

Claude Code

完全ドキュメント

By Anthropic

作成日：2026-03-17

ターミナルで動作するエージェント型コーディングツール。
コードベースを理解し、より速くコーディングできるよう支援します。

code.claude.com/docs/ja からコンパイル
最新版はウェブサイトをご覧ください。

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

Table of Contents

Part 1: Getting Started	5
Claude Code の概要	5
クイックスタート	
デスクトップアプリを始める	20
Claude Code をセットアップする	
Claude Code の仕組み	35
Part 2: Core Usage	45
Claude Code を拡張する	45
インタラクティブモード	57
Claude Code のベストプラクティス	
一般的なワークフロー	94
Part 3: Commands & Reference	
CLI リファレンス	118
キーボードショートカットのカスタマイズ	129
Part 4: Configuration	140
権限を設定する	140
Claude があなたのプロジェクトを記憶する方法	
Claude Code の設定	163
モデル設定	213
出力スタイル	222
高速モードでレスポンスを高速化	225
コストを効果的に管理する	230
Part 5: Extensibility	238
Hooks リファレンス	238
hooks でワークフローを自動化する	
MCP を使用して Claude Code をツールに接続する	
スキルで Claude を拡張する	365
プラグインを作成する	390
プラグインリファレンス	403
プラグインマーケットプレイスの作成と配布	425
マーケットプレイスから事前構築されたプラグインを発見してインストールする	455
Part 6: Advanced & Automation	468
カスタムサブエージェントの作成	468

Claude Code セッションのチームを調整する	
Claude Code をプログラムで実行する	
任意のデバイスからローカルセッションを続行する Remote Control	
スケジュールに従ってプロンプトを実行する	522
コードレビュー	527

Part 7: CI/CD & Integrations

Claude Code GitHub Actions	533
Claude Code GitLab CI/CD	553

Part 8: IDE & Platform Integration

VS Code で Claude Code を使用する	
JetBrains IDEs	584
Claude Code Desktop を使用する	
Chrome で Claude Code を使用する（ベータ版）	
Slack での Claude Code	621
ウェブ上の Claude Code	628

Part 9: Security & Privacy

セキュリティ	649
サンドボックス	654
Checkpointing	665
データ使用	667
法的およびコンプライアンス	673
ゼロデータ保持	675

Part 10: Enterprise & Monitoring

チームの使用状況を分析で追跡する	678
監視	685
エンタープライズデプロイメント概要	701
サーバー管理設定を構成する（パブリックベータ版）	707

Part 11: Cloud Providers

Amazon Bedrock 上の Claude Code	
Google Vertex AI 上の Claude Code	
Claude Code on Microsoft Foundry	726
LLM gateway設定	729
エンタープライズネットワーク設定	734

Part 12: Environment Setup

開発コンテナ	737
ターミナルセットアップを最適化する	740

ステータスラインをカスタマイズする

743

Part 13: Troubleshooting & Changelog

776

トラブルシューティング

776

Part 1: Getting Started

Claude Code の概要

Claude Code は agentic coding ツールで、コードベースを読み取り、ファイルを編集し、コマンドを実行し、開発ツールと統合します。ターミナル、IDE、デスクトップアプリ、ブラウザで利用できます。

Claude Code は AI を活用したコーディングアシスタントで、機能の構築、バグの修正、開発タスクの自動化を支援します。コードベース全体を理解し、複数のファイルとツール全体で作業して、タスクを完了できます。

はじめに

環境を選択してはじめましょう。ほとんどのサーフェスには [Claude サブスクリプション](#) または [Anthropic Console](#) アカウントが必要です。Terminal CLI と VS Code は [サードパーティプロバイダー](#) もサポートしています。

Terminal

ターミナルで Claude Code を直接操作するための機能豊富な CLI です。ファイルを編集し、コマンドを実行し、コマンドラインからプロジェクト全体を管理できます。

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.

その後、任意のプロジェクトで Claude Code を開始します：

```
cd your-project  
claude
```

初回使用時にログインするよう求められます。これで完了です！ [クイックスタートに進む](#)
[→](#)

Tip:

高度なセットアップでインストールオプション、手動更新、またはアンインストール手順を参照してください。問題が発生した場合は [トラブルシューティング](#) にアクセスしてください。

VS Code

VS Code 拡張機能は、インラインの差分、@-メンション、プラン確認、会話履歴をエディター内で直接提供します。

- [VS Code 用にインストール](#)
- [Cursor 用にインストール](#)

または、拡張機能ビュー（Mac では `Cmd+Shift+X`、Windows/Linux では `Ctrl+Shift+X`）で「Claude Code」を検索します。インストール後、コマンドパレット（`Cmd+Shift+P` / `Ctrl+Shift+P`）を開き、「Claude Code」と入力して、**新しいタブで開く**を選択します。

[VS Code ではじめる →](#)

Desktop app

IDE またはターミナルの外で Claude Code を実行するためのスタンドアロンアプリです。差分を視覚的に確認し、複数のセッションを並行実行し、定期的なタスクをスケジュール設定し、クラウドセッションを開始できます。

ダウンロードしてインストール：

- [macOS](#)（Intel および Apple Silicon）
- [Windows](#)（x64）
- [Windows ARM64](#)（リモートセッションのみ）

インストール後、Claude を起動し、サインインして、**Code** タブをクリックしてコーディングを開始します。[有料サブスクリプション](#)が必要です。

[デスクトップアプリについて詳しく →](#)

Web

ローカルセットアップなしでブラウザで Claude Code を実行します。長時間実行されるタスクを開始して完了を待つ、ローカルにないリポジトリで作業する、または複数のタスクを並行実行できます。デスクトップブラウザと Claude iOS アプリで利用できます。

claude.ai/code でコーディングを開始します。

[Web ではじめる →](#)

JetBrains

IntelliJ IDEA、PyCharm、WebStorm、その他の JetBrains IDE 用のプラグインで、インタラクティブな差分表示と選択コンテキスト共有機能があります。

JetBrains Marketplace から [Claude Code プラグイン](#) をインストールして、IDE を再起動します。

[JetBrains ではじめる →](#)

できること

Claude Code を使用できるいくつかの方法を紹介します：

Claude Code は、1 日を費やす退屈なタスクを処理します：テストされていないコードのテスト作成、プロジェクト全体のリントエラー修正、マージ競合の解決、依存関係の更新、リリースノートの作成。

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

プレーンテキストで必要なことを説明します。Claude Code はアプローチを計画し、複数のファイル全体でコードを作成し、動作を確認します。

バグの場合は、エラーメッセージを貼り付けるか、症状を説明します。Claude Code はコードベース全体で問題をトレースし、根本原因を特定し、修正を実装します。詳細な例は [一般的なワークフロー](#) を参照してください。

Claude Code は git と直接連携します。変更をステージングし、コミットメッセージを作成し、ブランチを作成し、プルリクエストを開きます。

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

CI では、[GitHub Actions](#) または [GitLab CI/CD](#) でコードレビューと問題トリアージを自動化できます。

[Model Context Protocol \(MCP\)](#) は、AI ツールを外部データソースに接続するためのオープンスタンダードです。MCP を使用すると、Claude Code は Google Drive のデザインドキュメントを読み取り、Jira のチケットを更新し、Slack からデータをプルするか、独自のカスタムツールリングを使用できます。

CLAUDE.md は、プロジェクトルートに追加するマークダウンファイルで、Claude Code はすべてのセッションの開始時に読み取ります。コーディング標準、アーキテクチャの決定、推奨ライブラリ、レビューチェックリストを設定するために使用します。Claude は、作業中に **自動メモリ** も構築し、ビルドコマンドやデバッグの洞察などの学習内容を保存し、何も記述することなくセッション全体で共有します。

カスタムコマンド を作成して、チームが共有できる反復可能なワークフローをパッケージ化します（`/review-pr` や `/deploy-staging` など）。

フック を使用すると、ファイル編集後の自動フォーマットやコミット前のリント実行など、Claude Code アクションの前後にシェルコマンドを実行できます。

複数の Claude Code エージェント を生成して、タスクの異なる部分で同時に作業します。リードエージェントが作業を調整し、サブタスクを割り当て、結果をマージします。

完全にカスタムなワークフローの場合、**Agent SDK** を使用すると、Claude Code のツールと機能を活用した独自のエージェントを構築でき、オーケストレーション、ツールアクセス、権限を完全に制御できます。

Claude Code は構成可能で、Unix 哲学に従います。ログをパイプして、CI で実行するか、他のツールと連鎖させます：

```
## ログを監視してアラートを取得
tail -f app.log | claude -p "Slack me if you see any anomalies"

## CI で翻訳を自動化
claude -p "translate new strings into French and raise a PR for review"

## ファイル全体でバルク操作
git diff main --name-only | claude -p "review these changed files for security issues"
```

すべてのコマンドとフラグのセットについては、**CLI リファレンス** を参照してください。

セッションは単一のサーフェスに限定されません。コンテキストが変わるにつれて、環境間で作業を移動します：

- デスクから離れて、携帯電話または **リモートコントロール** を使用した任意のブラウザから作業を続けます
- **Web** または **iOS アプリ** で長時間実行されるタスクを開始し、`/teleport` を使用してターミナルにプルします

- ・ターミナルセッションを [デスクトップアプリ](#) に `/desktop` で渡して、視覚的な差分確認を行います
- ・チームチャットからタスクをルーティング：[Slack](#) で `@Claude` にメンションしてバグレポートを送信し、プルリクエストを取得します

Claude Code をどこでも使用する

各サーフェスは同じ基盤となる Claude Code エンジンに接続するため、CLAUDE.md ファイル、設定、MCP サーバーはすべてのサーフェスで機能します。

上記の [Terminal](#)、[VS Code](#)、[JetBrains](#)、[Desktop](#)、[Web](#) 環境を超えて、Claude Code は CI/CD、チャット、ブラウザワークフローと統合します：

実行したいこと	最適なオプション
ローカルセッションを携帯電話または別のデバイスから続行する	リモートコントロール
ローカルでタスクを開始し、モバイルで続行する	Web または Claude iOS アプリ
PR レビューと問題トリアージを自動化する	GitHub Actions または GitLab CI/CD
すべての PR で自動コードレビューを取得する	GitHub Code Review
Slack からプルリクエストへバグレポートをルーティングする	Slack
ライブ Web アプリケーションをデバッグする	Chrome
独自のワークフロー用のカスタムエージェントを構築する	Agent SDK

次のステップ

Claude Code をインストールしたら、これらのガイドでさらに詳しく学べます。

- [クイックスタート](#)：コードベースの探索から修正のコミットまで、最初の実際のタスクを実行します
- [指示とメモリを保存する](#)：CLAUDE.md ファイルと自動メモリで Claude に永続的な指示を与えます
- [一般的なワークフロー](#)と [ベストプラクティス](#)：Claude Code から最大限の価値を得るためのパターン
- [設定](#)：ワークフローに合わせて Claude Code をカスタマイズします
- [トラブルシューティング](#)：一般的な問題の解決策
- code.claude.com：デモ、価格設定、製品の詳細

クイックスタート

Claude Code へようこそ！

このクイックスタートガイドを使用すれば、数分で AI を活用したコーディング支援を利用できます。このガイドを終了する頃には、一般的な開発タスクに Claude Code を使用する方法を理解できるようになります。

始める前に

以下を確認してください：

- ・ターミナルまたはコマンドプロンプトが開いている
 - ターミナルを使用したことがない場合は、[ターミナルガイド](#)をご覧ください
- ・作業するコードプロジェクトがある
- ・[Claude サブスクリプション](#)（Pro、Max、Teams、または Enterprise）、[Claude Console](#) アカウント、または[サポートされているクラウドプロバイダー](#)経由のアクセスがある

Note:

このガイドは、ターミナル CLI について説明しています。Claude Code は、[ウェブ](#)、[デスクトップアプリ](#)、[VS Code](#) および [JetBrains IDE](#)、[Slack](#)、および [GitHub Actions](#) と [GitLab](#) を使用した CI/CD でも利用できます。[すべてのインターフェース](#)を参照してください。

ステップ 1：Claude Code をインストールする

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.

ステップ 2: アカウントにログインする

Claude Code を使用するにはアカウントが必要です。 `claude` コマンドでインタラクティブセッションを開始すると、ログインが必要になります:

```
claude
```

```
## 初回使用時にログインするよう求められます
```

```
/login
```

```
## プロンプトに従ってアカウントでログインします
```

以下のいずれかのアカウントタイプを使用してログインできます：

- [Claude Pro](#)、[Max](#)、[Teams](#)、または [Enterprise](#)（推奨）
- [Claude Console](#)（プリペイドクレジット付き API アクセス）。初回ログイン時に、「Claude Code」ワークスペースが Console に自動的に作成され、コスト追跡が一元化されます。
- [Amazon Bedrock](#)、[Google Vertex AI](#)、または [Microsoft Foundry](#)（エンタープライズクラウドプロバイダー）

ログイン後、認証情報がシステムに保存され、再度ログインする必要はありません。後でアカウントを切り替えるには、`/login` コマンドを使用します。

ステップ 3：最初のセッションを開始する

任意のプロジェクトディレクトリでターミナルを開き、Claude Code を開始します：

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

```
claude
```

Claude Code のウェルカムスクリーンが表示され、セッション情報、最近の会話、および最新の更新が表示されます。利用可能なコマンドについては `/help` を入力するか、前のセッションを続行するには `/resume` を入力します。

Tip:

ログイン後（ステップ 2）、認証情報がシステムに保存されます。詳細については、[認証情報管理](#)を参照してください。

ステップ 4：最初の質問をする

コードベースを理解することから始めましょう。以下のコマンドのいずれかを試してください：

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

Claude がファイルを分析し、概要を提供します。より具体的な質問をすることもできます：

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

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

```
explain the folder structure
```

Claude 自体の機能について質問することもできます：

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

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

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

Note:

Claude Code は必要に応じてプロジェクトファイルを読み込みます。コンテキストを手動で追加する必要はありません。

ステップ 5：最初のコード変更を行う

次に、Claude Code に実際のコーディングを行わせましょう。簡単なタスクを試してください：

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

Claude Code は以下を実行します：

1. 適切なファイルを検索する
2. 提案された変更を表示する
3. 承認を求める
4. 編集を行う

Note:

Claude Code は、ファイルを変更する前に常に許可を求めます。個別の変更を承認するか、セッション中に「すべて承認」モードを有効にすることができます。

ステップ 6：Claude Code で Git を使用する

Claude Code は Git 操作を会話形式にします：

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

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

より複雑な Git 操作を求めることもできます：

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

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

```
help me resolve merge conflicts
```

ステップ 7：バグを修正するか機能を追加する

Claude はデバッグと機能実装に長けています。

自然言語で実現したいことを説明します：


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

または既存の問題を修正します：

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

Claude Code は以下を実行します：

- 関連するコードを特定する
- コンテキストを理解する
- ソリューションを実装する
- 利用可能な場合はテストを実行する

ステップ 8：その他の一般的なワークフローを試す

Claude と連携する方法は多数あります：

コードをリファクタリングする

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

テストを作成する

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

ドキュメントを更新する

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

コードレビュー

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

Tip:

Claude に、役立つ同僚に話しかけるように話しかけてください。実現したいことを説明すれば、Claude がそこに到達するのを支援します。

重要なコマンド

日常的に使用する最も重要なコマンドは以下の通りです：

コマンド	機能	例
<code>claude</code>	インタラクティブモードを開始する	<code>claude</code>
<code>claude "task"</code>	1 回限りのタスクを実行する	<code>claude "fix the build error"</code>
<code>claude -p "query"</code>	1 回限りのクエリを実行してから終了する	<code>claude -p "explain this function"</code>
<code>claude -c</code>	現在のディレクトリで最新の会話を続行する	<code>claude -c</code>
<code>claude -r</code>	前のセッションを再開する	<code>claude -r</code>
<code>claude commit</code>	Git コミットを作成する	<code>claude commit</code>
<code>/clear</code>	会話履歴をクリアする	<code>/clear</code>
<code>/help</code>	利用可能なコマンドを表示する	<code>/help</code>
<code>exit</code> または Ctrl+C	Claude Code を終了する	<code>exit</code>

コマンドの完全なリストについては、[CLI リファレンス](#)を参照してください。

初心者向けのプロのヒント

詳細については、[ベストプラクティス](#)および[一般的なワークフロー](#)を参照してください。

代わりに：‘バグを修正する’

試してください：‘ユーザーが間違った認証情報を入力した後に空白の画面が表示されるログインバグを修正する’

複雑なタスクをステップに分割します：

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

変更を加える前に、Claude にコードを理解させます：

```
analyze the database schema
```

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

- **?** を押してすべての利用可能なキーボードショートカットを表示する
- Tab キーでコマンド補完を使用する
- ↑ キーでコマンド履歴を表示する
- **/** を入力してすべてのコマンドと skills を表示する

次のステップ

基本を学習したので、より高度な機能を探索してください：

- [Claude Code の仕組み](#) agentic ループ、組み込みツール、および Claude Code がプロジェクトと相互作用する方法を理解する
- [ベストプラクティス](#) 効果的なプロンプティングとプロジェクト設定でより良い結果を得る
- [一般的なワークフロー](#) 一般的なタスクのステップバイステップガイド
- [Claude Code を拡張する](#) CLAUDE.md、skills、hooks、MCP などカスタマイズする

ヘルプを取得する

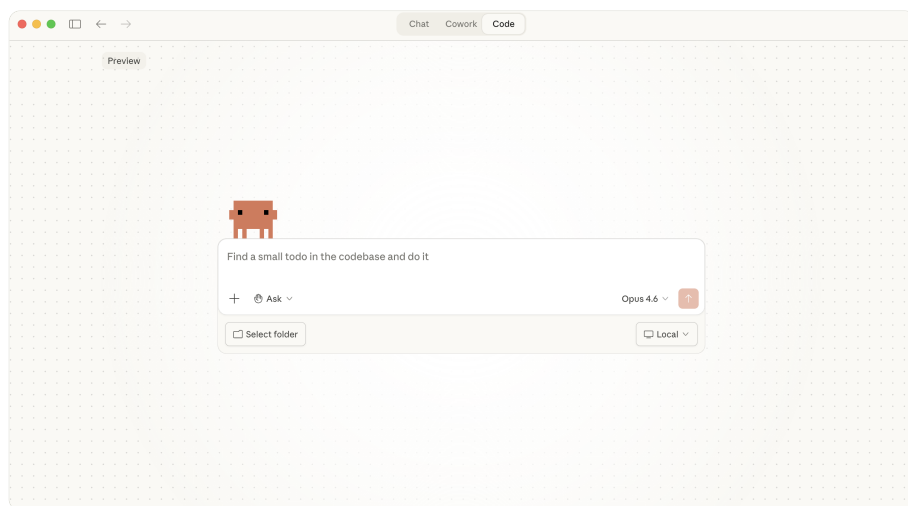
- **Claude Code 内**： **/help** を入力するか、「how do I...」と質問する
- **ドキュメント**：ここにいます！他のガイドを参照してください
- **コミュニティ**： [Discord](#) に参加して、ヒントとサポートを受けてください

デスクトップアプリを始める

Claude Code をデスクトップにインストールして、最初のコーディングセッションを開始します

デスクトップアプリは、グラフィカルインターフェース付きの Claude Code を提供します。ビジュアル diff レビュー、ライブアプリプレビュー、GitHub PR 監視と自動マージ、Git worktree 分離による並列セッション、スケジュール済みタスク、リモートでタスクを実行する機能があります。ターミナルは不要です。

このページでは、アプリのインストールと最初のセッションの開始について説明します。既にセットアップが完了している場合は、[Claude Code Desktop を使用する](#)で完全なリファレンスを参照してください。



Code タブが選択されている Claude Code Desktop インターフェースを示しており、プロンプトボックス、権限モードセクター（Ask permissions に設定）、モデルピッカー、フォルダセクター、Local environment オプションが表示されています

デスクトップアプリには3つのタブがあります。

- **Chat:** ファイルアクセスなしの一般的な会話。claude.ai と同様です。
- **Cowork:** クラウド VM で独自の環境を持つ自律型バックグラウンドエージェント。あなたが他の作業をしている間も独立して実行できます。

- **Code:** ローカルファイルへの直接アクセスを備えたインタラクティブなコーディングアシスタント。各変更をリアルタイムでレビューして承認します。

Chat と Cowork は [Claude Desktop サポート記事](#)で説明されています。このページは **Code** タブに焦点を当てています。

Note:

Claude Code には [Pro](#)、[Max](#)、[Teams](#)、または [Enterprise サブスクリプション](#)が必要です。

インストール

Step 1: アプリをダウンロードする

お使いのプラットフォーム用に Claude をダウンロードしてください。

- [macOS](#) Intel と Apple Silicon 用のユニバーサルビルド
- [Windows](#) x64 プロセッサ用

Windows ARM64 の場合は、[ここからダウンロード](#)してください。

Linux は現在サポートされていません。

Step 2: サインインする

Applications フォルダ (macOS) または Start メニュー (Windows) から Claude を起動します。Anthropic アカウントでサインインしてください。

Step 3: Code タブを開く

上部中央の **Code** タブをクリックします。Code をクリックするとアップグレードを促すメッセージが表示される場合は、最初に[有料プランにサブスクライブ](#)する必要があります。オンラインでサインインするよう促すメッセージが表示される場合は、サインインを完了してアプリを再起動してください。403 エラーが表示される場合は、[認証のトラブルシューティング](#)を参照してください。

デスクトップアプリには Claude Code が含まれています。Node.js または CLI を別途インストールする必要はありません。ターミナルから **claude** を使用するには、CLI を別途インストールしてください。[CLI を始める](#)を参照してください。

最初のセッションを開始する

Code タブを開いた状態で、プロジェクトを選択して Claude に何かをさせます。

Step 1: 環境とフォルダを選択する

Local を選択して、Claude をマシン上で実行し、ファイルを直接使用します。**Select folder** をクリックして、プロジェクトディレクトリを選択します。

Tip:

よく知っている小さなプロジェクトから始めてください。Claude Code が何ができるかを見るための最速の方法です。Windows では、ローカルセッションが機能するために [Git](#) がインストールされている必要があります。ほとんどの Mac には Git がデフォルトで含まれています。

以下も選択できます。

- **Remote:** Anthropic のクラウドインフラストラクチャでセッションを実行します。アプリを閉じてでも続行します。リモートセッションは [Claude Code on the web](#) と同じインフラストラクチャを使用します。
- **SSH:** SSH 経由でリモートマシンに接続します（独自のサーバー、クラウド VM、または dev コンテナ）。Claude Code はリモートマシンにインストールされている必要があります。

Step 2: モデルを選択する

送信ボタンの横のドロップダウンからモデルを選択します。Opus、Sonnet、Haiku の比較については、[モデル](#) を参照してください。セッション開始後はモデルを変更できません。

Step 3: Claude に何をするかを伝える

Claude に何をしてほしいかを入力します。

- `T000` コメントを見つけて修正する
- メイン関数のテストを追加する
- このコードベースの指示を含む `CLAUDE.md` を作成する

[セッション](#) は、コードについて Claude との会話です。各セッションは独自のコンテキストと変更を追跡するため、複数のタスクに取り組む際に相互に干渉することなく作業できます。

Step 4: 変更をレビューして受け入れる

デフォルトでは、Code タブは [Ask permissions モード](#) で開始されます。このモードでは、Claude が変更を提案し、適用する前にあなたの承認を待ちます。以下が表示されます。

1. 各ファイルで何がかわるかを正確に示す [diff ビュー](#)
2. 各変更を承認または拒否する Accept/Reject ボタン

3. Claude があなたのリクエストを処理する際のリアルタイム更新

変更を拒否すると、Claude は別の方法で進めたいかを尋ねます。承認するまでファイルは変更されません。

次は何をしますか？

最初の編集が完了しました。Desktop ができることすべての完全なリファレンスについては、[Claude Code Desktop を使用する](#)を参照してください。次に試すべきことをいくつか紹介します。

割り込みと操舵。 Claude をいつでも割り込むことができます。間違った方向に進んでいる場合は、停止ボタンをクリックするか、修正を入力して **Enter** を押します。Claude は何をしているかを停止し、入力に基づいて調整します。完了するまで待つ必要も、やり直す必要もありません。

Claude により多くのコンテキストを提供する。 プロンプトボックスに `@filename` と入力して特定のファイルを会話に取り込み、添付ボタンを使用して画像と PDF を添付するか、ファイルをプロンプトに直接ドラッグアンドドロップします。Claude が持つコンテキストが多いほど、結果は良くなります。[ファイルとコンテキストを追加する](#)を参照してください。

繰り返し可能なタスクにスキルを使用する。 `/` を入力するか、`+ → Slash commands` をクリックして、[組み込みコマンド](#)、[カスタムスキル](#)、およびプラグインスキルを参照します。スキルは、コードレビューチェックリストやデプロイメント手順など、必要なときに呼び出せる再利用可能なプロンプトです。

コミット前に変更をレビューする。 Claude がファイルを編集した後、`+12 -1` インジケータが表示されます。それをクリックして [diff ビュー](#)を開き、ファイルごとに変更をレビューし、特定の行にコメントします。Claude はコメントを読んで修正します。

Review code をクリックして、Claude に diff を自分で評価させ、インライン提案を残します。

コントロール量を調整する。 [権限モード](#)は、バランスを制御します。Ask permissions (デフォルト) は、すべての編集の前に承認が必要です。Auto accept edits は、ファイル編集を自動的に受け入れて、より高速な反復を実現します。Plan mode では、Claude がファイルに触れずにアプローチをマップアウトできます。これは大規模なリファクタリング前に便利です。

プラグインを追加してさらに多くの機能を追加する。 プロンプトボックスの横の `+` ボタンをクリックして **Plugins** を選択し、スキル、エージェント、MCP サーバーなどを追加する [プラグイン](#)を参照してインストールします。

アプリをプレビューする。 **Preview** ドロップダウンをクリックして、dev サーバーをデスクトップで直接実行します。Claude は実行中のアプリを表示し、エンドポイントをテストし、ログを検査し、見たものに対して反復できます。[アプリをプレビューする](#)を参照してください。

プルリクエストを追跡する。 PR を開いた後、Claude Code は CI チェック結果を監視し、失敗を自動的に修正するか、すべてのチェックが成功したら PR をマージできます。[プルリクエストステータスを監視する](#)を参照してください。

Claude をスケジュールに配置する。 [スケジュール済みタスク](#)を設定して、Claude を定期的に自動実行します。毎朝のコードレビュー、週次の依存関係監査、または接続されたツールから情報を取得するブリーフィング。

準備ができたならスケールアップする。 サイドバーから[並列セッション](#)を開いて、複数のタスクに同時に取り組みます。各タスクは独自の Git worktree にあります。[長時間実行される作業をクラウドに送信](#)して、アプリを閉じてでも続行するか、タスクが予想より長くかかる場合は [web またはあなたの IDE でセッションを続行](#)します。[GitHub](#)、[Slack](#)、[Linear](#) などの外部ツールを接続して、ワークフローをまとめます。

CLI から来ましたか？

Desktop は、グラフィカルインターフェース付きの CLI と同じエンジンを実行します。同じプロジェクトで両方を同時に実行でき、設定（CLAUDE.md ファイル、MCP サーバー、hooks、スキル、設定）を共有します。機能、フラグの同等物、Desktop で利用できないものの完全な比較については、[CLI 比較](#)を参照してください。

次のステップ

- [Claude Code Desktop を使用する](#): 権限モード、並列セッション、diff ビュー、コネクター、エンタープライズ設定
- [トラブルシューティング](#): 一般的なエラーとセットアップの問題の解決策
- [ベストプラクティス](#): 効果的なプロンプトを書き、Claude Code を最大限に活用するためのヒント
- [一般的なワークフロー](#): デバッグ、リファクタリング、テストなどのチュートリアル

Claude Code をセットアップする

開発マシンに Claude Code をインストール、認証し、使用を開始します。

システム要件

- ・ **オペレーティングシステム:**
 - macOS 13.0 以上
 - Windows 10 1809 以上または Windows Server 2019 以上 ([セットアップノート](#)を参照)
 - Ubuntu 20.04 以上
 - Debian 10 以上
 - Alpine Linux 3.19 以上 ([追加の依存関係が必要](#))
- ・ **ハードウェア:** 4 GB 以上の RAM
- ・ **ネットワーク:** インターネット接続が必要です ([ネットワーク設定](#)を参照)
- ・ **シェル:** Bash または Zsh で最適に動作します
- ・ **場所:** [Anthropic がサポートしている国](#)

追加の依存関係

- ・ **ripgrep:** 通常は Claude Code に含まれています。検索が失敗する場合は、[検索のトラブルシューティング](#)を参照してください。
- ・ **Node.js 18 以上:** [非推奨の npm インストール](#)にのみ必要です

インストール

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.

