Modus können Sie, wenn Claude eine Nachricht zu Chat leitet, aber Sie wollten eine Codierungssitzung, auf „Als Code erneut versuchen“ klicken, um stattdessen eine Claude Code-Sitzung zu erstellen. Wenn es zu Code weitergeleitet wird, aber Sie wollten eine Chat-Sitzung, können Sie diese Option in diesem Thread wählen.

Wie es funktioniert

Automatische Erkennung

Wenn Sie @Claude in einem Slack-Kanal oder Thread erwähnen, analysiert Claude automatisch Ihre Nachricht, um festzustellen, ob es sich um eine Codierungsaufgabe handelt. Wenn Claude Codierungsabsicht erkennt, leitet es Ihre Anfrage stattdessen zu Claude Code im Web weiter, anstatt als regulärer Chat-Assistent zu antworten.

Sie können Claude auch explizit anweisen, eine Anfrage als Codierungsaufgabe zu behandeln, auch wenn es diese nicht automatisch erkennt.

Note:

Claude Code in Slack funktioniert nur in Kanälen (öffentlich oder privat). Es funktioniert nicht in direkten Nachrichten (DMs).

Kontexterfassung

Aus Threads: Wenn Sie Claude in einem Thread @erwähnen, sammelt es Kontext aus allen Nachrichten in diesem Thread, um das vollständige Gespräch zu verstehen.

Aus Kanälen: Wenn es direkt in einem Kanal erwähnt wird, schaut Claude sich aktuelle Kanalnachrichten auf relevanten Kontext an.

Dieser Kontext hilft Claude, das Problem zu verstehen, das entsprechende Repository auszuwählen und seinen Ansatz zur Aufgabe zu informieren.

Warning:

Wenn @Claude in Slack aufgerufen wird, erhält Claude Zugriff auf den Gesprächskontext, um Ihre Anfrage besser zu verstehen. Claude kann Anweisungen aus anderen Nachrichten im Kontext befolgen, daher sollten Benutzer sicherstellen, dass sie Claude nur in vertrauenswürdigen Slack-Gesprächen verwenden.

Sitzungsfluss

1. **Initiierung:** Sie @erwähnen Claude mit einer Codierungsanfrage
2. **Erkennung:** Claude analysiert Ihre Nachricht und erkennt Codierungsabsicht
3. **Sitzungserstellung:** Eine neue Claude Code-Sitzung wird auf claude.ai/code erstellt
4. **Fortschritts-Updates:** Claude veröffentlicht Status-Updates in Ihrem Slack-Thread, während die Arbeit fortschreitet
5. **Abschluss:** Nach Abschluss @erwähnt Claude Sie mit einer Zusammenfassung und Aktionsschaltflächen
6. **Überprüfung:** Klicken Sie auf „Sitzung anzeigen“, um das vollständige Transkript zu sehen, oder auf „PR erstellen“, um einen Pull Request zu öffnen

Benutzeroberflächenelemente

App Home

Die Registerkarte „App Home“ zeigt Ihren Verbindungsstatus an und ermöglicht es Ihnen, Ihr Claude-Konto von Slack zu verbinden oder zu trennen.

Nachrichtenaktionen

- **Sitzung anzeigen:** Öffnet die vollständige Claude Code-Sitzung in Ihrem Browser, wo Sie alle durchgeführten Arbeiten sehen, die Sitzung fortsetzen oder zusätzliche Anfragen stellen können.
- **PR erstellen:** Erstellt einen Pull Request direkt aus den Änderungen der Sitzung.
- **Als Code erneut versuchen:** Wenn Claude zunächst als Chat-Assistent antwortet, aber Sie wollten eine Codierungssitzung, klicken Sie auf diese Schaltfläche, um die Anfrage als Claude Code-Aufgabe erneut zu versuchen.
- **Repository ändern:** Ermöglicht es Ihnen, ein anderes Repository auszuwählen, wenn Claude falsch gewählt hat.

Repository-Auswahl

Claude wählt automatisch ein Repository basierend auf dem Kontext aus Ihrem Slack-Gespräch aus. Wenn mehrere Repositories zutreffen könnten, zeigt Claude möglicherweise ein Dropdown-Menü an, mit dem Sie das richtige auswählen können.

Zugriff und Berechtigungen

Zugriff auf Benutzerebene

Zugriffstyp	Anforderung
Claude Code-Sitzungen	Jeder Benutzer führt Sitzungen unter seinem eigenen Claude-Konto aus
Nutzung & Ratenlimits	Sitzungen werden gegen die individuellen Plan-Limits des Benutzers angerechnet
Repository-Zugriff	Benutzer können nur auf Repositories zugreifen, die sie persönlich verbunden haben
Sitzungsverlauf	Sitzungen erscheinen in Ihrem Claude Code-Verlauf auf claude.ai/code

Berechtigungen für Workspace-Administratoren

Slack-Workspace-Administratoren kontrollieren, ob die Claude-App im Workspace installiert werden kann. Einzelne Benutzer authentifizieren sich dann mit ihren eigenen Claude-Konten, um die Integration zu verwenden.

Was wo zugänglich ist

In Slack: Sie sehen Status-Updates, Abschluss-Zusammenfassungen und Aktionsschaltflächen. Das vollständige Transkript wird beibehalten und ist immer zugänglich.

Im Web: Die vollständige Claude Code-Sitzung mit vollständiger Gesprächsverlauf, alle Code-Änderungen, Dateivorgänge und die Möglichkeit, die Sitzung fortzusetzen oder Pull Requests zu erstellen.

Best Practices

Schreiben effektiver Anfragen

- **Seien Sie spezifisch:** Geben Sie Dateinamen, Funktionsnamen oder Fehlermeldungen an, wenn relevant.

- **Geben Sie Kontext:** Erwähnen Sie das Repository oder Projekt, wenn es nicht aus dem Gespräch klar ist.
- **Definieren Sie Erfolg:** Erklären Sie, wie „fertig“ aussieht – sollte Claude Tests schreiben? Dokumentation aktualisieren? Einen PR erstellen?
- **Verwenden Sie Threads:** Antworten Sie in Threads, wenn Sie über Fehler oder Funktionen diskutieren, damit Claude den vollständigen Kontext sammeln kann.

Wann Slack vs. Web verwenden

Verwenden Sie Slack, wenn: Kontext bereits in einer Slack-Diskussion vorhanden ist, Sie eine Aufgabe asynchron starten möchten oder mit Teamkollegen zusammenarbeiten, die Sichtbarkeit benötigen.

Verwenden Sie das Web direkt, wenn: Sie Dateien hochladen müssen, Echtzeit-Interaktion während der Entwicklung möchten oder an längeren, komplexeren Aufgaben arbeiten.

Fehlerbehebung

Sitzungen starten nicht

1. Überprüfen Sie, ob Ihr Claude-Konto in der Claude App Home verbunden ist
2. Überprüfen Sie, ob Sie Claude Code im Web-Zugriff aktiviert haben
3. Stellen Sie sicher, dass Sie mindestens ein GitHub-Repository mit Claude Code verbunden haben

Repository wird nicht angezeigt

1. Verbinden Sie das Repository in Claude Code im Web unter claude.ai/code
2. Überprüfen Sie Ihre GitHub-Berechtigungen für dieses Repository
3. Versuchen Sie, Ihr GitHub-Konto zu trennen und erneut zu verbinden

Falsches Repository ausgewählt

1. Klicken Sie auf die Schaltfläche „Repository ändern“, um ein anderes Repository auszuwählen
2. Geben Sie den Repository-Namen in Ihrer Anfrage an, um eine genauere Auswahl zu erhalten

Authentifizierungsfehler

1. Trennen Sie Ihr Claude-Konto in der App Home und verbinden Sie es erneut

2. Stellen Sie sicher, dass Sie in Ihrem Browser mit dem richtigen Claude-Konto angemeldet sind
3. Überprüfen Sie, ob Ihr Claude-Plan Claude Code-Zugriff umfasst

Sitzungsablauf

1. Sitzungen bleiben in Ihrem Claude Code-Verlauf im Web zugänglich
2. Sie können vergangene Sitzungen von claude.ai/code aus fortsetzen oder referenzieren

Aktuelle Einschränkungen

- **Nur GitHub:** Unterstützt derzeit nur Repositories auf GitHub.
- **Ein PR gleichzeitig:** Jede Sitzung kann einen Pull Request erstellen.
- **Ratenlimits gelten:** Sitzungen verwenden die Ratenlimits Ihres individuellen Claude-Plans.
- **Web-Zugriff erforderlich:** Benutzer müssen Claude Code im Web-Zugriff haben; diejenigen ohne erhalten nur Standard-Claude-Chat-Antworten.

Verwandte Ressourcen

- [Claude Code im Web](#) Erfahren Sie mehr über Claude Code im Web
- [Claude für Slack](#) Allgemeine Claude for Slack-Dokumentation
- [Slack App Marketplace](#) Installieren Sie die Claude-App aus dem Slack Marketplace
- [Claude Help Center](#) Erhalten Sie zusätzliche Unterstützung

Claude Code im Web

Führen Sie Claude Code-Aufgaben asynchron auf sicherer Cloud-Infrastruktur aus

Note:

Claude Code im Web befindet sich derzeit in der Forschungsvorschau.

Was ist Claude Code im Web?

Claude Code im Web ermöglicht es Entwicklern, Claude Code aus der Claude-App zu starten. Dies ist perfekt für:

- **Fragen beantworten:** Fragen zur Code-Architektur und zur Implementierung von Funktionen stellen
- **Fehlerbehebung und Routineaufgaben:** Gut definierte Aufgaben, die keine häufige Steuerung erfordern
- **Parallele Arbeit:** Mehrere Fehlerbehebungen parallel durchführen
- **Repositories nicht auf Ihrem lokalen Computer:** Arbeiten Sie an Code, den Sie nicht lokal ausgecheckt haben
- **Backend-Änderungen:** Wo Claude Code Tests schreiben und dann Code schreiben kann, um diese Tests zu bestehen

Claude Code ist auch in der Claude-App für [iOS](#) und [Android](#) verfügbar, um Aufgaben unterwegs zu starten und laufende Arbeiten zu überwachen.

Sie können [neue Aufgaben im Web von Ihrem Terminal aus starten](#) mit `--remote`, oder [Web-Sitzungen zurück zu Ihrem Terminal teleportieren](#), um lokal fortzufahren. Um die Web-Oberfläche zu verwenden, während Sie Claude Code auf Ihrem eigenen Computer statt auf Cloud-Infrastruktur ausführen, siehe [Remote Control](#).

Wer kann Claude Code im Web nutzen?

Claude Code im Web ist in der Forschungsvorschau verfügbar für:

- **Pro-Benutzer**
- **Max-Benutzer**
- **Team-Benutzer**

- **Enterprise-Benutzer** mit Premium-Sitzen oder Chat + Claude Code-Sitzen

Erste Schritte

1. Besuchen Sie claude.ai/code
2. Verbinden Sie Ihr GitHub-Konto
3. Installieren Sie die Claude GitHub-App in Ihren Repositories
4. Wählen Sie Ihre Standardumgebung
5. Reichen Sie Ihre Coding-Aufgabe ein
6. Überprüfen Sie Änderungen in der Diff-Ansicht, iterieren Sie mit Kommentaren und erstellen Sie dann einen Pull Request

Wie es funktioniert

Wenn Sie eine Aufgabe in Claude Code im Web starten:

1. **Repository-Klonen:** Ihr Repository wird auf eine von Anthropic verwaltete virtuelle Maschine geklont
2. **Umgebungssetup:** Claude bereitet eine sichere Cloud-Umgebung mit Ihrem Code vor und führt dann Ihr [Setup-Skript](#) aus, falls konfiguriert
3. **Netzwerkconfiguration:** Der Internetzugriff wird basierend auf Ihren Einstellungen konfiguriert
4. **Aufgabenausführung:** Claude analysiert Code, nimmt Änderungen vor, führt Tests aus und überprüft seine Arbeit
5. **Fertigstellung:** Sie werden benachrichtigt, wenn die Aufgabe abgeschlossen ist, und können einen PR mit den Änderungen erstellen
6. **Ergebnisse:** Änderungen werden in einen Branch gepusht, bereit für die Pull Request-Erstellung

Überprüfen Sie Änderungen mit der Diff-Ansicht

Die Diff-Ansicht zeigt Ihnen genau, was Claude geändert hat, bevor Sie einen Pull Request erstellen. Anstatt auf „PR erstellen“ zu klicken, um Änderungen in GitHub zu überprüfen, sehen Sie den Diff direkt in der App und iterieren Sie mit Claude, bis die Änderungen bereit sind.

Wenn Claude Änderungen an Dateien vornimmt, wird ein Diff-Statistik-Indikator angezeigt, der die Anzahl der hinzugefügten und entfernten Zeilen anzeigt (z. B. **+12 -1**). Wählen Sie diesen Indikator, um den Diff-Viewer zu öffnen, der eine Dateiliste auf der linken Seite und die Änderungen für jede Datei auf der rechten Seite anzeigt.

Aus der Diff-Ansicht können Sie:

- Änderungen Datei für Datei überprüfen
- Kommentare zu bestimmten Änderungen hinterlassen, um Änderungen anzufordern
- Mit Claude basierend auf dem, was Sie sehen, weiter iterieren

Dies ermöglicht es Ihnen, Änderungen durch mehrere Runden von Feedback zu verfeinern, ohne Draft-PRs zu erstellen oder zu GitHub zu wechseln.

Verschieben von Aufgaben zwischen Web und Terminal

Sie können neue Aufgaben im Web von Ihrem Terminal aus starten oder Web-Sitzungen in Ihr Terminal ziehen, um lokal fortzufahren. Web-Sitzungen bleiben bestehen, auch wenn Sie Ihren Laptop schließen, und Sie können sie von überall aus überwachen, einschließlich der Claude Mobile-App.

Note:

Die Sitzungsübergabe ist unidirektional: Sie können Web-Sitzungen in Ihr Terminal ziehen, aber Sie können keine vorhandene Terminal-Sitzung ins Web verschieben. Das Flag `--remote` erstellt eine *neue* Web-Sitzung für Ihr aktuelles Repository.

Vom Terminal zum Web

Starten Sie eine Web-Sitzung von der Befehlszeile mit dem Flag `--remote` :

```
claude --remote "Fix the authentication bug in src/auth/login.ts"
```

Dies erstellt eine neue Web-Sitzung auf `claude.ai`. Die Aufgabe wird in der Cloud ausgeführt, während Sie lokal weiterarbeiten. Verwenden Sie `/tasks` , um den Fortschritt zu überprüfen, oder öffnen Sie die Sitzung auf `claude.ai` oder der Claude Mobile-App, um direkt zu interagieren. Von dort aus können Sie Claude steuern, Feedback geben oder Fragen beantworten, genau wie in jedem anderen Gespräch.

Tipps für Remote-Aufgaben

Planen Sie lokal, führen Sie remote aus: Für komplexe Aufgaben starten Sie Claude im Plan Mode, um den Ansatz zu besprechen, und senden Sie dann die Arbeit ins Web:

```
claude --permission-mode plan
```

Im Plan Mode kann Claude nur Dateien lesen und das Codebase erkunden. Sobald Sie mit dem Plan zufrieden sind, starten Sie eine Remote-Sitzung für autonome Ausführung:

```
claude --remote "Execute the migration plan in docs/migration-plan.md"
```

Dieses Muster gibt Ihnen Kontrolle über die Strategie, während Claude autonom in der Cloud ausgeführt wird.

Führen Sie Aufgaben parallel aus: Jeder `--remote`-Befehl erstellt seine eigene Web-Sitzung, die unabhängig ausgeführt wird. Sie können mehrere Aufgaben starten und sie werden alle gleichzeitig in separaten Sitzungen ausgeführt:

```
claude --remote "Fix the flaky test in auth.spec.ts"
claude --remote "Update the API documentation"
claude --remote "Refactor the logger to use structured output"
```

Überwachen Sie alle Sitzungen mit `/tasks`. Wenn eine Sitzung abgeschlossen ist, können Sie einen PR aus der Web-Oberfläche erstellen oder die [Sitzung teleportieren](#), um im Terminal fortzufahren.

Vom Web zum Terminal

Es gibt mehrere Möglichkeiten, eine Web-Sitzung in Ihr Terminal zu ziehen:

- **Mit `/teleport`**: Führen Sie innerhalb von Claude Code `/teleport` (oder `/tp`) aus, um eine interaktive Auswahl Ihrer Web-Sitzungen zu sehen. Wenn Sie nicht committe Änderungen haben, werden Sie aufgefordert, diese zuerst zu stashen.
- **Mit `--teleport`**: Führen Sie von der Befehlszeile `claude --teleport` für eine interaktive Sitzungsauswahl aus, oder `claude --teleport <session-id>`, um eine bestimmte Sitzung direkt fortzusetzen.
- **Von `/tasks`**: Führen Sie `/tasks` aus, um Ihre Hintergrund-Sitzungen zu sehen, drücken Sie dann `t`, um in eine zu teleportieren
- **Von der Web-Oberfläche**: Klicken Sie auf „Open in CLI“, um einen Befehl zu kopieren, den Sie in Ihr Terminal einfügen können

Wenn Sie eine Sitzung teleportieren, überprüft Claude, dass Sie sich im richtigen Repository befinden, ruft den Branch aus der Remote-Sitzung ab und checkt ihn aus, und lädt die vollständige Gesprächshistorie in Ihr Terminal.

Anforderungen für das Teleportieren

Teleport überprüft diese Anforderungen, bevor eine Sitzung fortgesetzt wird. Wenn eine Anforderung nicht erfüllt ist, sehen Sie einen Fehler oder werden aufgefordert, das Problem zu beheben.

Anfor- de- rung	Details
Saube- rer Git- Status	Ihr Arbeitsverzeichnis darf keine nicht committeten Änderungen haben. Teleport fordert Sie auf, Änderungen zu stashen, falls erforderlich.
Korrekt- es Re- pository	Sie müssen <code>--teleport</code> aus einem Checkout desselben Repositories ausführen, nicht aus einem Fork.
Branch verfüg- bar	Der Branch aus der Web-Sitzung muss in das Remote gepusht worden sein. Teleport ruft ihn automatisch ab und checkt ihn aus.
Gleiches Konto	Sie müssen sich bei demselben Claude.ai-Konto authentifizieren, das in der Web-Sitzung verwendet wurde.

Sitzungen teilen

Um eine Sitzung zu teilen, schalten Sie ihre Sichtbarkeit gemäß den folgenden Kontotypen um. Danach teilen Sie den Sitzungslink wie gewohnt. Empfänger, die Ihre gemeinsame Sitzung öffnen, sehen den neuesten Status der Sitzung beim Laden, aber die Seite des Empfängers wird nicht in Echtzeit aktualisiert.

Teilen von einem Enterprise- oder Teams-Konto

Für Enterprise- und Teams-Konten sind die beiden Sichtbarkeitsoptionen **Privat** und **Team**. Die Team-Sichtbarkeit macht die Sitzung für andere Mitglieder Ihrer Claude.ai-Organisation sichtbar. Die Überprüfung des Repository-Zugriffs ist standardmäßig aktiviert, basierend auf dem GitHub-Konto, das mit dem Konto des Empfängers verbunden ist. Der Anzeigename Ihres Kontos ist für alle Empfänger mit Zugriff sichtbar. [Claude in Slack](#)-Sitzungen werden automatisch mit Team-Sichtbarkeit geteilt.

Teilen von einem Max- oder Pro-Konto

Für Max- und Pro-Konten sind die beiden Sichtbarkeitsoptionen **Privat** und **Öffentlich**. Die öffentliche Sichtbarkeit macht die Sitzung für jeden Benutzer sichtbar, der bei claude.ai angemeldet ist.

Überprüfen Sie Ihre Sitzung auf sensible Inhalte, bevor Sie sie teilen. Sitzungen können Code und Anmeldedaten aus privaten GitHub-Repositories enthalten. Die Überprüfung des Repository-Zugriffs ist standardmäßig nicht aktiviert.

Aktivieren Sie die Überprüfung des Repository-Zugriffs und/oder halten Sie Ihren Namen in Ihren gemeinsamen Sitzungen zurück, indem Sie zu Einstellungen > Claude Code > Freigabeeinstellungen gehen.

Verwalten von Sitzungen

Archivieren von Sitzungen

Sie können Sitzungen archivieren, um Ihre Sitzungsliste organisiert zu halten. Archivierte Sitzungen sind in der Standard-Sitzungsliste ausgeblendet, können aber durch Filtern nach archivierten Sitzungen angezeigt werden.

Um eine Sitzung zu archivieren, bewegen Sie den Mauszeiger über die Sitzung in der Seitenleiste und klicken Sie auf das Archiv-Symbol.

Löschen von Sitzungen

Das Löschen einer Sitzung entfernt die Sitzung und ihre Daten dauerhaft. Diese Aktion kann nicht rückgängig gemacht werden. Sie können eine Sitzung auf zwei Arten löschen:

- **Von der Seitenleiste:** Filtern Sie nach archivierten Sitzungen, bewegen Sie dann den Mauszeiger über die Sitzung, die Sie löschen möchten, und klicken Sie auf das Lösch-Symbol
- **Vom Sitzungsmenü:** Öffnen Sie eine Sitzung, klicken Sie auf das Dropdown-Menü neben dem Sitzungstitel und wählen Sie **Löschen**

Sie werden aufgefordert, vor dem Löschen einer Sitzung zu bestätigen.

Cloud-Umgebung

Standardimage

Wir erstellen und verwalten ein universelles Image mit vorinstallierten gängigen Toolchains und Sprachökosystemen. Dieses Image enthält:

- Beliebte Programmiersprachen und Laufzeiten

- Gängige Build-Tools und Paketmanager
- Test-Frameworks und Linter

Überprüfung verfügbarer Tools

Um zu sehen, was in Ihrer Umgebung vorinstalliert ist, bitten Sie Claude Code, Folgendes auszuführen:

```
check-tools
```

Dieser Befehl zeigt:

- Programmiersprachen und ihre Versionen
- Verfügbare Paketmanager
- Installierte Entwicklungstools

Sprachspezifische Setups

Das universelle Image enthält vorkonfigurierte Umgebungen für:

- **Python:** Python 3.x mit pip, poetry und gängigen wissenschaftlichen Bibliotheken
- **Node.js:** Neueste LTS-Versionen mit npm, yarn, pnpm und bun
- **Ruby:** Versionen 3.1.6, 3.2.6, 3.3.6 (Standard: 3.3.6) mit gem, bundler und rbenv für Versionsverwaltung
- **PHP:** Version 8.4.14
- **Java:** OpenJDK mit Maven und Gradle
- **Go:** Neueste stabile Version mit Modulunterstützung
- **Rust:** Rust-Toolchain mit cargo
- **C++:** GCC- und Clang-Compiler

Datenbanken

Das universelle Image enthält die folgenden Datenbanken:

- **PostgreSQL:** Version 16
- **Redis:** Version 7.0

Umgebungsconfiguration

Wenn Sie eine Sitzung in Claude Code im Web starten, passiert folgendes im Hintergrund:

1. **Umgebungsvorbereitung:** Wir klonen Ihr Repository und führen alle konfigurierten [Setup-Skripte](#) aus. Das Repo wird mit dem Standard-Branch Ihres GitHub-Repos

geklont. Wenn Sie einen bestimmten Branch auschecken möchten, können Sie dies in der Eingabeaufforderung angeben.

2. **Netzwerkconfiguration:** Wir konfigurieren den Internetzugriff für den Agent. Der Internetzugriff ist standardmäßig begrenzt, aber Sie können die Umgebung so konfigurieren, dass sie keinen Internetzugriff oder vollständigen Internetzugriff hat, je nach Ihren Anforderungen.
3. **Claude Code-Ausführung:** Claude Code wird ausgeführt, um Ihre Aufgabe zu erfüllen, Code zu schreiben, Tests auszuführen und seine Arbeit zu überprüfen. Sie können Claude während der Sitzung über die Web-Oberfläche steuern und lenken. Claude respektiert den Kontext, den Sie in Ihrer `CLAUDE.md` definiert haben.
4. **Ergebnis:** Wenn Claude seine Arbeit abgeschlossen hat, pusht es den Branch zu Remote. Sie können einen PR für den Branch erstellen.

Note:

Claude arbeitet vollständig über das Terminal und die CLI-Tools, die in der Umgebung verfügbar sind. Es verwendet die vorinstallierten Tools im universellen Image und alle zusätzlichen Tools, die Sie über hooks oder Abhängigkeitsverwaltung installieren.

Um eine neue Umgebung hinzuzufügen: Wählen Sie die aktuelle Umgebung, um die Umgebungsauswahl zu öffnen, und wählen Sie dann „Umgebung hinzufügen“. Dies öffnet einen Dialog, in dem Sie den Umgebungsnamen, die Netzwerkzugriffsstufe, Umgebungsvariablen und ein [Setup-Skript](#) angeben können.

Um eine vorhandene Umgebung zu aktualisieren: Wählen Sie die aktuelle Umgebung, rechts neben dem Umgebungsnamen, und wählen Sie die Schaltfläche Einstellungen. Dies öffnet einen Dialog, in dem Sie den Umgebungsnamen, den Netzwerkzugriff, Umgebungsvariablen und das Setup-Skript aktualisieren können.

Um Ihre Standardumgebung vom Terminal aus auszuwählen: Wenn Sie mehrere Umgebungen konfiguriert haben, führen Sie `/remote-env` aus, um auszuwählen, welche verwendet werden soll, wenn Sie Web-Sitzungen von Ihrem Terminal mit `--remote` starten. Mit einer einzelnen Umgebung zeigt dieser Befehl Ihre aktuelle Konfiguration an.

Note:

Umgebungsvariablen müssen als Schlüssel-Wert-Paare im `.env` [-Format](#) angegeben werden. Zum Beispiel:

```
API_KEY=your_api_key
DEBUG=true
```

Setup-Skripte

Ein Setup-Skript ist ein Bash-Skript, das ausgeführt wird, wenn eine neue Cloud-Sitzung startet, bevor Claude Code startet. Verwenden Sie Setup-Skripte, um Abhängigkeiten zu installieren, Tools zu konfigurieren oder alles vorzubereiten, das die Cloud-Umgebung benötigt und nicht im [Standardimage](#) enthalten ist.

Skripte werden als Root auf Ubuntu 24.04 ausgeführt, daher funktionieren `apt install` und die meisten Sprachpaketmanager.

Tip:

Um zu überprüfen, was bereits installiert ist, bevor Sie es zu Ihrem Skript hinzufügen, bitten Sie Claude, `check-tools` in einer Cloud-Sitzung auszuführen.

Um ein Setup-Skript hinzuzufügen, öffnen Sie den Dialog Umgebungseinstellungen und geben Sie Ihr Skript in das Feld **Setup-Skript** ein.

Dieses Beispiel installiert die `gh` CLI, die nicht im Standardimage enthalten ist:

```
#!/bin/bash
apt update && apt install -y gh
```

Setup-Skripte werden nur beim Erstellen einer neuen Sitzung ausgeführt. Sie werden übersprungen, wenn eine vorhandene Sitzung fortgesetzt wird.

Wenn das Skript mit einem Nicht-Null-Wert beendet wird, schlägt die Sitzung fehl zu starten. Fügen Sie `|| true` an nicht kritische Befehle an, um zu vermeiden, dass die Sitzung bei einer fehlerhaften Installation blockiert wird.

Note:

Setup-Skripte, die Pakete installieren, benötigen Netzwerkzugriff, um Registries zu erreichen. Der Standard-Netzwerkzugriff ermöglicht Verbindungen zu [gängigen Paketregistries](#), einschließlich npm, PyPI, RubyGems und crates.io. Skripte schlagen fehl, Pakete zu installieren, wenn Ihre Umgebung Netzwerkzugriff deaktiviert hat.

Setup-Skripte vs. SessionStart-Hooks

Verwenden Sie ein Setup-Skript, um Dinge zu installieren, die die Cloud benötigt, aber Ihr Laptop bereits hat, wie eine Sprachlaufzeit oder ein CLI-Tool. Verwenden Sie einen [SessionStart-Hook](#) für Projekt-Setup, das überall ausgeführt werden sollte, Cloud und lokal, wie `npm install`.

Beide werden am Anfang einer Sitzung ausgeführt, aber sie gehören an verschiedene Orte:

	Setup-Skripte	SessionStart-Hooks
Angehängt an	Die Cloud-Umgebung	Ihr Repository
Konfiguriert in	Cloud-Umgebungs-UI	<code>.claude/settings.json</code> in Ihrem Repo
Wird ausgeführt	Bevor Claude Code startet, nur bei neuen Sitzungen	Nach Claude Code startet, bei jeder Sitzung einschließlich fortgesetzter
Umfang	Nur Cloud-Umgebungen	Sowohl lokal als auch Cloud

SessionStart-Hooks können auch in Ihrer Benutzer-Level-Datei `~/.claude/settings.json` lokal definiert werden, aber Benutzer-Level-Einstellungen werden nicht zu Cloud-Sitzungen übertragen. In der Cloud werden nur Hooks ausgeführt, die zum Repo committed sind.

Abhängigkeitsverwaltung

Benutzerdefinierte Umgebungsimages und Snapshots werden noch nicht unterstützt. Verwenden Sie [Setup-Skripte](#), um Pakete zu installieren, wenn eine Sitzung startet, oder [SessionStart-Hooks](#) für die Abhängigkeitsinstallation, die auch in lokalen Umgebungen ausgeführt werden sollte. SessionStart-Hooks haben [bekannte Einschränkungen](#).

Um die automatische Abhängigkeitsinstallation mit einem Setup-Skript zu konfigurieren, öffnen Sie Ihre Umgebungseinstellungen und fügen Sie ein Skript hinzu:

```
#!/bin/bash
npm install
pip install -r requirements.txt
```


Alternativ können Sie SessionStart-Hooks in der Datei `.claude/settings.json` Ihres Repositories für die Abhängigkeitsinstallation verwenden, die auch in lokalen Umgebungen ausgeführt werden sollte:

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/scripts/install_pkgs.sh"
          }
        ]
      }
    ]
  }
}
```

Erstellen Sie das entsprechende Skript unter `scripts/install_pkgs.sh`:

```
#!/bin/bash

## Only run in remote environments
if [ "$CLAUDE_CODE_REMOTE" ≠ "true" ]; then
  exit 0
fi

npm install
pip install -r requirements.txt
exit 0
```

Machen Sie es ausführbar: `chmod +x scripts/install_pkgs.sh`

Umgebungsvariablen beibehalten

SessionStart-Hooks können Umgebungsvariablen für nachfolgende Bash-Befehle beibehalten, indem sie in die Datei schreiben, die in der Umgebungsvariablen

`CLAUDE_ENV_FILE` angegeben ist. Weitere Informationen finden Sie unter [SessionStart-Hooks](#) in der Hooks-Referenz.

Abhängigkeitsverwaltung Einschränkungen

- **Hooks werden für alle Sitzungen ausgelöst:** SessionStart-Hooks werden in lokalen und Remote-Umgebungen ausgeführt. Es gibt keine Hook-Konfiguration, um einen Hook nur auf Remote-Sitzungen zu beschränken. Um die lokale Ausführung zu überspringen, überprüfen Sie die Umgebungsvariable `CLAUDE_CODE_REMOTE` in Ihrem Skript wie oben gezeigt.
- **Erfordert Netzwerkzugriff:** Installationsbefehle benötigen Netzwerkzugriff, um Paketregistries zu erreichen. Wenn Ihre Umgebung mit „Kein Internet“-Zugriff konfiguriert ist, schlagen diese Hooks fehl. Verwenden Sie „Begrenzt“ (Standard) oder „Vollständig“ Netzwerkzugriff. Die [Standard-Allowlist](#) enthält gängige Registries wie npm, PyPI, RubyGems und crates.io.
- **Proxy-Kompatibilität:** Der gesamte ausgehende Datenverkehr in Remote-Umgebungen läuft durch einen [Sicherheits-Proxy](#). Einige Paketmanager funktionieren mit diesem Proxy nicht korrekt. Bun ist ein bekanntes Beispiel.
- **Wird bei jedem Sitzungsstart ausgeführt:** Hooks werden jedes Mal ausgeführt, wenn eine Sitzung startet oder fortgesetzt wird, was die Startup-Latenz erhöht. Halten Sie Installationsskripte schnell, indem Sie überprüfen, ob Abhängigkeiten bereits vorhanden sind, bevor Sie sie neu installieren.

Netzwerkzugriff und Sicherheit

Netzwerkrichtlinie

GitHub-Proxy

Aus Sicherheitsgründen werden alle GitHub-Operationen über einen dedizierten Proxy-Service durchgeführt, der alle Git-Interaktionen transparent verwaltet. Innerhalb der Sandbox authentifiziert sich der Git-Client mit einem benutzerdefinierten Scoped-Credential. Dieser Proxy:

- Verwaltet GitHub-Authentifizierung sicher - der Git-Client verwendet ein Scoped-Credential innerhalb der Sandbox, das der Proxy überprüft und in Ihr tatsächliches GitHub-Authentifizierungstoken übersetzt

- Beschränkt Git-Push-Operationen auf den aktuellen Arbeitsbranch aus Sicherheitsgründen
- Ermöglicht nahtloses Klonen, Abrufen und PR-Operationen bei Beibehaltung von Sicherheitsgrenzen

Sicherheits-Proxy

Umgebungen werden aus Sicherheits- und Missbrauchspräventionsgründen hinter einem HTTP/HTTPS-Netzwerk-Proxy ausgeführt. Der gesamte ausgehende Internetdatenverkehr läuft durch diesen Proxy, der Folgendes bietet:

- Schutz vor böswilligen Anfragen
- Ratenbegrenzung und Missbrauchsprävention
- Inhaltsfilterung für erhöhte Sicherheit

Zugriffsstufen

Standardmäßig ist der Netzwerkzugriff auf [Allowlist-Domains](#) begrenzt.

Sie können benutzerdefinierten Netzwerkzugriff konfigurieren, einschließlich Deaktivierung des Netzwerkzugriffs.

Standard-Allowlist-Domains

Bei Verwendung von „Begrenzt“ Netzwerkzugriff sind die folgenden Domains standardmäßig zulässig:

Anthropic-Services

- api.anthropic.com
- statsig.anthropic.com
- platform.claude.com
- code.claude.com
- claude.ai

Versionskontrolle

- github.com
- [www.github.com](#)
- api.github.com
- npm.pkg.github.com
- raw.githubusercontent.com
- pkg-npm.githubusercontent.com

- objects.githubusercontent.com
- codeload.github.com
- avatars.githubusercontent.com
- camo.githubusercontent.com
- gist.github.com
- gitlab.com
- www.gitlab.com
- registry.gitlab.com
- bitbucket.org
- www.bitbucket.org
- api.bitbucket.org

Container-Registries

- registry-1.docker.io
- auth.docker.io
- index.docker.io
- hub.docker.com
- www.docker.com
- production.cloudflare.docker.com
- download.docker.com
- gcr.io
- *.gcr.io
- ghcr.io
- mcr.microsoft.com
- *.data.mcr.microsoft.com
- public.ecr.aws

Cloud-Plattformen

- cloud.google.com
- accounts.google.com
- gcloud.google.com
- *.googleapis.com
- storage.googleapis.com
- compute.googleapis.com

- container.googleapis.com
- azure.com
- portal.azure.com
- microsoft.com
- www.microsoft.com
- *.microsoftonline.com
- packages.microsoft.com
- dotnet.microsoft.com
- dot.net
- visualstudio.com
- dev.azure.com
- *.amazonaws.com
- *.api.aws
- oracle.com
- www.oracle.com
- java.com
- www.java.com
- java.net
- www.java.net
- download.oracle.com
- yum.oracle.com

Paketmanager - JavaScript/Node

- registry.npmjs.org
- www.npmjs.com
- www.npmjs.org
- npmjs.com
- npmjs.org
- yarnpkg.com
- registry.yarnpkg.com

Paketmanager - Python

- pypi.org
- www.pypi.org

- files.pythonhosted.org
- pythonhosted.org
- test.pypi.org
- pypi.python.org
- pypa.io
- www.pypa.io

Paketmanager - Ruby

- rubygems.org
- www.rubygems.org
- api.rubygems.org
- index.rubygems.org
- ruby-lang.org
- www.ruby-lang.org
- rubyforge.org
- www.rubyforge.org
- rubyonrails.org
- www.rubyonrails.org
- rvm.io
- get.rvm.io

Paketmanager - Rust

- crates.io
- www.crates.io
- index.crates.io
- static.crates.io
- rustup.rs
- static.rust-lang.org
- www.rust-lang.org

Paketmanager - Go

- proxy.golang.org
- sum.golang.org
- index.golang.org

- golang.org
- www.golang.org
- goproxy.io
- pkg.go.dev

Paketmanager - JVM

- maven.org
- repo.maven.org
- central.maven.org
- repo1.maven.org
- jcenter.bintray.com
- gradle.org
- www.gradle.org
- services.gradle.org
- plugins.gradle.org
- kotlin.org
- www.kotlin.org
- spring.io
- repo.spring.io

Paketmanager - Andere Sprachen

- packagist.org (PHP Composer)
- www.packagist.org
- repo.packagist.org
- nuget.org (.NET NuGet)
- www.nuget.org
- api.nuget.org
- pub.dev (Dart/Flutter)
- api.pub.dev
- hex.pm (Elixir/Erlang)
- www.hex.pm
- cpan.org (Perl CPAN)
- www.cpan.org
- metacpan.org

- www.metacpan.org
- api.metacpan.org
- cocoapods.org (iOS/macOS)
- www.cocoapods.org
- cdn.cocoapods.org
- haskell.org
- www.haskell.org
- hackage.haskell.org
- swift.org
- www.swift.org

Linux-Distributionen

- archive.ubuntu.com
- security.ubuntu.com
- ubuntu.com
- www.ubuntu.com
- *.ubuntu.com
- ppa.launchpad.net
- launchpad.net
- www.launchpad.net

Entwicklungstools & Plattformen

- dl.k8s.io (Kubernetes)
- pkgs.k8s.io
- k8s.io
- www.k8s.io
- releases.hashicorp.com (HashiCorp)
- apt.releases.hashicorp.com
- rpm.releases.hashicorp.com
- archive.releases.hashicorp.com
- hashicorp.com
- www.hashicorp.com
- repo.anaconda.com (Anaconda/Conda)
- conda.anaconda.org

- anaconda.org
- www.anaconda.com
- anaconda.com
- continuum.io
- apache.org (Apache)
- www.apache.org
- archive.apache.org
- downloads.apache.org
- eclipse.org (Eclipse)
- www.eclipse.org
- download.eclipse.org
- nodejs.org (Node.js)
- www.nodejs.org

Cloud-Services & Überwachung

- statsig.com
- www.statsig.com
- api.statsig.com
- sentry.io
- *.sentry.io
- http-intake.logs.datadoghq.com
- *.datadoghq.com
- *.datadoghq.eu

Content Delivery & Mirrors

- sourceforge.net
- *.sourceforge.net
- packagecloud.io
- *.packagecloud.io

Schema & Konfiguration

- json-schema.org
- www.json-schema.org
- json.schemastore.org

- www.schemastore.org

Model Context Protocol

- *.modelcontextprotocol.io

Note:

Domains, die mit * gekennzeichnet sind, zeigen Wildcard-Subdomain-Matching an. Zum Beispiel erlaubt *.gcr.io den Zugriff auf jede Subdomain von gcr.io.

Sicherheits-Best-Practices für benutzerdefinierten Netzwerkzugriff

1. **Prinzip der geringsten Berechtigung:** Aktivieren Sie nur den minimalen erforderlichen Netzwerkzugriff
2. **Regelmäßig überprüfen:** Überprüfen Sie zulässige Domains regelmäßig
3. **Verwenden Sie HTTPS:** Bevorzugen Sie immer HTTPS-Endpunkte gegenüber HTTP

Sicherheit und Isolation

Claude Code im Web bietet starke Sicherheitsgarantien:

- **Isolierte virtuelle Maschinen:** Jede Sitzung wird in einer isolierten, von Anthropic verwalteten VM ausgeführt
- **Netzwerkzugriffskontrolle:** Der Netzwerkzugriff ist standardmäßig begrenzt und kann deaktiviert werden

Note:

Wenn Claude Code mit deaktiviertem Netzwerkzugriff ausgeführt wird, darf Claude Code mit der Anthropic API kommunizieren, was möglicherweise immer noch ermöglicht, dass Daten die isolierte Claude Code VM verlassen.

- **Schutz von Anmeldedaten:** Sensible Anmeldedaten (wie Git-Anmeldedaten oder Signaturschlüssel) befinden sich niemals in der Sandbox mit Claude Code. Die Authentifizierung wird über einen sicheren Proxy mit Scoped-Credentials verwaltet
- **Sichere Analyse:** Code wird in isolierten VMs analysiert und geändert, bevor PRs erstellt werden

Preisgestaltung und Ratenlimits

Claude Code im Web teilt Ratenlimits mit allen anderen Claude- und Claude Code-Nutzungen in Ihrem Konto. Das Ausführen mehrerer Aufgaben parallel verbraucht proportional mehr Ratenlimits.

Einschränkungen

- **Repository-Authentifizierung:** Sie können Sitzungen nur vom Web zum lokalen Computer verschieben, wenn Sie sich bei demselben Konto authentifizieren
- **Plattformbeschränkungen:** Claude Code im Web funktioniert nur mit Code, der in GitHub gehostet wird. GitLab und andere Nicht-GitHub-Repositories können nicht mit Cloud-Sitzungen verwendet werden

Best Practices

1. **Automatisieren Sie das Umgebungssetup:** Verwenden Sie [Setup-Skripte](#), um Abhängigkeiten zu installieren und Tools zu konfigurieren, bevor Claude Code startet. Für fortgeschrittenere Szenarien konfigurieren Sie [SessionStart-Hooks](#).
2. **Dokumentieren Sie Anforderungen:** Geben Sie Abhängigkeiten und Befehle klar in Ihrer `CLAUDE.md`-Datei an. Wenn Sie eine `AGENTS.md`-Datei haben, können Sie sie in Ihrer `CLAUDE.md` mit `@AGENTS.md` sourcing verwenden, um eine einzige Quelle der Wahrheit zu erhalten.

Verwandte Ressourcen

- [Hooks-Konfiguration](#)
- [Einstellungsreferenz](#)
- [Sicherheit](#)
- [Datennutzung](#)

Part 9: Security & Privacy

Sicherheit

Erfahren Sie mehr über die Sicherheitsvorkehrungen von Claude Code und Best Practices für sichere Nutzung.

Wie wir Sicherheit angehen

Sicherheitsfundament

Die Sicherheit Ihres Codes ist von größter Bedeutung. Claude Code ist mit Sicherheit im Kern entwickelt worden, gemäß Anthropics umfassendem Sicherheitsprogramm. Erfahren Sie mehr und greifen Sie auf Ressourcen zu (SOC 2 Type 2 Bericht, ISO 27001 Zertifikat usw.) im [Anthropic Trust Center](#).

Berechtigungs-basierte Architektur

Claude Code verwendet standardmäßig strikte Nur-Lesen-Berechtigungen. Wenn zusätzliche Aktionen erforderlich sind (Dateien bearbeiten, Tests ausführen, Befehle ausführen), fordert Claude Code explizite Genehmigung an. Benutzer kontrollieren, ob sie Aktionen einmalig genehmigen oder automatisch zulassen möchten.

Wir haben Claude Code so gestaltet, dass es transparent und sicher ist. Beispielsweise erfordern wir Genehmigung für Bash-Befehle vor ihrer Ausführung, was Ihnen direkte Kontrolle gibt. Dieser Ansatz ermöglicht es Benutzern und Organisationen, Berechtigungen direkt zu konfigurieren.

Für detaillierte Berechtigungskonfiguration siehe [Berechtigungen](#).

Integrierte Schutzmaßnahmen

Um Risiken in agentengestützten Systemen zu mindern:

- **Sandbox-Bash-Tool:** [Sandbox](#) Bash-Befehle mit Dateisystem- und Netzwerkisolation, wodurch Berechtigungsaufforderungen reduziert werden, während die Sicherheit gewährleistet bleibt. Aktivieren Sie mit `/sandbox`, um Grenzen zu definieren, in denen Claude Code autonom arbeiten kann
- **Schreibzugriffsbeschränkung:** Claude Code kann nur in den Ordner schreiben, in dem es gestartet wurde, und in dessen Unterordner – es kann Dateien in übergeordnete

ten Verzeichnissen nicht ohne explizite Genehmigung ändern. Während Claude Code Dateien außerhalb des Arbeitsverzeichnisses lesen kann (nützlich für den Zugriff auf Systembibliotheken und Abhängigkeiten), sind Schreibvorgänge streng auf den Projektumfang beschränkt und schaffen eine klare Sicherheitsgrenze

- **Minderung von Genehmigungsmüdigkeit:** Unterstützung für das Zulassen häufig verwendeter sicherer Befehle pro Benutzer, pro Codebasis oder pro Organisation
- **Accept Edits-Modus:** Mehrere Bearbeitungen stapelweise akzeptieren, während Berechtigungsaufforderungen für Befehle mit Nebenwirkungen beibehalten werden

Benutzerverantwortung

Claude Code hat nur die Berechtigungen, die Sie ihm gewähren. Sie sind verantwortlich für die Überprüfung vorgeschlagener Code und Befehle auf Sicherheit vor der Genehmigung.

Schutz vor Prompt-Injection

Prompt-Injection ist eine Technik, bei der ein Angreifer versucht, die Anweisungen eines KI-Assistenten durch das Einfügen bössartiger Texte zu überschreiben oder zu manipulieren. Claude Code enthält mehrere Schutzmaßnahmen gegen diese Angriffe:

Kernschutzmaßnahmen

- **Berechtigungssystem:** Sensible Operationen erfordern explizite Genehmigung
- **Kontextbewusste Analyse:** Erkennt potenziell schädliche Anweisungen durch Analyse der vollständigen Anfrage
- **Eingabebereinigung:** Verhindert Befehlsinjektionen durch Verarbeitung von Benutzereingaben
- **Befehlsblockliste:** Blockiert standardmäßig riskante Befehle, die beliebige Inhalte aus dem Web abrufen, wie `curl` und `wget`. Wenn explizit zulässig, beachten Sie [Einschränkungen des Berechtigungsmusters](#)

Datenschutzvorkehrungen

Wir haben mehrere Schutzmaßnahmen implementiert, um Ihre Daten zu schützen, einschließlich:

- Begrenzte Aufbewahrungszeiträume für sensible Informationen (siehe [Privacy Center](#), um mehr zu erfahren)
- Eingeschränkter Zugriff auf Benutzersitzungsdaten
- Benutzerkontrolle über Datenschulungspräferenzen. Verbraucherbenutzer können ihre [DatenschutzEinstellungen](#) jederzeit ändern.

Für vollständige Details überprüfen Sie bitte unsere [Commercial Terms of Service](#) (für Team-, Enterprise- und API-Benutzer) oder [Consumer Terms](#) (für Free-, Pro- und Max-Benutzer) und [Privacy Policy](#).

Zusätzliche Schutzmaßnahmen

- **Genehmigung von Netzwerkanfragen:** Tools, die Netzwerkanfragen stellen, erfordern standardmäßig Benutzergenehmigung
- **Isolierte Kontextfenster:** Web Fetch verwendet ein separates Kontextfenster, um die Injection potenziell bösartiger Prompts zu vermeiden
- **Vertrauensüberprüfung:** Erste Codebasis-Ausführungen und neue MCP-Server erfordern Vertrauensüberprüfung
 - Hinweis: Vertrauensüberprüfung ist deaktiviert, wenn nicht-interaktiv mit dem `-p` -Flag ausgeführt wird
- **Erkennung von Befehlsinjektionen:** Verdächtige Bash-Befehle erfordern manuelle Genehmigung, auch wenn sie zuvor auf die Zulassungsliste gesetzt wurden
- **Fail-Closed-Matching:** Nicht übereinstimmende Befehle erfordern standardmäßig manuelle Genehmigung
- **Beschreibungen in natürlicher Sprache:** Komplexe Bash-Befehle enthalten Erklärungen zum Verständnis des Benutzers
- **Sichere Anmeldedatenspeicherung:** API-Schlüssel und Token sind verschlüsselt. Siehe [Credential Management](#)

Warning:

Windows WebDAV-Sicherheitsrisiko: Wenn Sie Claude Code unter Windows ausführen, empfehlen wir, WebDAV nicht zu aktivieren oder Claude Code keinen Zugriff auf Pfade wie `\*` zu gewähren, die WebDAV-Unterverzeichnisse enthalten können. [WebDAV wurde von Microsoft als veraltet eingestuft](#) aufgrund von Sicherheitsrisiken. Das Aktivieren von WebDAV kann Claude Code ermöglichen, Netzwerkanfragen an Remote-Hosts auszulösen und das Berechtigungssystem zu umgehen.

Best Practices für die Arbeit mit nicht vertrauenswürdigen Inhalten:

1. Überprüfen Sie vorgeschlagene Befehle vor der Genehmigung
2. Vermeiden Sie es, nicht vertrauenswürdige Inhalte direkt an Claude zu pipen
3. Überprüfen Sie vorgeschlagene Änderungen an kritischen Dateien
4. Verwenden Sie virtuelle Maschinen (VMs), um Skripte auszuführen und Tool-Aufrufe zu tätigen, besonders bei der Interaktion mit externen Webdiensten

5. Melden Sie verdächtiges Verhalten mit `/bug`

Warning:

Während diese Schutzmaßnahmen das Risiko erheblich reduzieren, ist kein System vollständig immun gegen alle Angriffe. Halten Sie immer gute Sicherheitspraktiken bei der Arbeit mit einem KI-Tool ein.

MCP-Sicherheit

Claude Code ermöglicht es Benutzern, Model Context Protocol (MCP)-Server zu konfigurieren. Die Liste der zulässigen MCP-Server wird in Ihrem Quellcode konfiguriert, als Teil der Claude Code-Einstellungen, die Ingenieure in die Versionskontrolle einchecken.

Wir ermutigen Sie, entweder Ihre eigenen MCP-Server zu schreiben oder MCP-Server von Anbietern zu verwenden, denen Sie vertrauen. Sie können Claude Code-Berechtigungen für MCP-Server konfigurieren. Anthropic verwaltet oder prüft keine MCP-Server.

IDE-Sicherheit

Siehe [VS Code-Sicherheit und Datenschutz](#) für weitere Informationen zum Ausführen von Claude Code in einer IDE.

Cloud-Ausführungssicherheit

Bei Verwendung von [Claude Code im Web](#) sind zusätzliche Sicherheitskontrollen vorhanden:

- **Isolierte virtuelle Maschinen:** Jede Cloud-Sitzung wird in einer isolierten, von Anthropic verwalteten VM ausgeführt
- **Netzwerkzugriffskontrollen:** Der Netzwerkzugriff ist standardmäßig begrenzt und kann so konfiguriert werden, dass er deaktiviert ist oder nur bestimmte Domänen zulässt
- **Anmeldedatenschutz:** Die Authentifizierung wird über einen sicheren Proxy durchgeführt, der einen scoped Credential in der Sandbox verwendet, der dann in Ihr tatsächliches GitHub-Authentifizierungstoken übersetzt wird
- **Branch-Einschränkungen:** Git-Push-Operationen sind auf den aktuellen Arbeitsbranch beschränkt
- **Audit-Protokollierung:** Alle Operationen in Cloud-Umgebungen werden zu Compliance- und Audit-Zwecken protokolliert

- **Automatische Bereinigung:** Cloud-Umgebungen werden nach Abschluss der Sitzung automatisch beendet

Weitere Details zur Cloud-Ausführung finden Sie unter [Claude Code im Web](#).

[Remote Control](#)-Sitzungen funktionieren anders: Die Weboberfläche verbindet sich mit einem Claude Code-Prozess, der auf Ihrem lokalen Computer ausgeführt wird. Alle Code-Ausführungen und Dateizugriffe bleiben lokal, und die gleichen Daten, die während einer lokalen Claude Code-Sitzung fließen, werden über die Anthropic API über TLS übertragen. Es sind keine Cloud-VMs oder Sandboxing beteiligt. Die Verbindung verwendet mehrere kurzlebige, eng begrenzte Anmeldedaten, die jeweils auf einen bestimmten Zweck beschränkt sind und unabhängig ablaufen, um den Blast-Radius eines einzelnen kompromittierten Credentials zu begrenzen.

Best Practices für Sicherheit

Arbeiten mit sensiblem Code

- Überprüfen Sie alle vorgeschlagenen Änderungen vor der Genehmigung
- Verwenden Sie projektspezifische Berechtigungseinstellungen für sensible Repositories
- Erwägen Sie die Verwendung von [devcontainers](#) für zusätzliche Isolierung
- Überprüfen Sie regelmäßig Ihre Berechtigungseinstellungen mit `/permissions`

Team-Sicherheit

- Verwenden Sie [verwaltete Einstellungen](#), um organisatorische Standards durchzusetzen
- Teilen Sie genehmigte Berechtigungskonfigurationen über Versionskontrolle
- Schulen Sie Teammitglieder in Best Practices für Sicherheit
- Überwachen Sie die Claude Code-Nutzung durch [OpenTelemetry-Metriken](#)
- Überprüfen oder blockieren Sie Einstellungsänderungen während Sitzungen mit `ConfigChange -Hooks`

Meldung von Sicherheitsproblemen

Wenn Sie eine Sicherheitslücke in Claude Code entdecken:

1. Offenbaren Sie sie nicht öffentlich
2. Melden Sie sie über unser [HackerOne-Programm](#)
3. Fügen Sie detaillierte Reproduktionsschritte ein
4. Geben Sie uns Zeit, das Problem zu beheben, bevor Sie es öffentlich offenbaren

Verwandte Ressourcen

- [Sandboxing](#) - Dateisystem- und Netzwerkisolation für Bash-Befehle
- [Berechtigungen](#) - Konfigurieren Sie Berechtigungen und Zugriffskontrolle
- [Nutzungsüberwachung](#) - Verfolgen und überprüfen Sie Claude Code-Aktivität
- [Entwicklungscontainer](#) - Sichere, isolierte Umgebungen
- [Anthropic Trust Center](#) - Sicherheitszertifizierungen und Compliance

Sandboxing

Erfahren Sie, wie das Sandboxing-Tool von Claude Code Dateisystem- und Netzwerkisolation für sicherere und autonomere Agent-Ausführung bietet.

Übersicht

Claude Code verfügt über natives Sandboxing, um eine sicherere Umgebung für die Agent-Ausführung bereitzustellen und die Notwendigkeit ständiger Genehmigungseingaben zu reduzieren. Anstatt für jeden Bash-Befehl eine Genehmigung zu erbitten, erstellt Sandboxing vordefinierte Grenzen, in denen Claude Code mit reduziertem Risiko freier arbeiten kann.

Das Sandboxing-Bash-Tool verwendet Betriebssystem-Primitive, um sowohl Dateisystem- als auch Netzwerkisolation durchzusetzen.

Warum Sandboxing wichtig ist

Die traditionelle genehmigungsbasierte Sicherheit erfordert ständige Benutzerbestätigung für Bash-Befehle. Während dies Kontrolle bietet, kann es zu Folgendem führen:

- **Genehmigungsmüdigkeit:** Wiederholtes Klicken auf „Genehmigen“ kann dazu führen, dass Benutzer weniger Aufmerksamkeit auf das legen, was sie genehmigen
- **Reduzierte Produktivität:** Ständige Unterbrechungen verlangsamen Entwicklungs-Workflows
- **Begrenzte Autonomie:** Claude Code kann nicht effizient arbeiten, wenn es auf Genehmigungen wartet

Sandboxing adressiert diese Herausforderungen durch:

1. **Klare Grenzen definieren:** Geben Sie genau an, auf welche Verzeichnisse und Netzwerk-Hosts Claude Code zugreifen kann
2. **Genehmigungseingaben reduzieren:** Sichere Befehle innerhalb der Sandbox erfordern keine Genehmigung
3. **Sicherheit beibehalten:** Versuche, auf Ressourcen außerhalb der Sandbox zuzugreifen, lösen sofortige Benachrichtigungen aus
4. **Autonomie ermöglichen:** Claude Code kann unabhängiger innerhalb definierter Grenzen arbeiten

Warning:

Effektives Sandboxing erfordert **sowohl** Dateisystem- als auch Netzwerkisolation. Ohne Netzwerkisolation könnte ein kompromittierter Agent sensible Dateien wie SSH-Schlüssel exfiltrieren. Ohne Dateisystem-Isolation könnte ein kompromittierter Agent System-Ressourcen manipulieren, um Netzwerkzugriff zu erlangen. Bei der Konfiguration von Sandboxing ist es wichtig sicherzustellen, dass Ihre konfigurierten Einstellungen keine Umgehungen in diesen Systemen schaffen.

Wie es funktioniert

Dateisystem-Isolation

Das Sandboxing-Bash-Tool beschränkt den Dateisystem-Zugriff auf spezifische Verzeichnisse:

- **Standard-Schreibverhalten:** Lese- und Schreibzugriff auf das aktuelle Arbeitsverzeichnis und seine Unterverzeichnisse
- **Standard-Leseverhalten:** Lesezugriff auf den gesamten Computer, außer bestimmten blockierten Verzeichnissen
- **Blockierter Zugriff:** Kann Dateien außerhalb des aktuellen Arbeitsverzeichnisses nicht ohne explizite Genehmigung ändern
- **Konfigurierbar:** Definieren Sie benutzerdefinierte zulässige und blockierte Pfade durch Einstellungen

Sie können Schreibzugriff auf zusätzliche Pfade mit `sandbox.filesystem.allowWrite` in Ihren Einstellungen gewähren. Diese Einschränkungen werden auf OS-Ebene durchgesetzt (Seatbelt auf macOS, bubblewrap auf Linux), daher gelten sie für alle Subprozess-Befehle, einschließlich Tools wie `kubectl`, `terraform` und `npm`, nicht nur für Claudes Datei-Tools.

Netzwerk-Isolation

Der Netzwerkzugriff wird durch einen Proxy-Server gesteuert, der außerhalb der Sandbox läuft:

- **Domain-Einschränkungen:** Nur genehmigte Domains können zugegriffen werden
- **Benutzerbestätigung:** Neue Domain-Anfragen lösen Genehmigungseingaben aus (es sei denn, `allowManagedDomainsOnly` ist aktiviert, was nicht zulässige Domains automatisch blockiert)
- **Benutzerdefinierte Proxy-Unterstützung:** Fortgeschrittene Benutzer können benutzerdefinierte Regeln für ausgehenden Datenverkehr implementieren

- **Umfassende Abdeckung:** Einschränkungen gelten für alle Skripte, Programme und Subprozesse, die durch Befehle erzeugt werden

OS-Level-Durchsetzung

Das Sandboxing-Bash-Tool nutzt Betriebssystem-Sicherheits-Primitive:

- **macOS:** Verwendet Seatbelt für Sandbox-Durchsetzung
- **Linux:** Verwendet [bubblewrap](#) für Isolation
- **WSL2:** Verwendet bubblewrap, wie Linux

WSL1 wird nicht unterstützt, da bubblewrap Kernel-Features erfordert, die nur in WSL2 verfügbar sind.

Diese OS-Level-Einschränkungen stellen sicher, dass alle Kindprozesse, die durch Claudes Befehle erzeugt werden, die gleichen Sicherheitsgrenzen erben.

Erste Schritte

Voraussetzungen

Auf **macOS** funktioniert Sandboxing sofort mit dem integrierten Seatbelt-Framework.

Auf **Linux und WSL2** installieren Sie zuerst die erforderlichen Pakete:

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

Sandboxing aktivieren

Sie können Sandboxing durch Ausführung des `/sandbox`-Befehls aktivieren:

```
/sandbox
```

Dies öffnet ein Menü, in dem Sie zwischen Sandbox-Modi wählen können. Wenn erforderliche Abhängigkeiten fehlen (wie `bubblewrap` oder `socat` auf Linux), zeigt das Menü Installationsanweisungen für Ihre Plattform an.

Sandbox-Modi

Claude Code bietet zwei Sandbox-Modi:

Auto-Allow-Modus: Bash-Befehle werden versuchen, innerhalb der Sandbox ausgeführt zu werden und sind automatisch zulässig, ohne dass eine Genehmigung erforderlich ist. Befehle, die nicht in der Sandbox ausgeführt werden können (wie solche, die Netzwerkzugriff auf nicht zulässige Hosts benötigen), fallen auf den regulären Genehmigungsfluss zurück. Explizite Ask/Deny-Regeln, die Sie konfiguriert haben, werden immer respektiert.

Regulärer Genehmigungsmodus: Alle Bash-Befehle durchlaufen den Standard-Genehmigungsfluss, auch wenn sie in der Sandbox ausgeführt werden. Dies bietet mehr Kontrolle, erfordert aber mehr Genehmigungen.

In beiden Modi erzwingt die Sandbox die gleichen Dateisystem- und Netzwerk-Einschränkungen. Der Unterschied liegt nur darin, ob Sandbox-Befehle automatisch genehmigt oder explizit genehmigt werden müssen.

Info:

Der Auto-Allow-Modus funktioniert unabhängig von Ihrer Genehmigungsmodus-Einstellung. Selbst wenn Sie sich nicht im „Bearbeitungen akzeptieren“-Modus befinden, werden Sandbox-Bash-Befehle automatisch ausgeführt, wenn Auto-Allow aktiviert ist. Dies bedeutet, dass Bash-Befehle, die Dateien innerhalb der Sandbox-Grenzen ändern, ohne Eingabeaufforderung ausgeführt werden, auch wenn Datei-Bearbeitungs-Tools normalerweise eine Genehmigung erfordern würden.

Sandboxing konfigurieren

Passen Sie das Sandbox-Verhalten durch Ihre `settings.json`-Datei an. Siehe [Einstellung](#) für eine vollständige Konfigurationsreferenz.

Schreibzugriff für Subprozesse auf spezifische Pfade gewähren

Standardmäßig können Sandbox-Befehle nur in das aktuelle Arbeitsverzeichnis schreiben. Wenn Subprozess-Befehle wie `kubectl`, `terraform` oder `npm` außerhalb des Projektverzeichnisses schreiben müssen, verwenden Sie `sandbox.filesystem.allowWrite`, um Zugriff auf spezifische Pfade zu gewähren:

```

{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "allowWrite": ["~/.kube", "//tmp/build"]
    }
  }
}

```

Diese Pfade werden auf OS-Ebene durchgesetzt, daher respektieren alle Befehle, die innerhalb der Sandbox ausgeführt werden, einschließlich ihrer Kindprozesse, diese. Dies ist der empfohlene Ansatz, wenn ein Tool Schreibzugriff auf einen bestimmten Ort benötigt, anstatt das Tool mit `excludedCommands` vollständig aus der Sandbox auszuschließen.

Wenn `allowWrite` (oder `denyWrite` / `denyRead`) in mehreren [Einstellungs-Scopes](#) definiert ist, werden die Arrays **zusammengeführt**, was bedeutet, dass Pfade aus jedem Scope kombiniert werden, nicht ersetzt. Wenn beispielsweise verwaltete Einstellungen Schreibvorgänge zu `//opt/company-tools` zulassen und ein Benutzer `~/.kube` in seinen persönlichen Einstellungen hinzufügt, sind beide Pfade in der endgültigen Sandbox-Konfiguration enthalten. Dies bedeutet, dass Benutzer und Projekte die Liste erweitern können, ohne Pfade zu duplizieren oder zu überschreiben, die durch höher priorisierte Scopes gesetzt sind.

Pfad-Präfixe steuern, wie Pfade aufgelöst werden:

Präfix	Bedeutung	Beispiel
<code>//</code>	Absoluter Pfad vom Dateisystem-Root	<code>//tmp/build</code> wird zu <code>/tmp/build</code>
<code>~/</code>	Relativ zum Home-Verzeichnis	<code>~/.kube</code> wird zu <code>\$HOME/.kube</code>
<code>/</code>	Relativ zum Verzeichnis der Einstellungsdatei	<code>/build</code> wird zu <code>\$SETTINGSDIR/build</code>
<code>./</code> oder kein Präfix	Relativer Pfad (aufgelöst durch Sandbox-Runtime)	<code>./output</code>

Sie können auch Schreib- oder Lesezugriff mit `sandbox.filesystem.denyWrite` und `sandbox.filesystem.denyRead` blockieren. Diese werden mit allen Pfaden aus `Edit(...)` und `Read(...)` Genehmigungsregeln zusammengeführt.

Tip:

Nicht alle Befehle sind sofort mit Sandboxing kompatibel. Einige Hinweise, die Ihnen helfen können, das Beste aus der Sandbox herauszuholen:

- Viele CLI-Tools erfordern Zugriff auf bestimmte Hosts. Wenn Sie diese Tools verwenden, werden sie um Genehmigung bitten, auf bestimmte Hosts zuzugreifen. Die Gewährung der Genehmigung ermöglicht ihnen, jetzt und in Zukunft auf diese Hosts zuzugreifen, was ihnen ermöglicht, sicher innerhalb der Sandbox ausgeführt zu werden.
- `watchman` ist nicht kompatibel mit der Ausführung in der Sandbox. Wenn Sie `jest` ausführen, erwägen Sie die Verwendung von `jest --no-watchman`
- `docker` ist nicht kompatibel mit der Ausführung in der Sandbox. Erwägen Sie, `docker` in `excludedCommands` anzugeben, um zu erzwingen, dass es außerhalb der Sandbox ausgeführt wird.

Note:

Claude Code enthält einen absichtlichen Escape-Hatch-Mechanismus, der es Befehlen ermöglicht, außerhalb der Sandbox ausgeführt zu werden, wenn nötig. Wenn ein Befehl aufgrund von Sandbox-Einschränkungen fehlschlägt (wie Netzwerkverbindungsprobleme oder inkompatible Tools), wird Claude aufgefordert, den Fehler zu analysieren und kann den Befehl mit dem `dangerouslyDisableSandbox`-Parameter erneut versuchen. Befehle, die diesen Parameter verwenden, durchlaufen den normalen Claude Code-Genehmigungsfluss, der Benutzererlaubnis zur Ausführung erfordert. Dies ermöglicht Claude Code, Grenzfälle zu handhaben, in denen bestimmte Tools oder Netzwerkoperationen nicht innerhalb von Sandbox-Einschränkungen funktionieren können.

Sie können diesen Escape-Hatch deaktivieren, indem Sie `"allowUnsandboxedCommands": false` in Ihren [Sandbox-Einstellungen](#) setzen. Wenn deaktiviert, wird der `dangerouslyDisableSandbox`-Parameter vollständig ignoriert und alle Befehle müssen in der Sandbox ausgeführt oder explizit in `excludedCommands` aufgelistet werden.

Sicherheitsvorteile

Schutz vor Prompt-Injection

Selbst wenn ein Angreifer Claude Code's Verhalten erfolgreich durch Prompt-Injection manipuliert, stellt die Sandbox sicher, dass Ihr System sicher bleibt:

Dateisystem-Schutz:

- Kann kritische Konfigurationsdateien wie `~/bashrc` nicht ändern
- Kann System-Level-Dateien in `/bin/` nicht ändern

- Kann Dateien nicht lesen, die in Ihren [Claude-Genehmigungseinstellungen](#) blockiert sind

Netzwerk-Schutz:

- Kann Daten nicht zu von Angreifern kontrollierten Servern exfiltrieren
- Kann böartige Skripte nicht von nicht autorisierten Domains herunterladen
- Kann keine unerwarteten API-Aufrufe an nicht genehmigten Diensten tätigen
- Kann keine Domains kontaktieren, die nicht explizit zulässig sind

Überwachung und Kontrolle:

- Alle Zugriffversuche außerhalb der Sandbox werden auf OS-Ebene blockiert
- Sie erhalten sofortige Benachrichtigungen, wenn Grenzen getestet werden
- Sie können wählen, zu verweigern, einmal zuzulassen oder Ihre Konfiguration dauerhaft zu aktualisieren

Reduzierte Angriffsfläche

Sandboxing begrenzt den potenziellen Schaden durch:

- **Bösartige Abhängigkeiten:** NPM-Pakete oder andere Abhängigkeiten mit schädlichem Code
- **Kompromitierte Skripte:** Build-Skripte oder Tools mit Sicherheitslücken
- **Social Engineering:** Angriffe, die Benutzer dazu verleiten, gefährliche Befehle auszuführen
- **Prompt-Injection:** Angriffe, die Claude dazu verleiten, gefährliche Befehle auszuführen

Transparente Bedienung

Wenn Claude Code versucht, auf Netzwerk-Ressourcen außerhalb der Sandbox zuzugreifen:

1. Der Vorgang wird auf OS-Ebene blockiert
2. Sie erhalten eine sofortige Benachrichtigung
3. Sie können wählen zu:
 - Die Anfrage verweigern
 - Sie einmal zulassen
 - Ihre Sandbox-Konfiguration aktualisieren, um sie dauerhaft zuzulassen

Sicherheitsbeschränkungen

- **Netzwerk-Sandboxing-Einschränkungen:** Das Netzwerk-Filtersystem funktioniert durch Einschränkung der Domains, mit denen Prozesse verbunden werden dürfen. Es inspiziert den Datenverkehr, der durch den Proxy fließt, nicht anderweitig, und Benutzer sind verantwortlich dafür, dass sie nur vertrauenswürdige Domains in ihrer Richtlinie zulassen.

Warning:

Benutzer sollten sich der potenziellen Risiken bewusst sein, die sich aus der Zulassung breiter Domains wie `github.com` ergeben, die Datenexfiltration ermöglichen können. Auch kann es in einigen Fällen möglich sein, das Netzwerk-Filtern durch `Domain Fronting` zu umgehen.

- **Privilege Escalation über Unix-Sockets:** Die `allowUnixSockets` -Konfiguration kann versehentlich Zugriff auf leistungsstarke System-Services gewähren, die zu Sandbox-Umgehungen führen könnten. Wenn sie beispielsweise verwendet wird, um Zugriff auf `/var/run/docker.sock` zu zulassen, würde dies effektiv Zugriff auf das Host-System durch Ausnutzung des Docker-Sockets gewähren. Benutzer werden ermutigt, sorgfältig zu überlegen, welche Unix-Sockets sie durch die Sandbox zulassen.
- **Dateisystem-Genehmigungseskalation:** Übermäßig breite Dateisystem-Schreibgenehmigungen können Privilege-Escalation-Angriffe ermöglichen. Das Zulassen von Schreibvorgängen zu Verzeichnissen, die ausführbare Dateien in `$PATH`, System-Konfigurationsverzeichnisse oder Benutzer-Shell-Konfigurationsdateien (`.bashrc`, `.zshrc`) enthalten, kann zu Code-Ausführung in verschiedenen Sicherheitskontexten führen, wenn andere Benutzer oder System-Prozesse auf diese Dateien zugreifen.
- **Linux-Sandbox-Stärke:** Die Linux-Implementierung bietet starke Dateisystem- und Netzwerk-Isolation, enthält aber einen `enableWeakerNestedSandbox` -Modus, der es ermöglicht, in Docker-Umgebungen ohne privilegierte Namespaces zu funktionieren. Diese Option schwächt die Sicherheit erheblich ab und sollte nur in Fällen verwendet werden, in denen zusätzliche Isolation anderweitig durchgesetzt wird.

Wie Sandboxing sich auf Genehmigungen bezieht

Sandboxing und [Genehmigungen](#) sind komplementäre Sicherheitsebenen, die zusammenarbeiten:

- **Genehmigungen** steuern, welche Tools Claude Code verwenden kann, und werden evaluiert, bevor ein Tool ausgeführt wird. Sie gelten für alle Tools: Bash, Read, Edit, WebFetch, MCP und andere.
- **Sandboxing** bietet OS-Level-Durchsetzung, die einschränkt, worauf Bash-Befehle auf Dateisystem- und Netzwerk-Ebene zugreifen können. Es gilt nur für Bash-Befehle und ihre Kindprozesse.

Dateisystem- und Netzwerk-Einschränkungen werden sowohl durch Sandbox-Einstellungen als auch durch Genehmigungsregeln konfiguriert:

- Verwenden Sie `sandbox.filesystem.allowWrite`, um Subprozess-Schreibzugriff auf Pfade außerhalb des Arbeitsverzeichnisses zu gewähren
- Verwenden Sie `sandbox.filesystem.denyWrite` und `sandbox.filesystem.denyRead`, um Subprozess-Zugriff auf spezifische Pfade zu blockieren
- Verwenden Sie `Read` und `Edit` Deny-Regeln, um Zugriff auf spezifische Dateien oder Verzeichnisse zu blockieren
- Verwenden Sie `WebFetch` Allow/Deny-Regeln, um Domain-Zugriff zu steuern
- Verwenden Sie Sandbox `allowedDomains`, um zu steuern, auf welche Domains Bash-Befehle zugreifen können

Pfade aus beiden `sandbox.filesystem`-Einstellungen und Genehmigungsregeln werden zusammengeführt in die endgültige Sandbox-Konfiguration.

Dieses [Repository](#) enthält Starter-Einstellungskonfigurationen für häufige Bereitstellungsszenarien, einschließlich Sandbox-spezifischer Beispiele. Verwenden Sie diese als Ausgangspunkte und passen Sie sie an Ihre Anforderungen an.

Erweiterte Verwendung

Benutzerdefinierte Proxy-Konfiguration

Für Organisationen, die erweiterte Netzwerk-Sicherheit benötigen, können Sie einen benutzerdefinierten Proxy implementieren, um:

- HTTPS-Datenverkehr zu entschlüsseln und zu inspizieren
- Benutzerdefinierte Filterregeln anzuwenden
- Alle Netzwerk-Anfragen zu protokollieren

- Mit bestehender Sicherheitsinfrastruktur zu integrieren