インストールプロセスが完了した後、プロジェクトに移動して Claude Code を開始します:

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

インストール中に問題が発生した場合は、[トラブルシューティングガイド](#)を参照してください。

Tip:

インストール後に `claude doctor` を実行して、インストールタイプとバージョンを確認します。

プラットフォーム固有のセットアップ

Windows: Claude Code をネイティブで実行するか ([Git Bash](#)が必要)、WSL 内で実行します。WSL 1 と WSL 2 の両方がサポートされていますが、WSL 1 はサポートが限定的であり、Bash ツールサンドボックスなどの機能をサポートしていません。

Alpine Linux およびその他の musl/uClibc ベースのディストリビューション:

Alpine およびその他の musl/uClibc ベースのディストリビューション上のネイティブインストーラーには、`libgcc`、`libstdc++`、および `ripgrep` が必要です。ディストリビューションのパッケージマネージャーを使用してこれらをインストールし、`USE_BUILTIN_RIPGREP=0` を設定します。

Alpine の場合:

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

認証

個人向け

1. **Claude Pro または Max プラン** (推奨) : Claude の [Pro または Max プラン](#) にサブスクリライブして、Claude Code と Web 上の Claude の両方を含む統一されたサブスクリプションを取得します。1 つの場所でアカウントを管理し、Claude.ai アカウントでログインします。
2. **Claude Console:** [Claude Console](#)を通じて接続し、OAuth プロセスを完了します。Anthropic Console でのアクティブな課金が必要です。「Claude Code」ワークスペースは使用状況の追跡とコスト管理のために自動的に作成されます。Claude Code ワークスペース用の API キーを作成することはできません。これは Claude Code の使用専用です。

チームおよび組織向け

1. **Claude for Teams または Enterprise (推奨)** : [Claude for Teams](#) または [Claude for Enterprise](#) にサブスクライブして、一元化された課金、チーム管理、および Claude Code と Web 上の Claude の両方へのアクセスを取得します。チームメンバーは Claude.ai アカウントでログインします。
2. **チーム課金を使用した Claude Console**: チーム課金を使用して共有 [Claude Console](#) 組織をセットアップします。チームメンバーを招待し、使用状況の追跡のためにロールを割り当てます。
3. **クラウドプロバイダー**: [Amazon Bedrock](#)、[Google Vertex AI](#)、または [Microsoft Foundry](#) を使用するように Claude Code を設定して、既存のクラウドインフラストラクチャでのデプロイメントを行います。

特定のバージョンをインストール

ネイティブインストーラーは、特定のバージョン番号またはリリースチャネル（`latest` または `stable`）のいずれかを受け入れます。インストール時に選択したチャネルが自動更新のデフォルトになります。詳細については、[リリースチャネルを設定](#)を参照してください。

最新バージョンをインストールするには（デフォルト）：

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
```

安定版をインストールするには:

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
```

特定のバージョン番号をインストールするには:

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
```

バイナリの整合性とコード署名

- すべてのプラットフォームの SHA256 チェックサムはリリースマニフェストで公開されており、現在 <https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json> に位置しています (例: {VERSION} を 2.0.30 に置き換えます)
- 署名されたバイナリは以下のプラットフォーム向けに配布されています:
 - macOS: 'Anthropic PBC' によって署名され、Apple によって公証されています
 - Windows: 'Anthropic, PBC' によって署名されています

NPM インストール (非推奨)

NPM インストールは非推奨です。可能な場合は[ネイティブインストール](#)方法を使用してください。既存の npm インストールをネイティブに移行するには、`claude install` を実行します。

グローバル npm インストール

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

Warning:

`sudo npm install -g` を使用しないでください。これは権限の問題とセキュリティリスクにつながる可能性があります。権限エラーが発生した場合は、[権限エラーのトラブルシューティング](#)を参照して、推奨されるソリューションを確認してください。

Windows セットアップ

オプション 1: WSL 内の Claude Code

- WSL 1 と WSL 2 の両方がサポートされています
- WSL 2 は強化されたセキュリティのための[サンドボックス](#)をサポートしています。WSL 1 はサンドボックスをサポートしていません。

オプション 2: Git Bash を使用したネイティブ Windows 上の Claude Code

- [Git for Windows](#)が必要です
- ポータブル Git インストールの場合、`bash.exe` へのパスを指定します:

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

Claude Code を更新

自動更新

Claude Code は最新の機能とセキュリティ修正を確保するために、自動的に最新の状態に保たれます。

- **更新チェック:** スタートアップ時および実行中に定期的に行われます
- **更新プロセス:** バックグラウンドで自動的にダウンロードおよびインストールされます

- **通知:** 更新がインストールされたときに通知が表示されます
- **更新の適用:** 更新は Claude Code を次回起動したときに有効になります

Note:

Homebrew および WinGet インストールは自動更新されません。 `brew upgrade claude-code` または `winget upgrade Anthropic.ClaudeCode` を使用して手動で更新します。

既知の問題: Claude Code は、新しいバージョンがこれらのパッケージマネージャーで利用可能になる前に、更新を通知する場合があります。アップグレードが失敗した場合は、待機してから後で再度試してください。

リリースチャネルを設定

`autoUpdatesChannel` 設定を使用して、自動更新と `claude update` の両方に対して Claude Code が従うリリースチャネルを設定します:

- `"latest"` (デフォルト): リリースされるとすぐに新機能を受け取ります
- `"stable"`: 通常約 1 週間前のバージョンを使用し、大きな回帰を伴うリリースをスキップします

これを `/config` → **自動更新チャネル** 経由で設定するか、[settings.json ファイル](#)に追加します:

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

エンタープライズデプロイメントの場合、[管理設定](#)を使用して、組織全体で一貫したリリースチャネルを強制できます。

自動更新を無効化

シェルまたは [settings.json ファイル](#) で `DISABLE_AUTOUPDATER` 環境変数を設定します:

```
export DISABLE_AUTOUPDATER=1
```

手動で更新

```
claude update
```

Claude Code をアンインストール

Claude Code をアンインストールする必要がある場合は、インストール方法の指示に従ってください。

ネイティブインストール

Claude Code バイナリとバージョンファイルを削除します:

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 インストール

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

WinGet インストール

```
winget uninstall Anthropic.ClaudeCode
```

NPM インストール

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


設定ファイルをクリーンアップ（オプション）

Warning:

設定ファイルを削除すると、すべての設定、許可されたツール、MCP サーバー設定、およびセッション履歴が削除されます。

Claude Code の設定とキャッシュされたデータを削除するには:

macOS、Linux、WSL:

```
## ユーザー設定と状態を削除
rm -rf ~/.claude
rm ~/.claude.json

## プロジェクト固有の設定を削除（プロジェクトディレクトリから実行）
rm -rf .claude
rm -f .mcp.json
```

Windows PowerShell:

```
## ユーザー設定と状態を削除
Remove-Item -Path "$env:USERPROFILE\.claude" -Recurse -Force
Remove-Item -Path "$env:USERPROFILE\.claude.json" -Force

## プロジェクト固有の設定を削除（プロジェクトディレクトリから実行）
Remove-Item -Path ".claude" -Recurse -Force
Remove-Item -Path ".mcp.json" -Force
```

Windows CMD:

REM ユーザー設定と状態を削除

```
rmmdir /s /q "%USERPROFILE%\.claude"  
del "%USERPROFILE%\.claude.json"
```

REM プロジェクト固有の設定を削除（プロジェクトディレクトリから実行）