```
{
  "sandbox": {
    "network": {
      "httpProxyPort": 8080,
      "socksProxyPort": 8081
    }
  }
}
```

Integration mit bestehenden Sicherheits-Tools

Das Sandboxing-Bash-Tool funktioniert zusammen mit:

- **Genehmigungsregeln:** Kombinieren Sie mit [Genehmigungseinstellungen](#) für Defense-in-Depth
- **Entwicklungs-Container:** Verwenden Sie mit [devcontainers](#) für zusätzliche Isolation
- **Enterprise-Richtlinien:** Erzwingen Sie Sandbox-Konfigurationen durch [verwaltete Einstellungen](#)

Best Practices

1. **Beginnen Sie restriktiv:** Beginnen Sie mit minimalen Genehmigungen und erweitern Sie nach Bedarf
2. **Überwachen Sie Protokolle:** Überprüfen Sie Sandbox-Verletzungsversuche, um Claude Code's Anforderungen zu verstehen
3. **Verwenden Sie umgebungsspezifische Konfigurationen:** Unterschiedliche Sandbox-Regeln für Entwicklungs- vs. Produktionsumgebungen
4. **Kombinieren Sie mit Genehmigungen:** Verwenden Sie Sandboxing zusammen mit IAM-Richtlinien für umfassende Sicherheit
5. **Testen Sie Konfigurationen:** Überprüfen Sie, dass Ihre Sandbox-Einstellungen legitime Workflows nicht blockieren

Open Source

Die Sandbox-Runtime ist als Open-Source-npm-Paket für die Verwendung in Ihren eigenen Agent-Projekten verfügbar. Dies ermöglicht der breiteren AI-Agent-Community, sicherere und autonomere Systeme zu bauen. Dies kann auch verwendet werden, um andere Programme zu sandboxen, die Sie ausführen möchten. Um beispielsweise einen MCP-Server zu sandboxen, könnten Sie ausführen:

```
npx @anthropic-ai/sandbox-runtime <command-to-sandbox>
```

Für Implementierungsdetails und Quellcode besuchen Sie das [GitHub-Repository](#).

Einschränkungen

- **Performance-Overhead:** Minimal, aber einige Dateisystem-Operationen können leicht langsamer sein
- **Kompatibilität:** Einige Tools, die spezifische System-Zugriffsmuster erfordern, benötigen möglicherweise Konfigurationsanpassungen oder müssen sogar außerhalb der Sandbox ausgeführt werden
- **Plattform-Unterstützung:** Unterstützt macOS, Linux und WSL2. WSL1 wird nicht unterstützt. Native Windows-Unterstützung ist geplant.

Siehe auch

- [Sicherheit](#) - Umfassende Sicherheitsfeatures und Best Practices
- [Genehmigungen](#) - Genehmigungskonfiguration und Zugriffskontrolle
- [Einstellungen](#) - Vollständige Konfigurationsreferenz
- [CLI-Referenz](#) - Befehlszeilenoptionen

Checkpointing

Verfolgen Sie automatisch die Änderungen von Claude und machen Sie unerwünschte Änderungen schnell rückgängig.

Claude Code verfolgt automatisch die Dateiänderungen von Claude während der Arbeit und ermöglicht es Ihnen, Änderungen schnell rückgängig zu machen und zu vorherigen Zuständen zurückzukehren, wenn etwas schiefgeht.

Wie Checkpointing funktioniert

Während Sie mit Claude arbeiten, erfasst Checkpointing automatisch den Zustand Ihres Codes vor jeder Änderung. Dieses Sicherheitsnetz ermöglicht es Ihnen, ehrgeizige, großflächige Aufgaben zu verfolgen, da Sie immer zu einem vorherigen Code-Zustand zurückkehren können.

Automatische Verfolgung

Claude Code verfolgt alle Änderungen, die von seinen Datei-Bearbeitungswerkzeugen vorgenommen werden:

- Jede Benutzereingabe erstellt einen neuen Checkpoint
- Checkpoints bleiben über Sitzungen hinweg erhalten, sodass Sie auf sie in fortgesetzten Gesprächen zugreifen können
- Werden automatisch zusammen mit Sitzungen nach 30 Tagen bereinigt (konfigurierbar)

Änderungen rückgängig machen

Drücken Sie **Esc** zweimal (**Esc** + **Esc**) oder verwenden Sie den **/rewind** Befehl, um das Rewind-Menü zu öffnen. Sie können wählen, um wiederherzustellen:

- **Nur Gespräch:** Zurückspulen zu einer Benutzernachricht, während Code-Änderungen beibehalten werden
- **Nur Code:** Dateiänderungen rückgängig machen, während das Gespräch beibehalten wird
- **Sowohl Code als auch Gespräch:** Beide zu einem früheren Punkt in der Sitzung wiederherstellen

Häufige Anwendungsfälle

Checkpoints sind besonders nützlich, wenn:

- **Alternativen erkunden:** Versuchen Sie verschiedene Implementierungsansätze, ohne Ihren Ausgangspunkt zu verlieren
- **Von Fehlern wiederherstellen:** Machen Sie schnell Änderungen rückgängig, die Fehler eingeführt oder Funktionalität unterbrochen haben
- **Funktionen iterieren:** Experimentieren Sie mit Variationen, da Sie zu funktionierenden Zuständen zurückkehren können

Einschränkungen

Bash-Befehlsänderungen werden nicht verfolgt

Checkpointing verfolgt keine Dateien, die durch Bash-Befehle geändert werden. Wenn Claude Code beispielsweise ausführt:

```
rm file.txt
mv old.txt new.txt
cp source.txt dest.txt
```

Diese Dateiänderungen können nicht durch Rewind rückgängig gemacht werden. Nur direkte Dateiänderungen, die durch Claudes Datei-Bearbeitungswerkzeuge vorgenommen werden, werden verfolgt.

Externe Änderungen werden nicht verfolgt

Checkpointing verfolgt nur Dateien, die in der aktuellen Sitzung bearbeitet wurden. Manuelle Änderungen, die Sie an Dateien außerhalb von Claude Code vornehmen, und Änderungen aus anderen gleichzeitigen Sitzungen werden normalerweise nicht erfasst, es sei denn, sie ändern zufällig dieselben Dateien wie die aktuelle Sitzung.

Kein Ersatz für Versionskontrolle

Checkpoints sind für schnelle, sitzungsebene Wiederherstellung konzipiert. Für permanente Versionsverlauf und Zusammenarbeit:

- Verwenden Sie weiterhin Versionskontrolle (z. B. Git) für Commits, Branches und langfristige Verlauf
- Checkpoints ergänzen, ersetzen aber nicht ordnungsgemäße Versionskontrolle

- Denken Sie an Checkpoints als „lokales Rückgängigmachen“ und Git als „permanente Verlauf“

Siehe auch

- [Interaktiver Modus](#) - Tastaturkürzel und Sitzungssteuerungen
- [Integrierte Befehle](#) - Zugriff auf Checkpoints mit `/rewind`
- [CLI-Referenz](#) - Befehlszeilenoptionen

Datennutzung

Erfahren Sie mehr über die Datennutzungsrichtlinien von Anthropic für Claude

Datenrichtlinien

Datentrainingsrichtlinie

Verbrauchernutzer (Free-, Pro- und Max-Pläne): Wir geben Ihnen die Möglichkeit, Ihre Daten zur Verbesserung zukünftiger Claude-Modelle zu nutzen. Wir trainieren neue Modelle mit Daten aus Free-, Pro- und Max-Konten, wenn diese Einstellung aktiviert ist (auch wenn Sie Claude Code aus diesen Konten verwenden).

Kommerzielle Nutzer: (Team- und Enterprise-Pläne, API, Plattformen von Drittanbietern und Claude Gov) behalten bestehende Richtlinien bei: Anthropic trainiert keine generativen Modelle mit Code oder Eingabeaufforderungen, die unter kommerziellen Bedingungen an Claude Code gesendet werden, es sei denn, der Kunde hat sich dafür entschieden, seine Daten für die Modellverbesserung bereitzustellen (zum Beispiel das [Developer Partner Program](#)).

Development Partner Program

Wenn Sie sich explizit dafür entscheiden, uns Materialien zum Trainieren bereitzustellen, z. B. über das [Development Partner Program](#), können wir diese Materialien zum Trainieren unserer Modelle verwenden. Ein Organisationsadministrator kann sich explizit für das Development Partner Program für seine Organisation anmelden. Beachten Sie, dass dieses Programm nur für die Anthropic-API von Erstanbietern verfügbar ist und nicht für Bedrock- oder Vertex-Nutzer.

Feedback mit dem `/bug` -Befehl

Wenn Sie uns Feedback zu Claude Code mit dem `/bug` -Befehl senden, können wir Ihr Feedback zur Verbesserung unserer Produkte und Dienstleistungen nutzen. Transkripte, die über `/bug` freigegeben werden, werden 5 Jahre lang aufbewahrt.

Sitzungsqualitätsumfragen

Wenn Sie in Claude Code die Eingabeaufforderung „Wie macht Claude das in dieser Sitzung?“ sehen, wird bei der Beantwortung dieser Umfrage (einschließlich der Auswahl von „Verwerfen“) nur Ihre numerische Bewertung (1, 2, 3 oder Verwerfen) aufgezeichnet. Wir

erfassen oder speichern keine Gesprächstranskripte, Eingaben, Ausgaben oder andere Sitzungsdaten als Teil dieser Umfrage. Im Gegensatz zu Daumen-hoch/runter-Feedback oder [/bug](#) -Berichten ist diese Sitzungsqualitätsumfrage eine einfache Produktzufriedenheitsmetrik. Ihre Antworten auf diese Umfrage beeinflussen nicht Ihre Datentrainingseinstellungen und können nicht zum Trainieren unserer KI-Modelle verwendet werden.

Um diese Umfragen zu deaktivieren, setzen Sie

`CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1`. Die Umfrage wird auch automatisch deaktiviert, wenn Sie Anbieter von Drittanbietern (Bedrock, Vertex, Foundry) verwenden oder wenn die Telemetrie deaktiviert ist.

Datenspeicherung

Anthropic speichert Claude Code-Daten basierend auf Ihrem Kontotyp und Ihren Einstellungen.

Verbrauchernutzer (Free-, Pro- und Max-Pläne):

- Nutzer, die die Datennutzung für die Modellverbesserung zulassen: 5-jährige Aufbewahrungsfrist zur Unterstützung der Modellentwicklung und Sicherheitsverbesserungen
- Nutzer, die die Datennutzung für die Modellverbesserung nicht zulassen: 30-tägige Aufbewahrungsfrist
- Datenschutzeinstellungen können jederzeit unter claude.ai/settings/data-privacy-controls geändert werden.

Kommerzielle Nutzer (Team, Enterprise und API):

- Standard: 30-tägige Aufbewahrungsfrist
- [Zero data retention](#): verfügbar für Claude Code auf Claude for Enterprise. ZDR wird auf Organisationsbasis aktiviert; jede neue Organisation muss ZDR separat von Ihrem Account-Team aktivieren lassen
- Lokales Caching: Claude Code-Clients können Sitzungen lokal bis zu 30 Tage speichern, um die Sitzungswiederaufnahme zu ermöglichen (konfigurierbar)

Sie können einzelne Claude Code- Websitzungen jederzeit löschen. Das Löschen einer Sitzung entfernt die Ereignisdaten der Sitzung dauerhaft. Anweisungen zum Löschen von Sitzungen finden Sie unter [Managing sessions](#).

Erfahren Sie mehr über Datenspeicherungspraktiken in unserem [Privacy Center](#).

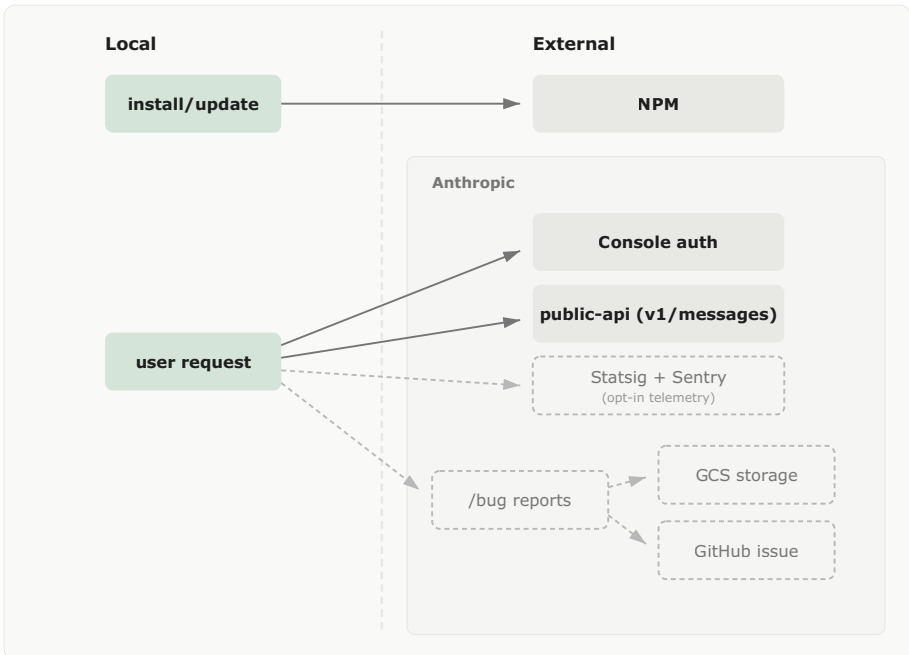
Vollständige Details finden Sie in unseren [Commercial Terms of Service](#) (für Team-, Enterprise- und API-Nutzer) oder [Consumer Terms](#) (für Free-, Pro- und Max-Nutzer) und [Privacy Policy](#).

Datenzugriff

Für alle Nutzer von Erstanbietern können Sie mehr über die protokollierten Daten für [lokales Claude Code](#) und [Remote Claude Code](#) erfahren. [Remote Control](#)-Sitzungen folgen dem lokalen Datenfluss, da die gesamte Ausführung auf Ihrem Computer stattfindet. Beachten Sie, dass Claude bei Remote Claude Code auf das Repository zugreift, in dem Sie Ihre Claude Code-Sitzung starten. Claude greift nicht auf Repositories zu, die Sie verbunden haben, aber in denen Sie keine Sitzung gestartet haben.

Local Claude Code: Datenfluss und Abhängigkeiten

Das folgende Diagramm zeigt, wie Claude Code während der Installation und des normalen Betriebs eine Verbindung zu externen Diensten herstellt. Durchgehende Linien zeigen erforderliche Verbindungen an, während gestrichelte Linien optionale oder vom Benutzer initiierte Datenflüsse darstellen.



Diagramm, das die externen Verbindungen von Claude Code zeigt: Installation/Update verbindet sich mit NPM, und Benutzeranfragen verbinden sich mit Anthropic-Diensten, einschließlich Console-Authentifizierung, öffentlicher API und optional Statsig, Sentry und Bug-Berichterstattung

Claude Code wird von [NPM](#) installiert. Claude Code wird lokal ausgeführt. Um mit dem LLM zu interagieren, sendet Claude Code Daten über das Netzwerk. Diese Daten umfassen alle Benutzereingabeaufforderungen und Modellausgaben. Die Daten werden während der Übertragung über TLS verschlüsselt und sind im Ruhezustand nicht verschlüsselt. Claude Code ist mit den meisten gängigen VPNs und LLM-Proxys kompatibel.

Claude Code basiert auf den APIs von Anthropic. Weitere Informationen zu den Sicherheitskontrollen unserer API, einschließlich unserer API-Protokollierungsverfahren, finden Sie in den Compliance-Artefakten im [Anthropic Trust Center](#).

Cloud-Ausführung: Datenfluss und Abhängigkeiten

Bei Verwendung von [Claude Code on the web](#) werden Sitzungen in von Anthropic verwalteten virtuellen Maschinen statt lokal ausgeführt. In Cloud-Umgebungen:

- **Code- und Datenspeicherung:** Ihr Repository wird auf eine isolierte VM geklont. Code und Sitzungsdaten unterliegen den Aufbewahrungs- und Nutzungsrichtlinien für Ihren Kontotyp (siehe Abschnitt Datenspeicherung oben)
- **Anmeldedaten:** Die GitHub-Authentifizierung wird über einen sicheren Proxy durchgeführt; Ihre GitHub-Anmeldedaten gelangen niemals in die Sandbox
- **Netzwerkverkehr:** Der gesamte ausgehende Datenverkehr wird über einen Sicherheitsproxy für Audit-Protokollierung und Missbrauchsprävention geleitet
- **Sitzungsdaten:** Eingabeaufforderungen, Codeänderungen und Ausgaben folgen den gleichen Datenrichtlinien wie die lokale Claude Code-Nutzung

Sicherheitsdetails zur Cloud-Ausführung finden Sie unter [Security](#).

Telemetrie-Dienste

Claude Code verbindet sich von den Maschinen der Benutzer mit dem Statsig-Dienst, um operative Metriken wie Latenz, Zuverlässigkeit und Nutzungsmuster zu protokollieren. Diese Protokollierung umfasst keinen Code oder Dateipfade. Die Daten werden während der Übertragung mit TLS und im Ruhezustand mit 256-Bit-AES-Verschlüsselung verschlüsselt. Weitere Informationen finden Sie in der [Statsig-Sicherheitsdokumentation](#). Um sich von der Statsig-Telemetrie abzumelden, setzen Sie die Umgebungsvariable

`DISABLE_TELEMETRY`.

Claude Code verbindet sich von den Maschinen der Benutzer mit Sentry für operative Fehlerprotokollierung. Die Daten werden während der Übertragung mit TLS und im Ruhezustand mit 256-Bit-AES-Verschlüsselung verschlüsselt. Weitere Informationen finden Sie in der [Sentry-Sicherheitsdokumentation](#). Um sich von der Fehlerprotokollierung abzumelden, setzen Sie die Umgebungsvariable `DISABLE_ERROR_REPORTING`.

Wenn Benutzer den `/bug` -Befehl ausführen, wird eine Kopie ihres vollständigen Gesprächsverlaufs einschließlich Code an Anthropic gesendet. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Optional wird ein Github-Problem in unserem öffentlichen Repository erstellt. Um sich von der Bug-Berichterstattung abzumelden, setzen Sie die Umgebungsvariable `DISABLE_BUG_COMMAND` .

Standardverhalten nach API-Anbieter

Standardmäßig deaktivieren wir den gesamten nicht wesentlichen Datenverkehr (einschließlich Fehlerberichterstattung, Telemetrie, Bug-Berichterstattungsfunktionalität und Sitzungsqualitätsumfragen), wenn Sie Bedrock, Vertex oder Foundry verwenden. Sie können sich auch auf einmal von all diesen abmelden, indem Sie die Umgebungsvariable `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` setzen. Hier sind die vollständigen Standardverhalten:

Dienst	Claude API	Vertex API	Bedrock API
Statsig (Metriken)	Standardmäßig aktiviert. <code>DISABLE_TELEMETRY=1</code> zum Deaktivieren.	Standardmäßig deaktiviert. <code>CLAUDE_CODE_USE_VERTEX</code> muss 1 sein.	Standardmäßig <code>CLAUDE_CODE_</code> muss 1 sein.

Dienst	Claude API	Vertex API	Bedrock API
Sentry (Fehler)	Standardmäßig aktiviert. DISABLE_ERROR_REPORTING=1 zum Deaktivieren.	Standardmäßig deaktiviert. CLAUDE_CODE_USE_VERTEX muss 1 sein.	Standardmäßig CLAUDE_CODE_ muss 1 sein.
Claude API (/ bug -Berichte)	Standardmäßig aktiviert. DISABLE_BUG_COMMAND=1 zum Deaktivieren.	Standardmäßig deaktiviert. CLAUDE_CODE_USE_VERTEX muss 1 sein.	Standardmäßig CLAUDE_CODE_ muss 1 sein.
Sitzungsqualitätsumfragen	Standardmäßig aktiviert. CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1 zum Deaktivieren.	Standardmäßig deaktiviert. CLAUDE_CODE_USE_VERTEX muss 1 sein.	Standardmäßig CLAUDE_CODE_ muss 1 sein.

Alle Umgebungsvariablen können in `settings.json` eingecheckt werden ([weitere Informationen](#)).

Rechtliche Bestimmungen und Compliance

Rechtliche Vereinbarungen, Compliance-Zertifizierungen und Sicherheitsinformationen für Claude Code.

Rechtliche Vereinbarungen

Lizenz

Ihre Nutzung von Claude Code unterliegt:

- [Geschäftsbedingungen](#) - für Team-, Enterprise- und Claude API-Benutzer
- [Verbraucherbedingungen](#) - für Free-, Pro- und Max-Benutzer

Geschäftliche Vereinbarungen

Unabhängig davon, ob Sie die Claude API direkt (1P) nutzen oder über AWS Bedrock oder Google Vertex (3P) darauf zugreifen, gilt Ihre bestehende Geschäftsvereinbarung für die Nutzung von Claude Code, sofern wir nicht gegenseitig etwas anderes vereinbart haben.

Compliance

Healthcare-Compliance (BAA)

Wenn ein Kunde eine Business Associate Agreement (BAA) mit uns hat und Claude Code nutzen möchte, wird die BAA automatisch erweitert, um Claude Code abzudecken, wenn der Kunde eine BAA ausgeführt hat und Zero Data Retention (ZDR) aktiviert hat. Die BAA gilt für den API-Datenverkehr dieses Kunden, der durch Claude Code fließt.

Sicherheit und Vertrauen

Vertrauen und Sicherheit

Weitere Informationen finden Sie im [Anthropic Trust Center](#) und im [Transparency Hub](#).

Meldung von Sicherheitslücken

Anthropic verwaltet unser Sicherheitsprogramm über HackerOne. [Verwenden Sie dieses Formular, um Sicherheitslücken zu melden.](#)

© Anthropic PBC. Alle Rechte vorbehalten. Die Nutzung unterliegt den geltenden Anthropic-Nutzungsbedingungen.

Null-Datenspeicherung

Erfahren Sie mehr über Null-Datenspeicherung (ZDR) für Claude Code auf Claude for Enterprise, einschließlich Umfang, deaktivierter Funktionen und wie Sie die Aktivierung anfordern.

Null-Datenspeicherung (ZDR) ist für Claude Code verfügbar, wenn es über Claude for Enterprise verwendet wird. Wenn ZDR aktiviert ist, werden Eingabeaufforderungen und Modellreaktionen, die während Claude Code-Sitzungen generiert werden, in Echtzeit verarbeitet und nicht von Anthropic gespeichert, nachdem die Antwort zurückgegeben wurde, außer wenn dies erforderlich ist, um das Gesetz einzuhalten oder Missbrauch zu bekämpfen.

ZDR auf Claude for Enterprise gibt Unternehmenskunden die Möglichkeit, Claude Code mit Null-Datenspeicherung zu verwenden und auf Verwaltungsfunktionen zuzugreifen:

- Kostenkontrolle pro Benutzer
- [Analytics](#)-Dashboard
- [Serververwaltete Einstellungen](#)
- Audit-Protokolle

ZDR für Claude Code auf Claude for Enterprise gilt nur für die direkte Plattform von Anthropic. Für Claude-Bereitstellungen auf AWS Bedrock, Google Vertex AI oder Microsoft Foundry beachten Sie die Datenspeicherungsrichtlinien dieser Plattformen.

ZDR-Umfang

ZDR deckt Claude Code-Inferenz auf Claude for Enterprise ab.

Warning:

ZDR wird auf Organisationsbasis aktiviert. Jede neue Organisation erfordert, dass ZDR separat von Ihrem Anthropic-Kontoteam aktiviert wird. ZDR wird nicht automatisch auf neue Organisationen angewendet, die unter demselben Konto erstellt werden. Wenden Sie sich an Ihr Kontoteam, um ZDR für neue Organisationen zu aktivieren.

Was ZDR abdeckt

ZDR deckt Modellrückschluss-Aufrufe ab, die über Claude Code auf Claude for Enterprise durchgeführt werden. Wenn Sie Claude Code in Ihrem Terminal verwenden, werden die Eingabeaufforderungen, die Sie senden, und die Antworten, die Claude generiert, nicht von Anthropic gespeichert. Dies gilt unabhängig davon, welches Claude-Modell verwendet wird.

Was ZDR nicht abdeckt

ZDR erstreckt sich nicht auf die folgenden Funktionen, auch nicht für Organisationen mit aktiviertem ZDR. Diese Funktionen folgen [Standard-Datenspeicherungsrichtlinien](#):

Funktion	Details
Chat auf claude.ai	Chat-Gespräche über die Claude for Enterprise-Weboberfläche werden nicht von ZDR abgedeckt.
Co-work	Cowork-Sitzungen werden nicht von ZDR abgedeckt.
Claude Code Analytics	Speichert keine Eingabeaufforderungen oder Modellreaktionen, erfasst aber Produktivitätsmetadaten wie Konto-E-Mails und Nutzungsstatistiken. Beitragskennzahlen sind für ZDR-Organisationen nicht verfügbar; das Analytics-Dashboard zeigt nur Nutzungsmetriken an.
Benutzer- und Platzverwaltung	Verwaltungsdaten wie Konto-E-Mails und Platzzuweisungen werden nach Standardrichtlinien beibehalten.
Integrationen von Drittanbietern	Daten, die von Drittanbieter-Tools, MCP servers oder anderen externen Integrationen verarbeitet werden, werden nicht von ZDR abgedeckt. Überprüfen Sie die Datenverwaltungspraktiken dieser Dienste unabhängig.

Funktionen, die unter ZDR deaktiviert sind

Wenn ZDR für eine Claude Code-Organisation auf Claude for Enterprise aktiviert ist, werden bestimmte Funktionen, die das Speichern von Eingabeaufforderungen oder Vervollständigungen erfordern, automatisch auf Backend-Ebene deaktiviert:

Funktion	Grund
Claude Code im Web	Erfordert serverseitige Speicherung des Gesprächsverlaufs.
Remote-Sitzungen aus der Desktop-App	Erfordert persistente Sitzungsdaten, die Eingabeaufforderungen und Vervollständigungen enthalten.
Feedback-Übermittlung (/feedback)	Das Übermitteln von Feedback sendet Gesprächsdaten an Anthropic.

Diese Funktionen werden im Backend blockiert, unabhängig von der clientseitigen Anzeige. Wenn Sie während des Starts eine deaktivierte Funktion im Claude Code-Terminal sehen, führt der Versuch, sie zu verwenden, zu einem Fehler, der angibt, dass die Richtlinien der Organisation diese Aktion nicht zulassen.

Zukünftige Funktionen können auch deaktiviert werden, wenn sie das Speichern von Eingabeaufforderungen oder Vervollständigungen erfordern.

Datenspeicherung bei Richtlinienverletzungen

Auch wenn ZDR aktiviert ist, kann Anthropic Daten speichern, wenn dies gesetzlich erforderlich ist oder um Verstöße gegen die Nutzungsrichtlinie zu beheben. Wenn eine Sitzung wegen eines Richtlinienverstosses gekennzeichnet wird, kann Anthropic die zugehörigen Ein- und Ausgaben bis zu 2 Jahre lang speichern, in Übereinstimmung mit Anthropic's Standard-ZDR-Richtlinie.

ZDR anfordern

Um ZDR für Claude Code auf Claude for Enterprise anzufordern, wenden Sie sich an Ihr Anthropic-Kontoteam. Ihr Kontoteam reicht die Anfrage intern ein, und Anthropic überprüft und aktiviert ZDR in Ihrer Organisation, nachdem die Berechtigung bestätigt wurde. Alle Aktivierungsmaßnahmen werden protokolliert.

Wenn Sie derzeit ZDR für Claude Code über Pay-as-you-go-API-Schlüssel verwenden, können Sie zu Claude for Enterprise wechseln, um Zugriff auf Verwaltungsfunktionen zu erhalten und gleichzeitig ZDR für Claude Code beizubehalten. Wenden Sie sich an Ihr Kontoteam, um die Migration zu koordinieren.

Part 10: Enterprise & Monitoring

Teamnutzung mit Analysen verfolgen

Zeigen Sie Claude Code-Nutzungsmetriken an, verfolgen Sie die Einführung und messen Sie die Engineering-Geschwindigkeit im Analytics-Dashboard.

Claude Code bietet Analytics-Dashboards, um Organisationen dabei zu helfen, Entwicklungsmuster zu verstehen, Beitragskennzahlen zu verfolgen und zu messen, wie Claude Code die Engineering-Geschwindigkeit beeinflusst. Greifen Sie auf das Dashboard für Ihren Plan zu:

Plan	Dashboard-URL	Enthält	Weitere Informationen
Claude for Teams / Enterprise	claude.ai/analytics/claude-code	Nutzungsmetriken, Beitragskennzahlen mit GitHub-Integration, Rangliste, Datenexport	Details
API (Claude Console)	platform.claude.com/claude-code	Nutzungsmetriken, Ausgabenverfolgung, Team-Insights	Details

Analytics für Teams und Enterprise aufrufen

Navigieren Sie zu claude.ai/analytics/claude-code. Admins und Owners können das Dashboard anzeigen.

Das Teams- und Enterprise-Dashboard enthält:

- **Nutzungsmetriken:** akzeptierte Codezeilen, Akzeptanzrate für Vorschläge, täglich aktive Benutzer und Sitzungen
- **Beitragskennzahlen:** PRs und Codezeilen, die mit Claude Code-Unterstützung versendet wurden, mit [GitHub-Integration](#)
- **Rangliste:** Top-Bertragsteller, sortiert nach Claude Code-Nutzung
- **Datenexport:** Beitragsdaten als CSV für benutzerdefinierte Berichte herunterladen

Beitragskennzahlen aktivieren

Note:

Beitragskennzahlen befinden sich in der öffentlichen Beta und sind in Claude for Teams und Claude for Enterprise-Plänen verfügbar. Diese Metriken decken nur Benutzer innerhalb Ihrer claude.ai-Organisation ab. Die Nutzung über die Claude Console API oder Integrationen von Drittanbietern ist nicht enthalten.

Nutzungs- und Einführungsdaten sind für alle Claude for Teams und Claude for Enterprise-Konten verfügbar. Beitragskennzahlen erfordern zusätzliche Einrichtung, um Ihre GitHub-Organisation zu verbinden.

Sie benötigen die Owner-Rolle, um Analytics-Einstellungen zu konfigurieren. Ein GitHub-Admin muss die GitHub-App installieren.

Warning:

Beitragskennzahlen sind nicht für Organisationen mit aktiviertem [Zero Data Retention](#) verfügbar. Das Analytics-Dashboard zeigt nur Nutzungsmetriken an.

Step 1: GitHub-App installieren

Ein GitHub-Admin installiert die Claude GitHub-App auf dem GitHub-Konto Ihrer Organisation unter github.com/apps/claude.

Step 2: Claude Code-Analysen aktivieren

Ein Claude Owner navigiert zu claude.ai/admin-settings/claude-code und aktiviert die Claude Code-Analytics-Funktion.

Step 3: GitHub-Analysen aktivieren

Aktivieren Sie auf der gleichen Seite den Schalter 'GitHub analytics'.

Step 4: Mit GitHub authentifizieren

Schließen Sie den GitHub-Authentifizierungsfluss ab und wählen Sie aus, welche GitHub-Organisationen in die Analyse einbezogen werden sollen.

Daten werden normalerweise innerhalb von 24 Stunden nach der Aktivierung angezeigt, mit täglichen Updates. Wenn keine Daten angezeigt werden, wird möglicherweise eine dieser Meldungen angezeigt:

- **„GitHub-App erforderlich“**: Installieren Sie die GitHub-App, um Beitragskennzahlen anzuzeigen
- **„Datenverarbeitung läuft“**: Überprüfen Sie in einigen Tagen erneut und bestätigen Sie, dass die GitHub-App installiert ist, falls keine Daten angezeigt werden

Beitragskennzahlen unterstützen GitHub Cloud und GitHub Enterprise Server.

Zusammenfassende Metriken überprüfen

Note:

Diese Metriken sind absichtlich konservativ und stellen eine Unterschätzung der tatsächlichen Auswirkungen von Claude Code dar. Nur Zeilen und PRs, bei denen hohes Vertrauen in die Beteiligung von Claude Code besteht, werden gezählt.

Das Dashboard zeigt diese zusammenfassenden Metriken oben an:

- **PRs mit CC**: Gesamtanzahl der zusammengeführten Pull Requests, die mindestens eine Codezeile enthalten, die mit Claude Code geschrieben wurde
- **Codezeilen mit CC**: Gesamtzahl der Codezeilen in allen zusammengeführten PRs, die mit Claude Code-Unterstützung geschrieben wurden. Nur „effektive Zeilen“ werden gezählt: Zeilen mit mehr als 3 Zeichen nach Normalisierung, ohne leere Zeilen und Zeilen mit nur Klammern oder trivialer Interpunktion.
- **PRs mit Claude Code (%)**: Prozentsatz aller zusammengeführten PRs, die Claude Code-unterstützten Code enthalten
- **Akzeptanzrate für Vorschläge**: Prozentsatz der Fälle, in denen Benutzer Claude Code-Codebearbeitungsvorschläge akzeptieren, einschließlich Edit, Write und NotebookEdit-Tool-Nutzung
- **Akzeptierte Codezeilen**: Gesamtzahl der Codezeilen, die von Claude Code geschrieben wurden und die Benutzer in ihren Sitzungen akzeptiert haben. Dies schließt abgelehnte Vorschläge aus und verfolgt keine nachfolgenden Löschungen.

Diagramme erkunden

Das Dashboard enthält mehrere Diagramme zur Visualisierung von Trends im Zeitverlauf.

Einführung verfolgen

Das Adoption-Diagramm zeigt tägliche Nutzungstrends:

- **users:** täglich aktive Benutzer
- **sessions:** Anzahl der aktiven Claude Code-Sitzungen pro Tag

PRs pro Benutzer messen

Dieses Diagramm zeigt die individuelle Entwickleraktivität im Zeitverlauf:

- **PRs per user:** Gesamtzahl der pro Tag zusammengeführten PRs geteilt durch täglich aktive Benutzer
- **users:** täglich aktive Benutzer

Verwenden Sie dies, um zu verstehen, wie sich die individuelle Produktivität mit zunehmender Claude Code-Einführung ändert.

Aufschlüsselung der Pull Requests anzeigen

Das Pull requests-Diagramm zeigt eine tägliche Aufschlüsselung der zusammengeführten PRs:

- **PRs with CC:** Pull Requests mit Claude Code-unterstütztem Code
- **PRs without CC:** Pull Requests ohne Claude Code-unterstützten Code

Wechseln Sie zur Ansicht **Lines of code**, um die gleiche Aufschlüsselung nach Codezeilen statt nach PR-Anzahl zu sehen.

Top-Beitragsteller finden

Die Rangliste zeigt die Top 10-Benutzer, sortiert nach Beitragsmenge. Wechseln Sie zwischen:

- **Pull requests:** zeigt PRs mit Claude Code vs. alle PRs für jeden Benutzer
- **Lines of code:** zeigt Zeilen mit Claude Code vs. alle Zeilen für jeden Benutzer

Klicken Sie auf **Export all users**, um vollständige Beitragsdaten für alle Benutzer als CSV-Datei herunterzuladen. Der Export enthält alle Benutzer, nicht nur die angezeigten Top 10.

PR-Zuordnung

Wenn Beitragskennzahlen aktiviert sind, analysiert Claude Code zusammengeführte Pull Requests, um zu bestimmen, welcher Code mit Claude Code-Unterstützung geschrieben wurde. Dies geschieht durch Abgleich der Claude Code-Sitzungsaktivität mit dem Code in jedem PR.

Tagging-Kriterien

PRs werden als „with Claude Code“ gekennzeichnet, wenn sie mindestens eine Codezeile enthalten, die während einer Claude Code-Sitzung geschrieben wurde. Das System verwendet konservatives Matching: Nur Code, bei dem hohes Vertrauen in die Beteiligung von Claude Code besteht, wird als unterstützt gezählt.

Zuordnungsprozess

Wenn ein Pull Request zusammengeführt wird:

1. Hinzugefügte Zeilen werden aus dem PR-Diff extrahiert
2. Claude Code-Sitzungen, die übereinstimmende Dateien innerhalb eines Zeitfensters bearbeitet haben, werden identifiziert
3. PR-Zeilen werden mit Claude Code-Ausgabe unter Verwendung mehrerer Strategien abgeglichen
4. Metriken werden für KI-unterstützte Zeilen und Gesamtzeilen berechnet

Vor dem Vergleich werden Zeilen normalisiert: Leerzeichen werden gekürzt, mehrere Leerzeichen werden zusammengefasst, Anführungszeichen werden standardisiert und Text wird in Kleinbuchstaben konvertiert.

Zusammengeführte Pull Requests mit Claude Code-unterstützten Zeilen werden in GitHub mit `claude-code-assisted` gekennzeichnet.

Zeitfenster

Sitzungen von 21 Tagen vor bis 2 Tage nach dem PR-Zusammenführungsdatum werden für den Zuordnungsabgleich berücksichtigt.

Ausgeschlossene Dateien

Bestimmte Dateien werden automatisch von der Analyse ausgeschlossen, da sie automatisch generiert werden:

- Lock-Dateien: package-lock.json, yarn.lock, Cargo.lock und ähnliche
- Generierter Code: Protobuf-Ausgaben, Build-Artefakte, minifizierte Dateien
- Build-Verzeichnisse: dist/, build/, node_modules/, target/
- Test-Fixtures: Snapshots, Cassetten, Mock-Daten
- Zeilen über 1.000 Zeichen, die wahrscheinlich minifiziert oder generiert sind

Zuordnungshinweise

Beachten Sie diese zusätzlichen Details bei der Interpretation von Zuordnungsdaten:

- Code, der von Entwicklern erheblich umgeschrieben wurde, mit mehr als 20% Unterschied, wird nicht Claude Code zugeordnet
- Sitzungen außerhalb des 21-Tage-Fensters werden nicht berücksichtigt
- Der Algorithmus berücksichtigt nicht den PR-Quell- oder Zielzweig bei der Durchführung der Zuordnung

Nutzen Sie Analytics optimal

Verwenden Sie Beitragskennzahlen, um ROI zu demonstrieren, Einführungsmuster zu identifizieren und Teammitglieder zu finden, die anderen beim Einstieg helfen können.

Einführung überwachen

Verfolgen Sie das Adoption-Diagramm und Benutzerzahlen, um Folgendes zu identifizieren:

- Aktive Benutzer, die Best Practices teilen können
- Gesamte Einführungstrends in Ihrer Organisation
- Nutzungsrückgänge, die auf Reibung oder Probleme hindeuten können

ROI messen

Beitragskennzahlen helfen bei der Beantwortung der Frage „Lohnt sich dieses Tool für die Investition?“ mit Daten aus Ihrer eigenen Codebasis:

- Verfolgen Sie Änderungen bei PRs pro Benutzer im Zeitverlauf, wenn die Einführung zunimmt
- Vergleichen Sie PRs und Codezeilen, die mit und ohne Claude Code versendet wurden
- Verwenden Sie zusammen mit [DORA-Metriken](#), Sprint-Geschwindigkeit oder anderen Engineering-KPIs, um Änderungen durch die Einführung von Claude Code zu verstehen

Power-User identifizieren

Die Rangliste hilft Ihnen, Teammitglieder mit hoher Claude Code-Einführung zu finden, die:

- Prompting-Techniken und Workflows mit dem Team teilen können
- Feedback geben können, was gut funktioniert
- Neue Benutzer onboarden können

Auf Daten programmgesteuert zugreifen

Um diese Daten über GitHub abzufragen, suchen Sie nach PRs mit dem Label `claude-code-assisted`.

Analytics für API-Kunden aufrufen

API-Kunden, die die Claude Console verwenden, können auf Analytics unter platform.claude.com/claude-code zugreifen. Sie benötigen die UsageView-Berechtigung, um auf das Dashboard zuzugreifen, die den Rollen Developer, Billing, Admin, Owner und Primary Owner gewährt wird.

Note:

Beitragskennzahlen mit GitHub-Integration sind derzeit nicht für API-Kunden verfügbar. Das Console-Dashboard zeigt nur Nutzungs- und Ausgabemetriken an.

Das Console-Dashboard zeigt:

- **Lines of code accepted:** Gesamtzahl der Codezeilen, die von Claude Code geschrieben wurden und die Benutzer in ihren Sitzungen akzeptiert haben. Dies schließt abgelehnte Vorschläge aus und verfolgt keine nachfolgenden Löschungen.
- **Suggestion accept rate:** Prozentsatz der Fälle, in denen Benutzer die Nutzung von Code-Bearbeitungstools akzeptieren, einschließlich Edit, Write und NotebookEdit-Tools.
- **Activity:** täglich aktive Benutzer und Sitzungen, die in einem Diagramm angezeigt werden.
- **Spend:** tägliche API-Kosten in Dollar neben der Benutzerzahl.

Team-Insights anzeigen

Die Team-Insights-Tabelle zeigt Metriken pro Benutzer:

- **Members:** alle Benutzer, die sich bei Claude Code authentifiziert haben. API-Schlüsselbenutzer werden nach Schlüsselkennung angezeigt, OAuth-Benutzer werden nach E-Mail-Adresse angezeigt.
- **Spend this month:** Gesamtkosten der API pro Benutzer für den aktuellen Monat.
- **Lines this month:** Gesamtzahl der akzeptierten Codezeilen pro Benutzer für den aktuellen Monat.

Note:

Die Ausgabenzahlen im Console-Dashboard sind Schätzungen für Analytics-Zwecke. Für tatsächliche Kosten beziehen Sie sich auf Ihre Abrechnungsseite.

Verwandte Ressourcen

- [Monitoring mit OpenTelemetry](#): Exportieren Sie Echtzeit-Metriken und Ereignisse in Ihren Observability-Stack
- [Kosten effektiv verwalten](#): Legen Sie Ausgabenlimits fest und optimieren Sie die Token-Nutzung
- [Berechtigungen](#): Konfigurieren Sie Rollen und Berechtigungen

Überwachung

Erfahren Sie, wie Sie OpenTelemetry für Claude Code aktivieren und konfigurieren.

Claude Code unterstützt OpenTelemetry (OTel) Metriken und Ereignisse für Überwachung und Observability.

Alle Metriken sind Zeitreihendaten, die über das Standard-Metriken-Protokoll von OpenTelemetry exportiert werden, und Ereignisse werden über das Logs/Events-Protokoll von OpenTelemetry exportiert. Es liegt in der Verantwortung des Benutzers, sicherzustellen, dass seine Metriken- und Logs-Backends ordnungsgemäß konfiguriert sind und dass die Aggregationsgranularität seine Überwachungsanforderungen erfüllt.

Schnellstart

Konfigurieren Sie OpenTelemetry mit Umgebungsvariablen:

```
## 1. Telemetrie aktivieren
export CLAUDE_CODE_ENABLE_TELEMETRY=1

## 2. Exporter auswählen (beide sind optional - konfigurieren Sie nur das, was Sie
benötigen)
export OTEL_METRICS_EXPORTER=otlp      # Optionen: otlp, prometheus, console
export OTEL_LOGS_EXPORTER=otlp        # Optionen: otlp, console

## 3. OTLP-Endpunkt konfigurieren (für OTLP-Exporter)
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## 4. Authentifizierung festlegen (falls erforderlich)
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer your-token"

## 5. Zum Debuggen: Exportintervalle reduzieren
export OTEL_METRIC_EXPORT_INTERVAL=10000 # 10 Sekunden (Standard: 60000ms)
export OTEL_LOGS_EXPORT_INTERVAL=5000    # 5 Sekunden (Standard: 5000ms)

## 6. Claude Code ausführen
claude
```

Note:

Die Standard-Exportintervalle betragen 60 Sekunden für Metriken und 5 Sekunden für Logs. Während der Einrichtung möchten Sie möglicherweise kürzere Intervalle für Debugging-Zwecke verwenden. Denken Sie daran, diese für die Produktionsnutzung zurückzusetzen.

Für vollständige Konfigurationsoptionen siehe die [OpenTelemetry-Spezifikation](#).

Administratorkonfiguration

Administratoren können OpenTelemetry-Einstellungen für alle Benutzer über die [verwaltete Einstellungsdatei](#) konfigurieren. Dies ermöglicht eine zentrale Kontrolle der Telemetrie-Einstellungen in einer Organisation. Weitere Informationen zur Anwendung von Einstellungen finden Sie unter [Einstellungspriorität](#).

Beispiel für verwaltete Einstellungskonfiguration:

```
{
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_METRICS_EXPORTER": "otlp",
    "OTEL_LOGS_EXPORTER": "otlp",
    "OTEL_EXPORTER_OTLP_PROTOCOL": "grpc",
    "OTEL_EXPORTER_OTLP_ENDPOINT": "http://collector.company.com:4317",
    "OTEL_EXPORTER_OTLP_HEADERS": "Authorization=Bearer company-token"
  }
}
```

Note:

Verwaltete Einstellungen können über MDM (Mobile Device Management) oder andere Geräteverwaltungslösungen verteilt werden. Umgebungsvariablen, die in der verwalteten Einstellungsdatei definiert sind, haben hohe Priorität und können von Benutzern nicht überschrieben werden.

Konfigurationsdetails

Allgemeine Konfigurationsvariablen

Umgebungsvariable	Beschreibung	Beispielwerte
CLAUDE_CODE_ENABLE_TELEMETRY	Aktiviert die Telemetrieerfassung (erforderlich)	1
OTEL_METRICS_EXPORTER	Metriken-Exporter-Typ(en) (kommagetrennt)	console , otlp , prometheus
OTEL_LOGS_EXPORTER	Logs/Events-Exporter-Typ(en) (kommagetrennt)	console , otlp

Umgebungsvariable	Beschreibung	Beispielwerte
<code>OTEL_EXPORTER_OTLP_PROTOCOL</code>	Protokoll für OTLP-Exporter (alle Signale)	<code>grpc</code> , <code>http</code> , <code>json</code> , <code>http/protobuf</code>
<code>OTEL_EXPORTER_OTLP_ENDPOINT</code>	OTLP-Collector-Endpunkt (alle Signale)	<code>http://localhost:4317</code>
<code>OTEL_EXPORTER_OTLP_METRICS_PROTOCOL</code>	Protokoll für Metriken (überschreibt allgemein)	<code>grpc</code> , <code>http</code> , <code>json</code> , <code>http/protobuf</code>
<code>OTEL_EXPORTER_OTLP_METRICS_ENDPOINT</code>	OTLP-Metriken-Endpunkt (überschreibt allgemein)	<code>http://localhost:4318/v1/metrics</code>
<code>OTEL_EXPORTER_OTLP_LOGS_PROTOCOL</code>	Protokoll für Logs (überschreibt allgemein)	<code>grpc</code> , <code>http</code> , <code>json</code> , <code>http/protobuf</code>
<code>OTEL_EXPORTER_OTLP_LOGS_ENDPOINT</code>	OTLP-Logs-Endpunkt (überschreibt allgemein)	<code>http://localhost:4318/v1/logs</code>
<code>OTEL_EXPORTER_OTLP_HEADERS</code>	Authentifizierungsholder für OTLP	<code>Authorization=Bearer token</code>
<code>OTEL_EXPORTER_OTLP_METRICS_CLIENT_KEY</code>	Client-Schlüssel für mTLS-Authentifizierung	Pfad zur Client-Schlüsseldatei
<code>OTEL_EXPORTER_OTLP_METRICS_CLIENT_CERTIFICATE</code>	Client-Zertifikat für mTLS-Authentifizierung	Pfad zur Client-Zertifikatsdatei

Umgebungsvariable	Beschreibung	Beispielwerte
OTEL_METRIC_EXPORT_INTERVAL	Exportintervall in Millisekunden (Standard: 60000)	5000 , 60000
OTEL_LOGS_EXPORT_INTERVAL	Logs-Exportintervall in Millisekunden (Standard: 5000)	1000 , 10000
OTEL_LOG_USER_PROMPTS	Aktiviert die Protokollierung von Benutzer-Prompt-Inhalten (Standard: deaktiviert)	1 zum Aktivieren
CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS	Intervall zum Aktualisieren dynamischer Header (Standard: 1740000ms / 29 Minuten)	900000

Metriken-Kardinalitätskontrolle

Die folgenden Umgebungsvariablen steuern, welche Attribute in Metriken enthalten sind, um die Kardinalität zu verwalten:

Umgebungsvariable	Beschreibung	Standardwert	Beispiel zum Deaktivieren
OTEL_METRICS_INCLUDE_SESSION_ID	Attribut session.id in Metriken einschließen	true	false
OTEL_METRICS_INCLUDE_APP_VERSION	Attribut app.version in Metriken einschließen	false	true

Umgebungsvariable	Beschreibung	Standardwert	Beispiel zum Deaktivieren
OTEL_METRICS_INCLUDE_ACCOUNT_UUID	Attribut user.account_uuid in Metriken einschließen	true	false

Diese Variablen helfen, die Kardinalität von Metriken zu kontrollieren, was sich auf die Speicheranforderungen und die Abfrageleistung in Ihrem Metriken-Backend auswirkt. Eine niedrigere Kardinalität bedeutet in der Regel bessere Leistung und niedrigere Speicherkosten, aber weniger granulare Daten für die Analyse.

Dynamische Header

Für Unternehmensumgebungen, die eine dynamische Authentifizierung erfordern, können Sie ein Skript konfigurieren, um Header dynamisch zu generieren:

Einstellungskonfiguration

Fügen Sie zu Ihrer `.claude/settings.json` hinzu:

```
{
  "otelHeadersHelper": "/bin/generate_opentelemetry_headers.sh"
}
```

Skriptanforderungen

Das Skript muss gültiges JSON mit Zeichenketten-Schlüssel-Wert-Paaren ausgeben, die HTTP-Header darstellen:

```
#!/bin/bash
## Beispiel: Mehrere Header
echo "{\"Authorization\": \"Bearer $(get-token.sh)\", \"X-API-Key\": \"$(get-api-key.sh)\"}"
```

Aktualisierungsverhalten

Das Headers-Helper-Skript wird beim Start und danach regelmäßig ausgeführt, um Token-Aktualisierung zu unterstützen. Standardmäßig wird das Skript alle 29 Minuten ausgeführt. Passen Sie das Intervall mit der Umgebungsvariable

`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS` an.

Unterstützung für Multi-Team-Organisationen

Organisationen mit mehreren Teams oder Abteilungen können benutzerdefinierte Attribute hinzufügen, um zwischen verschiedenen Gruppen zu unterscheiden, indem sie die Umgebungsvariable `OTEL_RESOURCE_ATTRIBUTES` verwenden:

```
## Benutzerdefinierte Attribute für Team-Identifikation hinzufügen
export OTEL_RESOURCE_ATTRIBUTES="department=engineering,team.id=platform,cost_center=eng-123"
```

Diese benutzerdefinierten Attribute werden in alle Metriken und Ereignisse einbezogen, sodass Sie:

- Metriken nach Team oder Abteilung filtern können
- Kosten pro Kostenstelle verfolgen können
- Team-spezifische Dashboards erstellen können
- Warnungen für bestimmte Teams einrichten können

Warning:

Wichtige Formatierungsanforderungen für `OTEL_RESOURCE_ATTRIBUTES`:

Die Umgebungsvariable `OTEL_RESOURCE_ATTRIBUTES` folgt der [W3C Baggage-Spezifikation](#), die strenge Formatierungsanforderungen hat:

- **Keine Leerzeichen erlaubt:** Werte dürfen keine Leerzeichen enthalten. Zum Beispiel ist `user.organizationName=My Company` ungültig
- **Format:** Muss kommagetrennte Schlüssel=Wert-Paare sein:
`key1=value1,key2=value2`
- **Zulässige Zeichen:** Nur US-ASCII-Zeichen ohne Steuerzeichen, Leerzeichen, doppelte Anführungszeichen, Kommas, Semikola und Backslashes
- **Sonderzeichen:** Zeichen außerhalb des zulässigen Bereichs müssen prozentcodiert sein

Beispiele:

```
## ❌ Ungültig - enthält Leerzeichen
export OTEL_RESOURCE_ATTRIBUTES="org.name=John's Organization"

## ✅ Gültig - verwenden Sie stattdessen Unterstriche oder camelCase
export OTEL_RESOURCE_ATTRIBUTES="org.name=Johns_Organization"
export OTEL_RESOURCE_ATTRIBUTES="org.name=JohnsOrganization"

## ✅ Gültig - prozentcodieren Sie Sonderzeichen, falls erforderlich
export OTEL_RESOURCE_ATTRIBUTES="org.name=John%27s%20Organization"
```

Hinweis: Das Einschließen von Werten in Anführungszeichen entkommt keine Leerzeichen. Zum Beispiel führt `org.name="My Company"` zum Literalwert `"My Company"` (mit Anführungszeichen enthalten), nicht zu `My Company`.

Beispielkonfigurationen

Console-Debugging (1-Sekunden-Intervalle)

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console
export OTEL_METRIC_EXPORT_INTERVAL=1000
```

OTLP/gRPC

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
```

Prometheus

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=prometheus
```

Mehrere Exporter

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console,otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=http/json
```

Unterschiedliche Endpunkte/Backends für Metriken und Logs

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_LOGS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_METRICS_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_METRICS_ENDPOINT=http://metrics.company.com:4318
export OTEL_EXPORTER_OTLP_LOGS_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_LOGS_ENDPOINT=http://logs.company.com:4317
```

Nur Metriken (keine Ereignisse/Logs)

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
```

Nur Ereignisse/Logs (keine Metriken)

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
```

```
export OTEL_LOGS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
```

Verfügbare Metriken und Ereignisse

Standardattribute

Alle Metriken und Ereignisse teilen diese Standardattribute:

Attribut	Beschreibung	Gesteuert durch
<code>session.id</code>	Eindeutige Sitzungskennung	<code>OTEL_METRICS_INCLUDE_SESSION_ID</code> (Standard: true)
<code>app.version</code>	Aktuelle Claude Code-Version	<code>OTEL_METRICS_INCLUDE_VERSION</code> (Standard: false)
<code>organization.id</code>	Organisations-UUID (wenn authentifiziert)	Immer enthalten, wenn verfügbar
<code>user.account.uuid</code>	Konto-UUID (wenn authentifiziert)	<code>OTEL_METRICS_INCLUDE_ACCOUNT_UUID</code> (Standard: true)
<code>terminal.type</code>	Terminal-Typ (zum Beispiel <code>iTerm.app</code> , <code>vscode</code> , <code>cursor</code> , <code>tmux</code>)	Immer enthalten, wenn erkannt

Metriken

Claude Code exportiert die folgenden Metriken:

Metrikname	Beschreibung	Einheit
<code>claude_code.session.count</code>	Anzahl der gestarteten CLI-Sitzungen	count
<code>claude_code.lines_of_code.count</code>	Anzahl der geänderten Codezeilen	count
<code>claude_code.pull_request.count</code>	Anzahl der erstellten Pull Requests	count

Metrikname	Beschreibung	Einheit
<code>claude_code.commit.count</code>	Anzahl der erstellten Git-Commits	count
<code>claude_code.cost.usage</code>	Kosten der Claude Code-Sitzung	USD
<code>claude_code.token.usage</code>	Anzahl der verwendeten Token	tokens
<code>claude_code.code_edit_tool.decision</code>	Anzahl der Entscheidungen zur Berechtigung des Code-Bearbeitungstools	count
<code>claude_code.active_time.total</code>	Gesamte aktive Zeit in Sekunden	s

Metrik-Details

Sitzungszähler

Wird zu Beginn jeder Sitzung erhöht.

Attribute:

- Alle [Standardattribute](#)

Codezeilen-Zähler

Wird erhöht, wenn Code hinzugefügt oder entfernt wird.

Attribute:

- Alle [Standardattribute](#)
- `type` : (`"added"` , `"removed"`)

Pull-Request-Zähler

Wird erhöht, wenn Pull Requests über Claude Code erstellt werden.

Attribute:

- Alle [Standardattribute](#)

Commit-Zähler

Wird erhöht, wenn Git-Commits über Claude Code erstellt werden.

Attribute:

- Alle [Standardattribute](#)

Kostenzähler

Wird nach jeder API-Anfrage erhöht.

Attribute:

- Alle [Standardattribute](#)
- `model`: Modellkennung (zum Beispiel "claude-sonnet-4-5-20250929")

Token-Zähler

Wird nach jeder API-Anfrage erhöht.

Attribute:

- Alle [Standardattribute](#)
- `type`: ("input", "output", "cacheRead", "cacheCreation")
- `model`: Modellkennung (zum Beispiel "claude-sonnet-4-5-20250929")

Code-Edit-Tool-Entscheidungszähler

Wird erhöht, wenn der Benutzer die Verwendung des Edit-, Write- oder NotebookEdit-Tools akzeptiert oder ablehnt.

Attribute:

- Alle [Standardattribute](#)
- `tool`: Toolname ("Edit", "Write", "NotebookEdit")
- `decision`: Benutzerentscheidung ("accept", "reject")
- `language`: Programmiersprache der bearbeiteten Datei (zum Beispiel "TypeScript", "Python", "JavaScript", "Markdown"). Gibt "unknown" für nicht erkannte Dateierweiterungen zurück.

Aktive-Zeit-Zähler

Verfolgt die tatsächliche Zeit, die aktiv Claude Code verwendet wird (nicht Leerlaufzeit). Diese Metrik wird während Benutzerinteraktionen wie dem Eingeben von Prompts oder dem Empfangen von Antworten erhöht.

Attribute:

- Alle [Standardattribute](#)

Ereignisse

Claude Code exportiert die folgenden Ereignisse über OpenTelemetry Logs/Events (wenn `OTEL_LOGS_EXPORTER` konfiguriert ist):

Benutzer-Prompt-Ereignis

Protokolliert, wenn ein Benutzer einen Prompt einreicht.

Ereignisname: `claude_code.user_prompt`

Attribute:

- Alle [Standardattribute](#)
- `event.name` : `"user_prompt"`
- `event.timestamp` : ISO 8601-Zeitstempel
- `prompt_length` : Länge des Prompts
- `prompt` : Prompt-Inhalt (standardmäßig geschwärzt, aktivieren Sie mit `OTEL_LOG_USER_PROMPTS=1`)

Tool-Ergebnis-Ereignis

Protokolliert, wenn ein Tool die Ausführung abgeschlossen hat.

Ereignisname: `claude_code.tool_result`

Attribute:

- Alle [Standardattribute](#)
- `event.name` : `"tool_result"`
- `event.timestamp` : ISO 8601-Zeitstempel
- `tool_name` : Name des Tools
- `success` : `"true"` oder `"false"`
- `duration_ms` : Ausführungszeit in Millisekunden
- `error` : Fehlermeldung (falls fehlgeschlagen)
- `decision` : Entweder `"accept"` oder `"reject"`
- `source` : Entscheidungsquelle - `"config"`, `"user_permanent"`, `"user_temporary"`, `"user_abort"` oder `"user_reject"`
- `tool_parameters` : JSON-Zeichenkette mit Tool-spezifischen Parametern (falls verfügbar)
 - Für Bash-Tool: enthält `bash_command`, `full_command`, `timeout`, `description`, `sandbox`

API-Anfrage-Ereignis

Protokolliert für jede API-Anfrage an Claude.

Ereignisname: `claude_code.api_request`

Attribute:

- Alle [Standardattribute](#)
- `event.name` : `"api_request"`
- `event.timestamp` : ISO 8601-Zeitstempel
- `model` : Verwendetes Modell (zum Beispiel "claude-sonnet-4-5-20250929")
- `cost_usd` : Geschätzte Kosten in USD
- `duration_ms` : Anfragedauer in Millisekunden
- `input_tokens` : Anzahl der Eingabe-Token
- `output_tokens` : Anzahl der Ausgabe-Token
- `cache_read_tokens` : Anzahl der aus dem Cache gelesenen Token
- `cache_creation_tokens` : Anzahl der Token, die für die Cache-Erstellung verwendet werden

API-Fehler-Ereignis

Protokolliert, wenn eine API-Anfrage an Claude fehlschlägt.

Ereignisname: `claude_code.api_error`

Attribute:

- Alle [Standardattribute](#)
- `event.name` : `"api_error"`
- `event.timestamp` : ISO 8601-Zeitstempel
- `model` : Verwendetes Modell (zum Beispiel "claude-sonnet-4-5-20250929")
- `error` : Fehlermeldung
- `status_code` : HTTP-Statuscode (falls zutreffend)
- `duration_ms` : Anfragedauer in Millisekunden
- `attempt` : Versuchsnummer (für wiederholte Anfragen)

Tool-Entscheidungs-Ereignis

Protokolliert, wenn eine Tool-Berechtigungsentscheidung getroffen wird (akzeptieren/ablehnen).

Ereignisname: `claude_code.tool_decision`

Attribute:

- Alle [Standardattribute](#)
- `event.name` : "tool_decision"
- `event.timestamp` : ISO 8601-Zeitstempel
- `tool_name` : Name des Tools (zum Beispiel "Read", "Edit", "Write", "NotebookEdit")
- `decision` : Entweder "accept" oder "reject"
- `source` : Entscheidungsquelle - "config", "user_permanent", "user_temporary", "user_abort" oder "user_reject"

Interpretation von Metriken- und Ereignisdaten

Die von Claude Code exportierten Metriken bieten wertvolle Einblicke in Nutzungsmuster und Produktivität. Hier sind einige häufige Visualisierungen und Analysen, die Sie erstellen können:

Nutzungsüberwachung

Metrik	Analysemöglichkeit
<code>claude_code.token.usage</code>	Aufschlüsselung nach <code>type</code> (input/output), Benutzer, Team oder Modell
<code>claude_code.session.count</code>	Verfolgung der Akzeptanz und des Engagements im Laufe der Zeit
<code>claude_code.lines_of_code.count</code>	Messung der Produktivität durch Verfolgung von Code-Hinzufügungen/Entfernungen
<code>claude_code.commit.count</code> & <code>claude_code.pull_request.count</code>	Verständnis der Auswirkungen auf Entwicklungs-Workflows

Kostenüberwachung

Die Metrik `claude_code.cost.usage` hilft bei:

- Verfolgung von Nutzungstrends über Teams oder Einzelpersonen hinweg
- Identifikation von Sitzungen mit hoher Nutzung zur Optimierung

Note:

Kostenmetriken sind Näherungswerte. Für offizielle Abrechnungsdaten konsultieren Sie Ihren API-Anbieter (Claude Console, AWS Bedrock oder Google Cloud Vertex).

Warnungen und Segmentierung

Häufige Warnungen, die Sie in Betracht ziehen sollten:

- Kostensteigerungen
- Ungewöhnlicher Token-Verbrauch
- Hohes Sitzungsvolumen von bestimmten Benutzern

Alle Metriken können nach `user.account_uuid`, `organization.id`, `session.id`, `model` und `app.version` segmentiert werden.

Ereignisanalyse

Die Ereignisdaten bieten detaillierte Einblicke in Claude Code-Interaktionen:

Tool-Nutzungsmuster: Analysieren Sie Tool-Ergebnis-Ereignisse, um zu identifizieren:

- Am häufigsten verwendete Tools
- Tool-Erfolgsquoten
- Durchschnittliche Tool-Ausführungszeiten
- Fehlermuster nach Tool-Typ

Leistungsüberwachung: Verfolgen Sie API-Anfrage-Dauern und Tool-Ausführungszeiten, um Leistungsengpässe zu identifizieren.

Backend-Überlegungen

Ihre Wahl des Metriken- und Logs-Backends bestimmt die Arten von Analysen, die Sie durchführen können:

Für Metriken

- **Zeitreihendatenbanken (zum Beispiel Prometheus):** Ratenberechnungen, aggregierte Metriken
- **Spaltenorientierte Speicher (zum Beispiel ClickHouse):** Komplexe Abfragen, eindeutige Benutzeranalyse
- **Vollständige Observability-Plattformen (zum Beispiel Honeycomb, Datadog):** Erweiterte Abfragen, Visualisierung, Warnungen

Für Ereignisse/Logs

- **Log-Aggregationssysteme (zum Beispiel Elasticsearch, Loki):** Volltextsuche, Log-Analyse

- **Spaltenorientierte Speicher (zum Beispiel ClickHouse):** Strukturierte Ereignisanalyse
- **Vollständige Observability-Plattformen (zum Beispiel Honeycomb, Datadog):** Korrelation zwischen Metriken und Ereignissen

Für Organisationen, die Daily/Weekly/Monthly Active User (DAU/WAU/MAU) Metriken benötigen, sollten Sie Backends in Betracht ziehen, die effiziente Abfragen eindeutiger Werte unterstützen.

Dienstinformationen

Alle Metriken und Ereignisse werden mit den folgenden Ressourcenattributen exportiert:

- `service.name` : `claude-code`
- `service.version` : Aktuelle Claude Code-Version
- `os.type` : Betriebssystemtyp (zum Beispiel `linux` , `darwin` , `windows`)
- `os.version` : Betriebssystem-Versionsnummer
- `host.arch` : Host-Architektur (zum Beispiel `amd64` , `arm64`)
- `wsl.version` : WSL-Versionsnummer (nur vorhanden, wenn auf Windows Subsystem for Linux ausgeführt)
- Meter-Name: `com.anthropic.claude_code`

ROI-Messung-Ressourcen

Für einen umfassenden Leitfaden zur Messung der Kapitalrendite für Claude Code, einschließlich Telemetrie-Setup, Kostenanalyse, Produktivitätsmetriken und automatisierter Berichterstattung, siehe den [Claude Code ROI Measurement Guide](#). Dieses Repository bietet einsatzbereite Docker Compose-Konfigurationen, Prometheus- und OpenTelemetry-Setups sowie Vorlagen zur Generierung von Produktivitätsberichten, die in Tools wie Linear integriert sind.

Sicherheits- /Datenschutzüberlegungen

- Telemetrie ist opt-in und erfordert explizite Konfiguration
- Vertrauliche Informationen wie API-Schlüssel oder Dateiinhalte werden niemals in Metriken oder Ereignisse einbezogen
- Benutzer-Prompt-Inhalte werden standardmäßig geschwärzt - nur die Prompt-Länge wird aufgezeichnet. Um Benutzer-Prompt-Protokollierung zu aktivieren, setzen Sie `OTEL_LOG_USER_PROMPTS=1`

Überwachung von Claude Code auf Amazon Bedrock

Für detaillierte Anleitung zur Überwachung der Claude Code-Nutzung für Amazon Bedrock siehe [Claude Code Monitoring Implementation \(Bedrock\)](#).

Übersicht zur Enterprise-Bereitstellung

Erfahren Sie, wie Claude Code mit verschiedenen Drittanbieter-Services und Infrastrukturen integriert werden kann, um Enterprise-Bereitstellungsanforderungen zu erfüllen.

Diese Seite bietet einen Überblick über verfügbare Bereitstellungsoptionen und hilft Ihnen, die richtige Konfiguration für Ihre Organisation zu wählen.

Anbietervergleich

Funktion Anthropic Amazon Bedrock Google Vertex AI Microsoft Foundry

Regionen Unterstützte [Länder](#) Mehrere AWS [Regionen](#) Mehrere GCP [Regionen](#) Mehrere Azure [Regionen](#)

Prompt-Caching Standardmäßig aktiviert Standardmäßig aktiviert Standardmäßig aktiviert Standardmäßig aktiviert

Authentifizierung API-Schlüssel API-Schlüssel oder AWS-Anmeldedaten GCP-Anmeldedaten API-Schlüssel oder Microsoft Entra ID

Kostenverfolgung Dashboard AWS Cost Explorer GCP Billing Azure Cost Management

Enterprise-Funktionen Teams, Nutzungsüberwachung IAM-Richtlinien, CloudTrail IAM-Rollen, Cloud Audit Logs RBAC-Richtlinien, Azure Monitor

Cloud-Anbieter

- [Amazon Bedrock](#) Nutzen Sie Claude-Modelle über AWS-Infrastruktur mit API-Schlüssel- oder IAM-basierter Authentifizierung und AWS-nativer Überwachung
- [Google Vertex AI](#) Greifen Sie auf Claude-Modelle über Google Cloud Platform mit Enterprise-Sicherheit und Compliance zu
- [Microsoft Foundry](#) Greifen Sie auf Claude über Azure mit API-Schlüssel- oder Microsoft Entra ID-Authentifizierung und Azure-Abrechnung zu

Unternehmensinfrastruktur

- [Enterprise Network](#) Konfigurieren Sie Claude Code für die Zusammenarbeit mit den Proxy-Servern und SSL/TLS-Anforderungen Ihrer Organisation

- **LLM Gateway** Stellen Sie zentralisierten Modellzugriff mit Nutzungsverfolgung, Budgetierung und Audit-Protokollierung bereit

Konfigurationsübersicht

Claude Code unterstützt flexible Konfigurationsoptionen, mit denen Sie verschiedene Anbieter und Infrastrukturen kombinieren können:

Note:

Verstehen Sie den Unterschied zwischen:

- **Unternehmens-Proxy:** Ein HTTP/HTTPS-Proxy für die Weiterleitung von Datenverkehr (festgelegt über `HTTPS_PROXY` oder `HTTP_PROXY`)
- **LLM Gateway:** Ein Service, der die Authentifizierung verwaltet und anbieterkompatible Endpunkte bereitstellt (festgelegt über `ANTHROPIC_BASE_URL` , `ANTHROPIC_BEDROCK_BASE_URL` oder `ANTHROPIC_VERTEX_BASE_URL`)

Beide Konfigurationen können zusammen verwendet werden.