```
rmmdir /s /q ".claude"  
del ".mcp.json"
```

Claude Code の仕組み

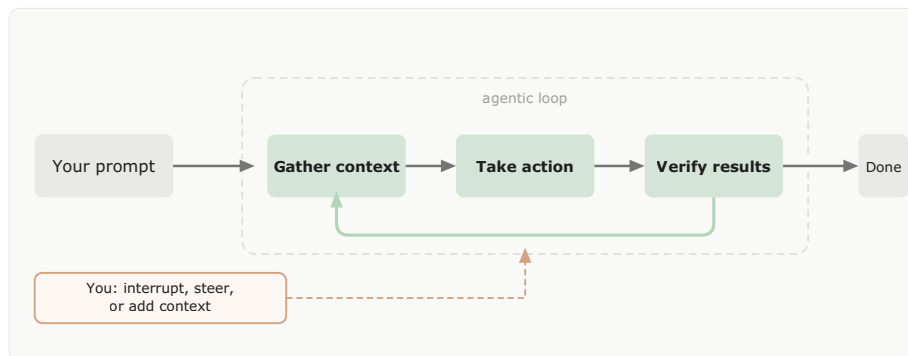
agentic ループ、組み込みツール、Claude Code がプロジェクトとどのように相互作用するかを理解します。

Claude Code はターミナルで実行される agentic アシスタントです。コーディングに優れていますが、コマンドラインからできることなら何でも支援できます。ドキュメント作成、ビルド実行、ファイル検索、トピック調査など、様々なタスクに対応します。

このガイドでは、コアアーキテクチャ、組み込み機能、および [Claude Code を効果的に使用するためのヒント](#) について説明します。ステップバイステップのウォークスルーについては、[一般的なワークフロー](#) を参照してください。スキル、MCP、フックなどの拡張機能については、[Claude Code を拡張する](#) を参照してください。

agentic ループ

Claude にタスクを与えると、3つのフェーズを通じて処理します。**コンテキストの収集、アクション実行、結果の検証** です。これらのフェーズは相互に融合します。Claude はツールを使用して、コードを理解するためのファイル検索、変更を加えるための編集、作業を確認するためのテスト実行など、様々な場面で活用します。



agentic ループ：プロンプトから Claude がコンテキストを収集し、アクションを実行し、結果を検証し、タスク完了まで繰り返します。任意の時点で中断できます。

ループは質問内容に応じて適応します。コードベースに関する質問はコンテキスト収集のみで済む場合もあります。バグ修正は3つのフェーズすべてを何度も繰り返します。リファクタリングは広範な検証を伴う場合があります。Claude は前のステップから学んだことに基づいて各ステップに必要なものを判断し、数十のアクションを連鎖させながら進路を修正していきます。

あなたもこのループの一部です。任意の時点で中断して Claude を別の方向に導いたり、追加のコンテキストを提供したり、別のアプローチを試すよう指示できます。Claude は自律的に動作しますが、入力に対して応答性を保ちます。

agentic ループは2つのコンポーネントで動作します。推論を行う [モデル](#) と、アクションを実行する [ツール](#) です。Claude Code は Claude の周りの **agentic ハーネス** として機能します。言語モデルを有能なコーディングエージェントに変えるツール、コンテキスト管理、実行環境を提供します。

モデル

Claude Code は Claude モデルを使用してコードを理解し、タスクについて推論します。Claude はあらゆる言語のコードを読み、コンポーネントがどのように接続されているかを理解し、目標を達成するために何を変更する必要があるかを判断できます。複雑なタスクの場合、作業をステップに分割し、実行し、学んだことに基づいて調整します。

[複数のモデル](#) が異なるトレードオフで利用可能です。Sonnet はほとんどのコーディングタスクをうまく処理します。Opus は複雑なアーキテクチャ上の決定に対してより強力な推論を提供します。セッション中に `/model` で切り替えるか、`claude --model <name>` で開始します。

このガイドで「Claude が選択する」または「Claude が決定する」と言う場合、モデルが推論を行っています。

ツール

ツールが Claude Code を agentic にします。ツールがなければ、Claude はテキストで応答することしかできません。ツールがあれば、Claude はアクションを実行できます。コードを読み、ファイルを編集し、コマンドを実行し、ウェブを検索し、外部サービスと相互作用します。各ツール使用は情報を返し、ループにフィードバックされ、Claude の次の決定に情報を与えます。

組み込みツールは一般的に5つのカテゴリに分類され、それぞれが異なる種類のエージェンシーを表します。

カテゴリ	Claude ができること
ファイル操作	ファイルの読み取り、コード編集、新規ファイル作成、名前変更と再編成
検索	パターンでファイルを検索、正規表現でコンテンツを検索、コードベースを探索
実行	シェルコマンド実行、サーバー起動、テスト実行、git 使用
ウェブ	ウェブ検索、ドキュメント取得、エラーメッセージ検索
コードインテリジェンス	編集後の型エラーと警告表示、定義へのジャンプ、参照の検索（ コードインテリジェンスプラグイン が必要）

これらが主な機能です。Claude には subagent の生成、質問、その他のオーケストレーションタスク用のツールもあります。完全なリストについては、[Claude が利用可能なツール](#)を参照してください。

Claude はプロンプトと学んだことに基づいて、どのツールを使用するかを選択します。「失敗しているテストを修正して」と言うと、Claude は以下のように対応する場合があります。

1. テストスイートを実行して何が失敗しているかを確認
2. エラー出力を読み取る
3. 関連するソースファイルを検索
4. それらのファイルを読んでコードを理解
5. ファイルを編集して問題を修正
6. テストを再度実行して検証

各ツール使用は Claude に新しい情報を与え、次のステップに情報を与えます。これが agentic ループの実際の動作です。

基本機能の拡張： 組み込みツールが基盤です。[スキル](#) で Claude の知識を拡張し、[MCP](#) で外部サービスに接続し、[フック](#) でワークフローを自動化し、[subagent](#) にタスクをオフロードできます。これらの拡張はコア agentic ループの上に層を形成します。ニーズに合った拡張を選択するためのガイダンスについては、[Claude Code を拡張する](#)を参照してください。

Claude がアクセスできるもの

このガイドはターミナルに焦点を当てています。Claude Code は [VS Code](#)、[JetBrains IDE](#)、その他の環境でも実行されます。

ディレクトリで **claude** を実行すると、Claude Code は以下にアクセスできます。

- **プロジェクト**。ディレクトリとサブディレクトリ内のファイル、および許可を得た他の場所のファイル。
- **ターミナル**。実行できるあらゆるコマンド。ビルドツール、git、パッケージマネージャ、システムユーティリティ、スクリプト。コマンドラインからできることなら、Claude もできます。
- **git の状態**。現在のブランチ、コミットされていない変更、最近のコミット履歴。
- **CLAUDE.md**。プロジェクト固有の指示、規約、Claude が毎回のセッションで知っておくべきコンテキストを保存するマークダウンファイル。
- **自動メモリ**。Claude が作業中に自動的に保存する学習内容。プロジェクトパターンと設定など。MEMORY.md の最初の 200 行は各セッションの開始時に読み込まれます。
- **設定した拡張機能**。外部サービス用の [MCP サーバー](#)、ワークフロー用の [スキル](#)、委譲作業用の [subagent](#)、ブラウザ相互作用用の [Claude in Chrome](#)。

Claude はプロジェクト全体を見ることができるため、プロジェクト全体で作業できます。「認証バグを修正して」と Claude に指示すると、関連ファイルを検索し、複数のファイルを読んでコンテキストを理解し、それらを調整して編集し、修正を検証するためにテストを実行し、要求に応じて変更をコミットします。これは現在のファイルのみを見るインラインコードアシスタントとは異なります。

環境とインターフェース

上記で説明した agentic ループ、ツール、機能は、Claude Code を使用するあらゆる場所で同じです。変わるのは、コードが実行される場所とそれとの相互作用方法です。

実行環境

Claude Code は 3 つの環境で実行され、各環境はコード実行場所に対して異なるトレードオフを持ちます。

環境	コード実行場所	ユースケース
ローカル	マシン	デフォルト。ファイル、ツール、環境への完全なアクセス
クラウド	Anthropic 管理 VM	タスクをオフロード、ローカルにないリポジトリで作業

環境	コード実行場所	ユースケース
リモートコントロール	マシン、ブラウザから制御	ウェブ UI を使用しながらすべてをローカルに保つ

インターフェース

Claude Code には、ターミナル、[デスクトップアプリ](#)、[IDE 拡張機能](#)、[claude.ai/code](#)、[リモートコントロール](#)、[Slack](#)、[CI/CD パイプライン](#) を通じてアクセスできます。インターフェースは Claude の表示方法と相互作用方法を決定しますが、基盤となる agentic ループは同じです。完全なリストについては、[Claude Code をどこでも使用する](#) を参照してください。

セッションで作業する

Claude Code は作業中にローカルで会話を保存します。各メッセージ、ツール使用、結果が保存され、[巻き戻し](#)、[再開](#)、[フォーク](#) セッションが可能になります。Claude がコード変更を加える前に、影響を受けるファイルのスナップショットも作成されるため、必要に応じて元に戻すことができます。

セッションは独立しています。 各新規セッションは新しいコンテキストウィンドウで開始され、前のセッションの会話履歴がありません。Claude は [自動メモリ](#) を使用してセッション間で学習を保持でき、[CLAUDE.md](#) に独自の永続的な指示を追加できます。

ブランチ間で作業する

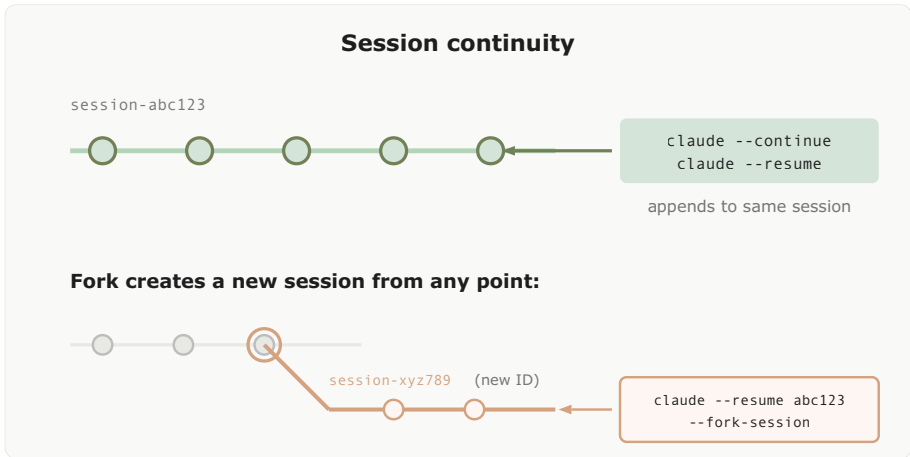
各 Claude Code 会話はセッションで、現在のディレクトリに関連付けられています。再開すると、そのディレクトリからのセッションのみが表示されます。

Claude は現在のブランチのファイルを見ます。ブランチを切り替えると、Claude は新しいブランチのファイルを見ますが、会話履歴は同じままです。Claude はブランチ切り替え後も、議論した内容を覚えています。

セッションはディレクトリに関連付けられているため、[git worktree](#) を使用して並列 Claude Code セッションを実行できます。これは個別のブランチ用に別のディレクトリを作成します。

セッションを再開またはフォークする

`claude --continue` または `claude --resume` でセッションを再開すると、同じセッション ID を使用して中断したところから再開します。新しいメッセージは既存の会話に追加されます。完全な会話履歴が復元されますが、セッションスコープの権限は復元されません。それらを再度承認する必要があります。



セッション継続性：再開は同じセッションを続行し、フォークは新しい ID で新しいブランチを作成します。

元のセッションに影響を与えずに別のアプローチを試すために分岐するには、`--fork-session` フラグを使用します。

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

これは会話履歴をその時点まで保持しながら新しいセッション ID を作成します。元のセッションは変更されません。再開と同様に、フォークされたセッションはセッションスコープの権限を継承しません。

複数のターミナルで同じセッション：複数のターミナルで同じセッションを再開すると、両方のターミナルが同じセッションファイルに書き込みます。両方からのメッセージがインターリーブされます。同じノートブックに 2 人が書き込むようなものです。何も破損しませんが、会話がごちゃごちゃになります。セッション中は各ターミナルは独自のメッセージのみを見ますが、後でそのセッションを再開すると、すべてがインターリーブされた状態で表示されます。同じ開始点から並列作業する場合は、`--fork-session` を使用して各ターミナルに独自のクリーンなセッションを与えます。

コンテキストウィンドウ

Claude のコンテキストウィンドウは、会話履歴、ファイルコンテンツ、コマンド出力、[CLAUDE.md](#)、読み込まれたスキル、システム指示を保持します。作業を進めるとコンテキストが満杯になります。Claude は自動的にコンパクト化しますが、会話の早い段階の指示が失われる可能性があります。永続的なルールは [CLAUDE.md](#) に入れ、`/context` を実行してスペースを使用しているものを確認します。

コンテキストが満杯になったとき

Claude Code はコンテキストウィンドウの制限に近づくとき、自動的にコンテキストを管理します。古いツール出力をクリアし、必要に応じて会話を要約します。リクエストと主要なコードスニペットは保持されます。会話の早い段階の詳細な指示は失われる可能性があります。会話履歴に依存するのではなく、永続的なルールを CLAUDE.md に入れます。

コンパクト化中に保持されるものを制御するには、CLAUDE.md に「Compact Instructions」セクションを追加するか、`/compact` をフォーカス付きで実行します（例：`/compact focus on the API changes`）。

`/context` を実行してスペースを使用しているものを確認します。MCP サーバーはすべてのリクエストにツール定義を追加するため、数個のサーバーで作業を開始する前に大量のコンテキストを消費できます。`/mcp` を実行してサーバーごとのコストを確認します。

スキルと subagent でコンテキストを管理する

コンパクト化を超えて、他の機能を使用してコンテキストに読み込まれるものを制御できます。

スキル はオンデマンドで読み込まれます。Claude はセッション開始時にスキル説明を見ますが、完全なコンテンツはスキルが使用されるときのみ読み込まれます。手動で呼び出すスキルの場合、`disable-model-invocation: true` を設定して、必要になるまで説明をコンテキストから除外します。

Subagent は独自の新しいコンテキストを取得し、メイン会話から完全に分離されます。それらの作業はコンテキストを膨張させません。完了すると、要約を返します。この分離が長いセッションで subagent が役立つ理由です。

コンテキストコスト で各機能のコストを確認し、**トークン使用量を削減** でコンテキスト管理のヒントを確認します。

チェックポイントと権限で安全に保つ

Claude には 2 つの安全メカニズムがあります。チェックポイントはファイル変更を元に戻すことができ、権限は Claude が質問なしでできることを制御します。

チェックポイントで変更を元に戻す

すべてのファイル編集は可逆的です。 Claude がファイルを編集する前に、現在のコンテンツのスナップショットを作成します。何か問題が発生した場合、`Esc` を 2 回押して前の状態に巻き戻すか、Claude に元に戻すよう指示します。

チェックポイントはセッションに対してローカルで、git とは別です。ファイル変更のみをカバーします。リモートシステムに影響を与えるアクション（データベース、API、デプロイメント）はチェックポイントできません。これが Claude が外部の副作用を持つコマンドを実行する前に質問する理由です。

Claude ができることを制御する

Shift+Tab を押して権限モードをサイクルします。

- **デフォルト**：Claude はファイル編集とシェルコマンドの前に質問します
- **自動受け入れ編集**：Claude はファイルを質問なしで編集し、コマンドについては質問します
- **プランモード**：Claude は読み取り専用ツールのみを使用し、実行前に承認できるプランを作成します

`.claude/settings.json` で特定のコマンドを許可することもできます。これにより、Claude は毎回質問しません。 `npm test` や `git status` などの信頼できるコマンドに便利です。設定は組織全体のポリシーから個人的な設定までスコープできます。詳細については、[権限](#) を参照してください。

Claude Code を効果的に使用する

これらのヒントは Claude Code からより良い結果を得るのに役立ちます。

Claude Code に助けを求める

Claude Code は使用方法を教えることができます。「フックの設定方法は？」や「CLAUDE.md を構造化する最良の方法は？」などの質問をすると、Claude が説明します。

組み込みコマンドもセットアップをガイドします。

- `/init` はプロジェクト用の CLAUDE.md 作成をウォークスルーします
- `/agents` はカスタム subagent の設定を支援します
- `/doctor` はインストールの一般的な問題を診断します

会話です

Claude Code は会話型です。完璧なプロンプトは必要ありません。やりたいことから始めて、その後改善します。

ログインバグを修正して

[Claude が調査し、何かを試す]

それは正確ではありません。問題はセッション処理にあります。

[Claude がアプローチを調整]

最初の試みが正しくない場合、最初からやり直す必要はありません。反復します。

中断して操舵する

任意の時点で Claude を中断できます。間違った方向に進んでいる場合は、修正を入力して Enter キーを押すだけです。Claude は何をしていたかを停止し、入力に基づいてアプローチを調整します。完了を待つ必要も、最初からやり直す必要もありません。

最初から具体的に

最初のプロンプトが正確であるほど、必要な修正が少なくなります。特定のファイルを参照し、制約を述べ、例のパターンを指摘します。

チェックアウトフローが期限切れカードのユーザーに対して機能していません。
src/payments/ で問題を確認してください。特にトークン更新。
最初に失敗するテストを書いて、その後修正してください。

曖昧なプロンプトは機能しますが、より多くの時間を操舵に費やします。上記のような具体的なプロンプトは最初の試みで成功することが多いです。

Claude が検証するものを与える

Claude は独自の作業を確認できるときにより良いパフォーマンスを発揮します。テストケース、期待される UI のスクリーンショット、または必要な出力を定義します。

validateEmail を実装します。テストケース: 'user@example.com' → true、
'invalid' → false、'user@.com' → false。その後テストを実行します。

ビジュアル作業の場合、デザインのスクリーンショットを貼り付けて、Claude に実装と比較するよう指示します。

実装する前に探索する

複雑な問題の場合、研究とコーディングを分離します。プランモード（**Shift+Tab** を 2 回）を使用してコードベースを最初に分析します。

src/auth/ を読んで、セッション処理方法を理解してください。
その後、OAuth サポート追加のプランを作成してください。

プランを確認し、会話を通じて改善し、その後 Claude に実装させます。このフェーズアプローチは、コードに直接ジャンプするよりも良い結果を生成します。

指示するのではなく委譲する

有能な同僚に委譲することを考えてください。コンテキストと方向を与え、その後 Claude が詳細を理解することを信頼します。

チェックアウトフローが期限切れカードのユーザーに対して機能していません。
関連するコードは src/payments/ にあります。調査して修正できますか？

どのファイルを読むか、どのコマンドを実行するかを指定する必要はありません。Claude がそれを理解します。

次のステップ

- ・ [機能で拡張する](#) スキル、MCP 接続、カスタムコマンドを追加
- ・ [一般的なワークフロー](#) 一般的なタスクのステップバイステップガイド

Part 2: Core Usage

Claude Code を拡張する

CLAUDE.md、Skills、subagents、hooks、MCP、plugins をいつ使用するかを理解します。

Claude Code は、コードについて推論するモデルと、ファイル操作、検索、実行、ウェブアクセス用の[組み込みツール](#)を組み合わせています。組み込みツールはほとんどのコーディングタスクをカバーしています。このガイドは拡張レイヤーについて説明します。つまり、Claude が何を知るかをカスタマイズし、外部サービスに接続し、ワークフローを自動化するために追加する機能です。

Note:

コア agentic ループがどのように機能するかについては、[How Claude Code works](#) を参照してください。

Claude Code は初めてですか？ [CLAUDE.md](#) でプロジェクト規約を開始します。必要に応じて他の拡張機能を追加してください。

概要

拡張機能は agentic ループのさまざまな部分に接続します。

- [CLAUDE.md](#) は、Claude がすべてのセッションで見る永続的なコンテキストを追加します
- [Skills](#) は再利用可能な知識と呼び出し可能なワークフローを追加します
- [MCP](#) は Claude を外部サービスとツールに接続します
- [Subagents](#) は独立したコンテキストで独自のループを実行し、サマリーを返します
- [Agent teams](#) は、共有タスクとピアツーピアメッセージングを使用して複数の独立したセッションを調整します
- [Hooks](#) はループの外側で決定論的スクリプトとして実行されます
- [Plugins](#) と [marketplaces](#) はこれらの機能をパッケージ化して配布します

Skills は最も柔軟な拡張機能です。スキルは知識、ワークフロー、または指示を含むマークダウンファイルです。`/deploy` のようなコマンドでスキルを呼び出すことができます。または Claude は関連する場合に自動的にスキルをロードできます。スキルは現在の会話で実行することも、subagents を介して独立したコンテキストで実行することもできます。

機能をあなたの目標に合わせる

機能は、Claude がすべてのセッションで見る常時オンのコンテキストから、あなたまたは Claude が呼び出すことができるオンデマンド機能、特定のイベントで実行される背景自動化まで、さまざまです。以下の表は、利用可能な機能と各機能が適切な場合を示しています。

機能	機能	使用する場合	例
CLAUDE.md	すべての会話で読み込まれる永続的なコンテキスト	プロジェクト規約、 「常に X を実行する」ルール	「pnpm を使用し、npm は使用しない。コミット前にテストを実行する。」
Skill	Claude が使用できる指示、知識、ワークフロー	再利用可能なコンテンツ、リファレンスドキュメント、繰り返しタスク	<code>/deploy</code> はデプロイメントチェックリストを実行します。エンドポイントパターンを持つ API ドキュメントスキル
Subagent	サマリー結果を返す独立した実行コンテキスト	コンテキスト分離、並列タスク、専門的なワーカー	多くのファイルを読み取るが、主要な結果のみを返す研究タスク
Agent teams	複数の独立した Claude Code セッションを調整	並列研究、新機能開発、競合する仮説でのデバッグ	セキュリティ、パフォーマンス、テストを同時にチェックするレビューアースポーン
MCP	外部サービスに接続	外部データまたはアクション	データベースをクエリ、Slack に投稿、ブラウザを制御
Hook	イベントで実行される決定論的スクリプト	予測可能な自動化、LLM は関与しない	すべてのファイル編集後に ESLint を実行

Plugins はパッケージングレイヤーです。プラグインはスキル、フック、subagents、MCP サーバーを単一のインストール可能なユニットにバンドルします。プラグインスキルは名前空間化されています（`/my-plugin:review` のように）ため、複数のプラグインが共存できます。同じセットアップを複数のリポジトリで再利用したい場合、または **marketplace** を通じて他のユーザーに配布したい場合はプラグインを使用します。

類似機能を比較する

一部の機能は似ているように見ることがあります。ここでは、それらを区別する方法を説明します。

Skill vs Subagent

スキルと subagents は異なる問題を解決します。

- **Skills** は任意のコンテキストにロードできる再利用可能なコンテンツです
- **Subagents** はメイン会話とは別に実行される独立したワーカーです

側面	Skill	Subagent
それは何か	再利用可能な指示、知識、またはワークフロー	独自のコンテキストを持つ独立したワーカー
主な利点	コンテキスト全体でコンテンツを共有	コンテキスト分離。作業は別々に行われ、サマリーのみが返される
最適な用途	リファレンスマテリアル、呼び出し可能なワークフロー	多くのファイルを読み取るタスク、並列作業、専門的なワーカー

スキルはリファレンスまたはアクションです。 リファレンススキルは Claude がセッション全体で使用する知識を提供します（API スタイルガイドなど）。アクションスキルは Claude に特定の操作を実行するよう指示します（デプロイメントワークフローを実行する `/deploy` など）。

コンテキスト分離が必要な場合、またはコンテキストウィンドウがいっぱいになっている場合は subagent を使用します。 subagent は数十のファイルを読み取るか、広範な検索を実行する可能性があります、メイン会話はサマリーのみを受け取ります。subagent の作業はメインコンテキストを消費しないため、中間作業を表示したままにする必要がない場合にも便利です。カスタム subagents は独自の指示を持つことができ、スキルをプリロードできます。

それらは組み合わせることができます。subagent は特定のスキルをプリロードできます（`skills:` フィールド）。スキルは `context: fork` を使用して独立したコンテキストで実行できます。詳細は [Skills](#) を参照してください。

CLAUDE.md vs Skill

どちらも指示を保存しますが、ロード方法と目的が異なります。

側面	CLAUDE.md	Skill
ロード	すべてのセッション、自動的に	オンデマンド
ファイルを含めることができます	はい、 <code>@path</code> インポート付き	はい、 <code>@path</code> インポート付き
ワークフローをトリガーできます	いいえ	はい、 <code>/<name></code> 付き
最適な用途	「常に X を実行する」ルール	リファレンスマテリアル、呼び出し可能なワークフロー

Claude が常に知っておくべき場合は CLAUDE.md に入れます。コーディング規約、ビルドコマンド、プロジェクト構造、「X を実行しない」ルール。

Claude が時々必要とするリファレンスマテリアル（API ドキュメント、スタイルガイド）の場合、またはあなたが `/<name>` でトリガーするワークフロー（デプロイ、レビュー、リリース）の場合は、スキルに入れます。

経験則：CLAUDE.md を 200 行以下に保ちます。増加している場合は、リファレンスコンテンツをスキルに移動するか、`.claude/rules/` ファイルに分割します。

CLAUDE.md vs Rules vs Skills

3 つすべてが指示を保存しますが、ロード方法が異なります。

側面	CLAUDE.md	<code>.claude/rules/</code>	Skill
ロード	すべてのセッション	すべてのセッション、またはマッチングファイルが開かれたとき	オンデマンド、呼び出されたときまたは関連するとき
スコープ	プロジェクト全体	ファイルパスにスコープできます	タスク固有

側面	CLAUDE.md	.claude/rules/	Skill
最適な用途	コア規約とビルドコマンド	言語固有またはディレクトリ固有のガイドライン	リファレンスマテリアル、繰り返しワークフロー

すべてのセッションが必要とする指示には **CLAUDE.md** を使用します。ビルドコマンド、テスト規約、プロジェクトアーキテクチャ。

CLAUDE.md を焦点を当てたままにするために **ルール** を使用します。 [paths](#) [frontmatter](#) を持つルールは、Claude がマッチングファイルで作業するときのみロードされ、コンテキストを節約します。

Claude が時々必要とするコンテンツ（API ドキュメントまたは `</name>` でトリガーするデプロイメントチェックリスト）には **スキル** を使用します。

Subagent vs Agent team

どちらも作業を並列化しますが、アーキテクチャ的には異なります。

- **Subagents** はセッション内で実行され、結果をメインコンテキストに報告します
- **Agent teams** は独立した Claude Code セッションであり、互いに通信します

側面	Subagent	Agent team
コンテキスト	独自のコンテキストウィンドウ。結果は呼び出し元に返される	独自のコンテキストウィンドウ。完全に独立
通信	結果をメインエージェントのみに報告	チームメイトが直接互いにメッセージを送信
調整	メインエージェントがすべての作業を管理	共有タスクリストと自己調整
最適な用途	結果のみが重要な焦点を絞ったタスク	議論と協力が必要な複雑な作業
トークンコスト	低い。結果がメインコンテキストに要約される	高い。各チームメイトは個別の Claude インスタンス

クイックで焦点を絞ったワーカーが必要な場合は subagent を使用します。 質問を研究する、主張を検証する、ファイルをレビューする。subagent は作業を実行し、サマリーを返します。メイン会話はクリーンなままです。

チームメイトが結果を共有し、互いに異議を唱え、独立して調整する必要がある場合は **agent team** を使用します。Agent teams は競合する仮説での研究、並列コードレビュー、各チームメイトが個別の部分に所有する新機能開発に最適です。

遷移ポイント： 並列 subagents を実行しているがコンテキスト制限に達している場合、または subagents が互いに通信する必要がある場合、agent teams は自然な次のステップです。

Note:

Agent teams は実験的であり、デフォルトで無効になっています。セットアップと現在の制限については [agent teams](#) を参照してください。

MCP vs Skill

MCP は Claude を外部サービスに接続します。スキルは Claude が知ることを拡張します。これらのサービスを効果的に使用方法を含みます。

側面	MCP	Skill
それは何か	外部サービスに接続するためのプロトコル	知識、ワークフロー、リファレンスマテリアル
提供	ツールとデータアクセス	知識、ワークフロー、リファレンスマテリアル
例	Slack 統合、データベースクエリ、ブラウザ制御	コードレビューチェックリスト、デプロイワークフロー、API スタイルガイド

これらは異なる問題を解決し、一緒に機能します。

MCP は Claude に外部システムと相互作用する能力を与えます。MCP がなければ、Claude はデータベースをクエリしたり、Slack に投稿したりできません。

スキル は Claude にこれらのツールを効果的に使用方法についての知識を与え、さらに `/<name>` でトリガーできるワークフローを提供します。スキルには、チームのデータベーススキーマとクエリパターン、または `/<name>` ワークフローとチームのメッセージフォーマットルールが含まれる場合があります。

例：MCP サーバーは Claude をデータベースに接続します。スキルは Claude にデータモデル、一般的なクエリパターン、さまざまなタスクに使用するテーブルを教えます。

機能がどのようにレイヤーするかを理解する

機能は複数のレベルで定義できます。ユーザー全体、プロジェクトごと、プラグイン経由、または管理ポリシーを通じて。また、CLAUDE.md ファイルをサブディレクトリにネストしたり、スキルをモノレポの特定のパッケージに配置したりすることもできます。同じ機能が複数のレベルに存在する場合、ここでそれらがどのようにレイヤーするかを示します。

- **CLAUDE.md ファイル** は加算的です。すべてのレベルがコンテンツを同時に Claude のコンテキストに提供します。作業ディレクトリ以上のファイルは起動時にロードされます。サブディレクトリは作業時にロードされます。指示が競合する場合、Claude は判断を使用してそれらを調整し、より具体的な指示が通常優先されます。
[CLAUDE.md ファイルがどのようにロードされるか](#) を参照してください。
- **スキルと subagents** は名前ですべてオーバーライドします。同じ名前が複数のレベルに存在する場合、優先度に基づいて 1 つの定義が勝ちます（スキルの場合は管理 > ユーザー > プロジェクト。subagents の場合は管理 > CLI フラグ > プロジェクト > ユーザー > プラグイン）。プラグインスキルは競合を避けるために [名前空間化](#) されています。[スキル検出](#) と [subagent スコープ](#) を参照してください。
- **MCP サーバー** は名前ですべてオーバーライドします。ローカル > プロジェクト > ユーザー。[MCP スコープ](#) を参照してください。
- **Hooks** はマージされます。すべての登録されたフックは、ソースに関係なく、マッチングイベントに対して発火します。[Hooks](#) を参照してください。

機能を組み合わせる

各拡張機能は異なる問題を解決します。CLAUDE.md は常時オンのコンテキストを処理し、スキルはオンデマンド知識とワークフローを処理し、MCP は外部接続を処理し、subagents は分離を処理し、フックは自動化を処理します。実際のセットアップはワークフローに基づいてそれらを組み合わせます。

たとえば、CLAUDE.md をプロジェクト規約に使用し、スキルをデプロイメントワークフローに使用し、MCP をデータベースに接続し、フックをすべての編集後にリントを実行するために使用する場合があります。各機能は最適な機能を処理します。

パターン	機能	例
Skill + MCP	MCP は接続を提供します。スキルは Claude にそれを効果的に使用する方法を教えます	MCP はデータベースに接続し、スキルはスキーマとクエリパターンを文書化します

パターン	機能	例
Skill + Subagent	スキルは並列作業のために subagents をスポンします	<code>/audit</code> スキルはセキュリティ、パフォーマンス、スタイル subagents を開始し、独立したコンテキストで作業します
CLAUDE.md + Skills	CLAUDE.md は常時オンのルールを保持します。スキルはオンデマンドでロードされるリファレンスマテリアルを保持します	CLAUDE.md は 'API 規約に従う' と言い、スキルは完全な API スタイルガイドを含みます
Hook + MCP	フックは MCP を通じて外部アクションをトリガーします	編集後フックは Claude が重要なファイルを変更するときに Slack 通知を送信します

コンテキストコストを理解する

追加する各機能は Claude のコンテキストの一部を消費します。多すぎるとコンテキストウィンドウがいっぱいになる可能性があります。Claude の効果を低下させるノイズを追加することもできます。スキルが正しくトリガーされない場合や、Claude が規約を失う場合があります。これらのトレードオフを理解することで、効果的なセットアップを構築するのに役立ちます。

機能別のコンテキストコスト

各機能には異なるロード戦略とコンテキストコストがあります。

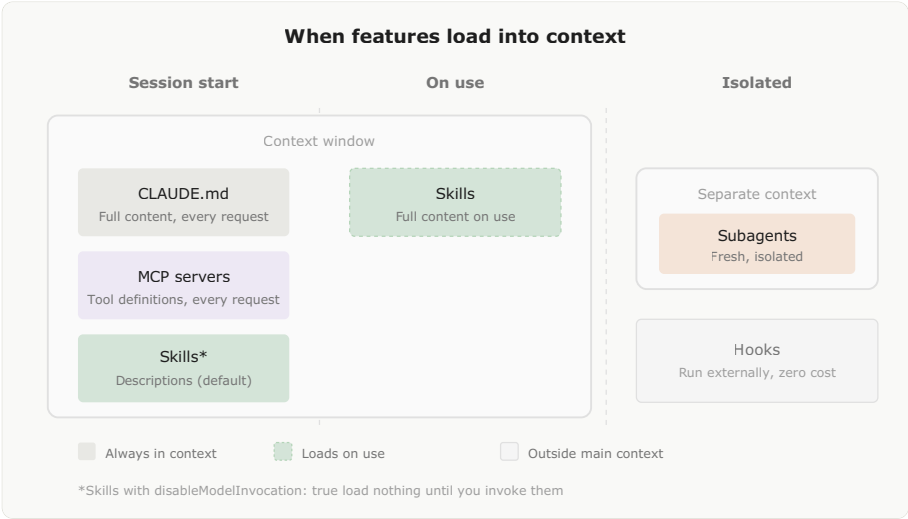
機能	ロード時期	ロード内容	コンテキストコスト
CLAUDE.md	セッション開始	完全なコンテンツ	すべてのリクエスト
Skills	セッション開始 + 使用時	開始時の説明、使用時の完全なコンテンツ	低い（毎リクエスト説明）*
MCP サーバー	セッション開始	すべてのツール定義と JSON スキーマ	すべてのリクエスト
Subagents	スポン時	指定されたスキルを持つ新しいコンテキスト	メインセッションから分離

機能	ロード時期	ロード内容	コンテキストコスト
Hooks	トリガー時	なし（外部で実行）	ゼロ、フックが追加 コンテキストを返さ ない限り

*デフォルトでは、スキル説明はセッション開始時にロードされるため、Claude はそれらを使用する時期を決定できます。スキルの frontmatter で `disable-model-invocation: true` を設定して、手動で呼び出すまで Claude から完全に非表示にします。これにより、自分でのみトリガーするスキルのコンテキストコストをゼロに削減します。

機能がどのようにロードされるかを理解する

各機能はセッション内の異なるポイントでロードされます。以下のタブは、各機能がいつロードされるか、およびコンテキストに何が入るかを説明しています。



コンテキストロード：CLAUDE.md と MCP はセッション開始時にロードされ、すべてのリクエストに留まります。スキルは開始時に説明をロードし、呼び出し時に完全なコンテンツをロードします。Subagents は独立したコンテキストを取得します。Hooks は外部で実行されます。

CLAUDE.md

時期：セッション開始

ロード内容：すべての CLAUDE.md ファイル（管理、ユーザー、プロジェクトレベル）の完全なコンテンツ。

継承： Claude は作業ディレクトリからルートまで CLAUDE.md ファイルを読み取り、サブディレクトリにネストされたものを、それらのファイルにアクセスするときに検出します。詳細は [How CLAUDE.md files load](#) を参照してください。

Tip:

CLAUDE.md を約 500 行以下に保ちます。リファレンスマテリアルをスキルに移動します。スキルはオンデマンドでロードされます。

Skills

スキルは Claude のツールキットの追加機能です。リファレンスマテリアル（API スタイルガイドなど）または `/<name>` でトリガーする呼び出し可能なワークフロー（`/deploy` など）です。Claude Code は `/simplify`、`/batch`、`/debug` などの [バンドルされたスキル](#) を備えており、すぐに機能します。独自のスキルを作成することもできます。Claude は適切な場合にスキルを使用するか、直接呼び出すことができます。

時期： スキルの設定によって異なります。デフォルトでは、説明はセッション開始時にロードされ、完全なコンテンツは使用時にロードされます。ユーザーのみのスキル（`disable-model-invocation: true`）の場合、呼び出すまで何もロードされません。

ロード内容： モデル呼び出し可能なスキルの場合、Claude はすべてのリクエストで名前と説明を見ます。`/<name>` でスキルを呼び出すか、Claude が自動的にロードする場合、完全なコンテンツが会話にロードされます。

Claude がスキルを選択する方法： Claude はタスクをスキル説明と照合して、関連するものを決定します。説明が曖昧または重複している場合、Claude は間違ったスキルをロードするか、役立つスキルを見落とす可能性があります。Claude に特定のスキルを使用するよう指示するには、`/<name>` で呼び出します。`disable-model-invocation: true` を持つスキルは、呼び出すまで Claude に見えません。

コンテキストコスト： 使用されるまで低い。ユーザーのみのスキルは呼び出されるまでゼロコストです。

Subagents 内： スキルは subagents で異なる動作をします。オンデマンドロードの代わりに、subagent に渡されるスキルは起動時にそのコンテキストに完全にプリロードされます。Subagents はメインセッションからスキルを継承しません。明示的に指定する必要があります。

Tip:

副作用を持つスキルには `disable-model-invocation: true` を使用します。これはコンテキストを節約し、あなたのみがそれらをトリガーすることを保証します。

MCP servers

時期： セッション開始。

ロード内容： 接続されたサーバーからのすべてのツール定義と JSON スキーマ。

コンテキストコスト： [Tool search](#) (デフォルトで有効) は MCP ツールをコンテキストの最大 10% までロードし、残りは必要になるまで遅延します。

信頼性に関する注記： MCP 接続はセッション中に静かに失敗する可能性があります。サーバーが切断されると、そのツールは警告なく消えます。Claude は以前アクセスできたツールを使用しようとする可能性があります。Claude が以前アクセスできた MCP ツールを使用できなくなったことに気付いた場合は、`/mcp` で接続を確認してください。

Tip:

`/mcp` を実行してサーバーごとのトークンコストを確認します。積極的に使用していないサーバーを切断します。

Subagents

時期： オンデマンド、タスクのために あなたまたは Claude がスポンするとき。

ロード内容： 以下を含む新しい独立したコンテキスト。

- ・ システムプロンプト (キャッシュ効率のため親と共有)
- ・ エージェントの `skills:` フィールドにリストされているスキルの完全なコンテンツ
- ・ CLAUDE.md と git ステータス (親から継承)
- ・ リードエージェントがプロンプトで渡すコンテキスト

コンテキストコスト： メインセッションから分離。Subagents は会話履歴または呼び出されたスキルを継承しません。

Tip:

フルな会話コンテキストが必要ない作業に subagents を使用します。それらの分離はメインセッションの膨張を防ぎます。

Hooks

時期：トリガー時。フックはツール実行、セッション境界、プロンプト送信、権限リクエスト、コンパクションなどの特定のライフサイクルイベントで発火します。完全なリストは [Hooks](#) を参照してください。

ロード内容：デフォルトではなし。フックは外部スクリプトとして実行されます。

コンテキストコスト：ゼロ、フックが会話にメッセージとして追加されるコンテキストを返さない限り。

Tip:

フックは Claude のコンテキストに影響する必要がない副作用（リント、ロギング）に最適です。

詳細を学ぶ

各機能には、セットアップ指示、例、設定オプションを含む独自のガイドがあります。

- [CLAUDE.md](#) プロジェクトコンテキスト、規約、指示を保存
- [Skills](#) Claude にドメイン専門知識と再利用可能なワークフローを提供
- [Subagents](#) 独立したコンテキストに作業をオフロード
- [Agent teams](#) 複数のセッションを並列で調整
- [MCP](#) Claude を外部サービスに接続
- [Hooks](#) フックでワークフローを自動化
- [Plugins](#) 機能セットをバンドルして共有
- [Marketplaces](#) プラグインコレクションをホストして配布

インタラクティブモード

Claude Code セッションのキーボードショートカット、入力モード、インタラクティブ機能の完全なリファレンス。

キーボードショートカット

Note:

キーボードショートカットはプラットフォームとターミナルによって異なる場合があります。**?** を押すと、お使いの環境で利用可能なショートカットが表示されます。

macOS ユーザー: Option/Alt キーショートカット (**Alt+B**、**Alt+F**、**Alt+Y**、**Alt+M**、**Alt+P**) を使用するには、ターミナルで Option を Meta として設定する必要があります。

- **iTerm2:** 設定 → プロファイル → キー → Left/Right Option キーを「Esc+」に設定
- **Terminal.app:** 設定 → プロファイル → キーボード → 「Option キーを Meta として使用」をチェック
- **VS Code:** 設定 → プロファイル → キー → Left/Right Option キーを「Esc+」に設定

詳細は [ターミナル設定](#) を参照してください。

一般的なコントロール

ショートカット	説明	コンテキスト
Ctrl+C	現在の入力または生成をキャンセル	標準割り込み
Ctrl+F	すべてのバックグラウンドエージェントを終了。3 秒以内に 2 回押して確認	バックグラウンドエージェント制御
Ctrl+D	Claude Code セッションを終了	EOF シグナル
Ctrl+G	デフォルトテキストエディタで開く	プロンプトまたはカスタム応答をデフォルトテキストエディタで編集
Ctrl+L	ターミナル画面をクリア	会話履歴を保持

ショートカット	説明	コンテキスト
Ctrl+O	詳細出力を切り替え	ツール使用と実行の詳細を表示
Ctrl+R	コマンド履歴を逆順検索	前のコマンドをインタラクティブに検索
Ctrl+V または Cmd+V (iTerm2) または Alt+V (Windows)	クリップボードから画像を貼り付け	画像またはイメージファイルへのパスを貼り付け
Ctrl+B	バックグラウンドで実行中のタスク	bash コマンドとエージェントをバックグラウンドで実行。Tmux ユーザーは 2 回押す
Ctrl+T	タスクリストを切り替え	ターミナルステータス領域の タスクリスト を表示または非表示
Left/Right 矢印	ダイアログタブを循環	パーミッションダイアログとメニューのタブ間を移動
Up/Down 矢印	コマンド履歴を移動	前の入力呼び出し
Esc + Esc	巻き戻すまたは要約	コードおよび/または会話を前の時点に復元するか、選択したメッセージから要約
Shift+Tab または Alt+M (一部の設定)	パーミッションモードを切り替え	Auto-Accept Mode、Plan Mode、通常モード間を切り替え
Option+P (macOS) または Alt+P (Windows/Linux)	モデルを切り替え	プロンプトをクリアせずにモデルを切り替え
Option+T (macOS) または Alt+T (Windows/Linux)	拡張思考を切り替え	拡張思考モードを有効または無効にします。このショートカットを有効にするには、まず /terminal-setup を実行してください

テキスト編集

ショートカット	説明	コンテキスト
<code>Ctrl+K</code>	行末まで削除	削除されたテキストは貼り付け用に保存
<code>Ctrl+U</code>	行全体を削除	削除されたテキストは貼り付け用に保存
<code>Ctrl+Y</code>	削除されたテキストを貼り付け	<code>Ctrl+K</code> または <code>Ctrl+U</code> で削除されたテキストを貼り付け
<code>Alt+Y</code> (<code>Ctrl+Y</code> の後)	貼り付け履歴を循環	貼り付け後、以前に削除されたテキストを循環します。macOS では Option を Meta として 設定が必要
<code>Alt+B</code>	カーソルを 1 単語戻す	単語ナビゲーション。 macOS では Option を Meta として 設定が必要
<code>Alt+F</code>	カーソルを 1 単語進める	単語ナビゲーション。 macOS では Option を Meta として 設定が必要

テーマと表示

ショートカット	説明	コンテキスト
<code>Ctrl+T</code>	コードブロックの構文強調表示を切り替え	<code>/theme</code> ピッカーメニュー内でのみ機能します。Claude の応答のコードが構文色付けを使用するかどうかを制御

Note:

構文強調表示は Claude Code のネイティブビルドでのみ利用可能です。

複数行入力

方法	ショートカット	コンテキスト
クイックエスケープ	<code>\ + Enter</code>	すべてのターミナルで機能
macOS デフォルト	<code>Option+Enter</code>	macOS のデフォルト
Shift+Enter	<code>Shift+Enter</code>	iTerm2、WezTerm、Ghostty、Kitty で標準で機能
制御シーケンス	<code>Ctrl+J</code>	複数行用のラインフィード文字
ペーストモード	直接貼り付け	コードブロック、ログ用

Tip:

Shift+Enter は iTerm2、WezTerm、Ghostty、Kitty で設定なしで機能します。その他のターミナル（VS Code、Alacritty、Zed、Warp）の場合は、`/terminal-setup` を実行してバインディングをインストールしてください。

クイックコマンド

ショートカット	説明	注記
<code>/</code> で開始	コマンドまたはスキル	組み込みコマンド と スキル を参照
<code>!</code> で開始	Bash モード	コマンドを直接実行し、実行出力をセッションに追加
<code>@</code>	ファイルパスメンション	ファイルパスオートコンプリートをトリガー

組み込みコマンド

Claude Code で `/` を入力すると、利用可能なすべてのコマンドが表示されます。または `/` の後に任意の文字を入力してフィルタリングできます。すべてのコマンドがすべてのユーザーに表示されるわけではありません。プラットフォーム、プラン、または環境によって異なります。たとえば、`/desktop` は macOS と Windows にのみ表示され、`/`

`upgrade` と `/privacy-settings` は Pro および Max プランでのみ利用可能で、`/terminal-setup` はターミナルがそのキーバインディングをネイティブにサポートしている場合は非表示です。

Claude Code には、`/` を入力したときに組み込みコマンドと一緒に表示される `/simplify`、`/batch`、`/debug` などの [バンドルされたスキル](#) も付属しています。独自のコマンドを作成するには、[スキル](#) を参照してください。

以下の表では、`<arg>` は必須引数を示し、`[arg]` はオプション引数を示します。

コマンド	目的
<code>/add-dir</code> <code><path></code>	現在のセッションに新しい作業ディレクトリを追加
<code>/agents</code>	エージェント 設定を管理
<code>/btw</code> <code><question></code>	会話に追加せずに サイドクエスチョン をすばやく質問
<code>/chrome</code>	Chrome の Claude 設定を設定
<code>/clear</code>	会話履歴をクリアしてコンテキストを解放。エイリアス: <code>/reset</code> 、 <code>/new</code>
<code>/compact</code> <code>[instructions]</code>	オプションのフォーカス指示付きで会話をコンパクト化
<code>/config</code>	設定 インターフェースを開いて、テーマ、モデル、 出力スタイル 、その他の設定を調整。エイリアス: <code>/settings</code>
<code>/context</code>	現在のコンテキスト使用状況をカラーグリッドとして視覚化
<code>/copy</code>	最後のアシスタント応答をクリップボードにコピー。コードブロックが存在する場合、個別ブロックまたは完全な応答を選択するインタラクティブピッカーを表示
<code>/cost</code>	トークン使用統計を表示。サブスクリプション固有の詳細は コスト追跡ガイド を参照
<code>/desktop</code>	現在のセッションを Claude Code デスクトップアプリで続行。macOS と Windows のみ。エイリアス: <code>/app</code>
<code>/diff</code>	コミットされていない変更と各ターンの差分を表示するインタラクティブ差分ビューアを開く。左右矢印を使用して現在の git 差分と個別の Claude ターン間を切り替え、上下で ファイル を参照

コマンド	目的
<code>/doctor</code>	Claude Code インストールと設定を診断および検証
<code>/exit</code>	CLIを終了。エイリアス: <code>/quit</code>
<code>/export [filename]</code>	現在の会話をプレーンテキストとしてエクスポート。ファイル名を指定すると、そのファイルに直接書き込みます。指定しない場合、クリップボードにコピーするか、ファイルに保存するダイアログを開く
<code>/extra-usage</code>	レート制限に達したときに作業を続行するための追加使用量を設定
<code>/fast [on off]</code>	高速モード をオンまたはオフに切り替え
<code>/feedback [report]</code>	Claude Code に関するフィードバックを送信。エイリアス: <code>/bug</code>
<code>/fork [name]</code>	この時点で現在の会話のフォークを作成
<code>/help</code>	ヘルプと利用可能なコマンドを表示
<code>/hooks</code>	ツールイベント用の フック 設定を管理
<code>/ide</code>	IDE 統合を管理し、ステータスを表示
<code>/init</code>	<code>CLAUDE.md</code> ガイドでプロジェクトを初期化
<code>/insights</code>	Claude Code セッションを分析するレポートを生成します。プロジェクト領域、インタラクションパターン、摩擦点を含む
<code>/install-github-app</code>	リポジトリ用の Claude GitHub Actions アプリをセットアップ。リポジトリを選択して統合を設定するプロセスを案内
<code>/install-slack-app</code>	Claude Slack アプリをインストール。ブラウザを開いて OAuth フローを完了
<code>/keybindings</code>	キーバインディング設定ファイルを開くまたは作成
<code>/login</code>	Anthropic アカウントにサインイン
<code>/logout</code>	Anthropic アカウントからサインアウト
<code>/mcp</code>	MCP サーバー接続と OAuth 認証を管理
<code>/memory</code>	<code>CLAUDE.md</code> メモリファイルを編集し、 自動メモリ を有効または無効にし、自動メモリエントリを表示

コマンド	目的
<code>/mobile</code>	Claude モバイルアプリをダウンロードするための QR コードを表示。エイリアス: <code>/ios</code> 、 <code>/android</code>
<code>/model [model]</code>	AI モデルを選択または変更。それをサポートするモデルの場合、左右矢印を使用して 努力レベル を調整。変更は現在の応答の完了を待たずに即座に有効になります
<code>/passes</code>	友人と Claude Code の無料 1 週間を共有。アカウントが対象の場合にのみ表示
<code>/permissions</code>	パーミッション を表示または更新。エイリアス: <code>/allowed-tools</code>
<code>/plan</code>	プロンプトから直接プランモードに入る
<code>/plugin</code>	Claude Code プラグイン を管理
<code>/pr-comments [PR]</code>	GitHub プルリクエストからコメントを取得して表示。現在のブランチの PR を自動検出するか、PR URL または番号を渡します。 <code>gh</code> CLI が必要
<code>/privacy-settings</code>	プライバシー設定を表示および更新。Pro および Max プランサブスクライバーのみ利用可能
<code>/release-notes</code>	完全なチェンジログを表示します。最新バージョンがプロンプトに最も近い
<code>/reload-plugins</code>	すべてのアクティブな プラグイン を再度読み込んで、再起動せずに保留中の変更を適用。読み込まれた内容を報告し、再起動が必要な変更を記録
<code>/remote-control</code>	このセッションを claude.ai から リモート制御 できるようにする。エイリアス: <code>/rc</code>
<code>/remote-env</code>	テレポートセッション のデフォルトリモート環境を設定
<code>/rename [name]</code>	現在のセッションの名前を変更。名前を指定しない場合、会話履歴から自動生成
<code>/resume [session]</code>	ID または名前で会話を再開するか、セッションピッカーを開く。エイリアス: <code>/continue</code>
<code>/review</code>	非推奨。代わりに <code>code-review</code> プラグイン をインストール: <code>claude plugin install code-review@claude-code-marketplace</code>
<code>/rewind</code>	会話またはコードを前の時点に巻き戻すか、選択したメッセージから要約。 チェックポイント を参照。エイリアス: <code>/checkpoint</code>

コマンド	目的
<code>/sandbox</code>	サンドボックスモード を切り替え。サポートされているプラットフォームでのみ利用可能
<code>/security-review</code>	現在のブランチの保留中の変更をセキュリティ脆弱性について分析。git 差分をレビューし、インジェクション、認証問題、データ露出などのリスクを特定
<code>/skills</code>	利用可能な スキル をリスト
<code>/stats</code>	日次使用状況、セッション履歴、ストリーク、モデル設定を視覚化
<code>/status</code>	設定インターフェース（ステータスタブ）を開いて、バージョン、モデル、アカウント、接続性を表示
<code>/statusline</code>	Claude Code の ステータスライン を設定。必要な内容を説明するか、引数なしで実行してシェルスプロンプトから自動設定
<code>/stickers</code>	Claude Code ステッカーを注文
<code>/tasks</code>	バックグラウンドタスクをリストおよび管理
<code>/terminal-setup</code>	Shift+Enter およびその他のショートカット用のターミナルキーバインディングを設定。VS Code、Alacritty、Warp などの必要なターミナルでのみ表示
<code>/theme</code>	カラーテーマを変更。ライトおよびダークバリエント、色覚異常対応（ダルトン化）テーマ、ターミナルのカラーパレットを使用する ANSI テーマを含む
<code>/upgrade</code>	アップグレードページを開いて、より高いプランティアに切り替え
<code>/usage</code>	プラン使用制限とレート制限ステータスを表示
<code>/vim</code>	Vim と通常編集モード間を切り替え

MCP プロンプト

MCP サーバーはコマンドとして表示されるプロンプトを公開できます。これらは `/mcp_<server>_<prompt>` 形式を使用し、接続されたサーバーから動的に検出されます。詳細は [MCP プロンプト](#) を参照してください。

Vim エディタモード

`/vim` コマンドで Vim スタイルの編集を有効にするか、`/config` で永続的に設定します。

モード切り替え

コマンド	アクション	モードから
<code>Esc</code>	NORMAL モードに入る	INSERT
<code>i</code>	カーソルの前に挿入	NORMAL
<code>I</code>	行の開始に挿入	NORMAL
<code>a</code>	カーソルの後に挿入	NORMAL
<code>A</code>	行の終わりに挿入	NORMAL
<code>o</code>	下に行を開く	NORMAL
<code>O</code>	上に行を開く	NORMAL

ナビゲーション (NORMAL モード)

コマンド	アクション
<code>h / j / k / l</code>	左/下/上/右に移動
<code>w</code>	次の単語
<code>e</code>	単語の終わり
<code>b</code>	前の単語
<code>0</code>	行の開始
<code>\$</code>	行の終わり
<code>^</code>	最初の空白以外の文字
<code>gg</code>	入力の開始
<code>G</code>	入力の終わり
<code>f{char}</code>	次の文字の出現にジャンプ

コマンド	アクション
<code>F{char}</code>	前の文字の出現にジャンプ
<code>t{char}</code>	次の文字の出現の直前にジャンプ
<code>T{char}</code>	前の文字の出現の直後にジャンプ
<code>;</code>	最後の f/F/t/T モーションを繰り返す
<code>,</code>	最後の f/F/t/T モーションを逆順で繰り返す

Note:

vim 通常モードでは、カーソルが入力の開始または終了にあり、それ以上移動できない場合、矢印キーはコマンド履歴を移動します。

編集 (NORMAL モード)

コマンド	アクション
<code>x</code>	文字を削除
<code>dd</code>	行を削除
<code>D</code>	行末まで削除
<code>dw / de / db</code>	単語を削除/終わりまで/戻す
<code>cc</code>	行を変更
<code>C</code>	行末まで変更
<code>cw / ce / cb</code>	単語を変更/終わりまで/戻す
<code>yy / Y</code>	行をヤंक (コピー)
<code>yw / ye / yb</code>	単語をヤंक/終わりまで/戻す
<code>p</code>	カーソルの後に貼り付け
<code>P</code>	カーソルの前に貼り付け
<code>>></code>	行をインデント
<code><<</code>	行をデデント

コマンド	アクション
<code>j</code>	行を結合
<code>.</code>	最後の変更を繰り返す

テキストオブジェクト (NORMAL モード)

テキストオブジェクトは `d`、`c`、`y` などのオペレータで機能します。

コマンド	アクション
<code>iw / aw</code>	内部/周囲の単語
<code>iW / aW</code>	内部/周囲の WORD (空白区切り)
<code>i" / a"</code>	内部/周囲のダブルクォート
<code>i' / a'</code>	内部/周囲のシングルクォート
<code>i(/ a(</code>	内部/周囲の括弧
<code>i[/ a[</code>	内部/周囲の角括弧
<code>if / af</code>	内部/周囲の波括弧

コマンド履歴

Claude Code は現在のセッションのコマンド履歴を保持します。

- 入力履歴は作業ディレクトリごとに保存
- 入力履歴は `/clear` を実行して新しいセッションを開始するとリセット。前のセッションの会話は保存され、再開できます。
- Up/Down 矢印を使用して移動（上記のキーボードショートカットを参照）
- **注記:** 履歴展開（`!`）はデフォルトで無効

Ctrl+R での逆順検索

`Ctrl+R` を押してコマンド履歴をインタラクティブに検索します。

1. **検索を開始:** `Ctrl+R` を押して逆順履歴検索を有効化
2. **クエリを入力:** 前のコマンドで検索するテキストを入力。検索用語は一致する結果で強調表示
3. **一致を移動:** `Ctrl+R` をもう一度押して、より古い一致を循環

4. 一致を受け入れ:

- **Tab** または **Esc** を押して現在の一致を受け入れ、編集を続行
- **Enter** を押して一致を受け入れ、コマンドを即座に実行

5. 検索をキャンセル:

- **Ctrl+C** を押してキャンセルし、元の入力を復元
- 空の検索で **Backspace** を押してキャンセル

検索は検索用語が強調表示された一致するコマンドを表示するため、前の入力を見つけて再利用できます。

バックグラウンド bash コマンド

Claude Code はバックグラウンドで bash コマンドを実行することをサポートしており、長時間実行されるプロセスが実行されている間に作業を続行できます。

バックグラウンド実行の仕組み

Claude Code がコマンドをバックグラウンドで実行すると、コマンドを非同期で実行し、すぐにバックグラウンドタスク ID を返します。Claude Code はコマンドがバックグラウンドで実行され続けている間、新しいプロンプトに応答できます。

コマンドをバックグラウンドで実行するには、以下のいずれかを実行できます。

- Claude Code にコマンドをバックグラウンドで実行するよう指示
- **Ctrl+B** を押して、通常の Bash ツール呼び出しをバックグラウンドに移動。(Tmux ユーザーは **tmux** のプレフィックスキーのため **Ctrl+B** を 2 回押す必要があります。)

主な機能:

- 出力はバッファリングされ、Claude は TaskOutput ツールを使用して取得できます
- バックグラウンドタスクは追跡と出力取得用の一意の ID を持ちます
- バックグラウンドタスクは Claude Code の終了時に自動的にクリーンアップされます

すべてのバックグラウンドタスク機能を無効にするには、

`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` 環境変数を `1` に設定します。詳細は [環境変数](#) を参照してください。

一般的なバックグラウンド実行コマンド:

- ビルドツール (webpack、vite、make)
- パッケージマネージャー (npm、yarn、pnpm)
- テストランナー (jest、pytest)

- 開発サーバー
- 長時間実行プロセス (docker、terraform)

! プレフィックス付き Bash モード

入力に **!** をプレフィックスして、Claude を経由せずに bash コマンドを直接実行します。

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

Bash モード:

- コマンドとその出力を会話コンテキストに追加
- リアルタイムの進捗と出力を表示
- 長時間実行コマンド用の同じ **Ctrl+B** バックグラウンド実行をサポート
- Claude がコマンドを解釈または承認する必要がない
- 履歴ベースのオートコンプリートをサポート: 部分的なコマンドを入力し、**Tab** を押して現在のプロジェクトの前の **!** コマンドから完了
- **Escape**、**Backspace**、または空のプロンプトで **Ctrl+U** で終了

これは会話コンテキストを保持しながら、クイックシェル操作に便利です。

プロンプト提案

セッションを最初に開くと、グレイアウトされた例コマンドがプロンプト入力に表示され、開始するのに役立ちます。Claude Code はプロジェクトの git 履歴からこれを選択するため、最近作業しているファイルが反映されます。

Claude が応答した後、マルチパートリクエストのフォローアップステップや、ワークフローの自然な継続など、会話履歴に基づいて提案が表示され続けます。

- **Tab** を押して提案を受け入れるか、**Enter** を押して受け入れて送信
- 入力を開始して却下

提案は親の会話のプロンプトキャッシュを再利用するバックグラウンドリクエストとして実行されるため、追加コストは最小限です。Claude Code はキャッシュがコールドの場合、不要なコストを避けるために提案生成をスキップします。

提案は会話の最初のターン後、非インタラクティブモード、およびプランモードで自動的にスキップされます。

プロンプト提案を完全に無効にするには、環境変数を設定するか、`/config` で設定を切り替えます。

```
export CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION=false
```

`/btw` でのサイドクエスチョン

`/btw` を使用して、会話履歴に追加せずに現在の作業についての簡単な質問をします。メイン コンテキストを乱さずに、または Claude を長時間実行中のタスクから脱線させずに、高速な回答が必要な場合に便利です。

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

サイドクエスチョンは現在の会話に完全に表示されるため、Claude が既に読んだコード、以前に行った決定、またはセッションの他の何かについて質問できます。質問と回答は一時的です。これらは却下可能なオーバーレイに表示され、会話履歴に入ることはありません。

- **Claude が作業中に利用可能:** Claude が応答を処理している間でも `/btw` を実行できます。サイドクエスチョンは独立して実行され、メインターンを中断しません。
- **ツールアクセスなし:** サイドクエスチョンはコンテキストに既にあるもののみから回答します。Claude はサイドクエスチョンに回答するときにファイルを読み取ったり、コマンドを実行したり、検索したりできません。
- **単一応答:** フォローアップターンはありません。バックアンドフォースが必要な場合は、通常のプロンプトを使用してください。
- **低コスト:** サイドクエスチョンは親の会話のプロンプトキャッシュを再利用するため、追加コストは最小限です。

Space、**Enter**、または **Escape** を押して回答を却下し、プロンプトに戻ります。

`/btw` は [サブエージェント](#) の逆です。完全な会話を見ることができますがツールがない一方、サブエージェントは完全なツールを持ちますが空のコンテキストで開始します。`/btw` を使用して Claude がこのセッションから既に知っていることについて質問します。サブエージェントを使用して何か新しいことを見つけます。

タスクリスト

複雑なマルチステップ作業に取り組む場合、Claude はタスクリストを作成して進捗を追跡します。タスクはターミナルのステータス領域に表示され、保留中、進行中、または完了を示すインジケータが表示されます。

- `Ctrl+T` を押してタスクリストビューを切り替え。表示は一度に最大 10 個のタスクを表示
- すべてのタスクを表示するか、クリアするには、Claude に直接質問します。「show me all tasks」または「clear all tasks」
- タスクはコンテキストコンパクション全体で保持され、Claude がより大きなプロジェクトで整理された状態を保つのに役立ちます
- タスクリストをセッション間で共有するには、`CLAUDE_CODE_TASK_LIST_ID` を `~/.claude/tasks/` の名前付きディレクトリに使用するように設定します。
`CLAUDE_CODE_TASK_LIST_ID=my-project claude`
- 前の TODO リストに戻すには、`CLAUDE_CODE_ENABLE_TASKS=false` を設定します。

PR レビューステータス

オープンなプルリクエストを持つブランチで作業する場合、Claude Code はフッターにクリック可能な PR リンクを表示します（例：「PR #446」）。リンクには、レビュー状態を示すカラーアンダーラインがあります。

- 緑: 承認済み
- 黄: レビュー保留中
- 赤: 変更をリクエスト
- グレー: ドラフト
- 紫: マージ済み

`Cmd+click` (Mac) または `Ctrl+click` (Windows/Linux) でリンクをクリックして、プルリクエストをブラウザで開きます。ステータスは 60 秒ごとに自動的に更新されます。

Note:

PR ステータスには、`gh` CLI がインストールされ、認証されている (`gh auth login`) 必要があります。

関連項目

- [スキル](#) - カスタムプロンプトとワークフロー

- [チェックポイント](#) - Claude の編集を巻き戻し、前の状態を復元
- [CLI リファレンス](#) - コマンドラインフラグとオプション
- [設定](#) - 設定オプション
- [メモリ管理](#) - CLAUDE.md ファイルの管理

Claude Code のベストプラクティス

環境設定から並列セッションでのスケーリングまで、Claude Code を最大限に活用するためのヒントとパターン。

Claude Code は agentic coding 環境です。質問に答えて待つチャットボットとは異なり、Claude Code はファイルを読み取り、コマンドを実行し、変更を加え、あなたが見守ったり、方向を変えたり、完全に任せたりしながら、自律的に問題を解決できます。

これはあなたの作業方法を変えます。自分でコードを書いて Claude にレビューしてもらう代わりに、やりたいことを説明すると Claude がそれをどのように構築するかを考え出します。Claude は探索し、計画し、実装します。

しかし、この自律性にも学習曲線があります。Claude は理解する必要がある特定の制約の中で動作します。

このガイドでは、Anthropic の内部チームと、様々なコードベース、言語、環境で Claude Code を使用しているエンジニアの間で効果的であることが証明されたパターンについて説明します。agentic ループがどのように機能するかについては、[Claude Code の仕組み](#)を参照してください。

ほとんどのベストプラクティスは 1 つの制約に基づいています。Claude のコンテキストウィンドウはすぐにいっぱいになり、満杯になるにつれてパフォーマンスが低下します。

Claude のコンテキストウィンドウは、すべてのメッセージ、Claude が読み取ったすべてのファイル、およびすべてのコマンド出力を含む、会話全体を保持します。ただし、これはすぐにいっぱいになる可能性があります。単一のデバッグセッションまたはコードベース探索でも、数万のトークンを生成および消費する可能性があります。

LLM のパフォーマンスはコンテキストが満杯になるにつれて低下するため、これは重要です。コンテキストウィンドウがいっぱいになると、Claude は以前の指示を「忘れる」か、より多くの間違いを犯す可能性があります。コンテキストウィンドウは管理する最も重要なリソースです。[カスタムステータスライン](#)でコンテキスト使用量を継続的に追跡し、トークン使用量を削減するための戦略については[トークン使用量を削減](#)を参照してください。

Claude に自分の作業を検証する方法を与える

Tip:

テスト、スクリーンショット、または期待される出力を含めて、Claude が自分自身をチェックできるようにします。これはあなたができる最も高いレバレッジのことです。

Claude は、テストを実行したり、スクリーンショットを比較したり、出力を検証したりするなど、自分の作業を検証できるときに劇的に良くなります。

明確な成功基準がないと、正しく見えるが実際には機能しないものを生成する可能性があります。あなたが唯一のフィードバックループになり、すべての間違いがあなたの注意を必要とします。

戦略	前	後
検証基準を提供する	「メールアドレスを検証する関数を実装する」	「validateEmail 関数を書く。テストケースの例: user@example.com は true、invalid は false、 user@.com は false。実装後にテストを実行する」
UI の変更を視覚的に検証する	「ダッシュボードをより良く見えるようにする」	「[スクリーンショットを貼り付け] このデザインを実装する。結果のスクリーンショットを撮り、元のものと比較する。違いをリストアップして修正する」
症状ではなく根本原因に対処する	「ビルドが失敗している」	「ビルドがこのエラーで失敗している: [エラーを貼り付け]。修正して、ビルドが成功することを確認する。根本原因に対処し、エラーを抑制しない」

UI の変更は [Chrome 拡張機能の Claude](#) を使用して検証できます。これはブラウザで新しいタブを開き、UI をテストし、コードが機能するまで反復します。

検証はテストスイート、リンター、または出力をチェックする Bash コマンドにすることもできます。検証を堅牢にすることに投資してください。

最初に探索し、次に計画し、その後コーディングする

Tip:

研究と計画を実装から分離して、間違った問題を解決することを避けます。

Claude が直接コーディングにジャンプさせると、間違った問題を解決するコードが生成される可能性があります。[Plan Mode](#) を使用して、探索を実行から分離します。

推奨されるワークフローには 4 つのフェーズがあります。

Step 1: 探索

Plan Mode に入ります。Claude はファイルを読み取り、変更を加えずに質問に答えます。

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

Step 2: 計画

Claude に詳細な実装計画を作成するよう依頼します。

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

Ctrl+G を押して、Claude が進む前に、テキストエディタで計画を開いて直接編集します。

Step 3: 実装

Normal Mode に戻り、Claude にコーディングさせ、計画に対して検証します。

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

Step 4: コミット

Claude に説明的なメッセージでコミットし、PR を作成するよう依頼します。

commit with a descriptive message and open a PR

Plan Mode は便利ですが、オーバーヘッドも追加します。

スコープが明確で修正が小さいタスク（タイプミスの修正、ログ行の追加、変数の名前変更など）の場合は、Claude に直接実行するよう依頼します。

計画は、アプローチについて不確実な場合、変更が複数のファイルを変更する場合、または変更されるコードに不慣れな場合に最も役立ちます。差分を 1 文で説明できる場合は、計画をスキップします。

プロンプトで具体的なコンテキストを提供する

Tip:

指示がより正確であるほど、必要な修正が少なくなります。

Claude は意図を推測できますが、あなたの心を読むことはできません。特定のファイルを参照し、制約を述べ、例のパターンを指摘します。

戦略	前	後
タスクをスコープする。 どのファイル、どのシナリオ、テスト設定を指定します。	「foo.py のテストを追加する」	「ユーザーがログアウトしているエッジケースをカバーする foo.py のテストを書く。モックを避ける。」
ソースを指摘する。 Claude を質問に答えることができるソースに向けます。	「ExecutionFactory がこんなに奇妙な API を持っているのはなぜですか？」	「ExecutionFactory の git 履歴を調べて、その API がどのようになったかを要約する」

戦略	前	後
既存のパターンを参照する。Claude をコードベースのパターンに向けます。	「カレンダーウィジェットを追加する」	「ホームページで既存のウィジェットがどのように実装されているかを見て、パターンを理解する。 HotDogWidget.php は良い例です。パターンに従って、ユーザーが月を選択し、前後にページネーションして年を選択できる新しいカレンダーウィジェットを実装する。コードベースで既に使用されているもの以外のライブラリを使用せずにゼロから構築する。」
症状を説明する。症状、可能性のある場所、「修正」の外観を提供します。	「ログインバグを修正する」	「ユーザーはセッションタイムアウト後にログインが失敗すると報告しています。src/auth/ の認証フロー、特にトークン更新を確認します。問題を再現する失敗するテストを書き、修正する」

曖昧なプロンプトは、探索していて方向転換を余裕を持つてできるときに役立つことがあります。「このファイルで何を改善しますか？」のようなプロンプトは、あなたが尋ねることを考えなかったことを表面化させることができます。

リッチコンテンツを提供する

Tip:

📎 を使用してファイルを参照したり、スクリーンショット/画像を貼り付けたり、データを直接パイプしたりします。

Claude にリッチデータを提供するにはいくつかの方法があります。

- 📎 でファイルを参照する コードがどこにあるかを説明する代わりに。Claude は応答する前にファイルを読み取ります。
- 画像を直接貼り付ける。画像をコピー/貼り付けまたはドラッグアンドドロップしてプロンプトに入れます。

- **ドキュメントと API リファレンスの URL を指定する。** `/permissions` を使用して、頻繁に使用されるドメインをホワイトリストに登録します。
- **データをパイプする** `cat error.log | claude` を実行してファイルの内容を直接送信します。
- **Claude に必要なものを取得させる。** Bash コマンド、MCP ツール、またはファイルを読み取ることを使用して、Claude 自身がコンテキストをプルするよう指示します。

環境を設定する

いくつかのセットアップステップにより、Claude Code はすべてのセッション全体で大幅に効果的になります。拡張機能の完全な概要と各機能をいつ使用するかについては、[Claude Code を拡張](#)を参照してください。

効果的な CLAUDE.md を書く

Tip:

`/init` を実行して、現在のプロジェクト構造に基づいてスターター CLAUDE.md ファイルを生成し、時間をかけて改善します。

CLAUDE.md は Claude がすべての会話の開始時に読む特別なファイルです。Bash コマンド、コードスタイル、ワークフロールールを含めます。これにより、Claude はコードだけでは推測できない永続的なコンテキストを取得します。

`/init` コマンドはコードベースを分析してビルドシステム、テストフレームワーク、コードパターンを検出し、改善するための堅牢な基盤を提供します。

CLAUDE.md ファイルに必須の形式はありませんが、短く人間が読める状態に保ちます。例えば：

```
## 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 はすべてのセッションで読み込まれるため、広く適用されるもののみを含めます。ドメイン知識またはときどきのみ関連するワークフローについては、代わりに [skills](#) を使用します。Claude はそれらをオンデマンドで読み込み、すべての会話を膨らませることなく使用します。

簡潔に保ちます。各行について、次のように尋ねます。「これを削除すると Claude が間違いを犯しますか？」 そうでない場合は、削除します。膨らんだ CLAUDE.md ファイルは Claude があなたの実際の指示を無視するようにします。

✓ 含める	✗ 除外する
Claude が推測できない Bash コマンド	Claude がコードを読むことで理解できるもの
デフォルトと異なるコードスタイルルール	Claude が既に知っている標準言語規約
テスト指示と推奨テストランナー	詳細な API ドキュメント（代わりにドキュメントにリンク）
リポジトリのエチケット（ブランチ命名、PR 規約）	頻繁に変わる情報
プロジェクト固有のアーキテクチャ決定	長い説明またはチュートリアル
開発者環境の癖（必須環境変数）	ファイルごとのコードベースの説明
一般的な落とし穴または明白でない動作	「きれいなコードを書く」のような自明なプラクティス

Claude が CLAUDE.md にルールがあるにもかかわらず、あなたが望まないことをし続ける場合、ファイルはおそらく長すぎて、ルールが失われています。Claude が CLAUDE.md で答えられている質問をあなたに尋ねる場合、フレーズが曖昧かもしれません。CLAUDE.md をコードのように扱う：物事がうまくいかないときにレビューし、定期的に削除し、Claude の動作が実際に変わるかどうかを観察することで変更をテストします。

`@path/to/import` 構文を使用して追加ファイルをインポートすることで、指示を調整できます。

```
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

CLAUDE.md ファイルはいくつかの場所に配置できます。

- ・ **ホームフォルダ** (`~/ .claude/CLAUDE.md`) : すべての Claude セッションに適用されます
- ・ **プロジェクトルート** (`./CLAUDE.md`) : git にチェックインしてチームと共有します
- ・ **親ディレクトリ** : `root/CLAUDE.md` と `root/foo/CLAUDE.md` の両方が自動的にプルされるモノレポに役立ちます
- ・ **子ディレクトリ** : Claude はそれらのディレクトリ内のファイル进行操作するときに、子 CLAUDE.md ファイルをオンデマンドでプルします

パーミッションを設定する

Tip:

`/permissions` を使用して安全なコマンドをホワイトリストに登録するか、 `/sandbox` を使用して OS レベルの分離を行います。これにより、制御を保ちながら中断を減らします。

デフォルトでは、Claude Code はシステムを変更する可能性のあるアクション（ファイル書き込み、Bash コマンド、MCP ツールなど）の許可をリクエストします。これは安全ですが、面倒です。10 回目の承認後、あなたは本当にレビューしていません。クリックしているだけです。これらの中断を減らすには 2 つの方法があります。

- ・ **パーミッションホワイトリスト** : 安全であることがわかっているツール (`npm run lint` や `git commit` など) を許可します
- ・ **サンドボックス** : OS レベルの分離を有効にして、ファイルシステムとネットワークアクセスを制限し、Claude が定義された境界内でより自由に動作できるようにします

または、 `--dangerously-skip-permissions` を使用して、lint エラーの修正やボイラープレートの生成などの含まれたワークフローのすべてのパーミッションチェックをバイパスします。

Warning:

Claude に任意のコマンドを実行させると、プロンプトインジェクションを介したデータ損失、システム破損、またはデータ流出が発生する可能性があります。 `--dangerously-skip-permissions` はインターネットアクセスのないサンドボックスでのみ使用します。

[パーミッションの設定](#)と[サンドボックスの有効化](#)の詳細をお読みください。

CLI ツールを使用する

Tip:

Claude Code に `gh`、`aws`、`gcloud`、`sentry-cli` などの CLI ツールを使用して外部サービスと対話するよう指示します。

CLI ツールは外部サービスと対話する最もコンテキスト効率的な方法です。GitHub を使用する場合は、`gh` CLI をインストールします。Claude は問題の作成、プルリクエストのオープン、コメントの読み取りにそれを使用する方法を知っています。`gh` がなければ、Claude は GitHub API を使用できますが、認証されていないリクエストはしばしばレート制限に達します。

Claude は、それが既に知らない CLI ツールを学ぶのにも効果的です。

`Use 'foo-cli-tool --help' to learn about foo tool, then use it to solve A, B, C.`
のようなプロンプトを試してください。

MCP サーバーを接続する

Tip:

`claude mcp add` を実行して、Notion、Figma、またはデータベースなどの外部ツールを接続します。

[MCP サーバー](#)を使用すると、Claude に問題トラッカーから機能を実装したり、データベースをクエリしたり、監視データを分析したり、Figma からデザインを統合したり、ワークフローを自動化したりするよう依頼できます。

フックを設定する

Tip:

例外なしで毎回発生する必要があるアクションにはフックを使用します。

フックは Claude のワークフロー内の特定のポイントで自動的にスクリプトを実行します。CLAUDE.md の指示とは異なり、フックは決定論的であり、アクションが発生することを保証します。

Claude はあなたのためにフックを書くことができます。「すべてのファイル編集後に eslint を実行するフックを書く」または「migrations フォルダへの書き込みをブロックするフックを書く」のようなプロンプトを試してください。対話的な設定については `/hooks` を実行するか、`.claude/settings.json` を直接編集します。

スキルを作成する

Tip:

`.claude/skills/` に `SKILL.md` ファイルを作成して、Claude にドメイン知識と再利用可能なワークフローを提供します。

スキルは、プロジェクト、チーム、またはドメイン固有の情報で Claude の知識を拡張します。Claude は関連するときに自動的にそれらを適用するか、`/skill-name` で直接呼び出すことができます。

`.claude/skills/` にディレクトリと `SKILL.md` を追加してスキルを作成します。

```
---
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/)
```

スキルは、直接呼び出す再利用可能なワークフローを定義することもできます。

```

---
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

```

`/fix-issue 1234` を実行して呼び出します。副作用のあるワークフローで、手動でトリガーしたい場合は `disable-model-invocation: true` を使用します。

カスタムサブエージェントを作成する

Tip:

`.claude/agents/` に特化したアシスタントを定義して、Claude が分離されたタスクに委譲できるようにします。

[サブエージェント](#)は独自のコンテキストと独自の許可されたツールセットで実行されます。メインの会話を乱さずに、多くのファイルを読み取ったり、特化した焦点が必要なタスクに役立ちます。

```

---
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.

```

Claude に明示的にサブエージェントを使用するよう指示します。「サブエージェントを使用してこのコードをセキュリティの問題についてレビューする。」

プラグインをインストールする

Tip:

`/plugin` を実行してマーケットプレイスを参照します。プラグインは設定なしでスキル、ツール、統合を追加します。

[プラグイン](#)は、コミュニティと Anthropic からの単一のインストール可能なユニットにスキル、フック、サブエージェント、MCP サーバーをバンドルします。型付き言語を使用する場合は、[コードインテリジェンスプラグイン](#)をインストールして、Claude に正確なシンボルナビゲーションと編集後の自動エラー検出を提供します。

スキル、サブエージェント、フック、MCP の選択に関するガイダンスについては、[Claude Code を拡張](#)を参照してください。

効果的にコミュニケーションする

Claude Code との通信方法は、結果の品質に大きく影響します。

コードベースの質問をする

Tip:

シニアエンジニアに尋ねるような質問を Claude にしてください。

新しいコードベースにオンボーディングするときは、Claude Code を学習と探索に使用します。別のエンジニアに尋ねるのと同じ種類の質問を Claude に尋ねることができます。

- ログインはどのように機能しますか？
- 新しい API エンドポイントを作成するにはどうすればよいですか？
- `foo.rs` の 134 行目の `async move { ... }` は何をしていますか？
- `CustomerOnboardingFlowImpl` はどのエッジケースを処理しますか？
- このコードが 333 行目で `bar()` の代わりに `foo()` を呼び出すのはなぜですか？

Claude Code をこのように使用することは、効果的なオンボーディングワークフローであり、ラップアップ時間を改善し、他のエンジニアの負荷を軽減します。特別なプロンプトは必要ありません。直接質問してください。

Claude にあなたにインタビューさせる

Tip:

より大きな機能については、Claude に最初あなたにインタビューさせます。最小限のプロンプトで開始し、Claude に `AskUserQuestion` ツールを使用してあなたにインタビューするよう依頼します。

Claude は、技術的な実装、UI/UX、エッジケース、トレードオフなど、あなたがまだ考えていないことについて質問します。

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.

仕様が完成したら、新しいセッションを開始して実行します。新しいセッションはクリーンなコンテキストを持ち、実装に完全に焦点を当てており、参照する書かれた仕様があります。

セッションを管理する

会話は永続的で可逆的です。これを有利に使用してください。

早期かつ頻繁に方向転換する

Tip:

Claude が軌道を外れていることに気付いたらすぐに修正します。

最良の結果は、タイトなフィードバックループから来ます。Claude は時々最初の試みで問題を完全に解決しますが、それを迅速に修正することは一般的により良い解決策をより速く生成します。

- **Esc** : **Esc** キーで Claude の中途半端なアクションを停止します。コンテキストは保持されるため、リダイレクトできます。
- **Esc + Esc** または **/rewind** : **Esc** を 2 回押すか **/rewind** を実行して、巻き戻しメニューを開き、以前の会話とコード状態を復元するか、選択したメッセージから要約します。
- **"Undo that"** : Claude に変更を元に戻すよう依頼します。
- **/clear** : 関連のないタスク間でコンテキストをリセットします。関連のないコンテキストを持つ長いセッションはパフォーマンスを低下させる可能性があります。

1つのセッションで同じ問題について Claude を 2 回以上修正した場合、コンテキストは失敗したアプローチで乱雑です。 `/clear` を実行し、学んだことを組み込んだより具体的なプロンプトで新しく開始します。より良いプロンプトを持つクリーンなセッションは、ほぼ常に蓄積された修正を持つ長いセッションを上回ります。

コンテキストを積極的に管理する

Tip:

関連のないタスク間で `/clear` を実行してコンテキストをリセットします。

Claude Code はコンテキスト制限に近づく则会話履歴を自動的にコンパクトにします。これにより、重要なコードと決定を保持しながらスペースを解放します。

長いセッション中に、Claude のコンテキストウィンドウは関連のない会話、ファイルの内容、コマンドで満杯になる可能性があります。これはパフォーマンスを低下させ、時々 Claude を気を散らすことができます。

- ・タスク間で頻繁に `/clear` を使用してコンテキストウィンドウを完全にリセットします
- ・自動コンパクト化がトリガーされると、Claude は最も重要なもの（コードパターン、ファイル状態、主要な決定を含む）を要約します
- ・より多くの制御のために、`/compact <instructions>` を実行します。例えば `/compact Focus on the API changes`
- ・会話の一部のみをコンパクトするには、`Esc + Esc` または `/rewind` を使用し、メッセージチェックポイントを選択し、**ここから要約**を選択します。これにより、そのポイント以降のメッセージが凝縮され、以前のコンテキストは保持されます。
- ・CLAUDE.md でコンパクト化動作をカスタマイズします。 `"When compacting, always preserve the full list of modified files and any test commands"` のような指示を使用して、重要なコンテキストが要約を生き残ることを確認します
- ・会話履歴に入らない簡単な質問については、`/btw` を使用します。答えは却下可能なオーバーレイに表示され、会話履歴に入らないため、コンテキストを増やさずに詳細をチェックできます。

調査にサブエージェントを使用する

Tip:

`"use subagents to investigate X"` で研究を委譲します。彼らは別のコンテキストで探索し、実装のためにメインの会話をクリーンに保ちます。

コンテキストが基本的な制約であるため、サブエージェントは利用可能な最も強力なツールの 1 つです。Claude がコードベースを研究するとき、多くのファイルを読み取り、すべてがコンテキストを消費します。サブエージェントは別のコンテキストウィンドウで実行され、要約を報告します。

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

サブエージェントはコードベースを探索し、関連するファイルを読み取り、メインの会話を乱さずにすべての調査結果を報告します。

Claude が何かを実装した後、検証にサブエージェントを使用することもできます。

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

チェックポイントで巻き戻す

Tip:

Claude が行うすべてのアクションはチェックポイントを作成します。以前のチェックポイントに会話、コード、またはその両方を復元できます。

Claude は変更前に自動的にチェックポイントを作成します。**Escape** をダブルタップするか **/rewind** を実行して、巻き戻しメニューを開きます。会話のみを復元したり、コードのみを復元したり、両方を復元したり、選択したメッセージから要約したりできます。詳細については、[チェックポイント](#)を参照してください。

すべての動きを慎重に計画する代わりに、Claude に何か危険なことを試すよう指示できます。うまくいかない場合は、巻き戻して別のアプローチを試してください。チェックポイントはセッション全体で保持されるため、ターミナルを閉じて後もで巻き戻すことができます。

Warning:

チェックポイントは Claude が行った変更のみを追跡します。外部プロセスではありません。これは git の代替ではありません。

会話を再開する

Tip:

`claude --continue` を実行して中断したところから再開するか、`--resume` を使用して最近のセッションから選択します。

Claude Code は会話をローカルに保存します。タスクが複数のセッションにまたがる場合、コンテキストを再度説明する必要はありません。

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

`/rename` を使用してセッションに `"oauth-migration"` や `"debugging-memory-leak"` などの説明的な名前を付けて、後で見つけやすくします。セッションをブランチのように扱う：異なるワークストリームは別々の永続的なコンテキストを持つことができます。

自動化とスケール

1つの Claude で効果的になったら、並列セッション、非対話型モード、ファンアウトパターンで出力を乗算します。

これまでのすべては、1人の人間、1つの Claude、1つの会話を想定しています。しかし、Claude Code は水平にスケールします。このセクションのテクニックは、より多くのことを成し遂げる方法を示しています。

非対話型モードを実行する

Tip:

CI、プリコミットフック、またはスクリプトで `claude -p "prompt"` を使用します。ストリーミング JSON 出力の場合は `--output-format stream-json` を追加します。

`claude -p "your prompt"` を使用すると、セッションなしで Claude を非対話的に実行できます。非対話型モードは、Claude を CI パイプライン、プリコミットフック、または自動化されたワークフローに統合する方法です。出力形式を使用すると、結果をプログラムで解析できます。プレーンテキスト、JSON、またはストリーミング 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
```

複数の Claude セッションを実行する

Tip:

複数の Claude セッションを並列で実行して、開発を高速化し、分離された実験を実行するか、複雑なワークフローを開始します。

並列セッションを実行するには 3 つの主な方法があります。

- [Claude Code デスクトップアプリ](#)：複数のローカルセッションを視覚的に管理します。各セッションは独自の分離されたワークツリーを取得します。
- [Web 上の Claude Code](#)：Anthropic のセキュアなクラウドインフラストラクチャで分離された VM で実行します。
- [エージェントチーム](#)：共有タスク、メッセージング、チームリーダーを備えた複数のセッションの自動調整。

作業を並列化することを超えて、複数のセッションは品質に焦点を当てたワークフローを有効にします。新しいコンテキストは、Claude がちょうど書いたコードに偏らないため、コードレビューを改善します。

例えば、Writer/Reviewer パターンを使用します。

セッション A (ライター)	セッション B (レビューア)
Implement a rate limiter for our API endpoints	

セッション A (ライター)	セッション B (レビューアー)
	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.	

テストで同様のことを行うことができます。1 つの Claude にテストを書かせ、別の Claude にそれらを渡すコードを書かせます。

ファイル全体にファンアウトする

Tip:

各タスクに対して `claude -p` を呼び出すループを実行します。バッチ操作のスコープパーミッションに `--allowedTools` を使用します。

大規模な移行または分析の場合、多くの並列 Claude 呼び出し全体で作業を配布できます。

Step 1: タスクリストを生成する

Claude に移行が必要なすべてのファイルをリストさせます（例えば、`list all 2,000 Python files that need migrating`）

Step 2: リストをループするスクリプトを書く

```
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: いくつかのファイルでテストしてから、スケールで実行する

最初の 2~3 ファイルで何が悪いかに基づいてプロンプトを改善し、完全なセットで実行します。 `--allowedTools` フラグは Claude が何ができるかを制限します。これは無人で実行しているときに重要です。

Claude を既存のデータ/処理パイプラインに統合することもできます。

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

開発中は `--verbose` を使用し、本番環境ではオフにします。

一般的な失敗パターンを避ける

これらは一般的な間違いです。早期に認識することで時間を節約できます。

- **キッチンシンクセッション**。1つのタスクで開始し、関連のないことを Claude に尋ねてから、最初のタスクに戻ります。コンテキストは関連のない情報でいっぱいです。> **修正**：関連のないタスク間で `/clear` を実行します。
- **何度も修正する**。Claude が何か間違ったことをし、修正し、まだ間違っています。修正します。コンテキストは失敗したアプローチで乱雑です。> **修正**：2 回の失敗した修正の後、`/clear` を実行し、学んだことを組み込んだより良い初期プロンプトを書きます。
- **過度に指定された CLAUDE.md**。CLAUDE.md が長すぎる場合、Claude は重要なルールがノイズに失われるため、半分を無視します。> **修正**：容赦なく削除します。Claude が指示なしで既に何かを正しく行う場合、削除するか、フックに変換します。
- **信頼してから検証するギャップ**。Claude はもっともらしく見える実装を生成しますが、エッジケースを処理しません。> **修正**：常に検証を提供します（テスト、スクリプト、スクリーンショット）。検証できない場合は、出荷しないでください。
- **無限探索**。スコープなしで何かを「調査」するよう Claude に依頼します。Claude は数百のファイルを読み取り、コンテキストを満たします。> **修正**：調査を狭くスコープするか、サブエージェントを使用して、探索がメインコンテキストを消費しないようにします。

直感を開発する

このガイドのパターンは固定されていません。それらはすべての状況で一般的にうまく機能する出発点ですが、すべての状況に最適ではない可能性があります。

時々、あなたは1つの複雑な問題に深く入り込んでいて、履歴が価値があるため、コンテキストを蓄積させるべきです。時々、タスクが探索的であるため、計画をスキップして Claude にそれを理解させるべきです。時々、曖昧なプロンプトは、Claude が問題をどのように解釈するかを制約する前に見たいため、正確です。

何が機能するかに注意を払います。Claude が素晴らしい出力を生成するとき、あなたが何をしたかに注意してください。プロンプト構造、提供したコンテキスト、あなたがいたモード。Claude が苦勞するとき、なぜ尋ねてください。コンテキストがノイズが多すぎましたか？プロンプトが曖昧すぎましたか？タスクが1回のパスには大きすぎましたか？

時間をかけて、ガイドが捉えることができない直感を開発します。具体的にすべき時と開放的にすべき時、計画すべき時と探索すべき時、コンテキストをクリアすべき時と蓄積させるべき時を知ります。

関連リソース

- [Claude Code の仕組み](#)：agentic ループ、ツール、コンテキスト管理
- [Claude Code を拡張](#)：スキル、フック、MCP、サブエージェント、プラグイン
- [一般的なワークフロー](#)：デバッグ、テスト、PR などのステップバイステップレシピ
- [CLAUDE.md](#)：プロジェクト規約と永続的なコンテキストを保存する

一般的なワークフロー

Claude Code を使用してコードベースの探索、バグ修正、リファクタリング、テスト、その他の日常的なタスクを実行するためのステップバイステップガイド。

このページでは、日常的な開発のための実践的なワークフローについて説明します。未知のコードの探索、デバッグ、リファクタリング、テストの作成、PR の作成、セッションの管理などです。各セクションには、自分のプロジェクトに適応させることができるプロンプトの例が含まれています。より高度なパターンとヒントについては、[ベストプラクティス](#)を参照してください。

新しいコードベースを理解する

コードベースの概要を素早く把握する

新しいプロジェクトに参加したばかりで、その構造を素早く理解する必要があるとします。

Step 1: プロジェクトのルートディレクトリに移動する

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

Step 2: Claude Code を起動する