Bedrock mit Unternehmens-Proxy verwenden

Leiten Sie Bedrock-Datenverkehr über einen Unternehmens-HTTP/HTTPS-Proxy weiter:

```
## Bedrock aktivieren
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1

## Unternehmens-Proxy konfigurieren
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Bedrock mit LLM Gateway verwenden

Verwenden Sie einen Gateway-Service, der Bedrock-kompatible Endpunkte bereitstellt:

```
## Bedrock aktivieren
export CLAUDE_CODE_USE_BEDROCK=1

## LLM Gateway konfigurieren
export ANTHROPIC_BEDROCK_BASE_URL='https://your-llm-gateway.com/bedrock'
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1 # Falls Gateway AWS-Authentifizierung
verwaltet
```

Foundry mit Unternehmens-Proxy verwenden

Leiten Sie Azure-Datenverkehr über einen Unternehmens-HTTP/HTTPS-Proxy weiter:

```
## Microsoft Foundry aktivieren
export CLAUDE_CODE_USE_FOUNDRY=1
export ANTHROPIC_FOUNDRY_RESOURCE=your-resource
export ANTHROPIC_FOUNDRY_API_KEY=your-api-key # Oder weglassen für Entra ID-
Authentifizierung

## Unternehmens-Proxy konfigurieren
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Foundry mit LLM Gateway verwenden

Verwenden Sie einen Gateway-Service, der Azure-kompatible Endpunkte bereitstellt:

```
## Microsoft Foundry aktivieren
export CLAUDE_CODE_USE_FOUNDRY=1

## LLM Gateway konfigurieren
export ANTHROPIC_FOUNDRY_BASE_URL='https://your-llm-gateway.com'
export CLAUDE_CODE_SKIP_FOUNDRY_AUTH=1 # Falls Gateway Azure-Authentifizierung
verwaltet
```

Vertex AI mit Unternehmens-Proxy verwenden

Leiten Sie Vertex AI-Datenverkehr über einen Unternehmens-HTTP/HTTPS-Proxy weiter:

```
## Vertex aktivieren
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
export ANTHROPIC_VERTEX_PROJECT_ID=your-project-id

## Unternehmens-Proxy konfigurieren
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Vertex AI mit LLM Gateway verwenden

Kombinieren Sie Google Vertex AI-Modelle mit einem LLM Gateway für zentralisierte Verwaltung:

```
## Vertex aktivieren
export CLAUDE_CODE_USE_VERTEX=1

## LLM Gateway konfigurieren
export ANTHROPIC_VERTEX_BASE_URL='https://your-llm-gateway.com/vertex'
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1 # Falls Gateway GCP-Authentifizierung
verwaltet
```

Authentifizierungskonfiguration

Claude Code verwendet `ANTHROPIC_AUTH_TOKEN` für den `Authorization`-Header, wenn erforderlich. Die `SKIP_AUTH`-Flags (`CLAUDE_CODE_SKIP_BEDROCK_AUTH` , `CLAUDE_CODE_SKIP_VERTEX_AUTH`) werden in LLM Gateway-Szenarien verwendet, in denen das Gateway die Anbieter-Authentifizierung verwaltet.

Wahl der richtigen Bereitstellungsconfiguration

Berücksichtigen Sie diese Faktoren bei der Auswahl Ihres Bereitstellungsansatzes:

Direkter Anbieter-Zugriff

Am besten für Organisationen, die:

- Das einfachste Setup wünschen
- Bereits über AWS- oder GCP-Infrastruktur verfügen
- Anbieter-native Überwachung und Compliance benötigen

Unternehmens-Proxy

Am besten für Organisationen, die:

- Bereits Anforderungen für Unternehmens-Proxy haben
- Datenverkehrsüberwachung und Compliance benötigen
- Alle Datenverkehr über bestimmte Netzwerkpfade leiten müssen

LLM Gateway

Am besten für Organisationen, die:

- Nutzungsverfolgung über Teams hinweg benötigen
- Dynamisch zwischen Modellen wechseln möchten
- Benutzerdefinierte Ratenbegrenzung oder Budgets benötigen
- Zentralisierte Authentifizierungsverwaltung benötigen

Debugging

Beim Debuggen Ihrer Bereitstellung:

- Verwenden Sie den `claude /status` [Schrägstrich-Befehl](#). Dieser Befehl bietet Observability in alle angewendeten Authentifizierungs-, Proxy- und URL-Einstellungen.
- Setzen Sie die Umgebungsvariable `export ANTHROPIC_LOG=debug`, um Anfragen zu protokollieren.

Best Practices für Organisationen

1. Investieren Sie in Dokumentation und Speicher

Wir empfehlen dringend, in Dokumentation zu investieren, damit Claude Code Ihre Codebasis versteht. Organisationen können CLAUDE.md-Dateien auf mehreren Ebenen bereitstellen:

- **Organisationsweit:** Stellen Sie in Systemverzeichnissen wie `/Library/Application Support/ClaudeCode/CLAUDE.md` (macOS) bereit, um unternehmensweite Standards zu setzen
 - **Repository-Ebene:** Erstellen Sie `CLAUDE.md`-Dateien in Repository-Wurzeln mit Projektarchitektur, Build-Befehlen und Beitragsrichtlinien. Checken Sie diese in die Versionskontrolle ein, damit alle Benutzer davon profitieren
- [Weitere Informationen.](#)

2. Vereinfachen Sie die Bereitstellung

Wenn Sie eine benutzerdefinierte Entwicklungsumgebung haben, stellen wir fest, dass die Schaffung einer „Ein-Klick“-Möglichkeit zur Installation von Claude Code der Schlüssel zum Wachstum der Akzeptanz in einer Organisation ist.

3. Beginnen Sie mit gesteuerter Nutzung

Ermutigen Sie neue Benutzer, Claude Code für Codebasis-Q&A oder bei kleineren Fehlerbehebungen oder Feature-Anfragen zu versuchen. Bitten Sie Claude Code, einen Plan zu erstellen. Überprüfen Sie Claudes Vorschläge und geben Sie Feedback, wenn es falsch ist. Mit der Zeit, wenn Benutzer dieses neue Paradigma besser verstehen, werden sie effektiver darin, Claude Code agentischer laufen zu lassen.

4. Konfigurieren Sie Sicherheitsrichtlinien

Sicherheitsteams können verwaltete Berechtigungen für das konfigurieren, was Claude Code darf und nicht darf, was nicht durch lokale Konfiguration überschrieben werden kann. [Weitere Informationen](#).

5. Nutzen Sie MCP für Integrationen

MCP ist eine großartige Möglichkeit, Claude Code mehr Informationen zu geben, z. B. die Verbindung zu Ticket-Management-Systemen oder Fehlerprotokollen. Wir empfehlen, dass ein zentrales Team MCP-Server konfiguriert und eine `.mcp.json`-Konfiguration in die Codebasis eincheckt, damit alle Benutzer davon profitieren. [Weitere Informationen](#).

Bei Anthropic vertrauen wir Claude Code, um die Entwicklung in jeder Anthropic-Codebasis voranzutreiben. Wir hoffen, dass Sie Claude Code genauso gerne verwenden wie wir.

Nächste Schritte

- [Amazon Bedrock einrichten](#) für AWS-native Bereitstellung
- [Google Vertex AI konfigurieren](#) für GCP-Bereitstellung
- [Microsoft Foundry einrichten](#) für Azure-Bereitstellung
- [Enterprise Network konfigurieren](#) für Netzwerkanforderungen
- [LLM Gateway bereitstellen](#) für Enterprise-Verwaltung
- [Einstellungen](#) für Konfigurationsoptionen und Umgebungsvariablen

Serververwaltete Einstellungen konfigurieren (öffentliche Beta)

Konfigurieren Sie Claude Code zentral für Ihre Organisation durch serververwaltete Einstellungen, ohne dass eine Geräteverwaltungsinfrastruktur erforderlich ist.

Serververwaltete Einstellungen ermöglichen es Administratoren, Claude Code zentral über eine webbasierte Schnittstelle auf Claude.ai zu konfigurieren. Claude Code-Clients erhalten diese Einstellungen automatisch, wenn sich Benutzer mit ihren Organisationsanmeldedaten authentifizieren.

Dieser Ansatz ist für Organisationen konzipiert, die keine Geräteverwaltungsinfrastruktur haben oder Einstellungen für Benutzer auf nicht verwalteten Geräten verwalten müssen.

Note:

Serververwaltete Einstellungen befinden sich in der öffentlichen Beta und sind für [Claude for Teams](#) und [Claude for Enterprise](#) Kunden verfügbar. Funktionen können sich vor der allgemeinen Verfügbarkeit noch ändern.

Anforderungen

Um serververwaltete Einstellungen zu verwenden, benötigen Sie:

- Claude for Teams oder Claude for Enterprise Plan
- Claude Code Version 2.1.38 oder später für Claude for Teams oder Version 2.1.30 oder später für Claude for Enterprise
- Netzwerkzugriff auf api.anthropic.com

Wählen Sie zwischen serververwalteten und endpunktverwalteten Einstellungen

Claude Code unterstützt zwei Ansätze für zentralisierte Konfiguration. Serververwaltete Einstellungen liefern Konfiguration von Anthropics Servern. [Endpunktverwaltete Einstellungen](#) werden direkt auf Geräten über native Betriebssystemrichtlinien (macOS verwaltete Einstellungen, Windows-Registrierung) oder verwaltete Einstellungsdateien bereitgestellt.

Ansatz	Am besten für	Sicherheitsmodell
Serververwaltete Einstellungen	Organisationen ohne MDM oder Benutzer auf nicht verwalteten Geräten	Einstellungen werden von Anthropic's Servern zum Authentifizierungszeitpunkt bereitgestellt
Endpunktverwaltete Einstellungen	Organisationen mit MDM oder Endpunktverwaltung	Einstellungen werden auf Geräten über MDM-Konfigurationsprofile, Registrierungsrichtlinien oder verwaltete Einstellungsdateien bereitgestellt

Wenn Ihre Geräte in einer MDM- oder Endpunktverwaltungslösung registriert sind, bieten endpunktverwaltete Einstellungen stärkere Sicherheitsgarantien, da die Einstellungsdatei auf Betriebssystemebene vor Benutzermodifikationen geschützt werden kann.

Serververwaltete Einstellungen konfigurieren

Step 1: Öffnen Sie die Admin-Konsole

Navigieren Sie in [Claude.ai](#) zu **Admin-Einstellungen** > **Claude Code** > **Verwaltete Einstellungen**.

Step 2: Definieren Sie Ihre Einstellungen

Fügen Sie Ihre Konfiguration als JSON hinzu. Alle [in settings.json verfügbaren Einstellungen](#) werden unterstützt, einschließlich [nur verwaltete Einstellungen](#) wie `disableBypassPermissionsMode`.

Dieses Beispiel erzwingt eine Berechtigungsverweigerungsliste und verhindert, dass Benutzer Berechtigungen umgehen:

```
{
  "permissions": {
    "deny": [
      "Bash(curl *)",
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)"
    ]
  },
  "disableBypassPermissionsMode": "disable"
}
```

Step 3: Speichern und bereitstellen

Speichern Sie Ihre Änderungen. Claude Code-Clients erhalten die aktualisierten Einstellungen beim nächsten Start oder während des stündlichen Abrufzyklus.

Einstellungsbereitstellung überprüfen

Um zu bestätigen, dass Einstellungen angewendet werden, bitten Sie einen Benutzer, Claude Code neu zu starten. Wenn die Konfiguration Einstellungen enthält, die den [Sicherheitsgenehmigungsdialo](#)g auslösen, sieht der Benutzer beim Start eine Eingabeaufforderung, die die verwalteten Einstellungen beschreibt. Sie können auch überprüfen, dass verwaltete Berechtigungsregeln aktiv sind, indem Sie einen Benutzer `/permissions` ausführen lassen, um seine geltenden Berechtigungsregeln anzuzeigen.

Zugriffskontrolle

Die folgenden Rollen können serververwaltete Einstellungen verwalten:

- **Primärer Eigentümer**
- **Eigentümer**

Beschränken Sie den Zugriff auf vertrauenswürdige Personal, da Einstellungsänderungen für alle Benutzer in der Organisation gelten.

Aktuelle Einschränkungen

Serververwaltete Einstellungen haben während der Beta-Phase die folgenden Einschränkungen:

- Einstellungen gelten einheitlich für alle Benutzer in der Organisation. Pro-Gruppen-Konfigurationen werden noch nicht unterstützt.

- [MCP-Serverkonfigurationen](#) können nicht über serververwaltete Einstellungen verteilt werden.

Einstellungsbereitstellung

Einstellungspriorität

Serververwaltete Einstellungen und [endpunktverwaltete Einstellungen](#) nehmen beide die höchste Ebene in der Claude Code [Einstellungshierarchie](#) ein. Keine andere Einstellungsebene kann sie überschreiben, einschließlich Befehlszeilenargumenten. Wenn beide vorhanden sind, haben serververwaltete Einstellungen Vorrang und endpunktverwaltete Einstellungen werden nicht verwendet.

Abruf- und Caching-Verhalten

Claude Code ruft Einstellungen beim Start von Anthropic Servern ab und fragt stündlich während aktiver Sitzungen nach Updates ab.

Erster Start ohne zwischengespeicherte Einstellungen:

- Claude Code ruft Einstellungen asynchron ab
- Wenn der Abruf fehlschlägt, wird Claude Code ohne verwaltete Einstellungen fortgesetzt
- Es gibt ein kurzes Fenster, bevor Einstellungen geladen werden, in dem Einschränkungen noch nicht erzwungen werden

Nachfolgende Starts mit zwischengespeicherten Einstellungen:

- Zwischengespeicherte Einstellungen werden beim Start sofort angewendet
- Claude Code ruft frische Einstellungen im Hintergrund ab
- Zwischengespeicherte Einstellungen bleiben bei Netzwerkfehlern erhalten

Claude Code wendet Einstellungsaktualisierungen automatisch ohne Neustart an, außer für erweiterte Einstellungen wie OpenTelemetry-Konfiguration, die einen vollständigen Neustart erfordern, um wirksam zu werden.

Sicherheitsgenehmigungsdialoge

Bestimmte Einstellungen, die Sicherheitsrisiken darstellen könnten, erfordern explizite Benutzergenehmigung, bevor sie angewendet werden:

- **Shell-Befehlseinstellungen:** Einstellungen, die Shell-Befehle ausführen
- **Benutzerdefinierte Umgebungsvariablen:** Variablen, die nicht in der bekannten sicheren Zulassungsliste enthalten sind

• **Hook-Konfigurationen:** jede Hook-Definition

Wenn diese Einstellungen vorhanden sind, sehen Benutzer einen Sicherheitsdialog, der erklärt, was konfiguriert wird. Benutzer müssen genehmigen, um fortzufahren. Wenn ein Benutzer die Einstellungen ablehnt, wird Claude Code beendet.

Note:

Im nicht-interaktiven Modus mit dem `-p` Flag überspringt Claude Code Sicherheitsdialoge und wendet Einstellungen ohne Benutzergenehmigung an.

Plattformverfügbarkeit

Serververwaltete Einstellungen erfordern eine direkte Verbindung zu `api.anthropic.com` und sind nicht verfügbar, wenn Drittanbieter-Modellprovider verwendet werden:

- Amazon Bedrock
- Google Vertex AI
- Microsoft Foundry
- Benutzerdefinierte API-Endpunkte über `ANTHROPIC_BASE_URL` oder [LLM-Gateways](#)

Audit-Protokollierung

Audit-Log-Ereignisse für Einstellungsänderungen sind über die Compliance-API oder den Audit-Log-Export verfügbar. Kontaktieren Sie Ihr Anthropic-Kontoteam für Zugriff.

Audit-Ereignisse enthalten den Typ der durchgeführten Aktion, das Konto und das Gerät, das die Aktion durchgeführt hat, sowie Verweise auf die vorherigen und neuen Werte.

Sicherheitsüberlegungen

Serververwaltete Einstellungen bieten zentralisierte Richtliniendurchsetzung, funktionieren aber als clientseitige Kontrolle. Auf nicht verwalteten Geräten können Benutzer mit Admin- oder Sudo-Zugriff die Claude Code-Binärdatei, das Dateisystem oder die Netzwerkkonfiguration ändern.

Szenario	Verhalten
Benutzer bearbeitet die zwischengespeicherte Einstellungsdatei	Manipulierte Datei wird beim Start angewendet, aber korrekte Einstellungen werden beim nächsten Serverfetch wiederhergestellt

Szenario	Verhalten
Benutzer löscht die zwischengespeicherte Einstellungsdatei	Verhalten beim ersten Start tritt auf: Einstellungen werden asynchron abgerufen mit einem kurzen nicht erzwungenen Fenster
API ist nicht verfügbar	Zwischengespeicherte Einstellungen werden angewendet, falls verfügbar, andernfalls werden verwaltete Einstellungen nicht erzwungen, bis der nächste erfolgreiche Abruf erfolgt
Benutzer authentifiziert sich mit einer anderen Organisation	Einstellungen werden nicht für Konten außerhalb der verwalteten Organisation bereitgestellt
Benutzer setzt eine nicht standardmäßige <code>ANTHROPIC_BASE_URL</code>	Serververwaltete Einstellungen werden umgangen, wenn Drittanbieter-API-Provider verwendet werden

Um Laufzeitkonfigurationsänderungen zu erkennen, verwenden Sie `ConfigChange hooks`, um Änderungen zu protokollieren oder nicht autorisierte Änderungen zu blockieren, bevor sie wirksam werden.

Für stärkere Durchsetzungsgarantien verwenden Sie [endpunktverwaltete Einstellungen](#) auf Geräten, die in einer MDM-Lösung registriert sind.

Siehe auch

Verwandte Seiten zur Verwaltung der Claude Code-Konfiguration:

- [Einstellungen](#): vollständige Konfigurationsreferenz einschließlich aller verfügbaren Einstellungen
- [Endpunktverwaltete Einstellungen](#): verwaltete Einstellungen, die von der IT auf Geräten bereitgestellt werden
- [Authentifizierung](#): Einrichtung des Benutzerzugriffs auf Claude Code
- [Sicherheit](#): Sicherheitsvorkehrungen und Best Practices

Part 11: Cloud Providers

Claude Code auf Amazon Bedrock

Erfahren Sie, wie Sie Claude Code über Amazon Bedrock konfigurieren, einschließlich Setup, IAM-Konfiguration und Fehlerbehebung.

Voraussetzungen

Bevor Sie Claude Code mit Bedrock konfigurieren, stellen Sie sicher, dass Sie über Folgendes verfügen:

- Ein AWS-Konto mit aktiviertem Bedrock-Zugriff
- Zugriff auf gewünschte Claude-Modelle (z. B. Claude Sonnet 4.6) in Bedrock
- AWS CLI installiert und konfiguriert (optional – nur erforderlich, wenn Sie keinen anderen Mechanismus zur Beschaffung von Anmeldedaten haben)
- Angemessene IAM-Berechtigungen

Note:

Wenn Sie Claude Code für mehrere Benutzer bereitstellen, [fixieren Sie Ihre Modellversionen](#), um Fehler zu vermeiden, wenn Anthropic neue Modelle veröffentlicht.

Setup

1. Anwendungsfalldetails einreichen

Erstmalige Benutzer von Anthropic-Modellen müssen Anwendungsfalldetails einreichen, bevor sie ein Modell aufrufen. Dies wird einmal pro Konto durchgeführt.

1. Stellen Sie sicher, dass Sie die richtigen IAM-Berechtigungen haben (siehe unten für weitere Informationen)
2. Navigieren Sie zur [Amazon Bedrock-Konsole](#)
3. Wählen Sie **Chat/Text playground**
4. Wählen Sie ein beliebiges Anthropic-Modell aus, und Sie werden aufgefordert, das Anwendungsfallformular auszufüllen

2. AWS-Anmeldedaten konfigurieren

Claude Code verwendet die Standard-AWS-SDK-Anmeldedatenkette. Richten Sie Ihre Anmeldedaten mit einer dieser Methoden ein:

Option A: AWS CLI-Konfiguration

```
aws configure
```

Option B: Umgebungsvariablen (Zugriffsschlüssel)

```
export AWS_ACCESS_KEY_ID=your-access-key-id
export AWS_SECRET_ACCESS_KEY=your-secret-access-key
export AWS_SESSION_TOKEN=your-session-token
```

Option C: Umgebungsvariablen (SSO-Profil)

```
aws sso login --profile=<your-profile-name>

export AWS_PROFILE=your-profile-name
```

Option D: AWS Management Console-Anmeldedaten

```
aws login
```

[Erfahren Sie mehr](#) über `aws login`.

Option E: Bedrock API-Schlüssel

```
export AWS_BEARER_TOKEN_BEDROCK=your-bedrock-api-key
```

Bedrock API-Schlüssel bieten eine einfachere Authentifizierungsmethode ohne vollständige AWS-Anmeldedaten. [Erfahren Sie mehr über Bedrock API-Schlüssel](#).

Erweiterte Anmeldedatenkonfiguration

Claude Code unterstützt die automatische Aktualisierung von Anmeldedaten für AWS SSO und Unternehmensidentitätsanbieter. Fügen Sie diese Einstellungen zu Ihrer Claude Code-Einstellungsdatei hinzu (siehe [Einstellungen](#) für Dateispeicherorte).

Wenn Claude Code erkennt, dass Ihre AWS-Anmeldedaten abgelaufen sind (entweder lokal basierend auf ihrem Zeitstempel oder wenn Bedrock einen Anmeldedatenfehler zurückgibt), führt es automatisch Ihre konfigurierten `awsAuthRefresh` - und/oder `awsCredentialExport` -Befehle aus, um neue Anmeldedaten zu erhalten, bevor die Anfrage erneut versucht wird.

Beispielkonfiguration

```
{
  "awsAuthRefresh": "aws sso login --profile myprofile",
  "env": {
    "AWS_PROFILE": "myprofile"
  }
}
```

Erklärung der Konfigurationseinstellungen

awsAuthRefresh : Verwenden Sie dies für Befehle, die das `.aws` -Verzeichnis ändern, z. B. zum Aktualisieren von Anmeldedaten, SSO-Cache oder Konfigurationsdateien. Die Ausgabe des Befehls wird dem Benutzer angezeigt, aber interaktive Eingaben werden nicht unterstützt. Dies funktioniert gut für browserbasierte SSO-Flows, bei denen die CLI eine URL oder einen Code anzeigt und Sie die Authentifizierung im Browser abschließen.

awsCredentialExport : Verwenden Sie dies nur, wenn Sie das `.aws` -Verzeichnis nicht ändern können und Anmeldedaten direkt zurückgeben müssen. Die Ausgabe wird stillschweigend erfasst und nicht dem Benutzer angezeigt. Der Befehl muss JSON in diesem Format ausgeben:

```
{
  "Credentials": {
    "AccessKeyId": "value",
    "SecretAccessKey": "value",
    "SessionToken": "value"
  }
}
```

3. Claude Code konfigurieren

Legen Sie die folgenden Umgebungsvariablen fest, um Bedrock zu aktivieren:

```
## Bedrock-Integration aktivieren
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1 # oder Ihre bevorzugte Region

## Optional: Region für das kleine/schnelle Modell (Haiku) überschreiben
export ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION=us-west-2
```

Beachten Sie beim Aktivieren von Bedrock für Claude Code Folgendes:

- `AWS_REGION` ist eine erforderliche Umgebungsvariable. Claude Code liest diese Einstellung nicht aus der `.aws`-Konfigurationsdatei.
- Bei Verwendung von Bedrock sind die `/login`- und `/logout`-Befehle deaktiviert, da die Authentifizierung über AWS-Anmeldedaten erfolgt.
- Sie können Einstellungsdateien für Umgebungsvariablen wie `AWS_PROFILE` verwenden, die Sie nicht an andere Prozesse weitergeben möchten. Weitere Informationen finden Sie unter [Einstellungen](#).

4. Modellversionen fixieren

Warning:

Fixieren Sie spezifische Modellversionen für jede Bereitstellung. Wenn Sie Modellalias (`sonnet`, `opus`, `haiku`) ohne Fixierung verwenden, versucht Claude Code möglicherweise, eine neuere Modellversion zu verwenden, die in Ihrem Bedrock-Konto nicht verfügbar ist, was bestehende Benutzer unterbricht, wenn Anthropic Updates veröffentlicht.

Legen Sie diese Umgebungsvariablen auf spezifische Bedrock-Modell-IDs fest:

```
export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-6-v1'
export ANTHROPIC_DEFAULT_SONNET_MODEL='us.anthropic.claude-sonnet-4-6'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'
```

Diese Variablen verwenden Cross-Region-Inferenzprofil-IDs (mit dem `us.`-Präfix). Wenn Sie ein anderes Regionspräfix oder Anwendungsinferenzprofile verwenden, passen Sie entsprechend an. Aktuelle und ältere Modell-IDs finden Sie unter [Modellübersicht](#). Siehe [Modellkonfiguration](#) für die vollständige Liste der Umgebungsvariablen.

Claude Code verwendet diese Standardmodelle, wenn keine Fixierungsvariablen gesetzt sind:

Modelltyp	Standardwert
Primäres Modell	<code>global.anthropic.claude-sonnet-4-6</code>
Kleines/schnelles Modell	<code>us.anthropic.claude-haiku-4-5-20251001-v1:0</code>

Um Modelle weiter anzupassen, verwenden Sie eine dieser Methoden:

```
## Verwendung der Inferenzprofil-ID
export ANTHROPIC_MODEL='global.anthropic.claude-sonnet-4-6'
export ANTHROPIC_SMALL_FAST_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'

## Verwendung des Anwendungsinferenzprofil-ARN
export ANTHROPIC_MODEL='arn:aws:bedrock:us-east-2:your-account-id:application-
inference-profile/your-model-id'

## Optional: Prompt Caching deaktivieren, falls erforderlich
export DISABLE_PROMPT_CACHING=1
```

Note:
[Prompt Caching](#) ist möglicherweise nicht in allen Regionen verfügbar.

Jede Modellversion einem Inferenzprofil zuordnen

Die Umgebungsvariablen `ANTHROPIC_DEFAULT_*_MODEL` konfigurieren ein Inferenzprofil pro Modellfamilie. Wenn Ihre Organisation mehrere Versionen derselben Familie in der `model`-Auswahl verfügbar machen muss, die jeweils zu ihrem eigenen Anwendungsinferenzprofil-ARN weitergeleitet werden, verwenden Sie stattdessen die `modelOverrides`-Einstellung in Ihrer [Einstellungsdatei](#).

Dieses Beispiel ordnet drei Opus-Versionen unterschiedlichen ARNs zu, damit Benutzer zwischen ihnen wechseln können, ohne die Inferenzprofile Ihrer Organisation zu umgehen:


```
{
  "modelOverrides": {
    "claude-opus-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-
inference-profile/opus-46-prod",
    "claude-opus-4-5-20251101": "arn:aws:bedrock:us-
east-2:123456789012:application-inference-profile/opus-45-prod",
    "claude-opus-4-1-20250805": "arn:aws:bedrock:us-
east-2:123456789012:application-inference-profile/opus-41-prod"
  }
}
```

Wenn ein Benutzer eine dieser Versionen in `/model` auswählt, ruft Claude Code Bedrock mit dem zugeordneten ARN auf. Versionen ohne Überschreibung fallen auf die integrierte Bedrock-Modell-ID oder ein beliebiges übereinstimmendes Inferenzprofil zurück, das beim Start erkannt wird. Siehe [Modell-IDs pro Version überschreiben](#) für Details, wie Überschreibungen mit `availableModels` und anderen Modelleinstellungen interagieren.

IAM-Konfiguration

Erstellen Sie eine IAM-Richtlinie mit den erforderlichen Berechtigungen für Claude Code:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowModelAndInferenceProfileAccess",
      "Effect": "Allow",
      "Action": [
        "bedrock:InvokeModel",
        "bedrock:InvokeModelWithResponseStream",
        "bedrock:ListInferenceProfiles"
      ],
      "Resource": [
        "arn:aws:bedrock:*:*:inference-profile/*",
        "arn:aws:bedrock:*:*:application-inference-profile/*",
        "arn:aws:bedrock:*:*:foundation-model/*"
      ]
    },
    {
      "Sid": "AllowMarketplaceSubscription",
      "Effect": "Allow",
      "Action": [
        "aws-marketplace:ViewSubscriptions",
        "aws-marketplace:Subscribe"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "bedrock.amazonaws.com"
        }
      }
    }
  ]
}

```

Für restriktivere Berechtigungen können Sie die Ressource auf spezifische Inferenzprofil-ARNs beschränken.

Weitere Details finden Sie in der [Bedrock IAM-Dokumentation](#).

Note:

Erstellen Sie ein dediziertes AWS-Konto für Claude Code, um die Kostenverfolgung und Zugriffskontrolle zu vereinfachen.

AWS Guardrails

[Amazon Bedrock Guardrails](#) ermöglichen es Ihnen, Inhaltsfilterung für Claude Code zu implementieren. Erstellen Sie einen Guardrail in der [Amazon Bedrock-Konsole](#), veröffentlichen Sie eine Version, und fügen Sie dann die Guardrail-Header zu Ihrer [Einstellungsdatei](#) hinzu. Aktivieren Sie Cross-Region-Inferenz auf Ihrem Guardrail, wenn Sie Cross-Region-Inferenzprofile verwenden.

Beispielkonfiguration:

```
{
  "env": {
    "ANTHROPIC_CUSTOM_HEADERS": "X-Amzn-Bedrock-GuardrailIdentifier: your-guardrail-id\nX-Amzn-Bedrock-GuardrailVersion: 1"
  }
}
```

Fehlerbehebung

Wenn Sie auf Regionsprobleme stoßen:

- Modellverfügbarkeit prüfen: `aws bedrock list-inference-profiles --region your-region`
- Zu einer unterstützten Region wechseln: `export AWS_REGION=us-east-1`
- Erwägen Sie die Verwendung von Inferenzprofilen für Cross-Region-Zugriff

Wenn Sie einen Fehler „on-demand throughput isn’t supported“ erhalten:

- Geben Sie das Modell als [Inferenzprofil](#)-ID an

Claude Code verwendet die Bedrock [Invoke API](#) und unterstützt die Converse API nicht.

Zusätzliche Ressourcen

- [Bedrock-Dokumentation](#)
- [Bedrock-Preisgestaltung](#)

- [Bedrock-Inferenzprofile](#)
- [Claude Code auf Amazon Bedrock: Schnellstartanleitung](#)
- [Claude Code Monitoring Implementation \(Bedrock\)](#)

Claude Code auf Google Vertex AI

Erfahren Sie, wie Sie Claude Code über Google Vertex AI konfigurieren, einschließlich Setup, IAM-Konfiguration und Fehlerbehebung.

Voraussetzungen

Bevor Sie Claude Code mit Vertex AI konfigurieren, stellen Sie sicher, dass Sie über Folgendes verfügen:

- Ein Google Cloud Platform (GCP)-Konto mit aktivierter Abrechnung
- Ein GCP-Projekt mit aktivierter Vertex AI API
- Zugriff auf gewünschte Claude-Modelle (z. B. Claude Sonnet 4.5)
- Google Cloud SDK (`gcloud`) installiert und konfiguriert
- Kontingent in der gewünschten GCP-Region zugewiesen

Regionskonfiguration

Claude Code kann sowohl mit Vertex AI [global](#) als auch mit regionalen Endpunkten verwendet werden.

Note:

Vertex AI unterstützt möglicherweise die Claude Code-Standardmodelle nicht in allen Regionen. Möglicherweise müssen Sie zu einer [unterstützten Region](#) oder einem [unterstützten Modell](#) wechseln.

Note:

Vertex AI unterstützt möglicherweise die Claude Code-Standardmodelle nicht auf globalen Endpunkten. Möglicherweise müssen Sie zu einem regionalen Endpunkt oder einem [unterstützten Modell](#) wechseln.

Setup

1. Vertex AI API aktivieren

Aktivieren Sie die Vertex AI API in Ihrem GCP-Projekt:

```
## Legen Sie Ihre Projekt-ID fest
gcloud config set project YOUR-PROJECT-ID

## Aktivieren Sie die Vertex AI API
gcloud services enable aiplatform.googleapis.com
```

2. Modellzugriff anfordern

Fordern Sie Zugriff auf Claude-Modelle in Vertex AI an:

1. Navigieren Sie zum [Vertex AI Model Garden](#)
2. Suchen Sie nach „Claude“-Modellen
3. Fordern Sie Zugriff auf gewünschte Claude-Modelle an (z. B. Claude Sonnet 4.5)
4. Warten Sie auf Genehmigung (kann 24–48 Stunden dauern)

3. GCP-Anmeldedaten konfigurieren

Claude Code verwendet die standardmäßige Google Cloud-Authentifizierung.

Weitere Informationen finden Sie in der [Google Cloud-Authentifizierungsdokumentation](#).

Note:

Bei der Authentifizierung verwendet Claude Code automatisch die Projekt-ID aus der Umgebungsvariablen `ANTHROPIC_VERTEX_PROJECT_ID`. Um dies zu überschreiben, legen Sie eine dieser Umgebungsvariablen fest: `GCLOUD_PROJECT`, `GOOGLE_CLOUD_PROJECT` oder `GOOGLE_APPLICATION_CREDENTIALS`.

4. Claude Code konfigurieren

Legen Sie die folgenden Umgebungsvariablen fest:

```
## Aktivieren Sie die Vertex AI-Integration
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=global
export ANTHROPIC_VERTEX_PROJECT_ID=YOUR-PROJECT-ID

## Optional: Deaktivieren Sie Prompt Caching bei Bedarf
export DISABLE_PROMPT_CACHING=1

## Wenn CLOUD_ML_REGION=global, überschreiben Sie die Region für nicht
unterstützte Modelle
export VERTEX_REGION_CLAUDE_3_5_HAIKU=us-east5

## Optional: Überschreiben Sie Regionen für andere spezifische Modelle
export VERTEX_REGION_CLAUDE_3_5_SONNET=us-east5
export VERTEX_REGION_CLAUDE_3_7_SONNET=us-east5
export VERTEX_REGION_CLAUDE_4_0_OPUS=europe-west1
export VERTEX_REGION_CLAUDE_4_0_SONNET=us-east5
export VERTEX_REGION_CLAUDE_4_1_OPUS=europe-west1
```

Note:

[Prompt Caching](#) wird automatisch unterstützt, wenn Sie das `cache_control` Ephemeral-Flag angeben. Um es zu deaktivieren, legen Sie `DISABLE_PROMPT_CACHING=1` fest. Für erhöhte Ratenlimits wenden Sie sich an den Google Cloud-Support.

Note:

Bei Verwendung von Vertex AI sind die Befehle `/login` und `/logout` deaktiviert, da die Authentifizierung über Google Cloud-Anmeldedaten erfolgt.

5. Modellkonfiguration

Claude Code verwendet diese Standardmodelle für Vertex AI:

Modelltyp	Standardwert
Primäres Modell	claude-sonnet-4-5@20250929
Kleines/schnelles Modell	claude-haiku-4-5@20251001

Note:

Für Vertex AI-Benutzer wird Claude Code nicht automatisch von Haiku 3.5 auf Haiku 4.5 aktualisiert. Um manuell zu einem neueren Haiku-Modell zu wechseln, legen Sie die Umgebungsvariable `ANTHROPIC_DEFAULT_HAIKU_MODEL` auf den vollständigen Modellnamen fest (z. B. `claude-haiku-4-5@20251001`).

Um Modelle anzupassen:

```
export ANTHROPIC_MODEL='claude-opus-4-1@20250805'  
export ANTHROPIC_SMALL_FAST_MODEL='claude-haiku-4-5@20251001'
```

IAM-Konfiguration

Weisen Sie die erforderlichen IAM-Berechtigungen zu:

Die Rolle `roles/aiplatform.user` umfasst die erforderlichen Berechtigungen:

- `aiplatform.endpoints.predict` - Erforderlich für Modellaufrufe und Token-Zählung

Für restriktivere Berechtigungen erstellen Sie eine benutzerdefinierte Rolle nur mit den oben genannten Berechtigungen.

Weitere Informationen finden Sie in der [Vertex IAM-Dokumentation](#).

Note:

Wir empfehlen, ein dediziertes GCP-Projekt für Claude Code zu erstellen, um die Kostenverfolgung und Zugriffskontrolle zu vereinfachen.

1M Token-Kontextfenster

Claude Sonnet 4 und Sonnet 4.5 unterstützen das [1M Token-Kontextfenster](#) auf Vertex AI.

Note:

Das 1M Token-Kontextfenster befindet sich derzeit in der Beta-Phase. Um das erweiterte Kontextfenster zu verwenden, fügen Sie den `context-1m-2025-08-07` Beta-Header in Ihre Vertex AI-Anfragen ein.

Fehlerbehebung

Wenn Sie auf Kontingentprobleme stoßen:

- Überprüfen Sie aktuelle Kontingente oder fordern Sie eine Kontingenterhöhung über die [Cloud Console](#) an

Wenn Sie auf „Modell nicht gefunden“ 404-Fehler stoßen:

- Bestätigen Sie, dass das Modell im [Model Garden](#) aktiviert ist
- Überprüfen Sie, dass Sie Zugriff auf die angegebene Region haben
- Wenn Sie `CLOUD_ML_REGION=global` verwenden, überprüfen Sie, dass Ihre Modelle globale Endpunkte im [Model Garden](#) unter „Unterstützte Funktionen“ unterstützen. Für Modelle, die globale Endpunkte nicht unterstützen, können Sie entweder:
 - Ein unterstütztes Modell über `ANTHROPIC_MODEL` oder `ANTHROPIC_SMALL_FAST_MODEL` angeben, oder
 - Einen regionalen Endpunkt mit `VERTEX_REGION_<MODEL_NAME>` Umgebungsvariablen festlegen

Wenn Sie auf 429-Fehler stoßen:

- Stellen Sie für regionale Endpunkte sicher, dass das primäre Modell und das kleine/schnelle Modell in Ihrer ausgewählten Region unterstützt werden
- Erwägen Sie, zu `CLOUD_ML_REGION=global` zu wechseln, um bessere Verfügbarkeit zu erreichen

Zusätzliche Ressourcen

- [Vertex AI-Dokumentation](#)
- [Vertex AI-Preisgestaltung](#)
- [Vertex AI-Kontingente und -Limits](#)

Claude Code auf Microsoft Foundry

Erfahren Sie, wie Sie Claude Code über Microsoft Foundry konfigurieren, einschließlich Setup, Konfiguration und Fehlerbehebung.