```
claude
```

Step 3: 高レベルの概要を要求する

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

Step 4: 特定のコンポーネントについてさらに詳しく調べる

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

what are the key data models?

how is authentication handled?

Tip:

ヒント：

- 広い質問から始めて、特定の領域に絞り込んでいく
- プロジェクトで使用されているコーディング規約とパターンについて質問する
- プロジェクト固有の用語の用語集をリクエストする

関連するコードを見つける

特定の機能または機能に関連するコードを見つける必要があるとします。

Step 1: Claude に関連ファイルを見つけるよう依頼する

find the files that handle user authentication

Step 2: コンポーネントがどのように相互作用するかについてのコンテキストを取得する

how do these authentication files work together?

Step 3: 実行フローを理解する

trace the login process from front-end to database

Tip:

ヒント：

- 探しているものについて具体的に説明する
- プロジェクトのドメイン言語を使用する
- 言語の [コード インテリジェンス プラグイン](#) をインストールして、Claude に正確な '定義に移動' と '参照を検索' のナビゲーションを提供する

バグを効率的に修正する

エラーメッセージが表示され、そのソースを見つけて修正する必要があるとします。

Step 1: Claude とエラーを共有する

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

Step 2: 修正の推奨事項を要求する

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

Step 3: 修正を適用する

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

Tip:

ヒント:

- Claude に問題を再現するコマンドとスタックトレースを伝える
- エラーを再現するための手順を記載する
- エラーが断続的か一貫しているかを Claude に知らせる

コードをリファクタリングする

古いコードを最新のパターンとプラクティスを使用するように更新する必要があるとします。

Step 1: リファクタリング対象のレガシーコードを特定する

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

Step 2: リファクタリングの推奨事項を取得する

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

Step 3: 変更を安全に適用する


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

Step 4: リファクタリングを検証する

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

Tip:

ヒント：

- Claude に最新のアプローチの利点を説明するよう依頼する
- 必要に応じて、変更が後方互換性を維持することをリクエストする
- リファクタリングを小さくテスト可能な増分で実行する

特化した subagent を使用する

特定のタスクをより効果的に処理するために、特化した AI subagent を使用したいとします。

Step 1: 利用可能な subagent を表示する

```
/agents
```

これにより、利用可能なすべての subagent が表示され、新しいものを作成できます。

Step 2: subagent を自動的に使用する

Claude Code は自動的に適切なタスクを特化した subagent に委譲します：

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

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

Step 3: 特定の subagent を明示的にリクエストする

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

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

Step 4: ワークフロー用のカスタム subagent を作成する

```
/agents
```

次に「Create New subagent」を選択し、プロンプトに従って以下を定義します：

- subagent の目的を説明する一意の識別子（例：`code-reviewer`、`api-designer`）。
- Claude がこのエージェントを使用する場合
- アクセスできるツール
- エージェントの役割と動作を説明するシステムプロンプト

Tip:

ヒント：

- チーム共有用に `.claude/agents/` にプロジェクト固有の subagent を作成する
- 自動委譲を有効にするために説明的な `description` フィールドを使用する
- ツールアクセスを各 subagent が実際に必要なものに制限する
- 詳細な例については、[subagents ドキュメント](#)を確認する

Plan Mode を使用して安全なコード分析を行う

Plan Mode は Claude に読み取り専用操作でコードベースを分析することで計画を作成するよう指示します。これはコードベースの探索、複雑な変更の計画、またはコードの安全なレビューに最適です。Plan Mode では、Claude は `AskUserQuestion` を使用して要件を収集し、計画を提案する前に目標を明確にします。

Plan Mode を使用する場合

- **マルチステップの実装**：機能が多くのファイルへの編集を必要とする場合
- **コード探索**：何かを変更する前にコードベースを徹底的に調査したい場合
- **インタラクティブな開発**：Claude との方向性について反復したい場合

Plan Mode の使用方法

セッション中に Plan Mode をオンにする

Shift+Tab を使用してセッション中に Plan Mode に切り替えることができます。

Normal Mode にいる場合、**Shift+Tab** は最初に Auto-Accept Mode に切り替わります。これはターミナルの下部に ▶ accept edits on で示されます。その後の **Shift+Tab** は Plan Mode に切り替わります。これは || plan mode on で示されます。

Plan Mode で新しいセッションを開始する

Plan Mode で新しいセッションを開始するには、`--permission-mode plan` フラグを使用します：

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

Plan Mode で「ヘッドレス」クエリを実行する

`-p` を使用して Plan Mode でクエリを直接実行することもできます（つまり、[「ヘッドレスモード」](#)で）：

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

例：複雑なリファクタリングの計画

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

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

Claude は現在の実装を分析し、包括的な計画を作成します。フォローアップで改善します：

```
What about backward compatibility?
```

How should we handle database migration?

Tip:

Ctrl+G を押して計画をデフォルトのテキストエディタで開き、Claude が続行する前に直接編集できます。

Plan Mode をデフォルトとして設定する

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

詳細な設定オプションについては、[設定ドキュメント](#)を参照してください。

テストを使用する

カバーされていないコードのテストを追加する必要があるとします。

Step 1: テストされていないコードを特定する

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

Step 2: テストスキャフォールディングを生成する

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

Step 3: 意味のあるテストケースを追加する

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

Step 4: テストを実行して検証する

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

Claude は、プロジェクトの既存のパターンと規約に従うテストを生成できます。テストをリクエストするときは、検証したい動作について具体的に説明してください。Claude は既存のテストファイルを調べて、既に使用されているスタイル、フレームワーク、アサーションパターンに一致させます。

包括的なカバレッジのために、Claude に見落とした可能性のあるエッジケースを特定するよう依頼してください。Claude はコードパスを分析し、エラー条件、境界値、見落としやすい予期しない入力のテストを提案できます。

プルリクエストを作成する

Claude に直接プルリクエストを作成するよう依頼することができます（「create a pr for my changes」）。または、ステップバイステップで Claude をガイドすることができます：

Step 1: 変更内容を要約する

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

Step 2: プルリクエストを生成する

```
create a pr
```

Step 3: レビューと改善

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

`gh pr create` を使用して PR を作成すると、セッションはその PR に自動的にリンクされます。後で `claude --from-pr <number>` で再開できます。

Tip:

Claude が生成した PR を送信する前にレビューし、Claude に潜在的なリスクや考慮事項を強調するよう依頼してください。

ドキュメントを処理する

コードのドキュメントを追加または更新する必要があります。

Step 1: ドキュメント化されていないコードを特定する

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

Step 2: ドキュメントを生成する

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

Step 3: レビューと改善

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

Step 4: ドキュメントを検証する

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

Tip:

ヒント：

- 必要なドキュメントスタイル（JSDoc、docstring など）を指定する
- ドキュメント内の例をリクエストする
- パブリック API、インターフェース、複雑なロジックのドキュメントをリクエストする

画像を使用する

コードベース内の画像を使用する必要があり、Claude の画像コンテンツ分析を支援したいとします。

Step 1: 会話に画像を追加する

次のいずれかの方法を使用できます：

1. Claude Code ウィンドウに画像をドラッグアンドドロップする
2. 画像をコピーして、CLI に ctrl+v で貼り付ける（cmd+v は使用しないでください）

3. Claude に画像パスを提供する。例：「Analyze this image: /path/to/your/image.png」

Step 2: Claude に画像を分析するよう依頼する

What does this image show?

Describe the UI elements in this screenshot

Are there any problematic elements in this diagram?

Step 3: コンテキストに画像を使用する

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: ビジュアルコンテンツからコード提案を取得する

Generate CSS to match this design mockup

What HTML structure would recreate this component?

Tip:

ヒント：

- テキスト説明が不明確または面倒な場合は画像を使用する
- より良いコンテキストのために、エラー、UI デザイン、図のスクリーンショットを含める
- 会話で複数の画像を使用できます
- 画像分析は図、スクリーンショット、モックアップなどで機能します
- Claude が画像を参照する場合（例：[Image #1]）、**Cmd+Click** (Mac) または **Ctrl+Click** (Windows/Linux) リンクをクリックして、デフォルトビューアで画像を開きます

ファイルとディレクトリを参照する

@ を使用して、Claude に読み込まれるのを待たずにファイルまたはディレクトリをすばやく含めます。

Step 1: 単一ファイルを参照する

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

これにより、ファイルの完全な内容が会話に含まれます。

Step 2: ディレクトリを参照する

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

これにより、ファイル情報を含むディレクトリリストが提供されます。

Step 3: MCP リソースを参照する

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

これにより、@server:resource 形式を使用して接続された MCP サーバーからデータを取得します。詳細については、[MCP リソース](#)を参照してください。

Tip:

ヒント：

- ・ ファイルパスは相対パスまたは絶対パスにできます
- ・ @ ファイル参照は、ファイルのディレクトリと親ディレクトリに **CLAUDE.md** を追加してコンテキストに含めます
- ・ ディレクトリ参照はコンテンツではなくファイルリストを表示します
- ・ 単一のメッセージで複数のファイルを参照できます（例：「@file1.js and @file2.js」）

拡張思考（思考モード）を使用する

[拡張思考](#)はデフォルトで有効になっており、Claude が複雑な問題をステップバイステップで推論するためのスペースを提供します。この推論は詳細モードで表示されます。詳細モードは **Ctrl+0** でトグルできます。

さらに、Opus 4.6 は適応的推論を導入しています。固定された思考トークン予算の代わりに、モデルは[努力レベル](#)設定に基づいて思考を動的に割り当てます。拡張思考と適応的推論は一緒に機能して、Claude が応答する前にどの程度深く推論するかを制御できます。

拡張思考は、複雑なアーキテクチャの決定、難しいバグ、マルチステップの実装計画、異なるアプローチ間のトレードオフの評価に特に価値があります。

Note:

「think」、「think hard」、「think more」などのフレーズは通常のプロンプト指示として解釈され、思考トークンを割り当てません。

思考モードを設定する

思考はデフォルトで有効になっていますが、調整または無効にすることができます。

スコープ	設定方法	詳細
努力レベル	<code>/model</code> で調整するか、 <code>CLAUDE_CODE_EFFORT_LEVEL</code> を設定する	Opus 4.6 と Sonnet 4.6 の思考の深さを制御します：低、中、高。 努力レベルを調整 を参照してください
<code>ultrathink</code> キーワード	プロンプトの任意の場所に 「ultrathink」を含める	Opus 4.6 と Sonnet 4.6 でそのターンの努力を高に設定します。努力設定を永続的に変更せずに深い推論が必要な 1 回限りのタスクに便利です
トグルショートカット	<code>Option+T</code> (macOS) または <code>Alt+T</code> (Windows/Linux) を押す	現在のセッションの思考をオン/オフに切り替えます（すべてのモデル）。 ターミナル設定 を有効にして Option キーショートカットを有効にする必要がある場合があります

スコープ	設定方法	詳細
グローバルデフォルト	<code>/config</code> を使用して思考モードをトグルする	すべてのプロジェクト（すべてのモデル）でデフォルトを設定します。 <code>~/.claude/settings.json</code> に <code>alwaysThinkingEnabled</code> として保存されます
トークン予算を制限する	<code>MAX_THINKING_TOKENS</code> 環境変数を設定する	思考予算を特定のトークン数に制限します（Opus 4.6 では 0 に設定されていない限り無視されます）。 例： <code>export MAX_THINKING_TOKENS=10000</code>

Claude の思考プロセスを表示するには、`Ctrl+0` を押して詳細モードをトグルし、内部推論をグレーのイタリック体で表示します。

拡張思考の仕組み

拡張思考は、Claude が応答する前に実行する内部推論の量を制御します。より多くの思考により、ソリューションを探索し、エッジケースを分析し、間違いを自己修正するためのより多くのスペースが提供されます。

Opus 4.6 では、思考は適応的推論を使用します。モデルは、選択した[努力レベル](#)（低、中、高）に基づいて思考トークンを動的に割り当てます。これは速度と推論の深さのトレードオフを調整するための推奨される方法です。

他のモデルでは、思考は出力予算から最大 31,999 トークンの固定予算を使用します。

`MAX_THINKING_TOKENS` 環境変数でこれを制限するか、`/config` または `Option+T` / `Alt+T` トグルで思考を完全に無効にすることができます。

`MAX_THINKING_TOKENS` は Opus 4.6 と Sonnet 4.6 では無視されます。適応的推論が思考の深さを制御するためです。1 つの例外： `MAX_THINKING_TOKENS=0` を設定すると、任意のモデルで思考が完全に無効になります。適応的思考を無効にして固定思考予算に戻すには、`CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1` を設定します。[環境変数](#)を参照してください。

Warning:

Claude 4 モデルが要約された思考を表示していても、使用されたすべての思考トークンに対して課金されます

以前の会話を再開する

Claude Code を起動するときは、以前のセッションを再開できます：

- `claude --continue` は現在のディレクトリで最新の会話を続行します
- `claude --resume` は会話ピッカーを開くか、名前で再開します
- `claude --from-pr 123` は特定のプルリクエストにリンクされたセッションを再開します

アクティブなセッション内から、`/resume` を使用して別の会話に切り替えます。

セッションはプロジェクトディレクトリごとに保存されます。`/resume` ピッカーは同じ git リポジトリからのセッションを表示します。これには `worktree` も含まれます。

セッションに名前を付ける

セッションに説明的な名前を付けて、後で見つけやすくします。これは複数のタスクまたは機能に取り組むときのベストプラクティスです。

Step 1: 現在のセッションに名前を付ける

セッション中に `/rename` を使用して、覚えやすい名前を付けます：

```
/rename auth-refactor
```

ピッカーからセッションの名前を変更することもできます。`/resume` を実行し、セッションに移動して、`R` を押します。

Step 2: 後で名前で再開する

コマンドラインから：

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

またはアクティブなセッション内から：

```
/resume auth-refactor
```

セッションピッカーを使用する

`/resume` コマンド（または引数なしの `claude --resume`）は、次の機能を備えたインタラクティブセッションピッカーを開きます：

ピッカーのキーボードショートカット：

ショートカット	アクション
↑ / ↓	セッション間を移動する
→ / ←	グループ化されたセッションを展開または折りたたむ
Enter	ハイライトされたセッションを選択して再開する
P	セッションコンテンツをプレビューする
R	ハイライトされたセッションの名前を変更する
/	検索してセッションをフィルタリングする
A	現在のディレクトリとすべてのプロジェクト間を切り替える
B	現在の git ブランチからのセッションにフィルタリングする
Esc	ピッカーまたは検索モードを終了する

セッション組織：

ピッカーは有用なメタデータを含むセッションを表示します：

- ・セッション名または初期プロンプト
- ・最後のアクティビティからの経過時間
- ・メッセージ数
- ・Git ブランチ（該当する場合）

フォークされたセッション（`/rewind` または `--fork-session` で作成）はルートセッションの下にグループ化されており、関連する会話を見つけやすくなっています。

Tip:

ヒント：

- ・ **セッションに早期に名前を付ける**：異なるタスクで作業を開始するときに `/rename` を使用します。後で「payment-integration」を見つける方が「explain this function」よりもはるかに簡単です

- 現在のディレクトリで最新の会話にすばやくアクセスするには `--continue` を使用します
- 必要なセッションがわかっている場合は `--resume session-name` を使用します
- 参照して選択する必要がある場合は `--resume` (名前なし) を使用します
- スクリプトの場合は、`claude --continue --print "prompt"` を使用して非対話モードで再開します
- ピッカーで **P** を押して、セッションを再開する前にプレビューします
- 再開されたセッションは、元のセッションと同じモデルと設定で開始されます

仕組み：

1. **会話ストレージ**：すべての会話は完全なメッセージ履歴とともにローカルに自動保存されます
2. **メッセージ逆シリアル化**：再開時に、コンテキストを維持するために全メッセージ履歴が復元されます
3. **ツール状態**：前の会話からのツール使用と結果が保持されます
4. **コンテキスト復元**：会話は以前のすべてのコンテキストを保持して再開されます

Git worktree を使用して並列 Claude Code セッションを実行する

複数のタスクに同時に取り組む場合、各 Claude セッションがコードベースの独自のコピーを持つ必要があります。そうしないと、変更が衝突します。Git worktree は、それぞれが独自のファイルとブランチを持つ別の作業ディレクトリを作成することで、これを解決します。同じリポジトリ履歴とリモート接続を共有しながら。つまり、Claude が 1 つの worktree で機能に取り組んでいる間に、別の worktree でバグを修正でき、どちらのセッションも相互に干渉しません。

`--worktree` (`-w`) フラグを使用して、分離された worktree を作成し、Claude をその中で起動します。渡す値は worktree ディレクトリ名とブランチ名になります：

```
## "feature-auth" という名前の worktree で Claude を起動する
## .claude/worktrees/feature-auth/ を新しいブランチで作成する
claude --worktree feature-auth

## 別の worktree で別のセッションを開始する
claude --worktree bugfix-123
```

名前を省略すると、Claude は自動的にランダムな名前を生成します：

```
## "bright-running-fox" のような名前を自動生成する
claude --worktree
```

Worktree は `<repo>/claude/worktrees/<name>` に作成され、デフォルトのリモートブランチから分岐します。worktree ブランチは `worktree-<name>` という名前が付けられます。

セッション中に Claude に「work in a worktree」または「start a worktree」を依頼することもでき、自動的に作成されます。

Subagent worktree

Subagent は worktree 分離を使用して、競合なしに並列で作業することもできます。Claude に「use worktrees for your agents」を依頼するか、[カスタム subagent](#) で `isolation: worktree` をエージェントの frontmatter に追加して設定します。各 subagent は独自の worktree を取得し、変更なしで subagent が終了すると自動的にクリーンアップされます。

Worktree クリーンアップ

worktree セッションを終了すると、Claude は変更があったかどうかに基づいてクリーンアップを処理します：

- ・ **変更なし**：worktree とそのブランチは自動的に削除されます
- ・ **変更またはコミットが存在する**：Claude は worktree を保持するか削除するかを促します。保持すると、ディレクトリとブランチが保存されるため、後で戻ることができます。削除すると、worktree ディレクトリとそのブランチが削除され、すべてのコミットされていない変更とコミットが破棄されます

Claude セッション外で worktree をクリーンアップするには、[worktree を手動で管理](#)を使用します。

Tip:

`.claude/worktrees/` を `.gitignore` に追加して、worktree コンテンツがメインリポジトリに追跡されていないファイルとして表示されるのを防ぎます。

Worktree を手動で管理する

worktree の場所とブランチ設定をより細かく制御するには、Git を使用して worktree を直接作成します。これは、特定の既存ブランチをチェックアウトするか、worktree をリポジトリの外に配置する必要がある場合に便利です。

```
## 新しいブランチで worktree を作成する
git worktree add ../project-feature-a -b feature-a

## 既存のブランチで worktree を作成する
git worktree add ../project-bugfix bugfix-123

## worktree で Claude を起動する
cd ../project-feature-a && claude

## 完了したらクリーンアップする
git worktree list
git worktree remove ../project-feature-a
```

詳細については、[公式 Git worktree ドキュメント](#)を参照してください。

Tip:

各新しい worktree でプロジェクトのセットアップに従って開発環境を初期化することを忘れないでください。スタックに応じて、これには依存関係のインストール（`npm install`、`yarn`）、仮想環境のセットアップ、またはプロジェクトの標準セットアッププロセスに従うことが含まれる場合があります。

Git 以外のバージョン管理

Worktree 分離はデフォルトで git で機能します。SVN、Perforce、Mercurial などの他のバージョン管理システムの場合は、[WorktreeCreate](#) と [WorktreeRemove フック](#)を設定して、カスタム worktree 作成とクリーンアップロジックを提供します。設定されている場合、これらのフックは `--worktree` を使用するときデフォルトの git 動作を置き換えます。

共有タスクとメッセージングを使用した並列セッションの自動調整については、[エージェント チーム](#)を参照してください。

Claude が注意を必要とするときに通知を受け取る

長時間実行されるタスクを開始して別のウィンドウに切り替えるときは、Claude が終了したときまたは入力が必要なときに知ることができるようにデスクトップ通知を設定できます。これは **Notification フックイベント**を使用します。これは、Claude が許可を待っている、アイドル状態で新しいプロンプトの準備ができている、または認証を完了しているときはいつでも発火します。

Step 1: フックメニューを開く

`/hooks` と入力し、イベントのリストから **Notification** を選択します。

Step 2: マッチャーを設定する

+ Match all (no filter) を選択して、すべての通知タイプで発火させます。特定のイベントのみに通知するには、**+ Add new matcher...** を選択して、次のいずれかの値を入力します：

マッチャー	発火する場合
<code>permission_prompt</code>	Claude がツール使用を承認する必要がある場合
<code>idle_prompt</code>	Claude が完了し、次のプロンプトを待っている場合
<code>auth_success</code>	認証が完了する場合
<code>elicitation_dialog</code>	Claude があなたに質問している場合

Step 3: 通知コマンドを追加する

+ Add new hook... を選択して、OS のコマンドを入力します：

macOS

`osascript` を使用して AppleScript を通じてネイティブ macOS 通知をトリガーします：

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

Linux

ほとんどの Linux デスクトップに通知デーモンがプリインストールされている `notify-send` を使用します：


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

Windows (PowerShell)

PowerShell を使用して .NET の Windows Forms を通じてネイティブメッセージボックスを表示します：

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

Step 4: ユーザー設定に保存する

User settings を選択して、すべてのプロジェクトに通知を適用します。

JSON 設定例を含む完全なウォークスルーについては、[フックを使用してワークフローを自動化する](#)を参照してください。完全なイベント スキーマと通知タイプについては、[通知リファレンス](#)を参照してください。

Claude を unix スタイルのユーティリティとして使用する

検証プロセスに Claude を追加する

Claude Code をリンターまたはコードレビューアーとして使用したいとします。

ビルドスクリプトに Claude を追加する：

```
// 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:

ヒント:

- CI/CD パイプラインで自動コードレビューに Claude を使用する
- プロンプトをカスタマイズして、プロジェクトに関連する特定の問題をチェックする
- 異なるタイプの検証用に複数のスクリプトを作成することを検討する

パイプイン、パイプアウト

Claude にデータをパイプインし、構造化された形式でデータを取得したいとします。

Claude を通じてデータをパイプする:

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

Tip:

ヒント:

- パイプを使用して Claude を既存のシェルスクリプトに統合する
- 他の Unix ツールと組み合わせて強力なワークフローを作成する
- 構造化出力に `-output-format` を使用することを検討する

出力形式を制御する

特に Claude Code をスクリプトまたは他のツールに統合する場合、Claude の出力が特定の形式である必要があるとします。

Step 1: テキスト形式を使用する（デフォルト）

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

これは Claude のプレーンテキスト応答のみを出力します（デフォルトの動作）。

Step 2: JSON 形式を使用する

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

これは、コストと期間を含むメタデータを含むメッセージの JSON 配列を出力します。

Step 3: ストリーミング JSON 形式を使用する

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

これは、Claude がリクエストを処理するときにリアルタイムで JSON オブジェクトの一連を出力します。各メッセージは有効な JSON オブジェクトですが、連結された場合、全体の出力は有効な JSON ではありません。

Tip:

ヒント：

- Claude の応答だけが必要な単純な統合には `--output-format text` を使用する
- 完全な会話ログが必要な場合は `--output-format json` を使用する
- 各会話ターンのリアルタイム出力には `--output-format stream-json` を使用する

Claude の機能について質問する

Claude は独自の機能と制限に関する質問に答えることができるドキュメントへの組み込みアクセスを持っています。

質問例

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 はこれらの質問に対してドキュメントベースの回答を提供します。実行可能な例と実践的なデモンストレーションについては、上記の特定のワークフローセクションを参照してください。

Tip:

ヒント：

- Claude は、使用しているバージョンに関係なく、常に最新の Claude Code ドキュメントにアクセスできます
- 詳細な回答を得るために具体的な質問をする
- Claude は、MCP 統合、エンタープライズ設定、高度なワークフローなどの複雑な機能を説明できます

次のステップ

- [ベストプラクティス](#) Claude Code から最大限の価値を得るためのパターン

- [Claude Code の仕組み](#) agentic ループとコンテキスト管理を理解する
- [Claude Code を拡張する](#) skill、フック、MCP、subagent、プラグインを追加する
- [リファレンス実装](#) 開発コンテナリファレンス実装をクローンする

Part 3: Commands & Reference

CLI リファレンス

Claude Code コマンドラインインターフェースの完全なリファレンス。コマンドとフラグを含みます。

CLI コマンド

これらのコマンドを使用して、セッションを開始し、コンテンツをパイプし、会話を再開し、更新を管理できます。

コマンド	説明	例
<code>claude</code>	インタラクティブセッションを開始	<code>claude</code>
<code>claude "query"</code>	初期プロンプト付きでインタラクティブセッションを開始	<code>claude "explain this project"</code>
<code>claude -p "query"</code>	SDK 経由でクエリを実行してから終了	<code>claude -p "explain this function"</code>
<code>cat file claude -p "query"</code>	パイプされたコンテンツを処理	<code>cat logs.txt claude -p "explain"</code>
<code>claude -c</code>	現在のディレクトリで最新の会話を続行	<code>claude -c</code>
<code>claude -c -p "query"</code>	SDK 経由で続行	<code>claude -c -p "Check for type errors"</code>
<code>claude -r "<session>" "query"</code>	ID または名前でセッションを再開	<code>claude -r "auth-refactor" "Finish this PR"</code>
<code>claude update</code>	最新バージョンに更新	<code>claude update</code>

コマンド	説明	例
<code>claude auth login</code>	Anthropic アカウントにサインイン。 <code>--email</code> を使用してメールアドレスを事前入力し、 <code>--sso</code> を使用して SSO 認証を強制します	<code>claude auth login --email user@example.com --sso</code>
<code>claude auth logout</code>	Anthropic アカウントからログアウト	<code>claude auth logout</code>
<code>claude auth status</code>	認証ステータスを JSON として表示。 <code>--text</code> を使用して人間が読める形式で表示。ログイン済みの場合はコード 0 で終了、ログインしていない場合は 1 で終了	<code>claude auth status</code>
<code>claude agents</code>	すべての設定済み subagents をソース別にグループ化して一覧表示	<code>claude agents</code>
<code>claude mcp</code>	Model Context Protocol (MCP) サーバーを設定	Claude Code MCP ドキュメント を参照してください。
<code>claude remote-control</code>	Remote Control セッション を開始して、ローカルで実行中の Claude Code を Claude.ai または Claude アプリから制御します。フラグについては Remote Control を参照してください	<code>claude remote-control</code>

CLI フラグ

これらのコマンドラインフラグを使用して Claude Code の動作をカスタマイズします。

フラグ	説明	例
<code>--add-dir</code>	Claude がアクセスできる追加の作業ディレクトリを追加します (各パスがディレクトリとして存在することを検証)	<code>claude --add-dir ../apps ../lib</code>
<code>--agent</code>	現在のセッションのエージェントを指定します (<code>agent</code> 設定をオーバーライド)	<code>claude --agent my-custom-agent</code>

フラグ	説明	例
<code>--agents</code>	カスタム subagents を JSON 経由で動的に定義します（形式については以下を参照）	<pre>claude --agents '{"reviewer": {"description": "Reviews code", "prompt": "You are a code reviewer"}}'</pre>
<code>--allow-dangerously-skip-permissions</code>	権限バイパスをオプションとして有効にします（すぐには有効化しません）。 <code>--permission-mode</code> との組み合わせを許可します（注意して使用）	<pre>claude --permission- mode plan --allow- dangerously-skip- permissions</pre>
<code>--allowedTools</code>	権限を求めずに実行するツール。パターンマッチングについては 権限ルール構文 を参照してください。利用可能なツールを制限するには、代わりに <code>--tools</code> を使用してください	<pre>"Bash(git log *)" "Bash(git diff *)" "Read"</pre>
<code>--append-system-prompt</code>	カスタムテキストをデフォルトシステムプロンプトの末尾に追加	<pre>claude --append- system-prompt "Always use TypeScript"</pre>
<code>--append-system-prompt-file</code>	ファイルから追加のシステムプロンプトテキストを読み込み、デフォルトプロンプトに追加	<pre>claude --append- system-prompt-file ./ extra-rules.txt</pre>
<code>--betas</code>	API リクエストに含めるベータヘッダー（API キーユーザーのみ）	<pre>claude --betas interleaved-thinking</pre>
<code>--chrome</code>	Web 自動化とテストのための Chrome ブラウザ統合 を有効化	<pre>claude --chrome</pre>
<code>--continue</code> , <code>-c</code>	現在のディレクトリで最新の会話を読み込む	<pre>claude --continue</pre>
<code>--dangerously-skip-permissions</code>	すべての権限プロンプトをスキップします（注意して使用）	<pre>claude --dangerously- skip-permissions</pre>

フラグ	説明	例
<code>--debug</code>	デバッグモードを有効化し、オプションのカテゴリフィルタリング（例：" <code>api,hooks</code> " または " <code>!statsig,!file</code> "）	<code>claude --debug "api,mcp"</code>
<code>--disable-slash-commands</code>	このセッションのすべてのスキルとコマンドを無効化	<code>claude --disable-slash-commands</code>
<code>--disallowedTools</code>	モデルのコンテキストから削除され、使用できないツール	<code>"Bash(git log *)"</code> <code>"Bash(git diff *)"</code> <code>"Edit"</code>
<code>--fallback-model</code>	デフォルトモデルがオーバーロードされた場合、指定されたモデルへの自動フォールバックを有効化（プリントモードのみ）	<code>claude -p --fallback-model sonnet "query"</code>
<code>--fork-session</code>	再開時に、元のセッション ID を再利用する代わりに新しいセッション ID を作成します（ <code>--resume</code> または <code>--continue</code> で使用）	<code>claude --resume abc123 --fork-session</code>
<code>--from-pr</code>	特定の GitHub PR にリンクされたセッションを再開します。PR 番号または URL を受け入れます。 <code>gh pr create</code> 経由で作成されたセッションは自動的にリンクされます	<code>claude --from-pr 123</code>
<code>--ide</code>	起動時に IDE に自動的に接続します（正確に 1 つの有効な IDE が利用可能な場合）	<code>claude --ide</code>
<code>--init</code>	初期化フックを実行してインタラクティブモードを開始	<code>claude --init</code>
<code>--init-only</code>	初期化フックを実行して終了（インタラクティブセッションなし）	<code>claude --init-only</code>
<code>--include-partial-messages</code>	出力に部分的なストリーミングイベントを含めます（ <code>--print</code> と <code>--output-format=stream-json</code> が必要）	<code>claude -p --output-format stream-json --include-partial-messages "query"</code>

フラグ	説明	例
<code>--input-format</code>	プリントモードの入力形式を指定します (オプション: <code>text</code> 、 <code>stream-json</code>)	<code>claude -p --output-format json --input-format stream-json</code>
<code>--json-schema</code>	エージェントがワークフローを完了した後、JSON Schema に一致する検証済み JSON 出力を取得します (プリントモードのみ、 構造化出力 を参照)	<code>claude -p --json-schema</code> <code>'{"type":"object","properties":{"... }}"</code> <code>"query"</code>
<code>--maintenance</code>	メンテナンスフックを実行して終了	<code>claude --maintenance</code>
<code>--max-budget-usd</code>	停止する前に API 呼び出しに費やす最大ドル金額 (プリントモードのみ)	<code>claude -p --max-budget-usd 5.00</code> <code>"query"</code>
<code>--max-turns</code>	agentic ターンの数を制限します (プリントモードのみ)。制限に達するとエラーで終了します。デフォルトでは制限なし	<code>claude -p --max-turns 3 "query"</code>
<code>--mcp-config</code>	JSON ファイルまたは文字列から MCP サーバーを読み込みます (スペース区切り)	<code>claude --mcp-config ./mcp.json</code>
<code>--model</code>	現在のセッションのモデルを設定します。最新モデルのエイリアス (<code>sonnet</code> または <code>opus</code>) またはモデルの完全な名前を使用	<code>claude --model</code> <code>claude-sonnet-4-6</code>
<code>--no-chrome</code>	このセッションの Chrome ブラウザ統合 を無効化	<code>claude --no-chrome</code>
<code>--no-session-persistence</code>	セッション永続化を無効化し、セッションがディスクに保存されず、再開できないようにします (プリントモードのみ)	<code>claude -p --no-session-persistence</code> <code>"query"</code>
<code>--output-format</code>	プリントモードの出力形式を指定します (オプション: <code>text</code> 、 <code>json</code> 、 <code>stream-json</code>)	<code>claude -p "query" --output-format json</code>