Voraussetzungen

Bevor Sie Claude Code mit Microsoft Foundry konfigurieren, stellen Sie sicher, dass Sie über Folgendes verfügen:

- Ein Azure-Abonnement mit Zugriff auf Microsoft Foundry
- RBAC-Berechtigungen zum Erstellen von Microsoft Foundry-Ressourcen und Bereitstellungen
- Azure CLI installiert und konfiguriert (optional - nur erforderlich, wenn Sie keinen anderen Mechanismus zum Abrufen von Anmeldedaten haben)

Setup

1. Microsoft Foundry-Ressource bereitstellen

Erstellen Sie zunächst eine Claude-Ressource in Azure:

1. Navigieren Sie zum [Microsoft Foundry-Portal](#)
2. Erstellen Sie eine neue Ressource und notieren Sie sich Ihren Ressourcennamen
3. Erstellen Sie Bereitstellungen für die Claude-Modelle:
 - Claude Opus
 - Claude Sonnet
 - Claude Haiku

2. Azure-Anmeldedaten konfigurieren

Claude Code unterstützt zwei Authentifizierungsmethoden für Microsoft Foundry. Wählen Sie die Methode, die Ihren Sicherheitsanforderungen am besten entspricht.

Option A: API-Schlüssel-Authentifizierung

1. Navigieren Sie zu Ihrer Ressource im Microsoft Foundry-Portal
2. Gehen Sie zum Abschnitt **Endpunkte und Schlüssel**
3. Kopieren Sie **API-Schlüssel**

4. Legen Sie die Umgebungsvariable fest:

```
export ANTHROPIC_FOUNDRY_API_KEY=your-azure-api-key
```

Option B: Microsoft Entra ID-Authentifizierung

Wenn `ANTHROPIC_FOUNDRY_API_KEY` nicht gesetzt ist, verwendet Claude Code automatisch die Azure SDK [Standard-Anmeldekette](#). Dies unterstützt eine Vielzahl von Methoden zur Authentifizierung lokaler und Remote-Workloads.

In lokalen Umgebungen können Sie häufig die Azure CLI verwenden:

```
az login
```

Note:

Bei Verwendung von Microsoft Foundry sind die Befehle `/login` und `/logout` deaktiviert, da die Authentifizierung über Azure-Anmeldedaten erfolgt.

3. Claude Code konfigurieren

Legen Sie die folgenden Umgebungsvariablen fest, um Microsoft Foundry zu aktivieren. Beachten Sie, dass die Namen Ihrer Bereitstellungen als Modellbezeichner in Claude Code festgelegt sind (möglicherweise optional, wenn Sie die vorgeschlagenen Bereitstellungsnamen verwenden).

```
## Microsoft Foundry-Integration aktivieren
export CLAUDE_CODE_USE_FOUNDRY=1

## Azure-Ressourcenname (ersetzen Sie {resource} durch Ihren Ressourcennamen)
export ANTHROPIC_FOUNDRY_RESOURCE={resource}

## Oder geben Sie die vollständige Basis-URL an:
## export ANTHROPIC_FOUNDRY_BASE_URL=https://{resource}.services.ai.azure.com

## Legen Sie Modelle auf die Bereitstellungsnamen Ihrer Ressource fest
export ANTHROPIC_DEFAULT_SONNET_MODEL='claude-sonnet-4-5'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='claude-haiku-4-5'
export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-1'
```

Weitere Informationen zu Modellkonfigurationsoptionen finden Sie unter [Modellkonfiguration](#).

Azure RBAC-Konfiguration

Die Standardrollen **Azure AI User** und **Cognitive Services User** enthalten alle erforderlichen Berechtigungen zum Aufrufen von Claude-Modellen.

Für restriktivere Berechtigungen erstellen Sie eine benutzerdefinierte Rolle mit Folgendem:

```
{
  "permissions": [
    {
      "dataActions": [
        "Microsoft.CognitiveServices/accounts/providers/*"
      ]
    }
  ]
}
```

Weitere Informationen finden Sie in der [Microsoft Foundry RBAC-Dokumentation](#).

Fehlerbehebung

Wenn Sie einen Fehler erhalten “Failed to get token from azureADTokenProvider: ChainedTokenCredential authentication failed”:

- Konfigurieren Sie Entra ID in der Umgebung, oder legen Sie **ANTHROPIC_FOUNDRY_API_KEY** fest.

Zusätzliche Ressourcen

- [Microsoft Foundry-Dokumentation](#)
- [Microsoft Foundry-Modelle](#)
- [Microsoft Foundry-Preise](#)

LLM-Gateway-Konfiguration

Erfahren Sie, wie Sie Claude Code für die Zusammenarbeit mit LLM-Gateway-Lösungen konfigurieren. Behandelt Gateway-Anforderungen, Authentifizierungskonfiguration, Modellauswahl und anbieter-spezifisches Endpoint-Setup.

LLM-Gateways bieten eine zentralisierte Proxy-Schicht zwischen Claude Code und Modellanbietern und bieten häufig:

- **Zentralisierte Authentifizierung** - Einzelner Punkt für die API-Schlüsselverwaltung
- **Nutzungsverfolgung** - Überwachen Sie die Nutzung über Teams und Projekte hinweg
- **Kostenkontrollen** - Implementieren Sie Budgets und Ratenlimits
- **Audit-Protokollierung** - Verfolgen Sie alle Modellinteraktionen zur Compliance
- **Modell-Routing** - Wechseln Sie zwischen Anbietern ohne Code-Änderungen

Gateway-Anforderungen

Damit ein LLM-Gateway mit Claude Code funktioniert, muss es die folgenden Anforderungen erfüllen:

API-Format

Das Gateway muss Clients mindestens eines der folgenden API-Formate bereitstellen:

1. **Anthropic Messages**: `/v1/messages` , `/v1/messages/count_tokens`
 - Muss Request-Header weiterleiten: `anthropic-beta` , `anthropic-version`
2. **Bedrock InvokeModel**: `/invoke` , `/invoke-with-response-stream`
 - Muss Request-Body-Felder beibehalten: `anthropic_beta` , `anthropic_version`
3. **Vertex rawPredict**: `:rawPredict` , `:streamRawPredict` , `/count-tokens:rawPredict`
 - Muss Request-Header weiterleiten: `anthropic-beta` , `anthropic-version`

Das Nichtweiterleiten von Headern oder das Nichtbeibehalten von Body-Feldern kann zu eingeschränkter Funktionalität oder der Unmöglichkeit führen, Claude Code-Funktionen zu nutzen.

Note:

Claude Code bestimmt, welche Funktionen aktiviert werden sollen, basierend auf dem API-Format. Bei Verwendung des Anthropic Messages-Formats mit Bedrock oder Vertex müssen Sie möglicherweise die Umgebungsvariable

`CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS=1` setzen.

Konfiguration

Modellauswahl

Standardmäßig verwendet Claude Code Standard-Modellnamen für das ausgewählte API-Format.

Wenn Sie benutzerdefinierte Modellnamen in Ihrem Gateway konfiguriert haben, verwenden Sie die in [Modellkonfiguration](#) dokumentierten Umgebungsvariablen, um Ihre benutzerdefinierten Namen zu entsprechen.

LiteLLM-Konfiguration

Note:

LiteLLM ist ein Drittanbieter-Proxy-Service. Anthropic befürwortet, wartet oder prüft nicht die Sicherheit oder Funktionalität von LiteLLM. Diese Anleitung wird zu Informationszwecken bereitgestellt und kann veraltet werden. Verwenden Sie sie nach eigenem Ermessen.

Voraussetzungen

- Claude Code auf die neueste Version aktualisiert
- LiteLLM Proxy Server bereitgestellt und zugänglich
- Zugriff auf Claude-Modelle über Ihren gewählten Anbieter

Grundlegende LiteLLM-Einrichtung

Konfigurieren Sie Claude Code:

Authentifizierungsmethoden

Statischer API-Schlüssel

Einfachste Methode mit einem festen API-Schlüssel:

```
## In Umgebung setzen
export ANTHROPIC_AUTH_TOKEN=sk-litellm-static-key

## Oder in Claude Code-Einstellungen
{
  "env": {
    "ANTHROPIC_AUTH_TOKEN": "sk-litellm-static-key"
  }
}
```

Dieser Wert wird als **Authorization**-Header gesendet.

Dynamischer API-Schlüssel mit Helper

Für rotierende Schlüssel oder Pro-Benutzer-Authentifizierung:

1. Erstellen Sie ein API-Schlüssel-Helper-Skript:

```
#!/bin/bash
## ~/bin/get-litellm-key.sh

## Beispiel: Schlüssel aus Vault abrufen
vault kv get -field=api_key secret/litellm/claude-code

## Beispiel: JWT-Token generieren
jwt encode \
  --secret="${JWT_SECRET}" \
  --exp="+1h" \
  '{"user":"'${USER}'","team":"engineering"}'
```

1. Konfigurieren Sie Claude Code-Einstellungen zur Verwendung des Helpers:

```
{
  "apiKeyHelper": "~/bin/get-litellm-key.sh"
}
```

1. Legen Sie das Token-Aktualisierungsintervall fest:

```
## Alle Stunde aktualisieren (3600000 ms)
export CLAUDE_CODE_API_KEY_HELPER_TTL_MS=3600000
```

Dieser Wert wird als `Authorization` - und `X-API-Key` -Header gesendet. Der `apiKeyHelper` hat eine niedrigere Priorität als `ANTHROPIC_AUTH_TOKEN` oder `ANTHROPIC_API_KEY`.

Einheitlicher Endpoint (empfohlen)

Verwendung von LiteLLMs [Anthropic-Format-Endpoint](#):

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000
```

Vorteile des einheitlichen Endpoints gegenüber Pass-Through-Endpoints:

- Lastverteilung
- Fallbacks
- Konsistente Unterstützung für Kosten-Tracking und End-Benutzer-Tracking

Anbieter-spezifische Pass-Through-Endpoints (Alternative)

Claude API über LiteLLM

Verwendung von [Pass-Through-Endpoint](#):

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000/anthropic
```

Amazon Bedrock über LiteLLM

Verwendung von [Pass-Through-Endpoint](#):

```
export ANTHROPIC_BEDROCK_BASE_URL=https://litellm-server:4000/bedrock
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1
export CLAUDE_CODE_USE_BEDROCK=1
```

Google Vertex AI über LiteLLM

Verwendung von [Pass-Through-Endpoint](#):


```
export ANTHROPIC_VERTEX_BASE_URL=https://litellm-server:4000/vertex_ai/v1
export ANTHROPIC_VERTEX_PROJECT_ID=your-gcp-project-id
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
```

Weitere detaillierte Informationen finden Sie in der [LiteLLM-Dokumentation](#).

Zusätzliche Ressourcen

- [LiteLLM-Dokumentation](#)
- [Claude Code-Einstellungen](#)
- [Enterprise-Netzwerkkonfiguration](#)
- [Übersicht über Drittanbieter-Integrationen](#)

Enterprise-Netzwerkconfiguration

Konfigurieren Sie Claude Code für Enterprise-Umgebungen mit Proxy-Servern, benutzerdefinierten Zertifizierungsstellen (CA) und gegenseitiger Transport Layer Security (mTLS)-Authentifizierung.

Claude Code unterstützt verschiedene Enterprise-Netzwerk- und Sicherheitskonfigurationen über Umgebungsvariablen. Dies umfasst das Routing von Datenverkehr über unternehmenseigene Proxy-Server, das Vertrauen in benutzerdefinierte Zertifizierungsstellen (CA) und die Authentifizierung mit gegenseitigen Transport Layer Security (mTLS)-Zertifikaten für erhöhte Sicherheit.

Note:

Alle auf dieser Seite gezeigten Umgebungsvariablen können auch in `settings.json` konfiguriert werden.

Proxy-Konfiguration

Umgebungsvariablen

Claude Code respektiert Standard-Proxy-Umgebungsvariablen:

```
## HTTPS-Proxy (empfohlen)
export HTTPS_PROXY=https://proxy.example.com:8080

## HTTP-Proxy (falls HTTPS nicht verfügbar)
export HTTP_PROXY=http://proxy.example.com:8080

## Proxy für spezifische Anfragen umgehen - durch Leerzeichen getrennt
export NO_PROXY="localhost 192.168.1.1 example.com .example.com"
## Proxy für spezifische Anfragen umgehen - durch Komma getrennt
export NO_PROXY="localhost,192.168.1.1,example.com,.example.com"
## Proxy für alle Anfragen umgehen
export NO_PROXY="*"
```

Note:

Claude Code unterstützt keine SOCKS-Proxies.

Basis-Authentifizierung

Wenn Ihr Proxy eine Basis-Authentifizierung erfordert, fügen Sie Anmeldedaten in die Proxy-URL ein:

```
export HTTPS_PROXY=http://username:password@proxy.example.com:8080
```

Warning:

Vermeiden Sie das Hartcodieren von Passwörtern in Skripten. Verwenden Sie stattdessen Umgebungsvariablen oder sichere Anmeldedatenspeicherung.

Tip:

Für Proxies, die erweiterte Authentifizierung erfordern (NTLM, Kerberos usw.), erwägen Sie die Verwendung eines LLM-Gateway-Dienstes, der Ihre Authentifizierungsmethode unterstützt.

Benutzerdefinierte CA-Zertifikate

Wenn Ihre Enterprise-Umgebung benutzerdefinierte CAs für HTTPS-Verbindungen verwendet (ob über einen Proxy oder direkten API-Zugriff), konfigurieren Sie Claude Code so, dass diese vertraut werden:

```
export NODE_EXTRA_CA_CERTS=/path/to/ca-cert.pem
```

mTLS-Authentifizierung

Für Enterprise-Umgebungen, die Client-Zertifikat-Authentifizierung erfordern:

```
## Client-Zertifikat für Authentifizierung
export CLAUDE_CODE_CLIENT_CERT=/path/to/client-cert.pem

## Privater Schlüssel des Clients
export CLAUDE_CODE_CLIENT_KEY=/path/to/client-key.pem

## Optional: Passphrase für verschlüsselten privaten Schlüssel
export CLAUDE_CODE_CLIENT_KEY_PASSPHRASE="your-passphrase"
```

Netzwerkzugriffanforderungen

Claude Code benötigt Zugriff auf die folgenden URLs:

- `api.anthropic.com` : Claude-API-Endpunkte
- `claude.ai` : Authentifizierung für claude.ai-Konten
- `platform.claude.com` : Authentifizierung für Anthropic Console-Konten

Stellen Sie sicher, dass diese URLs in Ihrer Proxy-Konfiguration und Firewall-Regeln auf die Whitelist gesetzt sind. Dies ist besonders wichtig, wenn Sie Claude Code in containerisierten oder eingeschränkten Netzwerkkumgebungen verwenden.

Zusätzliche Ressourcen

- [Claude Code-Einstellungen](#)
- [Referenz für Umgebungsvariablen](#)
- [Leitfaden zur Fehlerbehebung](#)

Part 12: Environment Setup

Entwicklungscontainer

Erfahren Sie mehr über den Claude Code-Entwicklungscontainer für Teams, die konsistente, sichere Umgebungen benötigen.

Die Referenz [devcontainer-Einrichtung](#) und die zugehörige [Dockerfile](#) bieten einen vor-konfigurierten Entwicklungscontainer, den Sie unverändert verwenden oder nach Ihren Anforderungen anpassen können. Dieser devcontainer funktioniert mit der Visual Studio Code [Dev Containers-Erweiterung](#) und ähnlichen Tools.

Die erweiterten Sicherheitsmaßnahmen des Containers (Isolierung und Firewall-Regeln) ermöglichen es Ihnen, `claude --dangerously-skip-permissions` auszuführen, um Berechtigungsaufforderungen für unbeaufsichtigte Operationen zu umgehen.

Warning:

Obwohl der devcontainer erhebliche Schutzmaßnahmen bietet, ist kein System vollständig immun gegen alle Angriffe. Bei Ausführung mit `--dangerously-skip-permissions` verhindern devcontainer nicht, dass ein bösartiges Projekt alles exfiltriert, das im devcontainer zugänglich ist, einschließlich Claude Code-Anmeldedaten. Wir empfehlen, devcontainer nur bei der Entwicklung mit vertrauenswürdigen Repositories zu verwenden. Halten Sie immer gute Sicherheitspraktiken ein und überwachen Sie die Aktivitäten von Claude.

Wichtigste Funktionen

- **Produktionsreifes Node.js:** Basierend auf Node.js 20 mit wesentlichen Entwicklungsabhängigkeiten
- **Sicherheit nach Design:** Benutzerdefinierte Firewall, die den Netzwerkzugriff auf nur notwendige Dienste beschränkt
- **Benutzerfreundliche Tools:** Umfasst git, ZSH mit Produktivitätsverbesserungen, fzf und mehr
- **Nahtlose VS Code-Integration:** Vorkonfigurierte Erweiterungen und optimierte Einstellungen
- **Sitzungspersistenz:** Bewahrt Befehlsverlauf und Konfigurationen zwischen Container-Neustarts

- **Funktioniert überall:** Kompatibel mit macOS-, Windows- und Linux-Entwicklungsumgebungen

Erste Schritte in 4 Schritten

1. Installieren Sie VS Code und die Remote - Containers-Erweiterung
2. Klonen Sie das [Claude Code-Referenzimplementierungs](#)-Repository
3. Öffnen Sie das Repository in VS Code
4. Wenn Sie dazu aufgefordert werden, klicken Sie auf „In Container erneut öffnen“ (oder verwenden Sie die Befehlspalette: Cmd+Shift+P → „Remote-Containers: In Container erneut öffnen“)

Konfigurationsübersicht

Die devcontainer-Einrichtung besteht aus drei Hauptkomponenten:

- **[devcontainer.json](#):** Steuert Container-Einstellungen, Erweiterungen und Volume-Mounts
- **[Dockerfile](#):** Definiert das Container-Image und installierte Tools
- **[init-firewall.sh](#):** Etabliert Netzwerksicherheitsregeln

Sicherheitsfunktionen

Der Container implementiert einen mehrschichtigen Sicherheitsansatz mit seiner Firewall-Konfiguration:

- **Präzise Zugriffskontrolle:** Beschränkt ausgehende Verbindungen auf nur auf die Whitelist gesetzten Domains (npm-Registry, GitHub, Claude API usw.)
- **Zulässige ausgehende Verbindungen:** Die Firewall erlaubt ausgehende DNS- und SSH-Verbindungen
- **Standard-Deny-Richtlinie:** Blockiert alle anderen externen Netzwerkzugriffe
- **Startupüberprüfung:** Validiert Firewall-Regeln, wenn der Container initialisiert wird
- **Isolierung:** Erstellt eine sichere Entwicklungsumgebung, die von Ihrem Hauptsystem getrennt ist

Anpassungsoptionen

Die devcontainer-Konfiguration ist so konzipiert, dass sie an Ihre Anforderungen angepasst werden kann:

- Fügen Sie VS Code-Erweiterungen basierend auf Ihrem Workflow hinzu oder entfernen Sie sie
- Ändern Sie Ressourcenzuordnungen für verschiedene Hardware-Umgebungen
- Passen Sie Netzwerkzugriffsberechtigungen an
- Passen Sie Shell-Konfigurationen und Entwickler-Tools an

Beispiel-Anwendungsfälle

Sichere Kundenarbeit

Verwenden Sie devcontainer, um verschiedene Kundenprojekte zu isolieren und sicherzustellen, dass Code und Anmeldedaten niemals zwischen Umgebungen vermischt werden.

Team-Onboarding

Neue Teammitglieder können in wenigen Minuten eine vollständig konfigurierte Entwicklungsumgebung erhalten, mit allen notwendigen Tools und Einstellungen, die vorinstalliert sind.

Konsistente CI/CD-Umgebungen

Spiegeln Sie Ihre devcontainer-Konfiguration in CI/CD-Pipelines, um sicherzustellen, dass Entwicklungs- und Produktionsumgebungen übereinstimmen.

Verwandte Ressourcen

- [VS Code devcontainers-Dokumentation](#)
- [Claude Code-Sicherheitsbest Practices](#)
- [Enterprise-Netzwerkkonfiguration](#)

Optimieren Sie Ihr Terminal-Setup

Claude Code funktioniert am besten, wenn Ihr Terminal richtig konfiguriert ist. Befolgen Sie diese Richtlinien, um Ihr Erlebnis zu optimieren.

Designs und Erscheinungsbild

Claude kann das Design Ihres Terminals nicht steuern. Das wird von Ihrer Terminalanwendung verwaltet. Sie können das Design von Claude Code jederzeit über den Befehl `/config` an Ihr Terminal anpassen.

Für zusätzliche Anpassungen der Claude Code-Oberfläche selbst können Sie eine [benutzerdefinierte Statuszeile](#) konfigurieren, um kontextbezogene Informationen wie das aktuelle Modell, das Arbeitsverzeichnis oder den Git-Branch am unteren Rand Ihres Terminals anzuzeigen.

Zeilenumbrüche

Sie haben mehrere Optionen, um Zeilenumbrüche in Claude Code einzugeben:

- **Schnelle Flucht:** Geben Sie `\` gefolgt von Enter ein, um einen Zeilenumbruch zu erstellen
- **Shift+Enter:** Funktioniert standardmäßig in iTerm2, WezTerm, Ghostty und Kitty
- **Tastaturkürzel:** Richten Sie eine Tastenkombination ein, um einen Zeilenumbruch in anderen Terminals einzufügen

Shift+Enter für andere Terminals einrichten

Führen Sie `/terminal-setup` in Claude Code aus, um Shift+Enter automatisch für VS Code, Alacritty, Zed und Warp zu konfigurieren.

Note:

Der Befehl `/terminal-setup` ist nur in Terminals sichtbar, die eine manuelle Konfiguration erfordern. Wenn Sie iTerm2, WezTerm, Ghostty oder Kitty verwenden, sehen Sie diesen Befehl nicht, da Shift+Enter bereits nativ funktioniert.

Option+Enter einrichten (VS Code, iTerm2 oder macOS Terminal.app)

Für Mac Terminal.app:

1. Öffnen Sie Einstellungen → Profile → Tastatur
2. Aktivieren Sie „Option als Meta-Taste verwenden“

Für iTerm2 und VS Code Terminal:

1. Öffnen Sie Einstellungen → Profile → Tasten
2. Stellen Sie unter Allgemein die linke/rechte Optionstaste auf „Esc+“ ein

Benachrichtigungseinrichtung

Verpassen Sie nie, wenn Claude eine Aufgabe mit ordnungsgemäßer Benachrichtigungskonfiguration abschließt:

iTerm 2 Systembenachrichtigungen

Für iTerm 2 Benachrichtigungen, wenn Aufgaben abgeschlossen sind:

1. Öffnen Sie iTerm 2 Einstellungen
2. Navigieren Sie zu Profile → Terminal
3. Aktivieren Sie „Silence bell“ und Filter Alerts → „Send escape sequence-generated alerts“
4. Stellen Sie Ihre bevorzugte Benachrichtigungsverzögerung ein

Beachten Sie, dass diese Benachrichtigungen spezifisch für iTerm 2 sind und im Standard-macOS-Terminal nicht verfügbar sind.

Benutzerdefinierte Benachrichtigungshooks

Für erweiterte Benachrichtigungsbehandlung können Sie [Benachrichtigungshooks](#) erstellen, um Ihre eigene Logik auszuführen.

Umgang mit großen Eingaben

Bei der Arbeit mit umfangreichem Code oder langen Anweisungen:

- **Vermeiden Sie direktes Einfügen:** Claude Code kann bei sehr langen eingefügten Inhalten Schwierigkeiten haben
- **Verwenden Sie dateibasierte Workflows:** Schreiben Sie Inhalte in eine Datei und bitten Sie Claude, diese zu lesen
- **Beachten Sie VS Code-Einschränkungen:** Das VS Code-Terminal neigt besonders dazu, lange Einfügungen zu kürzen

Vim-Modus

Claude Code unterstützt eine Teilmenge von Vim-Tastenkombinationen, die mit `/vim` aktiviert oder über `/config` konfiguriert werden können.

Die unterstützte Teilmenge umfasst:

- Modusumschaltung: `Esc` (zu NORMAL), `i / I`, `a / A`, `o / O` (zu INSERT)
- Navigation: `h / j / k / l`, `w / e / b`, `0 / $ / ^`, `gg / G`, `f / F / t / T` mit `;` / `,` Wiederholung
- Bearbeitung: `x`, `dw / de / db / dd / D`, `cw / ce / cb / cc / C`, `.` (Wiederholung)
- Yank/Einfügen: `yy / Y`, `yw / ye / yb`, `p / P`
- Textobjekte: `iw / aw`, `iW / aW`, `i" / a"`, `i' / a'`, `i( / a(`, `i[ / a[`, `i{ / a{`
- Einrückung: `>> / <<`
- Zeilenoperationen: `J` (Zeilen verbinden)

Siehe [Interaktiver Modus](#) für die vollständige Referenz.

Passen Sie Ihre Statuszeile an

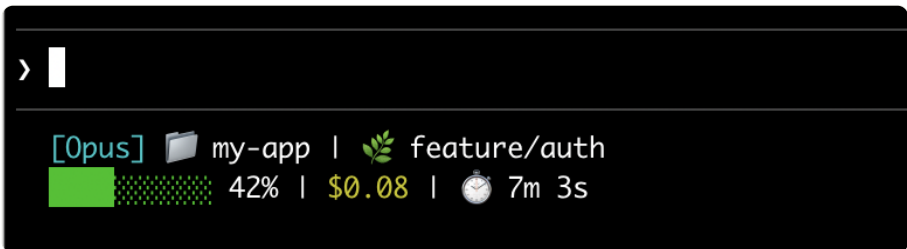
Konfigurieren Sie eine benutzerdefinierte Statusleiste, um die Kontextfensternutzung, Kosten und Git-Status in Claude Code zu überwachen

Die Statuszeile ist eine anpassbare Leiste am unteren Rand von Claude Code, die jedes Shell-Skript ausführt, das Sie konfigurieren. Sie empfängt JSON-Sitzungsdaten auf stdin und zeigt alles an, was Ihr Skript ausgibt, und bietet Ihnen eine persistente, auf einen Blick sichtbare Ansicht der Kontextnutzung, Kosten, Git-Status oder alles andere, das Sie verfolgen möchten.

Statuszeilen sind nützlich, wenn Sie:

- Die Kontextfensternutzung während der Arbeit überwachen möchten
- Sitzungskosten verfolgen müssen
- Über mehrere Sitzungen hinweg arbeiten und diese unterscheiden müssen
- Git-Branch und Status immer sichtbar haben möchten

Hier ist ein Beispiel einer [mehrzeiligen Statuszeile](#), die Git-Informationen in der ersten Zeile und einen farbcodierten Kontextbalken in der zweiten Zeile anzeigt.



Eine mehrzeilige Statuszeile, die Modellname, Verzeichnis, Git-Branch in der ersten Zeile und einen Kontextnutzungs-Fortschrittsbalken mit Kosten und Dauer in der zweiten Zeile anzeigt

Diese Seite führt Sie durch [das Einrichten einer grundlegenden Statuszeile](#), erklärt [wie die Daten fließen](#) von Claude Code zu Ihrem Skript, listet [alle Felder auf, die Sie anzeigen können](#), und bietet [einsatzbereite Beispiele](#) für häufige Muster wie Git-Status, Kostenverfolgung und Fortschrittsbalken.

Richten Sie eine Statuszeile ein

Verwenden Sie den `/statusline -Befehl`, um Claude Code ein Skript für Sie generieren zu lassen, oder [erstellen Sie manuell ein Skript](#) und fügen Sie es zu Ihren Einstellungen hinzu.

Verwenden Sie den `/statusline-Befehl`

Der `/statusline` -Befehl akzeptiert Anweisungen in natürlicher Sprache, die beschreiben, was Sie angezeigt haben möchten. Claude Code generiert eine Skriptdatei in `~/.claude/` und aktualisiert Ihre Einstellungen automatisch:

```
/statusline show model name and context percentage with a progress bar
```

Konfigurieren Sie eine Statuszeile manuell

Fügen Sie ein `statusLine` -Feld zu Ihren Benutzereinstellungen (`~/.claude/settings.json`, wobei `~` Ihr Basisverzeichnis ist) oder [Projekteinstellungen](#) hinzu. Setzen Sie `type` auf `"command"` und verweisen Sie `command` auf einen Skriptpfad oder einen Inline-Shell-Befehl. Eine vollständige Anleitung zum Erstellen eines Skripts finden Sie unter [Erstellen Sie eine Statuszeile Schritt für Schritt](#).

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh",
    "padding": 2
  }
}
```

Das `command` -Feld wird in einer Shell ausgeführt, sodass Sie auch Inline-Befehle anstelle einer Skriptdatei verwenden können. Dieses Beispiel verwendet `jq`, um die JSON-Eingabe zu analysieren und den Modellnamen und den Kontextprozentsatz anzuzeigen:

```
{
  "statusLine": {
    "type": "command",
    "command": "jq -r '\[\\(.model.display_name)] \\(.context_window.used_percent  
age // 0)% context'"
  }
}
```

Das optionale `padding`-Feld fügt zusätzlichen horizontalen Abstand (in Zeichen) zum Inhalt der Statuszeile hinzu. Standardmäßig `0`. Dieser Abstand wird zusätzlich zum integrierten Abstand der Benutzeroberfläche hinzugefügt, sodass er die relative Einrückung steuert, anstatt den absoluten Abstand vom Terminalrand zu steuern.

Deaktivieren Sie die Statuszeile

Führen Sie `/statusline` aus und bitten Sie es, Ihre Statuszeile zu entfernen oder zu löschen (z. B. `/statusline delete`, `/statusline clear`, `/statusline remove it`). Sie können auch das `statusLine`-Feld manuell aus Ihrer `settings.json` löschen.

Erstellen Sie eine Statuszeile Schritt für Schritt

Diese Anleitung zeigt, was unter der Haube passiert, indem Sie manuell eine Statuszeile erstellen, die das aktuelle Modell, das Arbeitsverzeichnis und den Prozentsatz der Kontextfensternutzung anzeigt.

Note:

Das Ausführen von `/statusline` mit einer Beschreibung dessen, was Sie möchten, konfiguriert all dies automatisch für Sie.

Diese Beispiele verwenden Bash-Skripte, die auf macOS und Linux funktionieren. Unter Windows finden Sie unter [Windows-Konfiguration](#) PowerShell- und Git Bash-Beispiele.



[Sonnet]  my-app | 24% context

Eine Statuszeile, die Modellname, Verzeichnis und Kontextprozentsatz anzeigt

Step 1: Erstellen Sie ein Skript, das JSON liest und Ausgabe druckt

Claude Code sendet JSON-Daten über stdin an Ihr Skript. Dieses Skript verwendet `jq`, einen Befehlszeilen-JSON-Parser, den Sie möglicherweise installieren müssen, um den Modellnamen, das Verzeichnis und den Kontextprozensatz zu extrahieren, und druckt dann eine formatierte Zeile.

Speichern Sie dies unter `~/.claude/statusline.sh` (wobei `~` Ihr Basisverzeichnis ist, z. B. `/Users/username` auf macOS oder `/home/username` auf Linux):

```
#!/bin/bash

## Read JSON data that Claude Code sends to stdin
input=$(cat)

## Extract fields using jq
MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
## The "// 0" provides a fallback if the field is null
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## Output the status line - ${DIR##*/} extracts just the folder name
echo "[$MODEL] 📁 ${DIR##*/} | ${PCT}% context"
```

Step 2: Machen Sie es ausführbar

Markieren Sie das Skript als ausführbar, damit Ihre Shell es ausführen kann:

```
chmod +x ~/.claude/statusline.sh
```

Step 3: Zu Einstellungen hinzufügen

Teilen Sie Claude Code mit, dass es Ihr Skript als Statuszeile ausführen soll. Fügen Sie diese Konfiguration zu `~/.claude/settings.json` hinzu, die `type` auf `"command"` setzt (was bedeutet „diesen Shell-Befehl ausführen“) und `command` auf Ihr Skript verweist:

```
{
  "statusLine": {
    "type": "command",
    "command": "~/claude/statusline.sh"
  }
}
```

Ihre Statuszeile wird am unteren Rand der Benutzeroberfläche angezeigt. Einstellungen werden automatisch neu geladen, aber Änderungen werden erst bei Ihrer nächsten Interaktion mit Claude Code angezeigt.

Wie Statuszeilen funktionieren

Claude Code führt Ihr Skript aus und leitet [JSON-Sitzungsdaten](#) über stdin an es weiter. Ihr Skript liest die JSON, extrahiert das, was es benötigt, und druckt Text auf stdout. Claude Code zeigt alles an, was Ihr Skript druckt.

Wann es aktualisiert wird

Ihr Skript wird nach jeder neuen Assistentenachricht ausgeführt, wenn sich der Berechtigungsmodus ändert oder wenn der Vim-Modus umgeschaltet wird. Aktualisierungen werden mit 300 ms entprellt, was bedeutet, dass schnelle Änderungen zusammengefasst werden und Ihr Skript einmal ausgeführt wird, wenn sich die Dinge beruhigt haben. Wenn eine neue Aktualisierung ausgelöst wird, während Ihr Skript noch läuft, wird die laufende Ausführung abgebrochen. Wenn Sie Ihr Skript bearbeiten, werden die Änderungen erst angezeigt, wenn Ihre nächste Interaktion mit Claude Code eine Aktualisierung auslöst.

Was Ihr Skript ausgeben kann

- **Mehrere Zeilen:** Jede `echo` - oder `print` -Anweisung wird als separate Zeile angezeigt. Siehe das [mehrzeilige Beispiel](#).
- **Farben:** Verwenden Sie [ANSI-Escape-Codes](#) wie `\033[32m` für Grün (Terminal muss diese unterstützen). Siehe das [Git-Status-Beispiel](#).
- **Links:** Verwenden Sie [OSC 8-Escape-Sequenzen](#), um Text anklickbar zu machen (Cmd+Klick auf macOS, Strg+Klick auf Windows/Linux). Erfordert ein Terminal, das Hyperlinks wie iTerm2, Kitty oder WezTerm unterstützt. Siehe das [Beispiel für anklickbare Links](#).

Note:

Die Statuszeile wird lokal ausgeführt und verbraucht keine API-Token. Sie wird vorübergehend während bestimmter UI-Interaktionen ausgeblendet, einschließlich Autovervollständigungsvorschläge, das Hilfenmenü und Berechtigungsaufforderungen.

Verfügbare Daten

Claude Code sendet die folgenden JSON-Felder über stdin an Ihr Skript:

Feld	Beschreibung
<code>model.id</code> , <code>model.display_name</code>	Aktuelle Modellkennung und Anzeigename
<code>cwd</code> , <code>workspace.current_dir</code>	Aktuelles Arbeitsverzeichnis. Beide Felder enthalten denselben Wert; <code>workspace.current_dir</code> wird für Konsistenz mit <code>workspace.project_dir</code> bevorzugt.
<code>workspace.project_dir</code>	Verzeichnis, in dem Claude Code gestartet wurde, das sich von <code>cwd</code> unterscheiden kann, wenn sich das Arbeitsverzeichnis während einer Sitzung ändert
<code>cost.total_cost_usd</code>	Gesamtsitzungskosten in USD
<code>cost.total_duration_ms</code>	Gesamtverstrichene Zeit seit Sitzungsbeginn in Millisekunden
<code>cost.total_api_duration_ms</code>	Gesamtzeit, die auf API-Antworten wartet, in Millisekunden
<code>cost.total_lines_added</code> , <code>cost.total_lines_removed</code>	Geänderte Codezeilen
<code>context_window.total_input_tokens</code> , <code>context_window.total_output_tokens</code>	Kumulative Token-Zählungen über die Sitzung
<code>context_window.context_window_size</code>	Maximale Kontextfenstergröße in Token. Standardmäßig 200.000 oder 1.000.000 für Modelle mit erweitertem Kontext.
<code>context_window.used_percentage</code>	Vorberechneter Prozentsatz der Kontextfensternutzung
<code>context_window.remaining_percentage</code>	Vorberechneter Prozentsatz des verbleibenden Kontextfensters

Feld	Beschreibung
<code>context_window.current_usage</code>	Token-Zählungen aus dem letzten API-Aufruf, beschrieben in Kontextfensterfelder
<code>exceeds_200k_tokens</code>	Ob die Gesamttoken-Zählung (Eingabe-, Cache- und Ausgabe-Token kombiniert) aus der letzten API-Antwort 200.000 überschreitet. Dies ist ein fester Schwellenwert unabhängig von der tatsächlichen Kontextfenstergröße.
<code>session_id</code>	Eindeutige Sitzungskennung
<code>transcript_path</code>	Pfad zur Gesprächstranskriptdatei
<code>version</code>	Claude Code-Version
<code>output_style.name</code>	Name des aktuellen Ausgabestils
<code>vim.mode</code>	Aktueller Vim-Modus (<code>NORMAL</code> oder <code>INSERT</code>), wenn Vim-Modus aktiviert ist
<code>agent.name</code>	Agent-Name bei Ausführung mit dem <code>--agent</code> -Flag oder konfigurierter Agent-Einstellung
<code>worktree.name</code>	Name des aktiven Worktree. Nur während <code>--worktree</code> -Sitzungen vorhanden
<code>worktree.path</code>	Absoluter Pfad zum Worktree-Verzeichnis
<code>worktree.branch</code>	Git-Branch-Name für den Worktree (z. B. <code>"worktree-my-feature"</code>). Fehlt bei Hook-basierten Worktrees
<code>worktree.original_cwd</code>	Das Verzeichnis, in dem Claude sich befand, bevor es den Worktree betrat
<code>worktree.original_branch</code>	Git-Branch, der vor dem Betreten des Worktree ausgecheckt wurde. Fehlt bei Hook-basierten Worktrees

Ihr Statuszeilen-Befehl empfängt diese JSON-Struktur über stdin:

```
{
  "cwd": "/current/working/directory",
  "session_id": "abc123...",
  "transcript_path": "/path/to/transcript.jsonl",
  "model": {
    "id": "claude-opus-4-6",
    "display_name": "Opus"
  },
  "workspace": {
    "current_dir": "/current/working/directory",
    "project_dir": "/original/project/directory"
  },
  "version": "1.0.80",
  "output_style": {
    "name": "default"
  },
  "cost": {
    "total_cost_usd": 0.01234,
    "total_duration_ms": 45000,
    "total_api_duration_ms": 2300,
    "total_lines_added": 156,
    "total_lines_removed": 23
  },
  "context_window": {
    "total_input_tokens": 15234,
    "total_output_tokens": 4521,
    "context_window_size": 200000,
    "used_percentage": 8,
    "remaining_percentage": 92,
    "current_usage": {
      "input_tokens": 8500,
      "output_tokens": 1200,
      "cache_creation_input_tokens": 5000,
      "cache_read_input_tokens": 2000
    }
  },
  "exceeds_200k_tokens": false,
  "vim": {
```

```
  "mode": "NORMAL"
},
"agent": {
  "name": "security-reviewer"
},
"worktree": {
  "name": "my-feature",
  "path": "/path/to/.claude/worktrees/my-feature",
  "branch": "worktree-my-feature",
  "original_cwd": "/path/to/project",
  "original_branch": "main"
}
}
```

Felder, die möglicherweise fehlen (nicht in JSON vorhanden):

- `vim` : wird nur angezeigt, wenn der Vim-Modus aktiviert ist
- `agent` : wird nur angezeigt, wenn mit dem `--agent` -Flag oder konfigurierter Agent-Einstellung ausgeführt wird
- `worktree` : wird nur während `--worktree` -Sitzungen angezeigt. Wenn vorhanden, können `branch` und `original_branch` auch bei Hook-basierten Worktrees fehlen

Felder, die `null` sein können:

- `context_window.current_usage` : `null` vor dem ersten API-Aufruf in einer Sitzung
- `context_window.used_percentage` , `context_window.remaining_percentage` : können früh in der Sitzung `null` sein

Behandeln Sie fehlende Felder mit bedingtem Zugriff und Null-Werte mit Fallback-Standardwerten in Ihren Skripten.

Kontextfensterfelder

Das `context_window` -Objekt bietet zwei Möglichkeiten, die Kontextnutzung zu verfolgen:

- **Kumulative Summen** (`total_input_tokens` , `total_output_tokens`): Summe aller Token über die gesamte Sitzung, nützlich für die Verfolgung des Gesamtverbrauchs
- **Aktuelle Nutzung** (`current_usage`): Token-Zählungen aus dem letzten API-Aufruf, verwenden Sie dies für einen genauen Kontextprozentsatz, da er den tatsächlichen Kontextzustand widerspiegelt

Das `current_usage`-Objekt enthält:

- `input_tokens` : Eingabe-Token im aktuellen Kontext
- `output_tokens` : generierte Ausgabe-Token
- `cache_creation_input_tokens` : Token, die in den Cache geschrieben wurden
- `cache_read_input_tokens` : Token, die aus dem Cache gelesen wurden

Das `used_percentage`-Feld wird nur aus Eingabe-Token berechnet: `input_tokens + cache_creation_input_tokens + cache_read_input_tokens`. Es enthält keine `output_tokens`.

Wenn Sie den Kontextprozensatz manuell aus `current_usage` berechnen, verwenden Sie die gleiche Formel nur für Eingaben, um `used_percentage` zu entsprechen.

Das `current_usage`-Objekt ist `null` vor dem ersten API-Aufruf in einer Sitzung.

Beispiele

Diese Beispiele zeigen häufige Statuszeilen-Muster. Um ein Beispiel zu verwenden:

1. Speichern Sie das Skript in einer Datei wie `~/.claude/statusline.sh` (oder `.py` / `.js`)
2. Machen Sie es ausführbar: `chmod +x ~/.claude/statusline.sh`
3. Fügen Sie den Pfad zu Ihren [Einstellungen](#) hinzu

Die Bash-Beispiele verwenden `jq` zum Analysieren von JSON. Python und Node.js haben integrierte JSON-Analyse.

Kontextfensternutzung

Zeigen Sie das aktuelle Modell und die Kontextfensternutzung mit einem visuellen Fortschrittsbalken an. Jedes Skript liest JSON von stdin, extrahiert das `used_percentage`-Feld und erstellt einen 10-Zeichen-Balken, wobei gefüllte Blöcke (█) die Nutzung darstellen:



Eine Statuszeile, die Modellname und einen Fortschrittsbalken mit Prozentsatz anzeigt

```
#!/bin/bash

## Read all of stdin into a variable
input=$(cat)

## Extract fields with jq, "// 0" provides fallback for null
MODEL=$(echo "$input" | jq -r '.model.display_name')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## Build progress bar: printf creates spaces, tr replaces with blocks
BAR_WIDTH=10
FILLED=$((PCT * BAR_WIDTH / 100))
EMPTY=$((BAR_WIDTH - FILLED))
BAR=""
[ "$FILLED" -gt 0 ] && BAR=$(printf "%${FILLED}s" | tr ' ' '█')
[ "$EMPTY" -gt 0 ] && BAR="${BAR}${(printf "%${EMPTY}s" | tr ' ' '░')}"

echo "[$MODEL] $BAR $PCT%"
```

```
#!/usr/bin/env python3

import json, sys

## json.load reads and parses stdin in one step
data = json.load(sys.stdin)
model = data['model']['display_name']
## "or 0" handles null values
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)

## String multiplication builds the bar
filled = pct * 10 // 100
bar = '█' * filled + '░' * (10 - filled)

print(f"[{model}] {bar} {pct}%")
```

```
#!/usr/bin/env node
// Node.js reads stdin asynchronously with events
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  // Optional chaining (?.) safely handles null fields
  const pct = Math.floor(data.context_window?.used_percentage || 0);

  // String.repeat() builds the bar
  const filled = Math.floor(pct * 10 / 100);
  const bar = '█'.repeat(filled) + '░'.repeat(10 - filled);

  console.log(`[${model}] ${bar} ${pct}%`);
});
```

Git-Status mit Farben

Zeigen Sie Git-Branch mit farbcodierten Indikatoren für bereitgestellte und geänderte Dateien an. Dieses Skript verwendet [ANSI-Escape-Codes](#) für Terminalfarben: `\033[32m` ist Grün, `\033[33m` ist Gelb und `\033[0m` setzt auf Standard zurück.

> █

[Sonnet]  my-app |  feature/auth +3~2

Eine Statuszeile, die Modell, Verzeichnis, Git-Branch und farbcodierte Indikatoren für bereitgestellte und geänderte Dateien anzeigt

Jedes Skript prüft, ob das aktuelle Verzeichnis ein Git-Repository ist, zählt bereitgestellte und geänderte Dateien und zeigt farbcodierte Indikatoren an:

```
#!/bin/bash

input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')

GREEN='\033[32m'
YELLOW='\033[33m'
RESET='\033[0m'

if git rev-parse --git-dir > /dev/null 2>&1; then
    BRANCH=$(git branch --show-current 2>/dev/null)
    STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
    MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')

    GIT_STATUS=""
    [ "$STAGED" -gt 0 ] && GIT_STATUS="${GREEN}+${STAGED}${RESET}"
    [ "$MODIFIED" -gt 0 ] && GIT_STATUS="${GIT_STATUS}${YELLOW}~${MODIFIED}${RESET}"

    echo -e "[$MODEL] 📁 ${DIR##*/} | 🌿 $BRANCH $GIT_STATUS"
else
    echo "[$MODEL] 📁 ${DIR##*/}"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])

GREEN, YELLOW, RESET = '\033[32m', '\033[33m', '\033[0m'

try:
    subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subprocess.D
EVNULL)
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=Tru
e).strip()
    staged_output = subprocess.check_output(['git', 'diff', '--cached', '--
numstat'], text=True).strip()
    modified_output = subprocess.check_output(['git', 'diff', '--numstat'], text=T
rue).strip()
    staged = len(staged_output.split('\n')) if staged_output else 0
    modified = len(modified_output.split('\n')) if modified_output else 0

    git_status = f"{GREEN}+{staged}{RESET}" if staged else ""
    git_status += f"{YELLOW}~{modified}{RESET}" if modified else ""

    print(f"[{model}] 📁 {directory} | 🌿 {branch} {git_status}")
except:
    print(f"[{model}] 📁 {directory}")
```



```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const dir = path.basename(data.workspace.current_dir);

  const GREEN = '\x1b[32m', YELLOW = '\x1b[33m', RESET = '\x1b[0m';

  try {
    execSync('git rev-parse --git-dir', { stdio: 'ignore' });
    const branch = execSync('git branch --show-current', { encoding:
'utf8' }).trim();
    const staged = execSync('git diff --cached --numstat', { encoding: 'utf8'
}).trim().split('\n').filter(Boolean).length;
    const modified = execSync('git diff --numstat', { encoding:
'utf8' }).trim().split('\n').filter(Boolean).length;

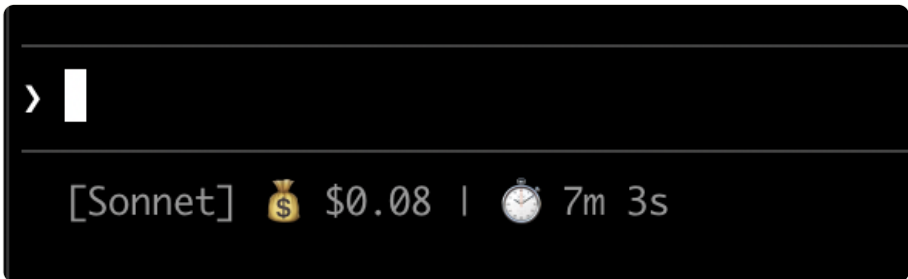
    let gitStatus = staged ? `${GREEN}+${staged}${RESET}` : '';
    gitStatus += modified ? `${YELLOW}~${modified}${RESET}` : '';

    console.log(`[${model}] 📁 ${dir} | 🌿 ${branch} ${gitStatus}`);
  } catch {
    console.log(`[${model}] 📁 ${dir}`);
  }
});
```

Kosten- und Dauer-Verfolgung

Verfolgen Sie die API-Kosten und verstrichene Zeit Ihrer Sitzung. Das `cost.total_cost_usd`-Feld sammelt die Kosten aller API-Aufrufe in der aktuellen Sitzung. Das `cost.total_duration_ms`-Feld misst die Gesamtverstrichene Zeit seit Sitzungsbeginn, während `cost.total_api_duration_ms` nur die Zeit verfolgt, die auf API-Antworten wartet.

Jedes Skript formatiert Kosten als Währung und konvertiert Millisekunden in Minuten und Sekunden:



Eine Statuszeile, die Modellname, Sitzungskosten und Dauer anzeigt

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

COST_FMT=$(printf '%.2f' "$COST")
DURATION_SEC=$((DURATION_MS / 1000))
MINS=$((DURATION_SEC / 60))
SECS=$((DURATION_SEC % 60))

echo "[$MODEL] 💰 $COST_FMT | 🕒 ${MINS}m ${SECS}s"
```

```
#!/usr/bin/env python3
import json, sys

data = json.load(sys.stdin)
model = data['model']['display_name']
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

duration_sec = duration_ms // 1000
mins, secs = duration_sec // 60, duration_sec % 60

print(f"[{model}] 💰 ${cost:.2f} | ⌚ {mins}m {secs}s")
```

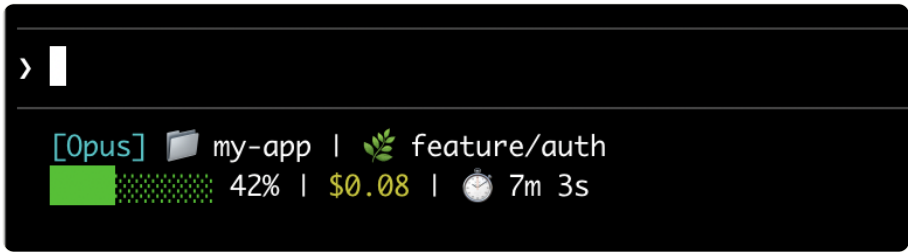
```
#!/usr/bin/env node
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const cost = data.cost?.total_cost_usd || 0;
  const durationMs = data.cost?.total_duration_ms || 0;

  const durationSec = Math.floor(durationMs / 1000);
  const mins = Math.floor(durationSec / 60);
  const secs = durationSec % 60;

  console.log(`[${model}] 💰 $$${cost.toFixed(2)} | ⌚ ${mins}m ${secs}s`);
});
```

Mehrere Zeilen anzeigen

Ihr Skript kann mehrere Zeilen ausgeben, um eine reichhaltigere Anzeige zu erstellen. Jede `echo`-Anweisung erzeugt eine separate Zeile im Statusbereich.



Eine mehrzeilige Statuszeile, die Modellname, Verzeichnis, Git-Branch in der ersten Zeile und einen Kontextnutzungs-Fortschrittsbalken mit Kosten und Dauer in der zweiten Zeile anzeigt

Dieses Beispiel kombiniert mehrere Techniken: schwellenwertbasierte Farben (Grün unter 70 %, Gelb 70–89 %, Rot 90 %+), einen Fortschrittsbalken und Git-Branch-Informationen. Jede `print` - oder `echo` -Anweisung erstellt eine separate Zeile:

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

CYAN='\033[36m'; GREEN='\033[32m'; YELLOW='\033[33m'; RED='\033[31m'; RESET='\033[
0m'

## Pick bar color based on context usage
if [ "$PCT" -ge 90 ]; then BAR_COLOR="$RED"
elif [ "$PCT" -ge 70 ]; then BAR_COLOR="$YELLOW"
else BAR_COLOR="$GREEN"; fi

FILLED=$((PCT / 10)); EMPTY=$((10 - FILLED))
BAR=$(printf "%${FILLED}s" | tr ' ' '█')$(printf "%${EMPTY}s" | tr ' ' '░')

MINS=$((DURATION_MS / 60000)); SECS=$((DURATION_MS % 60000 / 1000))

BRANCH=""
git rev-parse --git-dir > /dev/null 2>&1 && BRANCH=" | 🌿 $(git branch --show-
current 2>/dev/null)"

echo -e "${CYAN}[$MODEL]${RESET} 📁 ${DIR##*/}${BRANCH}"
COST_FMT=$(printf '=%.2f' "$COST")
echo -e "${BAR_COLOR}${BAR}${RESET} ${PCT}% | ${YELLOW}${COST_FMT}${RESET} | ⌚ $
{MINS}m ${SECS}s"
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

CYAN, GREEN, YELLOW, RED, RESET = '\033[36m', '\033[32m', '\033[33m', '\033[31m',
'\033[0m'

bar_color = RED if pct ≥ 90 else YELLOW if pct ≥ 70 else GREEN
filled = pct // 10
bar = '█' * filled + '░' * (10 - filled)

mins, secs = duration_ms // 60000, (duration_ms % 60000) // 1000

try:
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=True,
stderr=subprocess.DEVNULL).strip()
    branch = f" | 🌿 {branch}" if branch else ""
except:
    branch = ""

print(f"{CYAN}[{model}]{RESET} 📁 {directory}{branch}")
print(f"{bar_color}{bar}{RESET} {pct}% | {YELLOW}${cost:.2f}{RESET} | 🕒 {mins}m
{secs}s")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const dir = path.basename(data.workspace.current_dir);
  const cost = data.cost?.total_cost_usd || 0;
  const pct = Math.floor(data.context_window?.used_percentage || 0);
  const durationMs = data.cost?.total_duration_ms || 0;

  const CYAN = '\x1b[36m', GREEN = '\x1b[32m', YELLOW = '\x1b[33m', RED = '\x1b[31m', RESET = '\x1b[0m';

  const barColor = pct ≥ 90 ? RED : pct ≥ 70 ? YELLOW : GREEN;
  const filled = Math.floor(pct / 10);
  const bar = '█'.repeat(filled) + '░'.repeat(10 - filled);

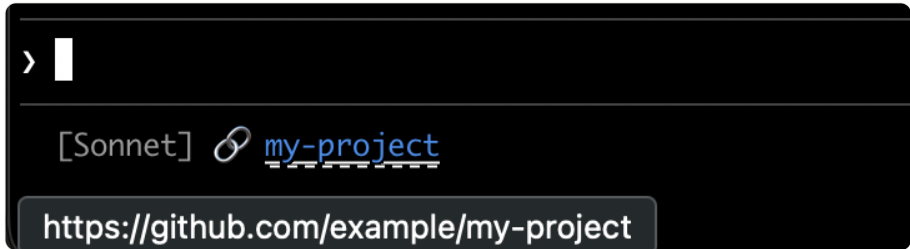
  const mins = Math.floor(durationMs / 60000);
  const secs = Math.floor((durationMs % 60000) / 1000);

  let branch = '';
  try {
    branch = execSync('git branch --show-current', { encoding: 'utf8', stdio:
['pipe', 'pipe', 'ignore'] }).trim();
    branch = branch ? ` | 🌿 ${branch}` : '';
  } catch {}

  console.log(`${CYAN}${model}${RESET} 📁 ${dir}${branch}`);
  console.log(`${barColor}${bar}${RESET} ${pct}% | ${YELLOW}${cost.toFixed(2)}$
${RESET} | 🕒 ${mins}m ${secs}s`);
});
```

Anklickbare Links

Dieses Beispiel erstellt einen anklickbaren Link zu Ihrem GitHub-Repository. Es liest die Git-Remote-URL, konvertiert das SSH-Format mit `sed` in HTTPS und umhüllt den Repository-Namen mit OSC 8-Escape-Codes. Halten Sie `Cmd` (macOS) oder `Strg` (Windows/Linux) gedrückt und klicken Sie, um den Link in Ihrem Browser zu öffnen.



Eine Statuszeile, die einen anklickbaren Link zu einem GitHub-Repository anzeigt

Jedes Skript ruft die Git-Remote-URL ab, konvertiert das SSH-Format in HTTPS und umhüllt den Repository-Namen mit OSC 8-Escape-Codes. Die Bash-Version verwendet `printf '%b'`, das Backslash-Escapes zuverlässiger interpretiert als `echo -e` über verschiedene Shells hinweg:

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')

## Convert git SSH URL to HTTPS
REMOTE=$(git remote get-url origin 2>/dev/null | sed 's/git@github.com:/https:\/\/github.com\/' | sed 's\/.git$\/')

if [ -n "$REMOTE" ]; then
    REPO_NAME=$(basename "$REMOTE")
    # OSC 8 format: \e]8;;URL\a then TEXT then \e]8;;\a
    # printf %b interprets escape sequences reliably across shells
    printf '%b' "[$MODEL]  \e]8;;${REMOTE}\a${REPO_NAME}\e]8;;\a\n"
else
    echo "[$MODEL]"
fi
```



```
#!/usr/bin/env python3
import json, sys, subprocess, re, os

data = json.load(sys.stdin)
model = data['model']['display_name']

## Get git remote URL
try:
    remote = subprocess.check_output(
        ['git', 'remote', 'get-url', 'origin'],
        stderr=subprocess.DEVNULL, text=True
    ).strip()
    # Convert SSH to HTTPS format
    remote = re.sub(r'^git@github\.com:', 'https://github.com/', remote)
    remote = re.sub(r'\.git$', '', remote)
    repo_name = os.path.basename(remote)
    # OSC 8 escape sequences
    link = f"\033]8;;{remote}\a{repo_name}\033]8;;\a"
    print(f"[{model}] 🔗 {link}")
except:
    print(f"[{model}]")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;

  try {
    let remote = execSync('git remote get-url origin', { encoding: 'utf8', std
io: ['pipe', 'pipe', 'ignore'] }).trim();
    // Convert SSH to HTTPS format
    remote = remote.replace(/^git@github\.com:/, 'https://github.com/').replac
e(/\.git$/, '');
    const repoName = path.basename(remote);
    // OSC 8 escape sequences
    const link = `\x1b]8;;${remote}\x07${repoName}\x1b]8;;\x07`;
    console.log(`[${model}] 🔗 ${link}`);
  } catch {
    console.log(`[${model}]`);
  }
});
```

Teure Operationen zwischenspeichern

Ihr Statuszeilen-Skript wird während aktiver Sitzungen häufig ausgeführt. Befehle wie `git status` oder `git diff` können langsam sein, besonders in großen Repositories. Dieses Beispiel speichert Git-Informationen in einer temporären Datei zwischen und aktualisiert sie nur alle 5 Sekunden.

Verwenden Sie einen stabilen, festen Dateinamen für die Cache-Datei wie `/tmp/statusline-git-cache`. Jede Statuszeilen-Invokation wird als neuer Prozess ausgeführt, daher erzeugen prozessbasierte Identifikatoren wie `$$`, `os.getpid()` oder `process.pid` jedes Mal einen anderen Wert und der Cache wird nie wiederverwendet.

Jedes Skript prüft, ob die Cache-Datei fehlt oder älter als 5 Sekunden ist, bevor Git-Befehle ausgeführt werden:

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')

CACHE_FILE="/tmp/statusline-git-cache"
CACHE_MAX_AGE=5 # seconds

cache_is_stale() {
    [ ! -f "$CACHE_FILE" ] || \
    # stat -f %m is macOS, stat -c %Y is Linux
    [ $($((date +%s)) - $(stat -f %m "$CACHE_FILE" 2>/dev/null || stat -c %Y "$CACHE_FILE" 2>/dev/null || echo 0)) -gt $CACHE_MAX_AGE ]
}

if cache_is_stale; then
    if git rev-parse --git-dir > /dev/null 2>&1; then
        BRANCH=$(git branch --show-current 2>/dev/null)
        STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
        MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')
        echo "$BRANCH|$STAGED|$MODIFIED" > "$CACHE_FILE"
    else
        echo "||" > "$CACHE_FILE"
    fi
fi

IFS='|' read -r BRANCH STAGED MODIFIED < "$CACHE_FILE"

if [ -n "$BRANCH" ]; then
    echo "[$MODEL] 📁 ${DIR##*/} | 🌿 $BRANCH +$STAGED ~$MODIFIED"
else
    echo "[$MODEL] 📁 ${DIR##*/}"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os, time

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])

CACHE_FILE = "/tmp/statusline-git-cache"
CACHE_MAX_AGE = 5 # seconds

def cache_is_stale():
    if not os.path.exists(CACHE_FILE):
        return True
    return time.time() - os.path.getmtime(CACHE_FILE) > CACHE_MAX_AGE

if cache_is_stale():
    try:
        subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subprocess.DEVNULL)
        branch = subprocess.check_output(['git', 'branch', '--show-current'],
text=True).strip()
        staged = subprocess.check_output(['git', 'diff', '--cached', '--numstat'],
text=True).strip()
        modified = subprocess.check_output(['git', 'diff', '--numstat'],
text=True).strip()
        staged_count = len(staged.split('\n')) if staged else 0
        modified_count = len(modified.split('\n')) if modified else 0
        with open(CACHE_FILE, 'w') as f:
            f.write(f"{branch}|{staged_count}|{modified_count}")
    except:
        with open(CACHE_FILE, 'w') as f:
            f.write("||")

with open(CACHE_FILE) as f:
    branch, staged, modified = f.read().strip().split('|')

if branch:
```

```
print(f"[{model}] 📁 {directory} | 🌿 {branch} +{staged} ~{modified}")  
else:  
    print(f"[{model}] 📁 {directory}")
```

```
#!/usr/bin/env node

const { execSync } = require('child_process');
const fs = require('fs');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const dir = path.basename(data.workspace.current_dir);

  const CACHE_FILE = '/tmp/statusline-git-cache';
  const CACHE_MAX_AGE = 5; // seconds

  const cacheIsStale = () => {
    if (!fs.existsSync(CACHE_FILE)) return true;
    return (Date.now() / 1000) - fs.statSync(CACHE_FILE).mtimeMs / 1000 >
    CACHE_MAX_AGE;
  };

  if (cacheIsStale()) {
    try {
      execSync('git rev-parse --git-dir', { stdio: 'ignore' });
      const branch = execSync('git branch --show-current', { encoding: 'utf8'
' }).trim();
      const staged = execSync('git diff --cached --numstat', { encoding: 'ut
f8' }).trim().split('\n').filter(Boolean).length;
      const modified = execSync('git diff --numstat', { encoding:
'utf8' }).trim().split('\n').filter(Boolean).length;
      fs.writeFileSync(CACHE_FILE, `${branch}|${staged}|${modified}`);
    } catch {
      fs.writeFileSync(CACHE_FILE, '||');
    }
  }

  const [branch, staged, modified] = fs.readFileSync(CACHE_FILE,
'utf8').trim().split('|');
```

```
if (branch) {
    console.log(`[${model}] 📁 ${dir} | 🌿 ${branch} +${staged} ~${modified}
`);
} else {
    console.log(`[${model}] 📁 ${dir}`);
}
});
```

Windows-Konfiguration

Unter Windows führt Claude Code Statuszeilen-Befehle über Git Bash aus. Sie können PowerShell von dieser Shell aus aufrufen:

```
{
  "statusLine": {
    "type": "command",
    "command": "powershell -NoProfile -File C:/Users/username/.claude/
statusline.ps1"
  }
}
```

```
$input_json = $input | Out-String | ConvertFrom-Json
$cwd = $input_json.cwd
$model = $input_json.model.display_name
$used = $input_json.context_window.used_percentage
$dirname = Split-Path $cwd -Leaf

if ($used) {
    Write-Host "$dirname [$model] ctx: $used%"
} else {
    Write-Host "$dirname [$model]"
}
```

Oder führen Sie ein Bash-Skript direkt aus:

```
{
  "statusLine": {
    "type": "command",
    "command": "~/claude/statusline.sh"
  }
}
```

```
#!/usr/bin/env bash
input=$(cat)
cwd=$(echo "$input" | grep -o '"cwd": "[^"]*"' | cut -d'"' -f4)
model=$(echo "$input" | grep -o '"display_name": "[^"]*"' | cut -d'"' -f4)
dirname="${cwd##*/[\\]}"
echo "$dirname [$model]"
```

Tipps

- **Mit Mock-Eingabe testen:** `echo '{"model": {"display_name": "Opus"}, "context_window": {"used_percentage": 25}}' | ./statusline.sh`
- **Ausgabe kurz halten:** Die Statusleiste hat eine begrenzte Breite, daher kann lange Ausgabe abgeschnitten oder unangenehm umgebrochen werden
- **Langsame Operationen zwischenspeichern:** Ihr Skript wird während aktiver Sitzungen häufig ausgeführt, daher können Befehle wie `git status` zu Verzögerungen führen. Siehe das [Caching-Beispiel](#), um damit umzugehen.

Community-Projekte wie [ccstatusline](#) und [starship-claude](#) bieten vorkonfigurierte Konfigurationen mit Designs und zusätzlichen Funktionen.

Fehlerbehebung

Statuszeile wird nicht angezeigt

- Überprüfen Sie, dass Ihr Skript ausführbar ist: `chmod +x ~/.claude/statusline.sh`
- Überprüfen Sie, dass Ihr Skript auf stdout ausgibt, nicht auf stderr
- Führen Sie Ihr Skript manuell aus, um zu überprüfen, dass es Ausgabe erzeugt
- Wenn `disableAllHooks` in Ihren Einstellungen auf `true` gesetzt ist, ist die Statuszeile auch deaktiviert. Entfernen Sie diese Einstellung oder setzen Sie sie auf `false`, um sie erneut zu aktivieren.

- Führen Sie `claude --debug` aus, um den Exit-Code und stderr aus der ersten Statuszeilen-Invokation in einer Sitzung zu protokollieren
- Bitten Sie Claude, Ihre Einstellungsdatei zu lesen und den `statusLine` -Befehl direkt auszuführen, um Fehler zu finden

Statuszeile zeigt `--` oder leere Werte

- Felder können `null` sein, bevor die erste API-Antwort abgeschlossen ist
- Behandeln Sie Null-Werte in Ihrem Skript mit Fallbacks wie `// 0` in jq
- Starten Sie Claude Code neu, wenn Werte nach mehreren Nachrichten leer bleiben

Kontextprozensatz zeigt unerwartete Werte

- Verwenden Sie `used_percentage` für einen genauen Kontextzustand anstelle von kumulativen Summen
- Die `total_input_tokens` und `total_output_tokens` sind kumulativ über die Sitzung und können die Kontextfenstergröße überschreiten
- Der Kontextprozensatz kann sich von der `/context` -Ausgabe unterscheiden, je nachdem, wann jeder berechnet wird

OSC 8-Links sind nicht anklickbar

- Überprüfen Sie, dass Ihr Terminal OSC 8-Hyperlinks unterstützt (iTerm2, Kitty, WezTerm)
- Terminal.app unterstützt keine anklickbaren Links
- SSH- und tmux-Sitzungen können OSC-Sequenzen je nach Konfiguration entfernen
- Wenn Escape-Sequenzen als Literaltext wie `\e]8;;` angezeigt werden, verwenden Sie `printf '%b'` anstelle von `echo -e` für zuverlässigere Escape-Behandlung

Anzeigeglitches mit Escape-Sequenzen

- Komplexe Escape-Sequenzen (ANSI-Farben, OSC 8-Links) können gelegentlich zu beschädigter Ausgabe führen, wenn sie mit anderen UI-Aktualisierungen überlappen
- Wenn Sie beschädigten Text sehen, versuchen Sie, Ihr Skript auf einfache Textausgabe zu vereinfachen
- Mehrzeilige Statuszeilen mit Escape-Codes sind anfälliger für Rendering-Probleme als einzeilige einfache Texte

Skriptfehler oder Hängen

- Skripte, die mit Nicht-Null-Codes beendet werden oder keine Ausgabe erzeugen, führen dazu, dass die Statuszeile leer wird

- Langsame Skripte blockieren die Statuszeilen-Aktualisierung, bis sie abgeschlossen sind. Halten Sie Skripte schnell, um veraltete Ausgabe zu vermeiden.
- Wenn eine neue Aktualisierung ausgelöst wird, während ein langsames Skript läuft, wird das laufende Skript abgebrochen
- Testen Sie Ihr Skript unabhängig mit Mock-Eingabe, bevor Sie es konfigurieren

Benachrichtigungen teilen sich die Statuszeilen-Zeile

- Systembenachrichtigungen wie MCP-Serverfehler, automatische Updates und Token-Warnungen werden auf der rechten Seite derselben Zeile wie Ihre Statuszeile angezeigt
- Das Aktivieren des ausführlichen Modus fügt einen Token-Zähler zu diesem Bereich hinzu
- Auf schmalen Terminals können diese Benachrichtigungen Ihre Statuszeilen-Ausgabe abschneiden

Part 13: Troubleshooting & Changelog

Fehlerbehebung

Entdecken Sie Lösungen für häufige Probleme bei der Installation und Verwendung von Claude Code.

Installationsprobleme beheben

Tip:

Wenn Sie das Terminal ganz vermeiden möchten, können Sie mit der [Claude Code Desktop-App](#) Claude Code über eine grafische Benutzeroberfläche installieren und verwenden. Laden Sie sie für [macOS](#) oder [Windows](#) herunter und beginnen Sie zu programmieren, ohne eine Befehlszeilenkonfiguration durchzuführen.

Finden Sie die Fehlermeldung oder das Symptom, das Sie sehen:

Was Sie sehen	Lösung
<code>command not found: claude</code> oder <code>'claude' is not recognized</code>	Beheben Sie Ihren PATH
<code>syntax error near unexpected token '<'</code>	Installationsskript gibt HTML zurück
<code>curl: (56) Failure writing output to destination</code>	Laden Sie das Skript zuerst herunter, führen Sie es dann aus
<code>Killed</code> während der Installation unter Linux	Fügen Sie Swap-Speicher für Server mit wenig Speicher hinzu
<code>TLS connect error</code> oder <code>SSL/TLS secure channel</code>	Aktualisieren Sie CA-Zertifikate

Was Sie sehen	Lösung
<code>Failed to fetch version</code> oder kann den Download-Server nicht erreichen	Überprüfen Sie Netzwerk- und Proxy-Einstellungen
<code>irm is not recognized</code> oder <code>&& is not valid</code>	Verwenden Sie den richtigen Befehl für Ihre Shell
<code>Claude Code on Windows requires git-bash</code>	Installieren oder konfigurieren Sie Git Bash
<code>Error loading shared library</code>	Falscher Binär-Variant für Ihr System
<code>Illegal instruction</code> unter Linux	Architektur-Nichtübereinstimmung
<code>dyld: cannot load</code> oder <code>Abort trap</code> unter macOS	Binär-Inkompatibilität
<code>Invoke-Expression: Missing argument in parameter list</code>	Installationsskript gibt HTML zurück
<code>App unavailable in region</code>	Claude Code ist in Ihrem Land nicht verfügbar. Siehe unterstützte Länder .
<code>unable to get local issuer certificate</code>	Konfigurieren Sie Unternehmens-CA-Zertifikate
<code>OAuth error</code> oder <code>403 Forbidden</code>	Beheben Sie die Authentifizierung

Wenn Ihr Problem nicht aufgelistet ist, führen Sie diese Diagnoseschritte durch.

Installationsprobleme debuggen

Überprüfen Sie die Netzwerkverbindung

Das Installationsprogramm lädt von `storage.googleapis.com` herunter. Überprüfen Sie, ob Sie es erreichen können:

```
curl -sI https://storage.googleapis.com
```

Wenn dies fehlschlägt, blockiert Ihr Netzwerk möglicherweise die Verbindung. Häufige Ursachen:

- Unternehmens-Firewalls oder Proxys, die Google Cloud Storage blockieren
- Regionale Netzwerkbeschränkungen: Versuchen Sie ein VPN oder ein alternatives Netzwerk
- TLS/SSL-Probleme: Aktualisieren Sie die CA-Zertifikate Ihres Systems, oder überprüfen Sie, ob `HTTPS_PROXY` konfiguriert ist

Wenn Sie sich hinter einem Unternehmens-Proxy befinden, setzen Sie `HTTPS_PROXY` und `HTTP_PROXY` auf die Adresse Ihres Proxys, bevor Sie installieren. Fragen Sie Ihr IT-Team nach der Proxy-URL, wenn Sie diese nicht kennen, oder überprüfen Sie die Proxy-Einstellungen Ihres Browsers.

Dieses Beispiel setzt beide Proxy-Variablen und führt dann das Installationsprogramm über Ihren Proxy aus:

```
export HTTP_PROXY=http://proxy.example.com:8080
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

Überprüfen Sie Ihren PATH

Wenn die Installation erfolgreich war, aber Sie einen `command not found` - oder `not recognized` -Fehler erhalten, wenn Sie `claude` ausführen, befindet sich das Installationsverzeichnis nicht in Ihrem PATH. Ihre Shell sucht nach Programmen in Verzeichnissen, die in PATH aufgelistet sind, und das Installationsprogramm platziert `claude` unter `~/.local/bin/claude` auf macOS/Linux oder `%USERPROFILE%\local\bin\claude.exe` unter Windows.