フラグ	説明	例
<code>--permission-mode</code>	指定された 権限モード で開始	<code>claude --permission-mode plan</code>
<code>--permission-prompt-tool</code>	非インタラクティブモードで権限プロンプトを処理する MCP ツールを指定	<code>claude -p --permission-prompt-tool mcp_auth_tool "query"</code>
<code>--plugin-dir</code>	このセッションのみのプラグインをディレクトリから読み込みます（繰り返し可能）	<code>claude --plugin-dir ./my-plugins</code>
<code>--print</code> , <code>-p</code>	インタラクティブモードなしで応答を出力します（プログラムの使用の詳細については Agent SDK ドキュメント を参照）	<code>claude -p "query"</code>
<code>--remote</code>	claude.ai で提供されたタスク説明を使用して新しい Web セッション を作成	<code>claude --remote "Fix the login bug"</code>
<code>--resume</code> , <code>-r</code>	ID または名前で特定のセッションを再開するか、セッションを選択するためのインタラクティブピッカーを表示	<code>claude --resume auth-refactor</code>
<code>--session-id</code>	会話に特定のセッション ID を使用します（有効な UUID である必要があります）	<code>claude --session-id "550e8400-e29b-41d4-a716-446655440000"</code>
<code>--setting-sources</code>	読み込む設定ソースのカンマ区切りリスト（ <code>user</code> 、 <code>project</code> 、 <code>local</code> ）	<code>claude --setting-sources user,project</code>
<code>--settings</code>	追加の設定を読み込むための settings JSON ファイルまたは JSON 文字列へのパス	<code>claude --settings ./settings.json</code>
<code>--strict-mcp-config</code>	<code>--mcp-config</code> からのみ MCP サーバーを使用し、他のすべての MCP 設定を無視	<code>claude --strict-mcp-config --mcp-config ./mcp.json</code>

フラグ	説明	例
<code>--system-prompt</code>	デフォルトのシステムプロンプト全体をカスタムテキストに置き換え	<code>claude --system-prompt "You are a Python expert"</code>
<code>--system-prompt-file</code>	ファイルからシステムプロンプトを読み込み、デフォルトプロンプトを置き換え	<code>claude --system-prompt-file ./custom-prompt.txt</code>
<code>--teleport</code>	Web セッション をローカルターミナルで再開	<code>claude --teleport</code>
<code>--teammate-mode</code>	エージェントチーム のチームメイトの表示方法を設定します: <code>auto</code> (デフォルト)、 <code>in-process</code> 、または <code>tmux</code> 。 エージェントチームのセットアップ を参照	<code>claude --teammate-mode in-process</code>
<code>--tools</code>	Claude が使用できる組み込みツールを制限します。 <code>" "</code> を使用してすべてを無効化、 <code>"default"</code> をすべてに使用、または <code>"Bash,Edit,Read"</code> のようなツール名を使用	<code>claude --tools "Bash,Edit,Read"</code>
<code>--verbose</code>	詳細ログを有効化し、ターンバイターンの完全な出力を表示	<code>claude --verbose</code>
<code>--version</code> , <code>-v</code>	バージョン番号を出力	<code>claude -v</code>
<code>--worktree</code> , <code>-w</code>	Claude を <code><repo>/ .claude/ worktrees/<name></code> の分離された git worktree で開始します。名前が指定されていない場合は、自動生成されます	<code>claude -w feature-auth</code>

Tip:

`--output-format json` フラグは、スクリプトと自動化に特に役立ち、Claude の応答をプログラマ的に解析できます。

Agents フラグ形式

`--agents` フラグは、1 つ以上のカスタム subagents を定義する JSON オブジェクトを受け入れます。各 subagent には、一意の名前（キーとして）と、以下のフィールドを持つ定義オブジェクトが必要です。

フィールド	必須	説明
<code>description</code>	はい	subagent を呼び出すべき時期の自然言語説明
<code>prompt</code>	はい	subagent の動作をガイドするシステムプロンプト
<code>tools</code>	いいえ	subagent が使用できる特定のツールの配列。例：[<code>"Read"</code> , <code>"Edit"</code> , <code>"Bash"</code>]。省略された場合は、すべてのツールを継承します。 <code>Agent(agent_type)</code> 構文をサポート
<code>disallowedTools</code>	いいえ	この subagent に対して明示的に拒否するツール名の配列
<code>model</code>	いいえ	使用するモデルエイリアス： <code>sonnet</code> 、 <code>opus</code> 、 <code>haiku</code> 、または <code>inherit</code> 。省略された場合は、デフォルトで <code>inherit</code>
<code>skills</code>	いいえ	subagent のコンテキストにプリロードする <code>skill</code> 名の配列
<code>mcpServers</code>	いいえ	この subagent の <code>MCP servers</code> の配列。各エントリはサーバー名文字列または <code>{name: config}</code> オブジェクト
<code>maxTurns</code>	いいえ	subagent が停止する前の agentic ターンの最大数

例：

```
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."
  }
}'
```

subagents の作成と使用の詳細については、[subagents ドキュメント](#) を参照してください。

システムプロンプトフラグ

Claude Code は、システムプロンプトをカスタマイズするための 4 つのフラグを提供します。4 つすべてが、インタラクティブモードと非インタラクティブモードの両方で機能します。

フラグ	動作	ユースケース
<code>--system-prompt</code>	デフォルトプロンプト全体を 置き換え	Claude の動作と指示を完全に制御
<code>--system-prompt-file</code>	ファイルの内容で 置き換え	再現性とバージョン管理のためにファイルからプロンプトを読み込む
<code>--append-system-prompt</code>	デフォルトプロンプトに 追加	デフォルトの Claude Code 動作を保持しながら特定の指示を追加
<code>--append-system-prompt-file</code>	ファイルの内容をデフォルトプロンプトに 追加	デフォルトを保持しながらファイルから追加の指示を読み込む

各フラグを使用する時期：

- **--system-prompt** : Claude のシステムプロンプトを完全に制御する必要がある場合に使用します。これにより、すべてのデフォルト Claude Code 指示が削除され、白紙の状態が得られます。

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

- **--system-prompt-file** : ファイルからカスタムプロンプトを読み込みたい場合に使用します。チームの一貫性またはバージョン管理されたプロンプトテンプレートに役立ちます。

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

- **--append-system-prompt** : Claude Code のデフォルト機能を保持しながら特定の指示を追加したい場合に使用します。これはほとんどのユースケースで最も安全なオプションです。

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

- **--append-system-prompt-file** : Claude Code のデフォルトを保持しながらファイルから指示を追加したい場合に使用します。バージョン管理された追加に役立ちます。

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

--system-prompt と **--system-prompt-file** は相互に排他的です。追加フラグは、置き換えフラグのいずれかと一緒に使用できます。

ほとんどのユースケースでは、**--append-system-prompt** または **--append-system-prompt-file** が推奨されます。これらは Claude Code の組み込み機能を保持しながらカスタム要件を追加します。システムプロンプトを完全に制御する必要がある場合にのみ **--system-prompt** または **--system-prompt-file** を使用してください。

関連項目

- [Chrome 拡張機能](#) - ブラウザ自動化と Web テスト
- [インタラクティブモード](#) - ショートカット、入力モード、インタラクティブ機能

- [クイックスタートガイド](#) - Claude Code の開始方法
- [一般的なワークフロー](#) - 高度なワークフローとパターン
- [設定](#) - 設定オプション
- [Agent SDK ドキュメント](#) - プログラム的な使用と統合

キーボードショートカットのカスタマイズ

キーボードショートカットをカスタマイズして、Claude Code でキーバインディング設定ファイルを使用します。

Note:

カスタマイズ可能なキーボードショートカットには Claude Code v2.1.18 以降が必要です。
`claude --version` でバージョンを確認してください。

Claude Code はカスタマイズ可能なキーボードショートカットをサポートしています。/
`keybindings` を実行して、`~/.claude/keybindings.json` に設定ファイルを作成または開きます。

設定ファイル

キーバインディング設定ファイルは、`bindings` 配列を持つオブジェクトです。各ブロックはコンテキストとキーストロークからアクションへのマップを指定します。

Note:

キーバインディングファイルへの変更は自動的に検出され、Claude Code を再起動することなく適用されます。

フィールド	説明
<code>\$schema</code>	エディタのオートコンプリート用のオプション JSON スキーマ URL
<code>\$docs</code>	オプションのドキュメント URL
<code>bindings</code>	コンテキスト別のバインディングブロックの配列

この例では、チャットコンテキストで `Ctrl+E` を外部エディタを開くにバインドし、`Ctrl+U` をアンバインドします。

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

コンテキスト

各バインディングブロックは、バインディングが適用される**コンテキスト**を指定します。

コンテキスト	説明
Global	アプリ全体に適用
Chat	メインチャット入力エリア
Autocomplete	オートコンプリートメニューが開いている
Settings	設定メニュー（Escape キーのみで閉じる）
Confirmation	権限と確認ダイアログ
Tabs	タブナビゲーションコンポーネント
Help	ヘルプメニューが表示されている
Transcript	トランスクリプトビューア
HistorySearch	履歴検索モード（Ctrl+R）
Task	バックグラウンドタスクが実行中
ThemePicker	テーマピッカーダイアログ
Attachments	画像/添付ファイルバーナビゲーション

コンテキスト	説明
<code>Footer</code>	フッターインジケータナビゲーション（タスク、チーム、diff）
<code>MessageSelector</code>	巻き戻しと要約ダイアログのメッセージ選択
<code>DiffDialog</code>	Diff ビューアナビゲーション
<code>ModelPicker</code>	モデルピッカー努力レベル
<code>Select</code>	汎用選択/リストコンポーネント
<code>Plugin</code>	プラグインダイアログ（参照、発見、管理）

利用可能なアクション

アクションは `namespace:action` 形式に従います。例えば、`chat:submit` はメッセージを送信し、`app:toggleTodos` はタスクリストを表示します。各コンテキストには特定のアクションが利用可能です。

アプリアクション

`Global` コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>app:interrupt</code>	Ctrl+C	現在の操作をキャンセル
<code>app:exit</code>	Ctrl+D	Claude Code を終了
<code>app:toggleTodos</code>	Ctrl+T	タスクリストの表示を切り替え
<code>app:toggleTranscript</code>	Ctrl+O	詳細トランスクリプトの表示を切り替え

履歴アクション

コマンド履歴をナビゲートするためのアクション：

アクション	デフォルト	説明
<code>history:search</code>	Ctrl+R	履歴検索を開く
<code>history:previous</code>	Up	前の履歴項目

アクション	デフォルト	説明
<code>history:next</code>	Down	次の履歴項目

チャットアクション

Chat コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>chat:cancel</code>	Escape	現在の入力をキャンセル
<code>chat:cycleMode</code>	Shift+Tab*	権限モードをサイクル
<code>chat:modelPicker</code>	Cmd+P / Meta+P	モデルピッカーを開く
<code>chat:thinkingToggle</code>	Cmd+T / Meta+T	拡張思考を切り替え
<code>chat:submit</code>	Enter	メッセージを送信
<code>chat:undo</code>	Ctrl+_	最後のアクションを元に戻す
<code>chat:externalEditor</code>	Ctrl+G	外部エディタで開く
<code>chat:stash</code>	Ctrl+S	現在のプロンプトを保存
<code>chat:imagePaste</code>	Ctrl+V (Windows では Alt+V)	画像を貼り付け

*VT モードなし (Node <24.2.0 / <22.17.0、Bun <1.2.23) の Windows では、デフォルトは Meta+M です。

オートコンプリートアクション

Autocomplete コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>autocomplete:accept</code>	Tab	提案を受け入れ
<code>autocomplete:dismiss</code>	Escape	メニューを閉じる
<code>autocomplete:previous</code>	Up	前の提案
<code>autocomplete:next</code>	Down	次の提案

確認アクション

Confirmation コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>confirm:yes</code>	Y、Enter	アクションを確認
<code>confirm:no</code>	N、Escape	アクションを拒否
<code>confirm:previous</code>	Up	前のオプション
<code>confirm:next</code>	Down	次のオプション
<code>confirm:nextField</code>	Tab	次のフィールド
<code>confirm:previousField</code>	(アンバインド)	前のフィールド
<code>confirm:cycleMode</code>	Shift+Tab	権限モードをサイクル
<code>confirm:toggleExplanation</code>	Ctrl+E	権限説明を切り替え

権限アクション

権限ダイアログの **Confirmation** コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>permission:toggleDebug</code>	Ctrl+D	権限デバッグ情報を切り替え

トランスクリプトアクション

Transcript コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>transcript:toggleShow</code> <code>All</code>	Ctrl+E	すべてのコンテンツの表示を切り替え
<code>transcript:exit</code>	Ctrl+C、Escape	トランスクリプトビューを終了

履歴検索アクション

HistorySearch コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>historySearch:next</code>	Ctrl+R	次のマッチ
<code>historySearch:accept</code>	Escape、Tab	選択を受け入れ
<code>historySearch:cancel</code>	Ctrl+C	検索をキャンセル
<code>historySearch:execute</code>	Enter	選択したコマンドを実行

タスクアクション

Task コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>task:background</code>	Ctrl+B	現在のタスクをバックグラウンドに

テーマアクション

ThemePicker コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>theme:toggleSyntaxHighlighting</code>	Ctrl+T	シンタックスハイライトを切り替え

ヘルプアクション

Help コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>help:dismiss</code>	Escape	ヘルプメニューを閉じる

タブアクション

Tabs コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>tabs:next</code>	Tab、Right	次のタブ
<code>tabs:previous</code>	Shift+Tab、Left	前のタブ

添付ファイルアクション

Attachments コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>attachments:next</code>	Right	次の添付ファイル
<code>attachments:previous</code>	Left	前の添付ファイル
<code>attachments:remove</code>	Backspace、Delete	選択した添付ファイルを削除
<code>attachments:exit</code>	Down、Escape	添付ファイルバーを終了

フッターアクション

Footer コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>footer:next</code>	Right	次のフッター項目
<code>footer:previous</code>	Left	前のフッター項目
<code>footer:openSelected</code>	Enter	選択したフッター項目を開く
<code>footer:clearSelection</code>	Escape	フッター選択をクリア

メッセージセレクトアアクション

MessageSelector コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>messageSelector:up</code>	Up、K、Ctrl+P	リストで上に移動
<code>messageSelector:down</code>	Down、J、Ctrl+N	リストで下に移動
<code>messageSelector:top</code>	Ctrl+Up、Shift+Up、Meta+Up、Shift+K	最上部にジャンプ
<code>messageSelector:bottom</code>	Ctrl+Down、Shift+Down、Meta+Down、Shift+J	最下部にジャンプ
<code>messageSelector:select</code>	Enter	メッセージを選択

Diff アクション

DiffDialog コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>diff:dismiss</code>	Escape	Diff ビューアを閉じる
<code>diff:previousSource</code>	Left	前の Diff ソース
<code>diff:nextSource</code>	Right	次の Diff ソース
<code>diff:previousFile</code>	Up	Diff の前のファイル
<code>diff:nextFile</code>	Down	Diff の次のファイル
<code>diff:viewDetails</code>	Enter	Diff の詳細を表示
<code>diff:back</code>	(コンテキスト固有)	Diff ビューアに戻る

モデルピッカーアクション

ModelPicker コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>modelPicker:decreaseEffort</code>	Left	努力レベルを低下
<code>modelPicker:increaseEffort</code>	Right	努力レベルを増加

選択アクション

Select コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>select:next</code>	Down、J、Ctrl+N	次のオプション
<code>select:previous</code>	Up、K、Ctrl+P	前のオプション
<code>select:accept</code>	Enter	選択を受け入れ
<code>select:cancel</code>	Escape	選択をキャンセル

プラグインアクション

Plugin コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>plugin:toggle</code>	Space	プラグイン選択を切り替え
<code>plugin:install</code>	I	選択したプラグインをインストール

設定アクション

`Settings` コンテキストで利用可能なアクション：

アクション	デフォルト	説明
<code>settings:search</code>	/	検索モードに入る
<code>settings:retry</code>	R	使用状況データの読み込みを再試行（エラー時）

キーストロークシンタックス

モディファイア

+ セパレータでモディファイアキーを使用します。

- `ctrl` または `control` - Control キー
- `alt`、`opt`、または `option` - Alt/Option キー
- `shift` - Shift キー
- `meta`、`cmd`、または `command` - Meta/Command キー

例えば：

<code>ctrl+k</code>	単一キーとモディファイア
<code>shift+tab</code>	Shift + Tab
<code>meta+p</code>	Command/Meta + P
<code>ctrl+shift+c</code>	複数のモディファイア

大文字

スタンドアロンの大文字は Shift を意味します。例えば、`K` は `shift+k` と同等です。これは大文字と小文字のキーが異なる意味を持つ vim スタイルのバインディングに便利です。

モディファイア付きの大文字（例：`ctrl+K`）はスタイル的に扱われ、Shift を意味しません — `ctrl+K` は `ctrl+k` と同じです。

コード

コードはスペースで区切られたキーストロークのシーケンスです。

```
ctrl+k ctrl+s    Ctrl+K を押して、リリースしてから Ctrl+S
```

特殊キー

- `escape` または `esc` - Escape キー
- `enter` または `return` - Enter キー
- `tab` - Tab キー
- `space` - スペースバー
- `up` 、 `down` 、 `left` 、 `right` - 矢印キー
- `backspace` 、 `delete` - Delete キー

デフォルトショートカットをアンバインド

アクションを `null` に設定して、デフォルトショートカットをアンバインドします。

```
{
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+s": null
      }
    }
  ]
}
```

予約済みショートカット

これらのショートカットは再バインドできません。

ショートカット	理由
Ctrl+C	ハードコードされた割り込み/キャンセル
Ctrl+D	ハードコードされた終了

ターミナルの競合

一部のショートカットはターミナルマルチプレクサと競合する可能性があります。

ショートカット	競合
Ctrl+B	tmux プレフィックス（2 回押して送信）
Ctrl+A	GNU screen プレフィックス
Ctrl+Z	Unix プロセス一時停止（SIGTSTP）

Vim モードの相互作用

Vim モードが有効な場合（`/vim`）、キーバインディングと Vim モードは独立して動作します。

- **Vim モード** はテキスト入力レベルで入力を処理します（カーソル移動、モード、モードシジョン）
- **キーバインディング** はコンポーネントレベルでアクションを処理します（todos を切り替え、送信など）
- Vim モードの Escape キーは INSERT から NORMAL モードに切り替わります。
`chat:cancel` をトリガーしません
- ほとんどの Ctrl+key ショートカットは Vim モードを通過してキーバインディングシステムに渡されます
- Vim NORMAL モードでは、`?` はヘルプメニューを表示します（Vim の動作）

検証

Claude Code はキーバインディングを検証し、以下の警告を表示します。

- 解析エラー（無効な JSON または構造）
- 無効なコンテキスト名
- 予約済みショートカットの競合
- ターミナルマルチプレクサの競合
- 同じコンテキスト内の重複バインディング

`/doctor` を実行して、キーバインディングの警告を確認します。

Part 4: Configuration

権限を設定する

きめ細かい権限ルール、モード、管理ポリシーを使用して、Claude Code がアクセスして実行できる内容を制御します。

Claude Code は、エージェントが実行できることと実行できないことを正確に指定できるようにするため、きめ細かい権限をサポートしています。権限設定はバージョン管理にチェックインでき、組織内のすべての開発者に配布できるほか、個々の開発者がカスタマイズできます。

権限システム

Claude Code は、パワーと安全性のバランスを取るために、段階的な権限システムを使用しています。

ツールタイプ	例	承認が必要	「はい、今後は聞かない」の動作
読み取り専用	ファイル読み取り、Grep	いいえ	N/A
Bash コマンド	シェル実行	はい	プロジェクトディレクトリとコマンドごとに永続的
ファイル変更	Edit/Write ファイル	はい	セッション終了まで

権限を管理する

`/permissions` を使用して、Claude Code のツール権限を表示および管理できます。この UI は、すべての権限ルールと、それらが取得される `settings.json` ファイルをリストします。

- **Allow** ルールは、Claude Code が手動承認なしで指定されたツールを使用できるようにします。

- **Ask** ルールは、Claude Code が指定されたツールを使用しようとするたびに確認を促します。
- **Deny** ルールは、Claude Code が指定されたツールを使用することを防止します。

ルールは順序で評価されます。**deny -> ask -> allow**。最初にマッチしたルールが優先されるため、deny ルールは常に優先されます。

権限モード

Claude Code は、ツールの承認方法を制御するいくつかの権限モードをサポートしています。設定ファイルで `defaultMode` を設定します。

モード	説明
<code>default</code>	標準動作。各ツールの最初の使用時に権限を促します
<code>acceptEdits</code>	セッション中のファイル編集権限を自動的に受け入れます
<code>plan</code>	Plan Mode。Claude はファイルを分析できますが、ファイルを変更したりコマンドを実行したりすることはできません
<code>dontAsk</code>	<code>/permissions</code> または <code>permissions.allow</code> ルールで事前に承認されていない限り、ツールを自動的に拒否します
<code>bypassPermissions</code>	すべての権限プロンプトをスキップします（安全な環境が必要です。以下の警告を参照してください）

Warning:

`bypassPermissions` モードはすべての権限チェックを無効にします。このモードは、Claude Code が損害を引き起こせないコンテナや VM などの隔離された環境でのみ使用してください。管理者は、管理設定で `disableBypassPermissionsMode` を `"disable"` に設定することで、このモードを防止できます。

権限ルール構文

権限ルールは、`Tool` または `Tool(specifier)` の形式に従います。

ツールのすべての使用をマッチさせる

ツールのすべての使用をマッチさせるには、括弧なしでツール名を使用します。

ルール	効果
<code>Bash</code>	すべての Bash コマンドをマッチさせます
<code>WebFetch</code>	すべてのウェブフェッチリクエストをマッチさせます
<code>Read</code>	すべてのファイル読み取りをマッチさせます

`Bash(*)` は `Bash` と同等で、すべての Bash コマンドをマッチさせます。

細かい制御のためにスペシファイアを使用する

括弧内にスペシファイアを追加して、特定のツール使用をマッチさせます。

ルール	効果
<code>Bash(npm run build)</code>	正確なコマンド <code>npm run build</code> をマッチさせます
<code>Read(./env)</code>	現在のディレクトリの <code>.env</code> ファイルを読み取 ることをマッチさせます
<code>WebFetch(domain:example.com)</code>	example.com へのフェッチリクエストをマッ チさせます

ワイルドカードパターン

Bash ルールは `*` を使用したグロブパターンをサポートしています。ワイルドカードはコマンド内の任意の位置に表示できます。この設定により、npm および git commit コマンドが許可され、git push がブロックされます。

```
{
  "permissions": {
    "allow": [
      "Bash(npm run *)",
      "Bash(git commit *)",
      "Bash(git * main)",
      "Bash(* --version)",
      "Bash(* --help *)"
    ],
    "deny": [
      "Bash(git push *)"
    ]
  }
}
```

`*` の前のスペースは重要です。 `Bash(ls *)` は `ls -la` にマッチしますが `ls of` にはマッチしません。一方、 `Bash(ls*)` は両方にマッチします。レガシーな `:*` サフィックス構文は `*` と同等ですが、非推奨です。

ツール固有の権限ルール

Bash

Bash 権限ルールは `*` を使用したワイルドカードマッチングをサポートしています。ワイルドカードは、開始、中央、終了を含むコマンド内の任意の位置に表示できます。

- `Bash(npm run build)` は正確な Bash コマンド `npm run build` をマッチさせます
- `Bash(npm run test *)` は `npm run test` で始まる Bash コマンドをマッチさせます
- `Bash(npm *)` は `npm` で始まるコマンドをマッチさせます
- `Bash(* install)` は `install` で終わるコマンドをマッチさせます
- `Bash(git * main)` は `git checkout main`、`git merge main` などのコマンドをマッチさせます

`*` が末尾にスペース付きで表示される場合（`Bash(ls *)` など）、単語境界を強制し、プレフィックスの後にスペースまたは文字列の終わりが続く必要があります。たとえば、`Bash(ls *)` は `ls -la` にマッチしますが `ls of` にはマッチしません。対照的に、スペースなしの `Bash(ls*)` は、単語境界制約がないため、`ls -la` と `ls of` の両方にマッチします。

Tip:

Claude Code はシェルオペレータ（`&&` など）を認識しているため、`Bash(safe-cmd *)` のようなプレフィックスマッチルールは、`safe-cmd && other-cmd` コマンドを実行する権限を与えません。

Warning:

コマンド引数を制約しようとする Bash 権限パターンは脆弱です。たとえば、`Bash(curl http://github.com/ *)` は curl を GitHub URL に制限することを意図していますが、次のようなバリエーションにはマッチしません。

- URL の前のオプション。 `curl -X GET http://github.com/...`
- 異なるプロトコル。 `curl https://github.com/...`
- リダイレクト。 `curl -L http://bit.ly/xyz` (github にリダイレクト)
- 変数。 `URL=http://github.com && curl $URL`
- 余分なスペース。 `curl http://github.com`

より信頼性の高い URL フィルタリングについては、以下を検討してください。

- **Bash ネットワークツールを制限する**。deny ルールを使用して `curl`、`wget` などのコマンドをブロックし、許可されたドメインに対して `WebFetch(domain:github.com)` 権限で WebFetch ツールを使用します
- **PreToolUse フックを使用する**。Bash コマンドの URL を検証し、許可されていないドメインをブロックするフックを実装します
- CLAUDE.md を通じて Claude Code に許可された curl パターンについて指示します

WebFetch のみを使用しても、ネットワークアクセスは防止されません。Bash が許可されている場合、Claude は `curl`、`wget` または他のツールを使用して任意の URL に到達できます。

Read と Edit

Edit ルールは、ファイルを編集するすべての組み込みツールに適用されます。Claude は、Grep や Glob などのファイルを読み取るすべての組み込みツールに **Read** ルールを適用するためにベストエフォートを試みます。

Read と Edit ルールの両方は、[gitignore](#) 仕様に従い、4 つの異なるパターンタイプがあります。

パターン	意味	例	マッチ
<code>//path</code>	ファイルシステムルートからの絶対パス	<code>Read(//Users/alice/secrets/***)</code>	<code>/Users/alice/secrets/**</code>

パターン	意味	例	マッチ
<code>~/path</code>	ホームディレクトリからのパス	<code>Read(~/Documents/ *.pdf)</code>	<code>/Users/alice/ Documents/*.pdf</code>
<code>/path</code>	プロジェクトルートからの相対パス	<code>Edit(/src/**/ *.ts)</code>	<code><project root>/ src/**/*.ts</code>
<code>path</code> または <code>./path</code>	現在のディレクトリからの相対パス	<code>Read(*.env)</code>	<code><cwd>/*.env</code>

Warning:

`/Users/alice/file` のようなパターンは絶対パスではありません。プロジェクトルートからの相対パスです。絶対パスには `//Users/alice/file` を使用してください。

例。

- `Edit(/docs/**)`: `<project>/docs/` での編集（`/docs/` ではなく、`<project>/.claude/docs/` でもありません）
- `Read(~/.zshrc)`: ホームディレクトリの `.zshrc` を読み取ります
- `Edit(/tmp/scratch.txt)`: 絶対パス `/tmp/scratch.txt` を編集します
- `Read(src/**)`: `<current-directory>/src/` から読み取ります

Note:

gitignore パターンでは、`*` は単一のディレクトリ内のファイルをマッチさせ、`**` はディレクトリ全体で再帰的にマッチさせます。すべてのファイルアクセスを許可するには、括弧なしでツール名を使用します。Read、Edit、または Write。

WebFetch

- `WebFetch(domain:example.com)` は `example.com` へのフェッチリクエストをマッチさせます

MCP

- `mcp__puppeteer` は `puppeteer` サーバーによって提供されるツール（Claude Code で設定された名前）をマッチさせます
- `mcp__puppeteer__*` ワイルドカード構文は、`puppeteer` サーバーからのすべてのツールもマッチさせます

- `mcp__puppeteer__puppeteer_navigate` は `puppeteer` サーバーによって提供される `puppeteer_navigate` ツールをマッチさせます

Agent (subagents)

`Agent(AgentName)` ルールを使用して、Claude が使用できる `subagents` を制御します。

- `Agent(Explore)` は Explore subagent をマッチさせます
- `Agent(Plan)` は Plan subagent をマッチさせます
- `Agent(my-custom-agent)` は `my-custom-agent` という名前のカスタム subagent をマッチさせます

これらのルールを設定の `deny` 配列に追加するか、`--disallowedTools` CLI フラグを使用して特定のエージェントを無効にします。Explore エージェントを無効にするには。

```
{
  "permissions": {
    "deny": ["Agent(Explore)"]
  }
}
```

フックで権限を拡張する

[Claude Code フック](#) は、実行時に権限評価を実行するカスタムシェルコマンドを登録する方法を提供します。Claude Code がツール呼び出しを行うと、PreToolUse フックは権限システムの前に実行され、フック出力は権限システムの代わりにツール呼び出しを承認または拒否するかどうかを決定できます。

作業ディレクトリ

デフォルトでは、Claude は起動されたディレクトリ内のファイルにアクセスできます。このアクセスを拡張できます。

- **起動時。** `--add-dir <path>` CLI 引数を使用します
- **セッション中。** `/add-dir` コマンドを使用します
- **永続的な設定。** [設定ファイル](#) の `additionalDirectories` に追加します

追加ディレクトリ内のファイルは、元の作業ディレクトリと同じ権限ルールに従います。プロンプトなしで読み取り可能になり、ファイル編集権限は現在の権限モードに従います。

権限がサンドボックスとどのように相互作用するか

権限と[サンドボックス](#)は、補完的なセキュリティレイヤーです。

- **権限**は、Claude Code が使用できるツール、およびアクセスできるファイルまたはドメインを制御します。すべてのツール（Bash、Read、Edit、WebFetch、MCP など）に適用されます。
- **サンドボックス**は、Bash ツールのファイルシステムとネットワークアクセスを制限する OS レベルの強制を提供します。Bash コマンドとその子プロセスにのみ適用されます。

防御を深くするために両方を使用します。

- 権限 deny ルールは、Claude が制限されたリソースへのアクセスを試みることを防止します
- サンドボックス制限は、プロンプトインジェクションが Claude の意思決定をバイパスしても、Bash コマンドが定義された境界外のリソースに到達することを防止します
- サンドボックス内のファイルシステム制限は、Read と Edit deny ルールを使用し、別のサンドボックス設定は使用しません
- ネットワーク制限は、WebFetch 権限ルールとサンドボックスの `allowedDomains` リストを組み合わせます

管理設定

Claude Code 設定の一元的な制御が必要な組織の場合、管理者はユーザーまたはプロジェクト設定でオーバーライドできない管理設定をデプロイできます。これらのポリシー設定は通常の設定ファイルと同じ形式に従い、MDM/OS レベルのポリシー、管理設定ファイル、または[サーバー管理設定](#)を通じて配信できます。配信メカニズムとファイルの場所については、[設定ファイル](#)を参照してください。

管理のみの設定

一部の設定は管理設定でのみ有効です。

設定	説明
<code>disableBypassPermissionsMode</code>	<code>bypassPermissions</code> モードと <code>--dangerously-skip-permissions</code> フラグを防止するために <code>"disable"</code> に設定します

設定	説明
<code>allowManagedPermissionRulesOnly</code>	<code>true</code> の場合、ユーザーおよびプロジェクト設定が <code>allow</code> 、 <code>ask</code> 、または <code>deny</code> 権限ルールを定義することを防止します。管理設定のルールのみが適用されます
<code>allowManagedHooksOnly</code>	<code>true</code> の場合、ユーザー、プロジェクト、およびプラグインフックの読み込みを防止します。管理フックと SDK フックのみが許可されます
<code>allowManagedMcpServersOnly</code>	<code>true</code> の場合、管理設定からの <code>allowedMcpServers</code> のみが尊重されます。 <code>deniedMcpServers</code> はすべてのソースからマージされます。 管理 MCP 設定 を参照してください
<code>blockedMarketplaces</code>	マーケットプレイスソースのブロックリスト。ブロックされたソースはダウンロード前にチェックされるため、ファイルシステムに触れることはありません。 管理マーケットプレイス制限 を参照してください
<code>sandbox.network.allowManagedDomainsOnly</code>	<code>true</code> の場合、管理設定からの <code>allowedDomains</code> と <code>WebFetch(domain:...) allow</code> ルールのみが尊重されます。許可されていないドメインはユーザーに促すことなく自動的にブロックされます。拒否されたドメインはすべてのソースからマージされます
<code>strictKnownMarketplaces</code>	ユーザーが追加できるプラグインマーケットプレイスを制御します。 管理マーケットプレイス制限 を参照してください
<code>allow_remote_sessions</code>	<code>true</code> の場合、ユーザーが リモートコントロール と ウェブセッション を開始できます。デフォルトは <code>true</code> です。リモートセッションアクセスを防止するには <code>false</code> に設定します