Überprüfen Sie, ob sich das Installationsverzeichnis in Ihrem PATH befindet, indem Sie Ihre PATH-Einträge auflisten und nach `local/bin` filtern:

macOS/Linux

```
echo $PATH | tr ':' '\n' | grep local/bin
```

Wenn es keine Ausgabe gibt, fehlt das Verzeichnis. Fügen Sie es zu Ihrer Shell-Konfiguration hinzu:

```
## Zsh (macOS-Standard)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc

## Bash (Linux-Standard)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Alternativ können Sie Ihr Terminal schließen und erneut öffnen.

Überprüfen Sie, ob die Behebung funktioniert hat:

```
claude --version
```

Windows PowerShell

```
$env:PATH -split ';' | Select-String 'local\\bin'
```

Wenn es keine Ausgabe gibt, fügen Sie das Installationsverzeichnis zu Ihrem Benutzer-PATH hinzu:

```
$currentPath = [Environment]::GetEnvironmentVariable('PATH', 'User')
[Environment]::SetEnvironmentVariable('PATH', "$currentPath;$env:USERPROFILE\.local\bin", 'User')
```

Starten Sie Ihr Terminal neu, damit die Änderung wirksam wird.

Überprüfen Sie, ob die Behebung funktioniert hat:

```
claude --version
```

Windows CMD

```
echo %PATH% | findstr /i "local\bin"
```

Wenn es keine Ausgabe gibt, öffnen Sie die Systemeinstellungen, gehen Sie zu Umgebungsvariablen, und fügen Sie `%USERPROFILE%\local\bin` zu Ihrer Benutzer-PATH-Variable hinzu. Starten Sie Ihr Terminal neu.

Überprüfen Sie, ob die Behebung funktioniert hat:

```
claude --version
```

Überprüfen Sie auf widersprüchliche Installationen

Mehrere Claude Code-Installationen können zu Versionskonflikten oder unerwartetem Verhalten führen. Überprüfen Sie, was installiert ist:

macOS/Linux

Listet alle `claude`-Binärdateien auf, die in Ihrem PATH gefunden werden:

```
which -a claude
```

Überprüfen Sie, ob die native Installer- und npm-Versionen vorhanden sind:

```
ls -la ~/.local/bin/claude
```

```
ls -la ~/.claude/local/
```

```
npm -g ls @anthropic-ai/claude-code 2>/dev/null
```

Windows PowerShell

```
where.exe claude
```

```
Test-Path "$env:LOCALAPPDATA\Claude Code\claude.exe"
```

Wenn Sie mehrere Installationen finden, behalten Sie nur eine. Die native Installation unter `~/.local/bin/claude` wird empfohlen. Entfernen Sie alle zusätzlichen Installationen:

Deinstallieren Sie eine globale npm-Installation:

```
npm uninstall -g @anthropic-ai/claude-code
```

Entfernen Sie eine Homebrew-Installation unter macOS:

```
brew uninstall --cask claude-code
```

Überprüfen Sie Verzeichnisberechtigungen

Das Installationsprogramm benötigt Schreibzugriff auf `~/.local/bin/` und `~/.claude/`. Wenn die Installation mit Berechtigungsfehlern fehlschlägt, überprüfen Sie, ob diese Verzeichnisse beschreibbar sind:

```
test -w ~/.local/bin && echo "writable" || echo "not writable"
test -w ~/.claude && echo "writable" || echo "not writable"
```

Wenn eines der Verzeichnisse nicht beschreibbar ist, erstellen Sie das Installationsverzeichnis und setzen Sie Ihren Benutzer als Eigentümer:

```
sudo mkdir -p ~/.local/bin
sudo chown -R $(whoami) ~/.local
```

Überprüfen Sie, ob die Binärdatei funktioniert

Wenn `claude` installiert ist, aber beim Start abstürzt oder hängt, führen Sie diese Überprüfungen durch, um die Ursache einzugrenzen.

Bestätigen Sie, dass die Binärdatei vorhanden und ausführbar ist:

```
ls -la $(which claude)
```

Überprüfen Sie unter Linux auf fehlende gemeinsame Bibliotheken. Wenn `ldd` fehlende Bibliotheken anzeigt, müssen Sie möglicherweise Systempakete installieren. Unter Alpine Linux und anderen musl-basierten Distributionen siehe [Alpine Linux-Setup](#).

```
ldd $(which claude) | grep "not found"
```


Führen Sie eine schnelle Integritätsprüfung durch, dass die Binärdatei ausgeführt werden kann:

```
claude --version
```

Häufige Installationsprobleme

Dies sind die am häufigsten auftretenden Installationsprobleme und deren Lösungen.

Installationsskript gibt HTML statt eines Shell-Skripts zurück

Wenn Sie den Installationsbefehl ausführen, können Sie einen dieser Fehler sehen:

```
bash: line 1: syntax error near unexpected token `<'
bash: line 1: `<!DOCTYPE html>'
```

Unter PowerShell erscheint das gleiche Problem als:

```
Invoke-Expression: Missing argument in parameter list.
```

Dies bedeutet, dass die Installations-URL eine HTML-Seite statt des Installationsskripts zurückgegeben hat. Wenn die HTML-Seite „App unavailable in region“ sagt, ist Claude Code in Ihrem Land nicht verfügbar. Siehe [unterstützte Länder](#).

Andernfalls kann dies aufgrund von Netzwerkproblemen, regionalem Routing oder einer vorübergehenden Dienstunterbrechung geschehen.

Lösungen:

1. Verwenden Sie eine alternative Installationsmethode:

Unter macOS oder Linux installieren Sie über Homebrew:

```
brew install --cask claude-code
```

Unter Windows installieren Sie über WinGet:

```
winget install Anthropic.ClaudeCode
```

1. **Versuchen Sie es nach ein paar Minuten erneut:** Das Problem ist oft vorübergehend. Warten Sie und versuchen Sie den ursprünglichen Befehl erneut.

command not found: claude nach der Installation

Die Installation ist abgeschlossen, aber `claude` funktioniert nicht. Die genaue Fehlermeldung variiert je nach Plattform:

Plattform	Fehlermeldung
macOS	<code>zsh: command not found: claude</code>
Linux	<code>bash: claude: command not found</code>
Windows CMD	<code>'claude' is not recognized as an internal or external command</code>
PowerShell	<code>claude : The term 'claude' is not recognized as the name of a cmdlet</code>

Dies bedeutet, dass sich das Installationsverzeichnis nicht im Suchpfad Ihrer Shell befindet. Siehe [Überprüfen Sie Ihren PATH](#) für die Behebung auf jeder Plattform.

curl: (56) Failure writing output to destination

Der Befehl `curl ... | bash` lädt das Skript herunter und übergibt es direkt an Bash zur Ausführung über eine Pipe (`|`). Dieser Fehler bedeutet, dass die Verbindung unterbrochen wurde, bevor das Skript vollständig heruntergeladen wurde. Häufige Ursachen sind Netzwerkunterbrechungen, das Blockieren des Downloads während der Übertragung oder Systemressourcenlimits.

Lösungen:

1. **Überprüfen Sie die Netzwerkstabilität:** Claude Code-Binärdateien werden auf Google Cloud Storage gehostet. Testen Sie, ob Sie es erreichen können:

```
curl -fsSL https://storage.googleapis.com -o /dev/null
```

Wenn der Befehl stillschweigend abgeschlossen wird, ist Ihre Verbindung in Ordnung und das Problem ist wahrscheinlich vorübergehend. Versuchen Sie den Installationsbefehl erneut. Wenn Sie einen Fehler sehen, blockiert Ihr Netzwerk möglicherweise den Download.

1. Versuchen Sie eine alternative Installationsmethode:

Unter macOS oder Linux:

```
brew install --cask claude-code
```

Unter Windows:

```
winget install Anthropic.ClaudeCode
```

TLS- oder SSL-Verbindungsfehler

Fehler wie `curl: (35) TLS connect error`, `schannel: next InitializeSecurityContext failed`, oder PowerShells `Could not establish trust relationship for the SSL/TLS secure channel` deuten auf TLS-Handshake-Fehler hin.

Lösungen:

1. Aktualisieren Sie Ihre System-CA-Zertifikate:

Unter Ubuntu/Debian:

```
sudo apt-get update && sudo apt-get install ca-certificates
```

Unter macOS über Homebrew:

```
brew install ca-certificates
```

1. Unter Windows aktivieren Sie TLS 1.2 in PowerShell, bevor Sie das Installationsprogramm ausführen:

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
irm https://claude.ai/install.ps1 | iex
```

1. **Überprüfen Sie auf Proxy- oder Firewall-Interferenz:** Unternehmens-Proxys, die TLS-Inspektion durchführen, können diese Fehler verursachen, einschließlich `unable to get local issuer certificate`. Setzen Sie `NODE_EXTRA_CA_CERTS` auf Ihr Unternehmens-CA-Zertifikat-Bundle:

```
export NODE_EXTRA_CA_CERTS=/path/to/corporate-ca.pem
```

Fragen Sie Ihr IT-Team nach der Zertifikatsdatei, wenn Sie diese nicht haben. Sie können auch auf einer direkten Verbindung versuchen, um zu bestätigen, dass der Proxy die Ursache ist.

`Failed to fetch version from storage.googleapis.com`

Das Installationsprogramm konnte den Download-Server nicht erreichen. Dies bedeutet normalerweise, dass `storage.googleapis.com` in Ihrem Netzwerk blockiert ist.

Lösungen:

1. **Testen Sie die Konnektivität direkt:**

```
curl -sI https://storage.googleapis.com
```

1. **Wenn Sie sich hinter einem Proxy befinden,** setzen Sie `HTTPS_PROXY`, damit das Installationsprogramm es durchleiten kann. Siehe [Proxy-Konfiguration](#) für Details.

```
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

1. **Wenn Sie sich in einem eingeschränkten Netzwerk befinden,** versuchen Sie ein anderes Netzwerk oder VPN, oder verwenden Sie eine alternative Installationsmethode:

Unter macOS oder Linux:

```
brew install --cask claude-code
```

Unter Windows:

```
winget install Anthropic.ClaudeCode
```

Windows: `irm` oder `&&` nicht erkannt

Wenn Sie `'irm' is not recognized` oder `The token '&&' is not valid` sehen, führen Sie den falschen Befehl für Ihre Shell aus.

- **`irm` nicht erkannt:** Sie befinden sich in CMD, nicht in PowerShell. Sie haben zwei Optionen:
Öffnen Sie PowerShell, indem Sie im Startmenü nach „PowerShell“ suchen, und führen Sie dann den ursprünglichen Installationsbefehl aus:

```
irm https://claude.ai/install.ps1 | iex
```

Oder bleiben Sie in CMD und verwenden Sie stattdessen das CMD-Installationsprogramm:

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del  
install.cmd
```

- **`&&` nicht gültig:** Sie befinden sich in PowerShell, haben aber den CMD-Installationsbefehl ausgeführt. Verwenden Sie das PowerShell-Installationsprogramm:

```
irm https://claude.ai/install.ps1 | iex
```

Installation auf Linux-Servern mit wenig Speicher beendet

Wenn Sie während der Installation auf einem VPS oder einer Cloud-Instanz `Killed` sehen:

```
Setting up Claude Code...  
Installing Claude Code native build latest...  
bash: line 142: 34803 Killed      "$binary_path" install ${TARGET:+"$TARGET"}
```

Der Linux OOM-Killer hat den Prozess beendet, da dem System der Speicher ausgegangen ist. Claude Code benötigt mindestens 4 GB verfügbaren RAM.

Lösungen:

1. **Fügen Sie Swap-Speicher hinzu**, wenn Ihr Server über begrenzte RAM verfügt. Swap verwendet Festplattenspeicher als Überlauf-Speicher, sodass die Installation auch bei wenig physischem RAM abgeschlossen werden kann. Erstellen Sie eine 2-GB-Swap-Datei und aktivieren Sie sie:

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```

Versuchen Sie dann die Installation erneut:

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

1. **Schließen Sie andere Prozesse**, um Speicher vor der Installation freizugeben.
2. **Verwenden Sie eine größere Instanz**, wenn möglich. Claude Code benötigt mindestens 4 GB RAM.

Installation hängt in Docker

Wenn Sie Claude Code in einem Docker-Container installieren, kann die Installation als Root in `/` zu Hängern führen.

Lösungen:

1. **Setzen Sie ein Arbeitsverzeichnis**, bevor Sie das Installationsprogramm ausführen. Bei Ausführung von `/` scannt das Installationsprogramm das gesamte Dateisystem, was zu übermäßiger Speichernutzung führt. Das Setzen von `WORKDIR` begrenzt den Scan auf ein kleines Verzeichnis:

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

1. **Erhöhen Sie die Docker-Speicherlimits**, wenn Sie Docker Desktop verwenden:

```
docker build --memory=4g .
```

Windows: Claude Desktop überschreibt `claude` CLI-Befehl

Wenn Sie eine ältere Version von Claude Desktop installiert haben, kann sie möglicherweise eine `Claude.exe` im `WindowsApps`-Verzeichnis registrieren, die PATH-Priorität über Claude Code CLI hat. Das Ausführen von `claude` öffnet die Desktop-App statt der CLI.

Aktualisieren Sie Claude Desktop auf die neueste Version, um dieses Problem zu beheben.

Windows: „Claude Code on Windows requires git-bash“

Claude Code unter nativem Windows benötigt [Git für Windows](#), das Git Bash enthält.

Wenn Git nicht installiert ist, laden Sie es von git-scm.com/downloads/win herunter und installieren Sie es. Wählen Sie während der Einrichtung „Add to PATH“ aus. Starten Sie Ihr Terminal nach der Installation neu.

Wenn Git bereits installiert ist, aber Claude Code es immer noch nicht finden kann, setzen Sie den Pfad in Ihrer [settings.json-Datei](#):

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

Wenn Ihr Git an einem anderen Ort installiert ist, finden Sie den Pfad, indem Sie `where.exe git` in PowerShell ausführen, und verwenden Sie den `bin\\bash.exe`-Pfad aus diesem Verzeichnis.

Linux: falscher Binär-Variant installiert (musl/glibc-Nichtübereinstimmung)

Wenn Sie nach der Installation Fehler über fehlende gemeinsame Bibliotheken wie `libstdc++.so.6` oder `libgcc_s.so.1` sehen, hat das Installationsprogramm möglicherweise den falschen Binär-Variant für Ihr System heruntergeladen.

```
Error loading shared library libstdc++.so.6: No such file or directory
```

Dies kann auf glibc-basierten Systemen geschehen, auf denen musl-Cross-Compilation-Pakete installiert sind, was das Installationsprogramm dazu veranlasst, das System fälschlicherweise als musl zu erkennen.

Lösungen:**1. Überprüfen Sie, welche libc Ihr System verwendet:**

```
ldd /bin/ls | head -1
```

Wenn es `linux-vdso.so` oder Verweise auf `/lib/x86_64-linux-gnu/` anzeigt, befinden Sie sich auf glibc. Wenn es `musl` anzeigt, befinden Sie sich auf musl.

- 1. Wenn Sie sich auf glibc befinden, aber die musl-Binärdatei erhalten haben**, entfernen Sie die Installation und installieren Sie erneut. Sie können die richtige Binärdatei auch manuell aus dem GCS-Bucket unter <https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json> herunterladen. Erstellen Sie ein [GitHub-Problem](#) mit der Ausgabe von `ldd /bin/ls` und `ls /lib/libc.musl*`.
- 2. Wenn Sie sich tatsächlich auf musl befinden** (Alpine Linux), installieren Sie die erforderlichen Pakete:

```
apk add libgcc libstdc++ ripgrep
```

Illegal instruction unter Linux

Wenn das Installationsprogramm `Illegal instruction` statt der OOM-Nachricht `Killed` ausgibt, stimmt die heruntergeladene Binärdatei nicht mit Ihrer CPU-Architektur überein. Dies geschieht häufig auf ARM-Servern, die eine x86-Binärdatei erhalten, oder auf älteren CPUs, denen erforderliche Befehlssätze fehlen.

```
bash: line 142: 2238232 Illegal instruction   "$binary_path" install ${TARGET:
+ "${TARGET"}}
```

Lösungen:**1. Überprüfen Sie Ihre Architektur:**

```
uname -m
```


`x86_64` bedeutet 64-Bit Intel/AMD, `aarch64` bedeutet ARM64. Wenn die Binärdatei nicht übereinstimmt, [erstellen Sie ein GitHub-Problem](#) mit der Ausgabe.

1. **Versuchen Sie eine alternative Installationsmethode**, während das Architektur-Problem gelöst wird:

```
brew install --cask claude-code
```

`dyld: cannot load` unter macOS

Wenn Sie während der Installation `dyld: cannot load` oder `Abort trap: 6` sehen, ist die Binärdatei mit Ihrer macOS-Version oder Hardware nicht kompatibel.

```
dyld: cannot load 'claude-2.1.42-darwin-x64' (load command 0x80000034 is unknown)
Abort trap: 6
```

Lösungen:

1. **Überprüfen Sie Ihre macOS-Version:** Claude Code benötigt macOS 13.0 oder später. Öffnen Sie das Apple-Menü und wählen Sie „About This Mac“, um Ihre Version zu überprüfen.
2. **Aktualisieren Sie macOS**, wenn Sie eine ältere Version verwenden. Die Binärdatei verwendet Load-Befehle, die ältere macOS-Versionen nicht unterstützen.
3. **Versuchen Sie Homebrew** als alternative Installationsmethode:

```
brew install --cask claude-code
```

Windows-Installationsprobleme: Fehler in WSL

Sie können die folgenden Probleme in WSL antreffen:

OS/Plattform-Erkennungsprobleme: Wenn Sie während der Installation einen Fehler erhalten, verwendet WSL möglicherweise Windows `npm`. Versuchen Sie:

- Führen Sie `npm config set os linux` vor der Installation aus
- Installieren Sie mit `npm install -g @anthropic-ai/claude-code --force --no-os-check`. Verwenden Sie nicht `sudo`.

Node nicht gefunden-Fehler: Wenn Sie `exec: node: not found` sehen, wenn Sie `claude` ausführen, verwendet Ihre WSL-Umgebung möglicherweise eine Windows-Installation von Node.js. Sie können dies mit `which npm` und `which node` bestätigen, die

auf Linux-Pfade zeigen sollten, die mit `/usr/` beginnen, anstatt mit `/mnt/c/`. Um dies zu beheben, versuchen Sie, Node über den Paketmanager Ihrer Linux-Distribution oder über `nvm` zu installieren.

nvm-Versionskonflikte: Wenn Sie `nvm` sowohl in WSL als auch in Windows installiert haben, können Sie Versionskonflikte erleben, wenn Sie Node-Versionen in WSL wechseln. Dies geschieht, weil WSL den Windows-PATH standardmäßig importiert, was dazu führt, dass Windows `nvm/npm` Vorrang vor der WSL-Installation hat.

Sie können dieses Problem identifizieren durch:

- Ausführen von `which npm` und `which node` - wenn sie auf Windows-Pfade zeigen (beginnend mit `/mnt/c/`), werden Windows-Versionen verwendet
- Fehlerhafte Funktionalität nach dem Wechsel von Node-Versionen mit `nvm` in WSL

Um dieses Problem zu beheben, beheben Sie Ihren Linux-PATH, um sicherzustellen, dass die Linux-Node/npm-Versionen Vorrang haben:

Primäre Lösung: Stellen Sie sicher, dass `nvm` ordnungsgemäß in Ihrer Shell geladen wird

Die häufigste Ursache ist, dass `nvm` nicht in nicht-interaktiven Shells geladen wird. Fügen Sie Folgendes zu Ihrer Shell-Konfigurationsdatei (`~/.bashrc`, `~/.zshrc` usw.) hinzu:

```
## Load nvm if it exists
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

Oder führen Sie direkt in Ihrer aktuellen Sitzung aus:

```
source ~/.nvm/nvm.sh
```

Alternative: Passen Sie die PATH-Reihenfolge an

Wenn `nvm` ordnungsgemäß geladen ist, aber Windows-Pfade immer noch Vorrang haben, können Sie Ihre Linux-Pfade explizit in Ihrer Shell-Konfiguration dem PATH voranstellen:

```
export PATH="$HOME/.nvm/versions/node/$(node -v)/bin:$PATH"
```

Warning:

Vermeiden Sie das Deaktivieren des Windows-PATH-Imports über `appendWindowsPath = false`, da dies die Möglichkeit bricht, Windows-Ausführbare aus WSL aufzurufen. Vermeiden Sie auch das Deinstallieren von Node.js von Windows, wenn Sie es für Windows-Entwicklung verwenden.

WSL2-Sandbox-Setup

[Sandboxing](#) wird auf WSL2 unterstützt, erfordert aber die Installation zusätzlicher Pakete. Wenn Sie einen Fehler wie „Sandbox requires socat and bubblewrap“ sehen, wenn Sie `/sandbox` ausführen, installieren Sie die Abhängigkeiten:

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

WSL1 unterstützt Sandboxing nicht. Wenn Sie „Sandboxing requires WSL2“ sehen, müssen Sie auf WSL2 aktualisieren oder Claude Code ohne Sandboxing ausführen.

Berechtigungsfehler während der Installation

Wenn das native Installationsprogramm mit Berechtigungsfehlern fehlschlägt, ist das Zielverzeichnis möglicherweise nicht beschreibbar. Siehe [Überprüfen Sie Verzeichnisberechtigungen](#).

Wenn Sie zuvor mit npm installiert haben und npm-spezifische Berechtigungsfehler erhalten, wechseln Sie zum nativen Installationsprogramm:

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

Berechtigungen und Authentifizierung

Diese Abschnitte behandeln Anmeldefehler, Token-Probleme und Verhalten bei Berechtigungsaufforderungen.

Wiederholte Berechtigungsaufforderungen

Wenn Sie sich wiederholt zum Genehmigen der gleichen Befehle aufgefordert finden, können Sie bestimmte Tools mit dem Befehl `/permissions` ohne Genehmigung ausführen lassen. Siehe [Berechtigungsdokumentation](#).

Authentifizierungsprobleme

Wenn Sie Authentifizierungsprobleme haben:

1. Führen Sie `/logout` aus, um sich vollständig abzumelden
2. Schließen Sie Claude Code
3. Starten Sie mit `claude` neu und schließen Sie den Authentifizierungsprozess ab

Wenn der Browser während der Anmeldung nicht automatisch geöffnet wird, drücken Sie `c`, um die OAuth-URL in Ihre Zwischenablage zu kopieren, und fügen Sie sie dann manuell in Ihren Browser ein.

OAuth-Fehler: Ungültiger Code

Wenn Sie `OAuth error: Invalid code. Please make sure the full code was copied` sehen, ist der Anmeldecode abgelaufen oder wurde während des Kopierens gekürzt.

Lösungen:

- Drücken Sie Enter, um die Anmeldung schnell nach dem Öffnen des Browsers erneut zu versuchen
- Geben Sie `c` ein, um die vollständige URL zu kopieren, wenn der Browser nicht automatisch geöffnet wird
- Wenn Sie eine Remote-/SSH-Sitzung verwenden, kann der Browser auf dem falschen Computer geöffnet werden. Kopieren Sie die im Terminal angezeigte URL und öffnen Sie sie stattdessen in Ihrem lokalen Browser.

403 Forbidden nach der Anmeldung

Wenn Sie `API Error: 403 {"error":{"type":"forbidden","message":"Request not allowed"}}` nach der Anmeldung sehen:

- **Claude Pro/Max-Benutzer:** Überprüfen Sie, ob Ihr Abonnement unter [claude.ai/settings](#) aktiv ist
- **Console-Benutzer:** Bestätigen Sie, dass Ihr Konto die Rolle „Claude Code“ oder „Developer“ hat, die von Ihrem Administrator zugewiesen wurde
- **Hinter einem Proxy:** Unternehmens-Proxys können API-Anfragen beeinträchtigen. Siehe [Netzwerkkonfiguration](#) für Proxy-Setup.

OAuth-Anmeldung schlägt in WSL2 fehl

Browser-basierte Anmeldung in WSL2 kann fehlschlagen, wenn WSL Ihren Windows-Browser nicht öffnen kann. Setzen Sie die Umgebungsvariable **BROWSER** :

```
export BROWSER="/mnt/c/Program Files/Google/Chrome/Application/chrome.exe"
claude
```

Oder kopieren Sie die URL manuell: Wenn die Anmeldungsaufforderung angezeigt wird, drücken Sie **c** , um die OAuth-URL zu kopieren, und fügen Sie sie dann in Ihren Windows-Browser ein.

„Not logged in“ oder Token abgelaufen

Wenn Claude Code Sie auffordert, sich nach einer Sitzung erneut anzumelden, ist Ihr OAuth-Token möglicherweise abgelaufen.

Führen Sie **/login** aus, um sich erneut zu authentifizieren. Wenn dies häufig geschieht, überprüfen Sie, ob Ihre Systemuhr genau ist, da die Token-Validierung von korrekten Zeitstempeln abhängt.

Konfigurationsdateispeicherorte

Claude Code speichert die Konfiguration an mehreren Orten:

Datei	Zweck
<code>~/.claude/settings.json</code>	Benutzereinstellungen (Berechtigungen, hooks, Modellüberschreibungen)
<code>.claude/settings.json</code>	Projekteinstellungen (in die Quellcodeverwaltung eingchecked)
<code>.claude/settings.local.json</code>	Lokale Projekteinstellungen (nicht committed)
<code>~/.claude.json</code>	Globaler Status (Design, OAuth, MCP-Server)
<code>.mcp.json</code>	Projekt-MCP-Server (in die Quellcodeverwaltung eingchecked)
<code>managed-mcp.json</code>	Verwaltete MCP-Server
Verwaltete Einstellungen	Verwaltete Einstellungen (serververwaltet, MDM/OS-Ebenen-Richtlinien oder dateibasiert)

Unter Windows bezieht sich `~` auf Ihr Benutzer-Stammverzeichnis, z. B. `C:\Users\YourName`.

Weitere Informationen zum Konfigurieren dieser Dateien finden Sie unter [Einstellungen](#) und [MCP](#).

Konfiguration zurücksetzen

Um Claude Code auf Standardeinstellungen zurückzusetzen, können Sie die Konfigurationsdateien entfernen:

```
## Reset all user settings and state
rm ~/.claude.json
rm -rf ~/.claude/

## Reset project-specific settings
rm -rf .claude/
rm .mcp.json
```

Warning:

Dies entfernt alle Ihre Einstellungen, MCP-Serverkonfigurationen und Sitzungsverlauf.

Leistung und Stabilität

Diese Abschnitte behandeln Probleme im Zusammenhang mit Ressourcennutzung, Reaktionsfähigkeit und Suchverhalten.

Hohe CPU- oder Speichernutzung

Claude Code ist für die Zusammenarbeit mit den meisten Entwicklungsumgebungen konzipiert, kann aber bei der Verarbeitung großer Codebases erhebliche Ressourcen verbrauchen. Wenn Sie Leistungsprobleme haben:

1. Verwenden Sie `/compact` regelmäßig, um die Kontextgröße zu reduzieren
2. Schließen und starten Sie Claude Code zwischen großen Aufgaben neu
3. Erwägen Sie, große Build-Verzeichnisse zu Ihrer `.gitignore`-Datei hinzuzufügen

Befehl hängt oder friert ein

Wenn Claude Code nicht reagiert:

1. Drücken Sie `Ctrl+C`, um zu versuchen, den aktuellen Vorgang abzubrechen

2. Wenn nicht reagiert, müssen Sie möglicherweise das Terminal schließen und neu starten

Such- und Erkennungsprobleme

Wenn das Such-Tool, `@file`-Erwähnungen, benutzerdefinierte Agenten und benutzerdefinierte Skills nicht funktionieren, installieren Sie das System `ripgrep` :

```
## macOS (Homebrew)
brew install ripgrep

## Windows (winget)
winget install BurntSushi.ripgrep.MSVC

## Ubuntu/Debian
sudo apt install ripgrep

## Alpine Linux
apk add ripgrep

## Arch Linux
pacman -S ripgrep
```

Setzen Sie dann `USE_BUILTIN_RIPGREP=0` in Ihrer [Umgebung](#).

Langsame oder unvollständige Suchergebnisse auf WSL

Leistungseinbußen beim Lesen von Festplatten beim [Arbeiten über Dateisysteme auf WSL](#) können zu weniger als erwarteten Übereinstimmungen führen, wenn Sie Claude Code auf WSL verwenden. Die Suche funktioniert immer noch, gibt aber weniger Ergebnisse zurück als auf einem nativen Dateisystem.

Note:

`/doctor` zeigt in diesem Fall die Suche als OK an.

Lösungen:

1. **Senden Sie spezifischere Suchen:** Reduzieren Sie die Anzahl der durchsuchten Dateien, indem Sie Verzeichnisse oder Dateitypen angeben: „Search for JWT validation logic in the auth-service package“ oder „Find use of md5 hash in JS files“.

2. **Verschieben Sie das Projekt auf das Linux-Dateisystem:** Stellen Sie sicher, dass sich Ihr Projekt auf dem Linux-Dateisystem (`/home/`) statt auf dem Windows-Dateisystem (`/mnt/c/`) befindet.
3. **Verwenden Sie stattdessen natives Windows:** Erwägen Sie, Claude Code nativ unter Windows statt über WSL auszuführen, um eine bessere Dateisystem-Leistung zu erhalten.

IDE-Integrationsprobleme

Wenn Claude Code sich nicht mit Ihrer IDE verbindet oder sich in einem IDE-Terminal unerwartet verhält, versuchen Sie die folgenden Lösungen.

JetBrains IDE nicht erkannt auf WSL2

Wenn Sie Claude Code auf WSL2 mit JetBrains IDEs verwenden und Fehler „No available IDEs detected“ erhalten, liegt dies wahrscheinlich an WSL2s Netzwerkkonfiguration oder Windows Firewall, die die Verbindung blockiert.

WSL2-Netzwerkmodi

WSL2 verwendet standardmäßig NAT-Netzwerk, das IDE-Erkennung verhindern kann. Sie haben zwei Optionen:

Option 1: Konfigurieren Sie Windows Firewall (empfohlen)

1. Finden Sie Ihre WSL2-IP-Adresse:

```
wsl hostname -I
## Example output: 172.21.123.45
```

1. Öffnen Sie PowerShell als Administrator und erstellen Sie eine Firewall-Regel:

```
New-NetFirewallRule -DisplayName "Allow WSL2 Internal Traffic" -Direction Inbound
-Protocol TCP -Action Allow -RemoteAddress 172.21.0.0/16 -LocalAddress 172.21.0.0/16
```

Passen Sie den IP-Bereich basierend auf Ihrem WSL2-Subnetz aus Schritt 1 an.

1. Starten Sie sowohl Ihre IDE als auch Claude Code neu

Option 2: Wechseln Sie zu gespiegeltem Netzwerk

Fügen Sie zu `.wslconfig` in Ihrem Windows-Benutzerverzeichnis hinzu:


```
[wsl2]
networkingMode=mirrored
```

Starten Sie dann WSL mit `wsl --shutdown` von PowerShell neu.

Note:

Diese Netzwerkprobleme betreffen nur WSL2. WSL1 verwendet das Netzwerk des Hosts direkt und erfordert diese Konfigurationen nicht.

Weitere JetBrains-Konfigurationstipps finden Sie im [JetBrains IDE-Leitfaden](#).

Melden Sie Windows IDE-Integrationsprobleme

Wenn Sie IDE-Integrationsprobleme unter Windows haben, [erstellen Sie ein Problem](#) mit den folgenden Informationen:

- Umgebungstyp: natives Windows (Git Bash) oder WSL1/WSL2
- WSL-Netzwerkmodus, falls zutreffend: NAT oder gespiegelt
- IDE-Name und -Version
- Claude Code-Erweiterungs-/Plugin-Version
- Shell-Typ: Bash, Zsh, PowerShell usw.

Escape-Taste funktioniert nicht in JetBrains IDE-Terminals

Wenn Sie Claude Code in JetBrains-Terminals verwenden und die `Esc`-Taste den Agenten nicht wie erwartet unterbricht, liegt dies wahrscheinlich an einem Tastenkombinations-Konflikt mit JetBrains' Standardverknüpfungen.

Um dieses Problem zu beheben:

1. Gehen Sie zu Einstellungen → Tools → Terminal
2. Entweder:
 - Deaktivieren Sie „Move focus to the editor with Escape“, oder
 - Klicken Sie auf „Configure terminal keybindings“ und löschen Sie die Verknüpfung „Switch focus to Editor“
3. Wenden Sie die Änderungen an

Dies ermöglicht der `Esc`-Taste, Claude Code-Operationen ordnungsgemäß zu unterbrechen.

Markdown-Formatierungsprobleme

Claude Code generiert manchmal Markdown-Dateien mit fehlenden Sprach-Tags auf Code-Zäunen, was sich auf Syntax-Hervorhebung und Lesbarkeit in GitHub, Editoren und Dokumentations-Tools auswirken kann.

Fehlende Sprach-Tags in Code-Blöcken

Wenn Sie Code-Blöcke wie diesen in generiertem Markdown bemerken:

```
```
function example() {
 return "hello";
}
```text
```

Statt ordnungsgemäß gekennzeichnete Blöcke wie:

```
```javascript
function example() {
 return "hello";
}
```text
```

Lösungen:

1. **Bitten Sie Claude, Sprach-Tags hinzuzufügen:** Fordern Sie „Add appropriate language tags to all code blocks in this markdown file“ an.
2. **Verwenden Sie Post-Processing-Hooks:** Richten Sie automatische Formatierungs-Hooks ein, um fehlende Sprach-Tags zu erkennen und hinzuzufügen. Siehe [Auto-format code after edits](#) für ein Beispiel eines PostToolUse-Formatierungs-Hooks.
3. **Manuelle Überprüfung:** Überprüfen Sie nach der Generierung von Markdown-Dateien diese auf ordnungsgemäße Code-Block-Formatierung und fordern Sie Korrekturen an, falls erforderlich.

Inkonsistente Abstände und Formatierung

Wenn generiertes Markdown übermäßige Leerzeilen oder inkonsistente Abstände hat:

Lösungen:

1. **Fordern Sie Formatierungskorrekturen an:** Bitten Sie Claude, „Fix spacing and formatting issues in this markdown file“ zu beheben.
2. **Verwenden Sie Formatierungs-Tools:** Richten Sie Hooks ein, um Markdown-Formatierer wie `prettier` oder benutzerdefinierte Formatierungs-Skripte auf generierte Markdown-Dateien auszuführen.
3. **Geben Sie Formatierungspräferenzen an:** Fügen Sie Formatierungsanforderungen in Ihre Aufforderungen oder Projekt-[Memory](#)-Dateien ein.

Reduzieren Sie Markdown-Formatierungsprobleme

Um Formatierungsprobleme zu minimieren:

- **Seien Sie explizit in Anforderungen:** Fordern Sie „properly formatted markdown with language-tagged code blocks“ an
- **Verwenden Sie Projektkonventionen:** Dokumentieren Sie Ihren bevorzugten Markdown-Stil in `CLAUDE.md`
- **Richten Sie Validierungs-Hooks ein:** Verwenden Sie Post-Processing-Hooks, um häufige Formatierungsprobleme automatisch zu überprüfen und zu beheben

Weitere Hilfe erhalten

Wenn Sie Probleme haben, die hier nicht behandelt werden:

1. Verwenden Sie den Befehl `/bug` in Claude Code, um Probleme direkt an Anthropic zu melden
2. Überprüfen Sie das [GitHub-Repository](#) auf bekannte Probleme
3. Führen Sie `/doctor` aus, um Probleme zu diagnostizieren. Es überprüft:
 - Installationstyp, Version und Such-Funktionalität
 - Auto-Update-Status und verfügbare Versionen
 - Ungültige Einstellungsdateien (fehlerhaftes JSON, falsche Typen)
 - MCP-Serverkonfigurationsfehler
 - Tastenkombinations-Konfigurationsprobleme
 - Kontext-Nutzungswarnungen (große CLAUDE.md-Dateien, hohe MCP-Token-Nutzung, unerreichbare Berechtigungsregeln)
 - Plugin- und Agent-Ladefehler
4. Fragen Sie Claude direkt nach seinen Fähigkeiten und Funktionen - Claude hat integrierten Zugriff auf seine Dokumentation