設定の優先順位

権限ルールは、他のすべての Claude Code 設定と同じ[設定優先順位](#)に従います。

1. **管理設定**。コマンドライン引数を含む他のレベルでオーバーライドできません
2. **コマンドライン引数**。一時的なセッションオーバーライド
3. **ローカルプロジェクト設定** (`.claude/settings.local.json`)
4. **共有プロジェクト設定** (`.claude/settings.json`)
5. **ユーザー設定** (`~/.claude/settings.json`)

ツールがいずれかのレベルで拒否されている場合、他のレベルはそれを許可できません。たとえば、管理設定 `deny` は `--allowedTools` でオーバーライドできず、`--disallowedTools` は管理設定が定義する内容を超えて制限を追加できます。

権限がユーザー設定で許可されているがプロジェクト設定で拒否されている場合、プロジェクト設定が優先され、権限はブロックされます。

設定例

この[リポジトリ](#)には、一般的なデプロイメントシナリオのスターター設定が含まれています。これらを出発点として使用し、ニーズに合わせて調整してください。

関連項目

- [設定](#)。権限設定テーブルを含む完全な設定リファレンス
- [サンドボックス](#)。Bash コマンドの OS レベルのファイルシステムとネットワーク分離
- [認証](#)。Claude Code へのユーザーアクセスを設定します
- [セキュリティ](#)。セキュリティ保護とベストプラクティス
- [フック](#)。ワークフローを自動化し、権限評価を拡張します

Claude があなたのプロジェクトを記憶する方法

CLAUDE.md ファイルで Claude に永続的な指示を与え、自動メモリで Claude が自動的に学習を蓄積できるようにします。

Claude Code の各セッションは、新しいコンテキストウィンドウで始まります。2つのメカニズムがセッション間で知識を保持します。

- **CLAUDE.md ファイル:** Claude に永続的なコンテキストを与えるために作成する指示
- **自動メモリ:** あなたの修正と好みに基づいて Claude が自分自身で作成するメモ

このページでは、以下の方法について説明します。

- [CLAUDE.md ファイルの作成と整理](#)
- [.claude/rules/ で特定のファイルタイプにルールをスコープする](#)
- [自動メモリを設定して Claude が自動的にメモを取るようになる](#)
- [指示が従われていない場合のトラブルシューティング](#)

CLAUDE.md と自動メモリの比較

Claude Code には 2 つの相互補完的なメモリシステムがあります。どちらも各会話の開始時に読み込まれます。Claude はこれらをコンテキストとして扱い、強制的な設定ではありません。指示がより具体的で簡潔であるほど、Claude はそれに従う可能性が高くなります。

	CLAUDE.md ファイル	自動メモリ
誰が作成するか	あなた	Claude
含まれる内容	指示とルール	学習とパターン
スコープ	プロジェクト、ユーザー、または組織	ワーキングツリーごと
読み込まれる場所	すべてのセッション	すべてのセッション（最初の 200 行）
用途	コーディング標準、ワークフロー、プロジェクトアーキテクチャ	ビルドコマンド、デバッグの洞察、Claude が発見した好み

CLAUDE.md ファイルは Claude の動作をガイドしたい場合に使用します。自動メモリにより、Claude は手動の作業なしにあなたの修正から学習できます。

Subagent も独自の自動メモリを保持できます。詳細については、[subagent 設定](#)を参照してください。

CLAUDE.md ファイル

CLAUDE.md ファイルは、プロジェクト、個人的なワークフロー、または組織全体に対して Claude に永続的な指示を与えるマークダウンファイルです。これらのファイルをブレンテキストで作成します。Claude は各セッションの開始時にそれらを読みます。

CLAUDE.md ファイルの配置場所を選択する

CLAUDE.md ファイルは複数の場所に配置でき、それぞれ異なるスコープを持ちます。より具体的な場所がより広い場所よりも優先されます。

スコープ	場所	目的	ユースケースの例	共有対象
管理ポリシー	• macOS: <code>/Library/Application Support/ClaudeCode/CLAUDE.md</code> • Linux と WSL: <code>/etc/claude-code/CLAUDE.md</code> • Windows: <code>C:\Program Files\ClaudeCode\CLAUDE.md</code>	IT/DevOps が管理する組織全体の指示	企業のコーディング標準、セキュリティポリシー、コンプライアンス要件	組織内のすべてのユーザー
プロジェクト指示	<code>./CLAUDE.md</code> または <code>./.claude/CLAUDE.md</code>	プロジェクトのチーム共有指示	プロジェクトアーキテクチャ、コーディング標準、一般的なワークフロー	ソース管理を通じたチームメンバー
ユーザー指示	<code>~/.claude/CLAUDE.md</code>	すべてのプロジェクトの個人的な好み	コードスタイルの好み、個人的なツールショートカット	あなただけ（すべてのプロジェクト）

ワーキングディレクトリより上のディレクトリ階層内の CLAUDE.md ファイルは起動時に完全に読み込まれます。サブディレクトリ内の CLAUDE.md ファイルは、Claude がそれらのディレクトリ内のファイルを読むときにオンデマンドで読み込まれます。完全な解決順序については、[CLAUDE.md ファイルの読み込み方法](#)を参照してください。

大規模なプロジェクトの場合、[プロジェクトルール](#)を使用してトピック固有のファイルに指示を分割できます。ルールを使用すると、特定のファイルタイプまたはサブディレクトリに指示をスコープできます。

プロジェクト CLAUDE.md を設定する

プロジェクト CLAUDE.md は `./CLAUDE.md` または `./.claude/CLAUDE.md` に保存できます。このファイルを作成し、プロジェクトで作業する誰もが適用できる指示を追加します。ビルドおよびテストコマンド、コーディング標準、アーキテクチャの決定、命名規則、一般的なワークフローです。これらの指示はバージョン管理を通じてチームと共有されるため、個人的な好みではなくプロジェクトレベルの標準に焦点を当てます。

Tip:

`/init` を実行して、開始用の CLAUDE.md を自動的に生成します。Claude はコードベースを分析し、発見したビルドコマンド、テスト指示、プロジェクト規約を含むファイルを作成します。CLAUDE.md が既に存在する場合、`/init` は上書きするのではなく改善を提案します。Claude が自分で発見しない指示でそこから改善します。

効果的な指示を作成する

CLAUDE.md ファイルは各セッションの開始時にコンテキストウィンドウに読み込まれ、会話と一緒にトークンを消費します。これらはコンテキストであり強制的な設定ではないため、指示の作成方法は Claude がそれに従う信頼性に影響します。具体的で簡潔で、よく構造化された指示が最も効果的です。

サイズ: CLAUDE.md ファイルあたり 200 行以下を目標にします。より長いファイルはより多くのコンテキストを消費し、遵守を減らします。指示が大きくなっている場合は、[インポート](#)または `./.claude/rules/` ファイルを使用して分割します。

構造: マークダウンヘッダーと箇条書きを使用して関連する指示をグループ化します。Claude は読者と同じ方法で構造をスキャンします。整理されたセクションは密集した段落よりも従いやすいです。

具体性: 検証できるほど具体的な指示を作成します。例えば：

- ・「コードを適切にフォーマットする」ではなく「2 スペースのインデントを使用する」
- ・「変更をテストする」ではなく「コミット前に `npm test` を実行する」

- ・「ファイルを整理しておく」ではなく「API ハンドラーは `src/api/handlers/` に配置する」

一貫性: 2つのルールが互いに矛盾している場合、Claude は 1つを任意に選択する可能性があります。CLAUDE.md ファイル、サブディレクトリ内のネストされた CLAUDE.md ファイル、および `.claude/rules/` を定期的に確認して、古い指示または矛盾する指示を削除します。モノレポでは、`claudeMdExcludes` を使用して、作業に関連のない他のチームの CLAUDE.md ファイルをスキップします。

追加ファイルをインポートする

CLAUDE.md ファイルは `@path/to/import` 構文を使用して追加ファイルをインポートできます。インポートされたファイルは展開され、それらを参照する CLAUDE.md と一緒に起動時にコンテキストに読み込まれます。

相対パスと絶対パスの両方が許可されます。相対パスはワーキングディレクトリではなく、インポートを含むファイルに相対的に解決されます。インポートされたファイルは他のファイルを再帰的にインポートでき、最大深度は 5 ホップです。

README、package.json、およびワークフローガイドを取得するには、CLAUDE.md の任意の場所で `@` 構文を使用してそれらを参照します。

プロジェクト概要については `@README` を参照し、このプロジェクトで利用可能な npm コマンドについては `@package.json` を参照してください。

```
## 追加の指示
- git ワークフロー @docs/git-instructions.md
```

チェックインしたくない個人的な好みについては、ホームディレクトリからファイルをインポートします。インポートは共有 CLAUDE.md に含まれますが、それが指すファイルはマシンに残ります。

```
## 個人的な好み
- ~/.claude/my-project-instructions.md
```

Warning:

Claude Code が初めてプロジェクトで外部インポートを検出すると、ファイルをリストする承認ダイアログが表示されます。拒否すると、インポートは無効のままになり、ダイアログは再度表示されません。

指示を整理するためのより構造化されたアプローチについては、[.claude/rules/](#) を参照してください。

CLAUDE.md ファイルの読み込み方法

Claude Code は現在のワーキングディレクトリからディレクトリツリーを上を歩き、途中の各ディレクトリをチェックして CLAUDE.md ファイルを読みます。つまり、`foo/bar/` で Claude Code を実行すると、`foo/bar/CLAUDE.md` と `foo/CLAUDE.md` の両方から指示を読み込みます。

Claude は現在のワーキングディレクトリの下の子ディレクトリ内の CLAUDE.md ファイルも発見します。起動時に読み込む代わりに、Claude がそれらのサブディレクトリ内のファイルを読むときに含まれます。

他のチームの CLAUDE.md ファイルが取得される大規模なモノレポで作業する場合は、`claudeMdExcludes` を使用してそれらをスキップします。

追加ディレクトリから読み込む

`--add-dir` フラグは、メインワーキングディレクトリの外の追加ディレクトリへのアクセスを Claude に与えます。デフォルトでは、これらのディレクトリからの CLAUDE.md ファイルは読み込まれません。

追加ディレクトリから CLAUDE.md ファイルを読み込むには、`CLAUDE.md`、`.claude/CLAUDE.md`、および `.claude/rules/*.md` を含めて、`CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD` 環境変数を設定します。

```
CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1 claude --add-dir ../shared-config
```

`.claude/rules/` でルールを整理する

大規模なプロジェクトの場合、`.claude/rules/` ディレクトリを使用して指示を複数のファイルに整理できます。これにより、指示がモジュール化され、チームが保守しやすくなります。ルールは[特定のファイルパスにスコープ](#)することもできるため、Claude が一致するファイルで作業する場合にのみコンテキストに読み込まれ、ノイズを減らしてコンテキストスペースを節約します。

Note:

ルールは各セッションまたは一致するファイルが開かれたときにコンテキストに読み込まれます。タスク固有の指示で常にコンテキストに含まれる必要がない場合は、[skills](#)を使用してください。これは呼び出すときまたは Claude がプロンプトに関連していると判断したときにのみ読み込まれます。

ルールを設定する

プロジェクトの `.claude/rules/` ディレクトリにマークダウンファイルを配置します。各ファイルは1つのトピックをカバーし、`testing.md` または `api-design.md` のような説明的なファイル名を持つ必要があります。すべての `.md` ファイルは再帰的に発見されるため、`frontend/` または `backend/` のようなサブディレクトリにルールを整理できます。

```
your-project/
├── .claude/
│   ├── CLAUDE.md          # メインプロジェクト指示
│   └── rules/
│       ├── code-style.md  # コードスタイルガイドライン
│       ├── testing.md     # テスト規約
│       └── security.md    # セキュリティ要件
```

`paths` [frontmatter](#) のないルールは、`.claude/CLAUDE.md` と同じ優先度で起動時に読み込まれます。

パス固有のルール

ルールはYAML frontmatter と `paths` フィールドを使用して特定のファイルにスコープできます。これらの条件付きルールは、Claude が指定されたパターンに一致するファイルで作業している場合にのみ適用されます。

```
---
paths:
  - "src/api/**/*.ts"
---

## API 開発ルール

- すべての API エンドポイントは入力検証を含める必要があります
- 標準エラーレスポンス形式を使用します
- OpenAPI ドキュメンテーションコメントを含めます
```

`paths` フィールドのないルールは無条件に読み込まれ、すべてのファイルに適用されます。パススコープルールは、すべてのツール使用時ではなく、パターンに一致するファイルを読むときにトリガーされます。

`paths` フィールドでグロブパターンを使用して、拡張子、ディレクトリ、またはその組み合わせでファイルを一致させます。

パターン	一致
<code>**/*.ts</code>	任意のディレクトリ内のすべての TypeScript ファイル
<code>src/**/*.*</code>	<code>src/</code> ディレクトリの下の子のすべてのファイル
<code>*.md</code>	プロジェクトルートのマークダウンファイル
<code>src/components/*.tsx</code>	特定のディレクトリ内の React コンポーネント

複数のパターンを指定し、ブレース展開を使用して 1 つのパターンで複数の拡張子を一致させることができます。

```
---
paths:
  - "src/**/*.{ts,tsx}"
  - "lib/**/*.ts"
  - "tests/**/*.test.ts"
---
```

シンボリックリンクでプロジェクト間でルールを共有する

`.claude/rules/` ディレクトリはシンボリックリンクをサポートしているため、共有ルールセットを保持し、複数のプロジェクトにリンクできます。シンボリックリンクは解決され、通常どおり読み込まれ、循環シンボリックリンクは検出され、適切に処理されます。

この例は、共有ディレクトリと個別ファイルの両方をリンクします。

```
ln -s ~/shared-claude-rules .claude/rules/shared
ln -s ~/company-standards/security.md .claude/rules/security.md
```

ユーザーレベルのルール

`~/.claude/rules/` の個人的なルールはマシン上のすべてのプロジェクトに適用されます。プロジェクト固有ではない好みに使用します。

```
~/ .claude/rules/  
├─ preferences.md    # あなたの個人的なコーディング好み  
└─ workflows.md     # あなたの好みのワークフロー
```

ユーザーレベルのルールはプロジェクトルールの前に読み込まれ、プロジェクトルールに高い優先度を与えます。

大規模なチーム向けに CLAUDE.md を管理する

Claude Code をチーム全体に展開する組織の場合、指示を一元化し、どの CLAUDE.md ファイルが読み込まれるかを制御できます。

組織全体の CLAUDE.md を展開する

組織は、マシン上のすべてのユーザーに適用される一元管理の CLAUDE.md を展開できます。このファイルは個別の設定で除外することはできません。

Step 1: 管理ポリシーの場所にファイルを作成する

- macOS: `/Library/Application Support/ClaudeCode/CLAUDE.md`
- Linux と WSL: `/etc/claude-code/CLAUDE.md`
- Windows: `C:\Program Files\ClaudeCode\CLAUDE.md`

Step 2: 設定管理システムで展開する

MDM、グループポリシー、Ansible、または同様のツールを使用して、開発者マシン全体にファイルを配布します。その他の組織全体の設定オプションについては、[管理設定](#)を参照してください。

特定の CLAUDE.md ファイルを除外する

大規模なモノレポでは、祖先の CLAUDE.md ファイルに作業に関連のない指示が含まれている可能性があります。`claudeMdExcludes` 設定を使用すると、パスまたはグロブパターンで特定のファイルをスキップできます。

この例は、トップレベルの CLAUDE.md と親フォルダのルールディレクトリを除外します。`.claude/settings.local.json` に追加して、除外をマシンにローカルに保ちます。

```
{
  "claudeMdExcludes": [
    "**/monorepo/CLAUDE.md",
    "/home/user/monorepo/other-team/.claude/rules/**"
  ]
}
```

パターンはグロブ構文を使用して絶対ファイルパスに対して一致します。

`claudeMdExcludes` は任意の [設定レイヤー](#) で設定できます。ユーザー、プロジェクト、ローカル、または管理ポリシー。配列はレイヤー全体でマージされます。

管理ポリシー CLAUDE.md ファイルは除外できません。これにより、個別の設定に関係なく、組織全体の指示が常に適用されることが保証されます。

自動メモリ

自動メモリにより、Claude は何も作成することなくセッション間で知識を蓄積できます。Claude は作業中に自分自身のためにメモを保存します。ビルドコマンド、デバッグの洞察、アーキテクチャノート、コードスタイルの好み、ワークフローの習慣です。Claude はすべてのセッションで何かを保存するわけではありません。情報が将来の会話で役立つかどうかに基づいて、何を記憶する価値があるかを決定します。

Note:

自動メモリには Claude Code v2.1.59 以降が必要です。 `claude --version` でバージョンを確認します。

自動メモリを有効または無効にする

自動メモリはデフォルトで有効です。切り替えるには、セッションで `/memory` を開き、自動メモリトグルを使用するか、プロジェクト設定で `autoMemoryEnabled` を設定します。

```
{
  "autoMemoryEnabled": false
}
```

環境変数を使用して自動メモリを無効にするには、`CLAUDE_CODE_DISABLE_AUTO_MEMORY=1` を設定します。

ストレージの場所

各プロジェクトは `~/.claude/projects/<project>/memory/` に独自のメモリディレクトリを取得します。`<project>` パスは git リポジトリから派生しているため、同じリポジトリ内のすべてのワーキングツリーとサブディレクトリは1つの自動メモリディレクトリを共有します。git リポジトリの外では、プロジェクトルートが代わりに使用されます。

自動メモリを別の場所に保存するには、ユーザーまたはローカル設定で `autoMemoryDirectory` を設定します。

```
{
  "autoMemoryDirectory": "~/my-custom-memory-dir"
}
```

この設定はポリシー、ローカル、およびユーザー設定から受け入れられます。共有プロジェクトが自動メモリ書き込みを機密の場所にリダイレクトするのを防ぐため、プロジェクト設定 (`.claude/settings.json`) からは受け入れられません。

ディレクトリには `MEMORY.md` エントリポイントとオプションのトピックファイルが含まれます。

```
~/.claude/projects/<project>/memory/
├─ MEMORY.md          # 簡潔なインデックス、すべてのセッションに読み込まれます
├─ debugging.md       # デバッグパターンの詳細なメモ
├─ api-conventions.md # API 設計の決定
└─ ...                # Claude が作成するその他のトピックファイル
```

`MEMORY.md` はメモリディレクトリのインデックスとして機能します。Claude はセッション全体を通じてこのディレクトリ内のファイルを読み書きし、`MEMORY.md` を使用して保存されている内容を追跡します。

自動メモリはマシンローカルです。同じ git リポジトリ内のすべてのワーキングツリーとサブディレクトリは1つの自動メモリディレクトリを共有します。ファイルはマシン間またはクラウド環境全体で共有されません。

動作方法

`MEMORY.md` の最初の 200 行は各会話の開始時に読み込まれます。200 行を超えるコンテンツはセッション開始時に読み込まれません。Claude は詳細なメモを別のトピックファイルに移動することで、`MEMORY.md` を簡潔に保ちます。

この 200 行の制限は `MEMORY.md` にのみ適用されます。CLAUDE.md ファイルは長さに関係なく完全に読み込まれますが、より短いファイルはより良い遵守を生成します。

`debugging.md` または `patterns.md` のようなトピックファイルは起動時に読み込まれません。Claude は必要な情報が必要な場合、標準ファイルツールを使用してセッション中にオンデマンドで読みます。

Claude はセッション中にメモリファイルを読み書きします。Claude Code インターフェイスで「Writing memory」または「Recalled memory」が表示される場合、Claude は `~/.claude/projects/<project>/memory/` から積極的に更新または読み取りを行っています。

メモリを監査および編集する

自動メモリファイルはブレーンマークダウンで、いつでも編集または削除できます。 `/memory` を実行して、セッション内からメモリファイルを参照および開きます。

`/memory` で表示および編集する

`/memory` コマンドは、現在のセッションに読み込まれたすべての CLAUDE.md およびルールファイルをリストし、自動メモリのオン/オフを切り替え、自動メモリフォルダを開くためのリンクを提供します。任意のファイルを選択してエディタで開きます。

Claude に何かを記憶するよう求めるとき、「常に npm ではなく pnpm を使用する」または「API テストがローカル Redis インスタンスを必要とすることを覚えておく」のように、Claude はそれを自動メモリに保存します。代わりに CLAUDE.md に指示を追加するには、Claude に直接「これを CLAUDE.md に追加する」と尋ねるか、 `/memory` を通じてファイルを自分で編集します。

メモリの問題をトラブルシューティングする

これらは CLAUDE.md と自動メモリの最も一般的な問題と、それらをデバッグするための手順です。

Claude が CLAUDE.md に従っていない

CLAUDE.md はコンテキストであり、強制ではありません。Claude はそれを読んでそれに従おうとしますが、特に曖昧または矛盾する指示の場合、厳密な遵守の保証はありません。

デバッグするには：

- `/memory` を実行して、CLAUDE.md ファイルが読み込まれていることを確認します。ファイルがリストされていない場合、Claude はそれを見ることができません。

- 関連する CLAUDE.md がセッションに読み込まれる場所にあることを確認します ([CLAUDE.md ファイルの配置場所を選択する](#)を参照)。
- 指示をより具体的にします。「2 スペースのインデントを使用する」は「コードを適切にフォーマットする」よりも効果的です。
- CLAUDE.md ファイル全体で矛盾する指示を探します。2 つのファイルが同じ動作に対して異なるガイダンスを提供する場合、Claude は 1 つを任意に選択する可能性があります。

Tip:

`InstructionsLoaded` `hook` を使用して、どの指示ファイルが読み込まれているか、いつ読み込まれているか、なぜ読み込まれているかを正確にログに記録します。これはバスの固有のルールまたはサブディレクトリ内の遅延読み込みファイルをデバッグするのに役立ちます。

自動メモリが何を保存したかわからない

`/memory` を実行し、自動メモリフォルダを選択して、Claude が保存したものを参照します。すべてはプレーンマークダウンで、読み取り、編集、または削除できます。

CLAUDE.md が大きすぎる

200 行を超えるファイルはより多くのコンテキストを消費し、遵守を減らす可能性があります。詳細なコンテンツを `@path` インポート ([追加ファイルをインポートする](#)を参照) で参照される別のファイルに移動するか、`.claude/rules/` ファイル全体に指示を分割します。

`/compact` 後に指示が失われたようです

CLAUDE.md は圧縮を完全に生き残ります。`/compact` 後、Claude はディスクから CLAUDE.md を再度読み込み、セッションに新しく再注入します。圧縮後に指示が消えた場合、それは会話でのみ与えられ、CLAUDE.md に書き込まれていません。セッション間で永続化するために CLAUDE.md に追加します。

[効果的な指示を作成する](#)を参照して、サイズ、構造、および具体性に関するガイダンスを確認してください。

関連リソース

- [Skills](#): オンデマンドで読み込まれる反復可能なワークフローをパッケージ化する
- [Settings](#): 設定ファイルで Claude Code の動作を設定する
- [セッションを管理する](#): コンテキストを管理し、会話を再開し、並列セッションを実行する

- [Subagent メモリ](#): subagent が独自の自動メモリを保持できるようにする

Claude Code の設定

グローバルおよびプロジェクトレベルの設定と環境変数を使用して Claude Code を構成します。

Claude Code は、ニーズに合わせて動作を構成するためのさまざまな設定を提供しています。インタラクティブ REPL を使用する際に `/config` コマンドを実行することで Claude Code を構成できます。これにより、ステータス情報を表示し、構成オプションを変更できるタブ付き設定インターフェースが開きます。

構成スコープ

Claude Code は、**スコープシステム**を使用して、構成が適用される場所と共有される相手を決めます。スコープを理解することで、個人使用、チーム協力、またはエンタープライズデプロイメント用に Claude Code を構成する方法を決定するのに役立ちます。

利用可能なスコープ

スコープ	場所	影響を受けるユーザー	チームと共有されるか
Managed	サーバー管理設定、plist / レジストリ、またはシステムレベルの <code>managed-settings.json</code>	マシン上のすべてのユーザー	はい（IT によってデプロイされます）
User	<code>~/.claude/</code> ディレクトリ	あなた、すべてのプロジェクト全体	いいえ
Project	リポジトリ内の <code>.claude/</code>	このリポジトリのすべてのコラボレーター	はい（git にコミットされます）
Local	<code>.claude/settings.local.json</code>	あなた、このリポジトリ内のみ	いいえ（gitignored）

各スコープを使用する場合

Managed スコープは以下の場合に使用します：

- ・組織全体で実施する必要があるセキュリティポリシー
- ・オーバーライドできないコンプライアンス要件

- ・ IT/DevOps によってデプロイされた標準化された構成

User スコープは以下の場合に最適です：

- ・ すべての場所で使用したい個人的な設定（テーマ、エディター設定）
- ・ すべてのプロジェクト全体で使用するツールとプラグイン
- ・ API キーと認証（安全に保存）

Project スコープは以下の場合に最適です：

- ・ チーム共有設定（権限、hooks、MCP servers）
- ・ チーム全体が持つべきプラグイン
- ・ コラボレーター全体でツールを標準化

Local スコープは以下の場合に最適です：

- ・ 特定のプロジェクトの個人的なオーバーライド
- ・ チームと共有する前に構成をテストする
- ・ 他のユーザーには機能しないマシン固有の設定

スコープの相互作用

同じ設定が複数のスコープで構成されている場合、より具体的なスコープが優先されます：

1. **Managed**（最高） - 何によってもオーバーライドできません
2. **コマンドライン引数** - 一時的なセッションオーバーライド
3. **Local** - プロジェクトおよびユーザー設定をオーバーライド
4. **Project** - ユーザー設定をオーバーライド
5. **User**（最低） - 他に何も設定を指定しない場合に適用

たとえば、ユーザー設定で権限が許可されているが、プロジェクト設定で拒否されている場合、プロジェクト設定が優先され、権限はブロックされます。

スコープを使用する機能

スコープは多くの Claude Code 機能に適用されます：

機能	ユーザーの場所	プロジェクトの場所	ローカルの場所
Settings	<code>~/.claude/settings.json</code>	<code>.claude/ settings.json</code>	<code>.claude/ settings.local.js on</code>

機能	ユーザーの場所	プロジェクトの場所	ローカルの場所
Subagents	<code>~/ .claude/agents/</code>	<code>.claude/agents/</code>	—
MCP servers	<code>~/ .claude.json</code>	<code>.mcp.json</code>	<code>~/ .claude.json</code> (プロジェクトごと)
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> または <code>.claude/CLAUDE.md</code>	—

設定ファイル

`settings.json` ファイルは、階層的な設定を通じて Claude Code を構成するための公式メカニズムです：

- **ユーザー設定**は `~/ .claude/settings.json` で定義され、すべてのプロジェクトに適用されます。
- **プロジェクト設定**はプロジェクトディレクトリに保存されます：
 - `.claude/settings.json` ソース管理にチェックインされ、チームと共有される設定用
 - `.claude/settings.local.json` チェックインされない設定用。個人的な設定と実験に役立ちます。Claude Code は作成時に `.claude/settings.local.json` を無視するように git を構成します。
- **Managed 設定**：集中管理が必要な組織向けに、Claude Code は managed 設定の複数の配信メカニズムをサポートしています。すべて同じ JSON 形式を使用し、ユーザーまたはプロジェクト設定によってオーバーライドできません：
 - **サーバー管理設定**：Anthropic のサーバーから Claude.ai 管理コンソール経由で配信されます。[サーバー管理設定](#)を参照してください。
 - **MDM/OS レベルのポリシー**：macOS および Windows のネイティブデバイス管理を通じて配信されます：
 - macOS：`com.anthropic.claudecode` managed preferences ドメイン (Jamf、Kandji、または他の MDM ツールの構成プロファイルを通じてデプロイ)

- Windows : `HKLM\SOFTWARE\Policies\ClaudeCode` レジストリキーと JSON を含む `Settings` 値 (`REG_SZ` または `REG_EXPAND_SZ`) (グループポリシーまたは Intune を通じてデプロイ)
- Windows (ユーザーレベル) : `HKCU\SOFTWARE\Policies\ClaudeCode` (最低ポリシー優先度、管理者レベルのソースが存在しない場合のみ使用)
- ファイルベース : `managed-settings.json` および `managed-mcp.json` がシステムディレクトリにデプロイされます :
- macOS : `/Library/Application Support/ClaudeCode/`
- Linux および WSL : `/etc/claude-code/`
- Windows : `C:\Program Files\ClaudeCode\`

詳細については、[managed 設定](#)および [Managed MCP 構成](#)を参照してください。

Note:

Managed デプロイメントは、`strictKnownMarketplaces` を使用して [プラグインマーケットプレイスの追加](#)を制限することもできます。詳細については、[Managed マーケットプレイス制限](#)を参照してください。

- **その他の構成**は `~/.claude.json` に保存されます。このファイルには、設定 (テーマ、通知設定、エディターモード)、OAuth セッション、[MCP server](#) 構成 (ユーザーおよびローカルスコープ)、プロジェクトスコープの状態 (許可されたツール、信頼設定)、およびさまざまなキャッシュが含まれます。プロジェクトスコープの MCP サーバーは `.mcp.json` に別途保存されます。

Note:

Claude Code は構成ファイルのタイムスタンプ付きバックアップを自動的に作成し、データ損失を防ぐために最新の 5 つのバックアップを保持します。

```
{
  "$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"
  ]
}
```

上記の例の `$schema` 行は、Claude Code 設定の[公式 JSON スキーマ](#)を指しています。これを `settings.json` に追加すると、VS Code、Cursor、および JSON スキーマ検証をサポートする他のエディターでオートコンプリートとインライン検証が有効になります。

利用可能な設定

`settings.json` は多くのオプションをサポートしています：

キー	説明	例
<code>apiKeyHelper</code>	<code>/bin/sh</code> で実行されるカスタムスクリプト。認証値を生成します。この値は <code>X-Api-Key</code> および <code>Authorization: Bearer</code> ヘッダーとしてモデルリクエストに送信されます	<code>/bin/ generate_temp_api_key .sh</code>
<code>cleanupPeriodDays</code>	この期間より長い間非アクティブなセッションは起動時に削除されます。0 に設定するとすべてのセッションが即座に削除されます。（デフォルト：30 日）	20
<code>companyAnnouncements</code>	起動時にユーザーに表示するアナウンスメント。複数のアナウンスメントが提供される場合、ランダムにサイクルされます。	<code>["Welcome to Acme Corp! Review our code guidelines at docs.acme.com"]</code>
<code>env</code>	すべてのセッションに適用される環境変数	<code>{"F00": "bar"}</code>
<code>attribution</code>	git コミットおよびプルリクエストの属性をカスタマイズします。属性設定を参照してください	<code>{"commit": " Generated with Claude Code", "pr": ""}</code>
<code>includeCoAuthoredBy</code>	非推奨 ：代わりに <code>attribution</code> を使用してください。git コミットおよびプルリクエストに <code>co-authored-by Claude</code> バイラインを含めるかどうか（デフォルト： <code>true</code> ）	<code>false</code>
<code>includeGitInstructions</code>	Claude のシステムプロンプトに組み込みコミットおよび PR ワークフロー命令を含めます（デフォルト： <code>true</code> ）。これらの命令を削除するには <code>false</code> に設定します。たとえば、独自の git ワークフロースキルを使用する場合です。 <code>CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS</code> 環境変数が設定されている場合、この設定よりも優先されます	<code>false</code>

キー	説明	例
<code>permissions</code>	権限の構造については以下の表を参照してください。	
<code>hooks</code>	ライフサイクルイベントで実行するカスタムコマンドを構成します。 hooks ドキュメント を参照してください	hooks を参照
<code>disableAllHooks</code>	すべての hooks およびカスタム status line を無効にします	<code>true</code>
<code>allowManagedHooksOnly</code>	(Managed 設定のみ) ユーザー、プロジェクト、およびプラグイン hooks の読み込みを防止します。Managed hooks および SDK hooks のみを許可します。 Hook 構成 を参照してください	<code>true</code>
<code>allowedHttpHookUrls</code>	HTTP hooks がターゲットにできる URL パターンのホワイトリスト。* をワイルドカードとしてサポートします。設定されている場合、一致しない URL を持つ hooks はブロックされます。未定義 = 制限なし、空配列 = すべての HTTP hooks をブロック。設定ソース全体で配列がマージされます。 Hook 構成 を参照してください	<code>["https://hooks.example.com/*"]</code>
<code>httpHookAllowedEnvVars</code>	HTTP hooks がヘッダーに補間できる環境変数名のホワイトリスト。設定されている場合、各 hook の有効な <code>allowedEnvVars</code> はこのリストとの交差です。未定義 = 制限なし。設定ソース全体で配列がマージされます。 Hook 構成 を参照してください	<code>["MY_TOKEN", "HOOK_SECRET"]</code>

キー	説明	例
<code>allowManagedPermissionsRulesOnly</code>	(Managed 設定のみ) ユーザーおよびプロジェクト設定が <code>allow</code> 、 <code>ask</code> 、または <code>deny</code> 権限ルールを定義するのを防止します。Managed 設定のルールのみが適用されます。 Managed のみの設定 を参照してください	<code>true</code>
<code>allowManagedMcpServersOnly</code>	(Managed 設定のみ) Managed 設定からの <code>allowedMcpServers</code> のみが尊重されます。 <code>deniedMcpServers</code> はすべてのソースからマージされます。ユーザーは MCP サーバーを追加できますが、管理者定義のホワイトリストのみが適用されます。 Managed MCP 構成 を参照してください	<code>true</code>
<code>model</code>	Claude Code に使用するデフォルトモデルをオーバーライドします	<code>"claude-sonnet-4-6"</code>
<code>availableModels</code>	<code>/model</code> 、 <code>--model</code> 、Config ツール、または <code>ANTHROPIC_MODEL</code> を通じてユーザーが選択できるモデルを制限します。デフォルトオプションには影響しません。 モデル選択を制限 を参照してください	<code>["sonnet", "haiku"]</code>
<code>modelOverrides</code>	Anthropic モデル ID を Bedrock 推論プロファイル ARN などのプロバイダー固有のモデル ID にマップします。各モデルピッカーエントリは、プロバイダー API を呼び出す際にマップされた値を使用します。 バージョンごとにモデル ID をオーバーライド を参照してください	<pre>{ "claude-opus-4-6": "arn:aws:bedrock:..." }</pre>
<code>otelHeadersHelper</code>	動的 OpenTelemetry ヘッダーを生成するスクリプト。起動時および定期的に行われます (動的ヘッダー を参照)	<code>/bin/generate_otel_headers.sh</code>

キー	説明	例
<code>statusLine</code>	コンテキストを表示するカスタムステータスラインを構成します。 <code>statusLine</code> ドキュメント を参照してください	<pre>{"type": "command", "command": "~/ .claude/ statusline.sh"}</pre>
<code>fileSuggestion</code>	@ ファイルオートコンプリート用のカスタムスクリプトを構成します。 ファイル提案設定 を参照してください	<pre>{"type": "command", "command": "~/ .claude/file- suggestion.sh"}</pre>
<code>respectGitignore</code>	@ ファイルピッカーが <code>.gitignore</code> パターンを尊重するかどうかを制御します。 <code>true</code> (デフォルト) の場合、 <code>.gitignore</code> パターンに一致するファイルは提案から除外されます	<code>false</code>
<code>outputStyle</code>	システムプロンプトを調整する出力スタイルを構成します。 出力スタイルドキュメント を参照してください	<code>"Explanatory"</code>
<code>forceLoginMethod</code>	<code>claudeai</code> を使用して Claude.ai アカウントへのログインを制限するか、 <code>console</code> を使用して Claude Console (API 使用量請求) アカウントへのログインを制限します	<code>claudeai</code>
<code>forceLoginOrgUUID</code>	ログイン中に自動的に選択する組織の UUID を指定します。組織選択ステップをバイパスします。 <code>forceLoginMethod</code> が設定されている必要があります	<code>"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"</code>
<code>enableAllProjectMcpServers</code>	プロジェクト <code>.mcp.json</code> ファイルで定義されたすべての MCP サーバーを自動的に承認します	<code>true</code>
<code>enabledMcpjsonServers</code>	<code>.mcp.json</code> ファイルから承認する特定の MCP サーバーのリスト	<code>["memory", "github"]</code>
<code>disabledMcpjsonServers</code>	<code>.mcp.json</code> ファイルから拒否する特定の MCP サーバーのリスト	<code>["filesystem"]</code>

キー	説明	例
<code>allowedMcpServers</code>	Managed-settings.json で設定された場合、ユーザーが構成できる MCP サーバーのホワイトリスト。未定義 = 制限なし、空配列 = ロックダウン。すべてのスコープに適用されます。デニールストが優先されます。 Managed MCP 構成 を参照してください	<pre>[{ "serverName": "github" }]</pre>
<code>deniedMcpServers</code>	Managed-settings.json で設定された場合、明示的にブロックされた MCP サーバーのデニールスト。すべてのスコープ（Managed サーバーを含む）に適用されます。デニールストがホワイトリストよりも優先されます。 Managed MCP 構成 を参照してください	<pre>[{ "serverName": "filesystem" }]</pre>
<code>strictKnownMarketplaces</code>	Managed-settings.json で設定された場合、ユーザーが追加できるプラグインマーケットプレイスのホワイトリスト。未定義 = 制限なし、空配列 = ロックダウン。マーケットプレイス追加のみに適用されます。 Managed マーケットプレイス制限 を参照してください	<pre>[{ "source": "github", "repo": "acme-corp/ plugins" }]</pre>
<code>blockedMarketplaces</code>	（Managed 設定のみ）マーケットプレイスソースのブロックリスト。ブロックされたソースはダウンロード前にチェックされるため、ファイルシステムに触れることはありません。 Managed マーケットプレイス制限 を参照してください	<pre>[{ "source": "github", "repo": "untrusted/ plugins" }]</pre>

キー	説明	例
<code>pluginTrustMessage</code>	(Managed 設定のみ) インストール前に表示されるプラグイン信頼警告に追加されるカスタムメッセージ。これを使用して、組織固有のコンテキストを追加します。たとえば、内部マーケットプレイスからのプラグインが検証されていることを確認します。	<code>"All plugins from our marketplace are approved by IT"</code>
<code>awsAuthRefresh</code>	<code>.aws</code> ディレクトリを変更するカスタムスクリプト (高度な認証情報構成 を参照)	<code>aws sso login --profile myprofile</code>
<code>awsCredentialExport</code>	AWS 認証情報を含む JSON を出力するカスタムスクリプト (高度な認証情報構成 を参照)	<code>/bin/generate_aws_grant.sh</code>
<code>alwaysThinkingEnabled</code>	すべてのセッションに対して 拡張思考 をデフォルトで有効にします。通常は <code>/config</code> コマンドを通じて構成されます。直接編集ではなく	<code>true</code>
<code>plansDirectory</code>	プランファイルが保存される場所をカスタマイズします。パスはプロジェクトルートに相対的です。デフォルト: <code>~/.claude/plans</code>	<code>"./plans"</code>
<code>showTurnDuration</code>	レスポンス後にターン期間メッセージを表示します (例: 「Cooked for 1m 6s」)。これらのメッセージを非表示にするには <code>false</code> に設定します	<code>true</code>
<code>spinnerVerbs</code>	スピナーおよびターン期間メッセージに表示されるアクション動詞をカスタマイズします。 <code>mode</code> を <code>"replace"</code> に設定して独自の動詞のみを使用するか、 <code>"append"</code> に設定してデフォルトに追加します	<code>{"mode": "append", "verbs": ["Pondering", "Crafting"]}</code>

キー	説明	例
<code>language</code>	Claude の優先応答言語を構成します (例: <code>"japanese"</code> 、 <code>"spanish"</code> 、 <code>"french"</code>)。 Claude はデフォルトでこの言語で応答します	<code>"japanese"</code>
<code>autoUpdatesChannel</code>	更新に従うリリースチャネル。 <code>"stable"</code> を使用して、通常約 1 週間古いバージョンを使用し、大きな回帰があるバージョンをスキップするか、 <code>"latest"</code> (デフォルト) を使用して最新リリースを使用します	<code>"stable"</code>
<code>spinnerTipsEnabled</code>	Claude が作業中にスピナーにヒントを表示します。ヒントを無効にするには <code>false</code> に設定します (デフォルト: <code>true</code>)	<code>false</code>
<code>spinnerTipsOverride</code>	スピナーヒントをカスタム文字列でオーバーライドします。 <code>tips</code> : ヒント文字列の配列。 <code>excludeDefault: true</code> の場合、カスタムヒントのみを表示します。 <code>false</code> または不在の場合、カスタムヒントは組み込みヒントとマージされます	<pre>{ "excludeDefault": true, "tips": ["Use our internal tool X"] }</pre>
<code>terminalProgressBarEnabled</code>	Windows Terminal および iTerm2 などのサポートされているターミナルで進捗を表示するターミナル進捗バーを有効にします (デフォルト: <code>true</code>)	<code>false</code>
<code>prefersReducedMotion</code>	アクセシビリティのために UI アニメーション (スピナー、シマー、フラッシュエフェクト) を削減または無効にします	<code>true</code>

キー	説明	例
<code>fastModePerSessionOptIn</code>	<code>true</code> の場合、高速モードはセッション全体で永続化されません。各セッションは高速モードがオフで開始され、ユーザーが <code>/fast</code> で有効にする必要があります。ユーザーの高速モード設定は保存されます。 セッションごとのオプトインを要求 を参照してください	<code>true</code>
<code>teammateMode</code>	エージェントチーム チームメイトの表示方法: <code>auto</code> (tmux または iTerm2 で分割ペインを選択、それ以外の場合はインプロセス)、 <code>in-process</code> 、または <code>tmux</code> 。 エージェントチームをセットアップ を参照してください	<code>"in-process"</code>

権限設定

キー	説明	例
<code>allow</code>	ツール使用を許可する権限ルールの配列。パターンマッチングの詳細については、以下の 権限ルール構文 を参照してください	<code>["Bash(git diff *)"]</code>
<code>ask</code>	ツール使用時に確認を求める権限ルールの配列。パターンマッチングの詳細については、以下の 権限ルール構文 を参照してください	<code>["Bash(git push *)"]</code>
<code>deny</code>	ツール使用を拒否する権限ルールの配列。これを使用して、Claude Code アクセスから機密ファイルを除外します。 権限ルール構文 および Bash 権限制限 を参照してください	<code>["WebFetch", "Bash(curl *)", "Read(./env)", "Read(./secrets/**)"]</code>
<code>additionalDirectories</code>	Claude がアクセスできる追加の 作業ディレクトリ	<code>["../docs/"]</code>
<code>defaultMode</code>	Claude Code を開く際のデフォルト 権限モード	<code>"acceptEdits"</code>

キー	説明	例
<code>disableBypassPermissionsMode</code>	"disable" に設定して <code>bypassPermissions</code> モードの有効化を防止します。これにより <code>--dangerously-skip-permissions</code> コマンドラインフラグが無効になります。 Managed 設定 を参照してください	"disable"

権限ルール構文

権限ルールは `Tool` または `Tool(specifier)` の形式に従います。ルールは順序で評価されます：最初に deny ルール、次に ask、次に allow。最初に一致するルールが優先されます。

クイック例：

ルール	効果
<code>Bash</code>	すべての Bash コマンドに一致
<code>Bash(npm run *)</code>	<code>npm run</code> で始まるコマンドに一致
<code>Read(./ .env)</code>	<code>.env</code> ファイルの読み取りに一致
<code>WebFetch(domain:example.com)</code>	example.com へのフェッチリクエストに一致

ワイルドカード動作、Read、Edit、WebFetch、MCP、および Agent ルール用のツール固有パターン、および Bash パターンのセキュリティ制限を含む完全なルール構文リファレンスについては、[権限ルール構文](#)を参照してください。

サンドボックス設定

高度なサンドボックス動作を構成します。サンドボックスは bash コマンドをファイルシステムとネットワークから分離します。詳細については、[サンドボックス](#)を参照してください。

キー	説明	例
<code>enabled</code>	bash サンドボックスを有効にします（macOS、Linux、および WSL2）。デフォルト：false	<code>true</code>

キー	説明	例
<code>autoAllowBashIfSandboxed</code>	サンドボックス化されている場合、bash コマンドを自動承認します。デフォルト: true	<code>true</code>
<code>excludedCommands</code>	サンドボックスの外で実行する必要があるコマンド	<code>["git", "docker"]</code>
<code>allowUnsandboxedCommands</code>	<code>dangerouslyDisableSandbox</code> パラメーターを通じてサンドボックスの外でコマンドを実行することを許可します。false に設定すると、 <code>dangerouslyDisableSandbox</code> エスケープハッチは完全に無効になり、すべてのコマンドはサンドボックス化されるか、 <code>excludedCommands</code> に含まれる必要があります。厳密なサンドボックスを必要とするエンタープライズポリシーに役立ちます。デフォルト: true	<code>false</code>
<code>filesystem.allowWrite</code>	サンドボックス化されたコマンドが書き込みできる追加パス。配列はすべての設定スコープ全体でマージされます: ユーザー、プロジェクト、および Managed パスが結合され、置き換えられません。Edit(...) allow 権限ルールからのパスともマージされます。以下の パスプレフィックス を参照してください。	<code>["//tmp/build", "~/.kube"]</code>
<code>filesystem.denyWrite</code>	サンドボックス化されたコマンドが書き込みできないパス。配列はすべての設定スコープ全体でマージされます。Edit(...) deny 権限ルールからのパスともマージされます。	<code>["//etc", "//usr/local/bin"]</code>
<code>filesystem.denyRead</code>	サンドボックス化されたコマンドが読み取りできないパス。配列はすべての設定スコープ全体でマージされます。Read(...) deny 権限ルールからのパスともマージされます。	<code>["~/.aws/credentials"]</code>

キー	説明	例
<code>network.allowUnixSockets</code>	サンドボックスでアクセス可能な Unix ソケットパス (SSH エージェントなど)	<code>["~/.ssh/agent-socket"]</code>
<code>network.allowAllUnixSockets</code>	サンドボックス内のすべての Unix ソケット接続を許可します。デフォルト: false	<code>true</code>
<code>network.allowLocalBinding</code>	localhost ポートへのバインドを許可します (macOS のみ)。デフォルト: false	<code>true</code>
<code>network.allowedDomains</code>	アウトバウンドネットワークトラフィックを許可するドメインの配列。ワイルドカード (例: <code>*.example.com</code>) をサポートします。	<code>["github.com", "*.npmjs.org"]</code>
<code>network.allowManagedDomainsOnly</code>	(Managed 設定のみ) Managed 設定からの <code>allowedDomains</code> および <code>WebFetch(domain: ...)</code> allow ルールのみが尊重されます。ユーザー、プロジェクト、およびローカル設定からのドメインは無視されます。許可されていないドメインはユーザーにプロンプトを表示せずに自動的にブロックされます。拒否されたドメインはすべてのソースから尊重されます。デフォルト: false	<code>true</code>
<code>network.httpProxyPort</code>	独自のプロキシを持ち込む場合に使用する HTTP プロキシポート。指定されていない場合、Claude は独自のプロキシを実行します。	8080
<code>network.socksProxyPort</code>	独自のプロキシを持ち込む場合に使用する SOCKS5 プロキシポート。指定されていない場合、Claude は独自のプロキシを実行します。	8081

キー	説明	例
<code>enableWeakerNestedSan dbox</code>	非特権 Docker 環境でより弱いサンドボックスを有効にします（Linux および WSL2 のみ）。 セキュリティを低下させます 。デフォルト：false	<code>true</code>
<code>enableWeakerNetworkIs olation</code>	（macOS のみ）サンドボックス内のシステム TLS 信頼サービス（ <code>com.apple.trustd.agent</code> ）へのアクセスを許可します。 <code>gh</code> 、 <code>gcloud</code> 、 <code>terraform</code> などの Go ベースのツールが、MITM プロキシとカスタム CA を使用する <code>httpProxyPort</code> で TLS 証明書を検証するために必要です。 セキュリティを低下させます 。潜在的なデータ流出パスを開きます。デフォルト：false	<code>true</code>

サンドボックスパスプレフィックス

`filesystem.allowWrite`、`filesystem.denyWrite`、および `filesystem.denyRead` のパスは、これらのプレフィックスをサポートしています：

プレフィックス	意味	例
<code>//</code>	ファイルシステムルートからの絶対パス	<code>//tmp/build</code> は <code>/tmp/build</code> になります
<code>~/</code>	ホームディレクトリに相対的	<code>~/.kube</code> は <code>\$HOME/.kube</code> になります
<code>/</code>	設定ファイルのディレクトリに相対的	<code>/build</code> は <code>\$SETTINGS_DIR/build</code> になります
<code>./</code> またはプレフィックスなし	相対パス（サンドボックスランタイムで解決）	<code>./output</code>

構成例：

```
{
  "sandbox": {
    "enabled": true,
    "autoAllowBashIfSandboxed": true,
    "excludedCommands": ["docker"],
    "filesystem": {
      "allowWrite": ["/tmp/build", "~/kube"],
      "denyRead": ["/.aws/credentials"]
    },
    "network": {
      "allowedDomains": ["github.com", "*.npmjs.org", "registry.yarnpkg.com"],
      "allowUnixSockets": [
        "/var/run/docker.sock"
      ],
      "allowLocalBinding": true
    }
  }
}
```

ファイルシステムおよびネットワーク制限は、一緒にマージされる 2 つの方法で構成できます：

- **sandbox.filesystem 設定**（上記）：OS レベルのサンドボックス境界でパスを制御します。これらの制限は、Claude のファイルツールだけでなく、すべてのサブプロセスコマンド（例：`kubectl`、`terraform`、`npm`）に適用されます。
- **権限ルール**：`Edit` allow/deny ルールを使用して Claude のファイルツールアクセスを制御し、`Read` deny ルールを使用して読み取りをブロックし、`WebFetch` allow/deny ルールを使用してネットワークドメインを制御します。これらのルールからのパスもサンドボックス構成にマージされます。

属性設定

Claude Code は git コミットおよびプルリクエストに属性を追加します。これらは個別に構成されます：

- コミットはデフォルトで [git trailers](#)（`Co-Authored-By` など）を使用し、カスタマイズまたは無効にできます
- プルリクエストの説明はプレーンテキストです

キー	説明
<code>commit</code>	git コミットの属性。任意の trailers を含みます。空の文字列はコミット属性を非表示にします
<code>pr</code>	プルリクエストの説明の属性。空の文字列はプルリクエスト属性を非表示にします

デフォルトコミット属性：

 Generated with [Claude Code](https://claude.com/claude-code)

Co-Authored-By: Claude Sonnet 4.6 <noreply@anthropic.com>

デフォルトプルリクエスト属性：

 Generated with [Claude Code](https://claude.com/claude-code)

例：

```
{
  "attribution": {
    "commit": "Generated with AI\n\nCo-Authored-By: AI <ai@example.com>",
    "pr": ""
  }
}
```

Note:

`attribution` 設定は非推奨の `includeCoAuthoredBy` 設定よりも優先されます。すべての属性を非表示にするには、`commit` および `pr` を空の文字列に設定します。

ファイル提案設定

@ ファイルパスオートコンプリート用のカスタムコマンドを構成します。組み込みファイル提案は高速ファイルシステムトラバーサルを使用しますが、大規模なモノレポは事前構築されたファイルインデックスやカスタムツールなどのプロジェクト固有のインデックスから利益を得る可能性があります。

```
{
  "fileSuggestion": {
    "type": "command",
    "command": "~/claude/file-suggestion.sh"
  }
}
```

コマンドは [hooks](#) と同じ環境変数で実行され、`CLAUDE_PROJECT_DIR` を含みます。 `query` フィールドを含む JSON を stdin 経由で受け取ります：

```
{"query": "src/comp"}
```

stdout にニューラインで区切られたファイルパスを出力します（現在 15 に制限）：

```
src/components/Button.tsx
src/components/Modal.tsx
src/components/Form.tsx
```

例：

```
#!/bin/bash
query=$(cat | jq -r '.query')
your-repo-file-index --query "$query" | head -20
```

Hook 構成

これらの設定は、実行が許可される hooks と HTTP hooks がアクセスできるものを制御します。 `allowManagedHooksOnly` 設定は [Managed 設定](#) でのみ構成できます。URL および環境変数ホワイトリストは任意の設定レベルで設定でき、ソース全体でマージされます。

`allowManagedHooksOnly` が `true` の場合の動作：

- Managed hooks および SDK hooks が読み込まれます
- ユーザー hooks、プロジェクト hooks、およびプラグイン hooks はブロックされます

HTTP hook URL を制限：

HTTP hooks がターゲットにできる URL を制限します。マッチングのワイルドカードとして `*` をサポートします。配列が定義されている場合、一致しない URL をターゲットにする HTTP hooks はサイレントにブロックされます。

```
{
  "allowedHttpHookUrls": ["https://hooks.example.com/*", "http://localhost:*"]
}
```

HTTP hook 環境変数を制限：

HTTP hooks がヘッダー値に補間できる環境変数名を制限します。各 hook の有効な `allowedEnvVars` はその独自のリストとこの設定の交差です。

```
{
  "httpHookAllowedEnvVars": ["MY_TOKEN", "HOOK_SECRET"]
}
```

設定優先度

設定は優先度の順に適用されます。最高から最低：

1. **Managed 設定** ([サーバー管理](#)、[MDM/OS レベルのポリシー](#)、または [Managed 設定](#))
 - IT がサーバー配信、MDM 構成プロファイル、レジストリポリシー、または Managed 設定ファイルを通じてデプロイしたポリシー
 - コマンドラインの引数を含む他のレベルによってオーバーライドできません
 - Managed ティア内では、優先度は：サーバー管理 > MDM/OS レベルのポリシー > `managed-settings.json` > HKCU レジストリ (Windows のみ) です。1 つの Managed ソースのみが使用されます。ソースはマージされません。
2. **コマンドラインの引数**
 - 特定のセッションの一時的なオーバーライド
3. **ローカルプロジェクト設定** (`.claude/settings.local.json`)
 - 個人的なプロジェクト固有の設定
4. **共有プロジェクト設定** (`.claude/settings.json`)
 - ソース管理のチーム共有プロジェクト設定

5. ユーザー設定 (`~/.claude/settings.json`)

- 個人的なグローバル設定

この階層は、組織のポリシーが常に実施されることを保証しながら、チームと個人が経験をカスタマイズできるようにします。

たとえば、ユーザー設定が `Bash(npm run *)` を許可しているが、プロジェクトの共有設定がそれを拒否している場合、プロジェクト設定が優先され、コマンドはブロックされます。

Note:

配列設定はスコープ全体でマージされます。 `sandbox.filesystem.allowWrite` または `permissions.allow` などの同じ配列値設定が複数のスコープに表示される場合、配列は連結および重複排除され、置き換えられません。これは、低優先度のスコープが高優先度のスコープで設定されたエントリをオーバーライドすることなくエントリを追加でき、その逆も同様です。たとえば、Managed 設定が `allowWrite` を `["//opt/company-tools"]` に設定し、ユーザーが `["~/.kube"]` を追加する場合、両方のパスが最終構成に含まれます。

アクティブな設定を確認

Claude Code 内で `/status` を実行して、どの設定ソースがアクティブで、どこから来ているかを確認します。出力は、`Enterprise managed settings (remote)`、`Enterprise managed settings (plist)`、`Enterprise managed settings (HKLM)`、または `Enterprise managed settings (file)` などの起源を含む各構成レイヤー (Managed、ユーザー、プロジェクト) を表示します。設定ファイルにエラーが含まれている場合、`/status` は問題を報告して修正できるようにします。

構成システムに関する重要なポイント

- **メモリファイル (`CLAUDE.md`)** : Claude が起動時に読み込む命令とコンテキストを含みます
- **設定ファイル (JSON)** : 権限、環境変数、およびツール動作を構成します
- **スキル** : `/skill-name` で呼び出さるか、Claude によって自動的に読み込むことができるカスタムプロンプト
- **MCP サーバー** : 追加のツールと統合で Claude Code を拡張します
- **優先度** : 高レベルの構成 (Managed) が低レベルの構成 (User/Project) をオーバーライドします
- **継承** : 設定はマージされ、より具体的な設定がより広い設定に追加またはオーバーライドされます

システムプロンプト

Claude Code の内部システムプロンプトは公開されていません。カスタム命令を追加するには、`CLAUDE.md` ファイルまたは `--append-system-prompt` フラグを使用してください。

機密ファイルを除外

API キー、シークレット、環境ファイルなどの機密情報を含むファイルへの Claude Code アクセスを防ぐには、`.claude/settings.json` ファイルで `permissions.deny` 設定を使用してください：

```
{
  "permissions": {
    "deny": [
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)",
      "Read(./config/credentials.json)",
      "Read(./build)"
    ]
  }
}
```

これは非推奨の `ignorePatterns` 構成に置き換わります。これらのパターンに一致するファイルはファイル検出と検索結果から除外され、これらのファイルの読み取り操作は拒否されます。

Subagent 構成

Claude Code は、カスタム AI subagent をサポートしており、ユーザーおよびプロジェクトレベルで構成できます。これらの subagent は YAML frontmatter を含む Markdown ファイルとして保存されます：

- **ユーザー subagent**：`~/.claude/agents/` - すべてのプロジェクト全体で利用可能
- **プロジェクト subagent**：`.claude/agents/` - プロジェクトに固有で、チームと共有できます

Subagent ファイルは、カスタムプロンプトとツール権限を持つ特殊な AI アシスタントを定義します。subagent の作成と使用の詳細については、[subagent ドキュメント](#)を参照してください。

プラグイン構成

Claude Code は、スキル、エージェント、hooks、および MCP サーバーで機能を拡張できるプラグインシステムをサポートしています。プラグインはマーケットプレイスを通じて配布され、ユーザーおよびリポジトリレベルで構成できます。

プラグイン設定

`settings.json` のプラグイン関連設定：

```
{
  "enabledPlugins": {
    "formatter@acme-tools": true,
    "deployer@acme-tools": true,
    "analyzer@security-plugins": false
  },
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": "github",
      "repo": "acme-corp/claude-plugins"
    }
  }
}
```

`enabledPlugins`

どのプラグインが有効かを制御します。形式：`"plugin-name@marketplace-name": true/false`

スコープ：

- **ユーザー設定**（`~/.claude/settings.json`）：個人的なプラグイン設定
- **プロジェクト設定**（`.claude/settings.json`）：チームと共有されるプロジェクト固有のプラグイン
- **ローカル設定**（`.claude/settings.local.json`）：マシンごとのオーバーライド（コミットされません）

例：

```
{
  "enabledPlugins": {
    "code-formatter@team-tools": true,
    "deployment-tools@team-tools": true,
    "experimental-features@personal": false
  }
}
```

extraKnownMarketplaces

リポジトリで利用可能にする必要がある追加マーケットプレイスを定義します。通常、チームメンバーが必要なプラグインソースにアクセスできるようにするためにリポジトリレベルの設定で使用されます。

リポジトリが **extraKnownMarketplaces** を含む場合：

1. チームメンバーはフォルダを信頼するときにマーケットプレイスをインストールするよう求められます
2. チームメンバーはそのマーケットプレイスからプラグインをインストールするよう求められます
3. ユーザーは不要なマーケットプレイスまたはプラグインをスキップできます（ユーザー設定に保存）
4. インストールは信頼境界を尊重し、明示的な同意が必要です

例：

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/claude-plugins"
      }
    },
    "security-plugins": {
      "source": {
        "source": "git",
        "url": "https://git.example.com/security/plugins.git"
      }
    }
  }
}
```

マーケットプレイスソースタイプ：

- `github`：GitHub リポジトリ（`repo` を使用）
- `git`：任意の git URL（`url` を使用）
- `directory`：ローカルファイルシステムパス（`path` を使用、開発のみ）
- `hostPattern`：マーケットプレイスホストに一致する正規表現パターン（`hostPattern` を使用）

`strictKnownMarketplaces`

Managed 設定のみ：ユーザーが追加できるプラグインマーケットプレイスを制御します。この設定は **Managed 設定** でのみ構成でき、管理者にマーケットプレイスソースに対する厳密な制御を提供します。

Managed 設定ファイルの場所：

- **macOS**：`/Library/Application Support/ClaudeCode/managed-settings.json`
- **Linux および WSL**：`/etc/claude-code/managed-settings.json`
- **Windows**：`C:\Program Files\ClaudeCode\managed-settings.json`

主な特性：

- Managed 設定（`managed-settings.json`）でのみ利用可能

- ユーザーまたはプロジェクト設定によってオーバーライドできません（最高優先度）
- ネットワーク/ファイルシステム操作の前に実施されます（ブロックされたソースは実行されません）
- ソース仕様の完全一致を使用します（git ソースの `ref`、`path` を含む）。ただし、`hostPattern` は正規表現マッチングを使用します

ホワイトリスト動作：

- `undefined`（デフォルト）：制限なし - ユーザーは任意のマーケットプレイスを追加できます
- 空配列 `[]`：完全ロックダウン - ユーザーは新しいマーケットプレイスを追加できません
- ソースのリスト：ユーザーは正確に一致するマーケットプレイスのみを追加できます

サポートされているすべてのソースタイプ：

ホワイトリストは7つのマーケットプレイスソースタイプをサポートしています。ほとんどのソースは完全一致を使用しますが、`hostPattern` はマーケットプレイスホストに対して正規表現マッチングを使用します。

1. GitHub リポジトリ：

```
{ "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" }
```

フィールド：`repo`（必須）、`ref`（オプション：ブランチ/タグ/SHA）、`path`（オプション：サブディレクトリ）

1. Git リポジトリ：

```
{ "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" }
```

フィールド: `url` (必須)、`ref` (オプション: ブランチ/タグ/SHA)、`path` (オプション: サブディレクトリ)

1. URL ベースのマーケットプレイス:

```
{ "source": "url", "url": "https://plugins.example.com/marketplace.json" }
{ "source": "url", "url": "https://cdn.example.com/marketplace.json", "headers":
{ "Authorization": "Bearer ${TOKEN}" } }
```

フィールド: `url` (必須)、`headers` (オプション: 認証アクセス用の HTTP ヘッダー)

Note:

URL ベースのマーケットプレイスは `marketplace.json` ファイルのみをダウンロードします。サーバーからプラグインファイルをダウンロードしません。URL ベースのマーケットプレイス内のプラグインは、相対パスではなく外部ソース (GitHub、npm、または git URL) を使用する必要があります。相対パスを持つプラグインの場合は、代わりに Git ベースのマーケットプレイスを使用してください。[トラブルシューティング](#)を参照してください。

1. NPM パッケージ:

```
{ "source": "npm", "package": "@acme-corp/claude-plugins" }
{ "source": "npm", "package": "@acme-corp/approved-marketplace" }
```

フィールド: `package` (必須、スコープ付きパッケージをサポート)

1. ファイルパス:

```
{ "source": "file", "path": "/usr/local/share/claude/acme-marketplace.json" }
{ "source": "file", "path": "/opt/acme-corp/plugins/marketplace.json" }
```

フィールド: `path` (必須: marketplace.json ファイルへの絶対パス)

1. ディレクトリパス:

```
{ "source": "directory", "path": "/usr/local/share/claude/acme-plugins" }
{ "source": "directory", "path": "/opt/acme-corp/approved-marketplaces" }
```

フィールド: `path` (必須: `.claude-plugin/marketplace.json` を含むディレクトリへの絶対パス)

1. ホストパターンマッチング:

```
{ "source": "hostPattern", "hostPattern": "^github\\.example\\.com$" }  
{ "source": "hostPattern", "hostPattern": "^gitlab\\.internal\\.example\\.com$" }
```

フィールド: `hostPattern` (必須: マーケットプレイスホストに対してマッチする正規表現パターン)

各リポジトリを列挙することなく、特定のホストからすべてのマーケットプレイスを許可する場合は、ホストパターンマッチングを使用してください。これは、開発者が独自のマーケットプレイスを作成する内部 GitHub Enterprise または GitLab サーバーを持つ組織に役立ちます。

ソースタイプ別のホスト抽出:

- `github`: 常に `github.com` に対してマッチ
- `git`: URL からホスト名を抽出 (HTTPS および SSH 形式をサポート)
- `url`: URL からホスト名を抽出
- `npm`、`file`、`directory`: ホストパターンマッチングではサポートされていません

構成例:

例: 特定のマーケットプレイスのみを許可:

```
{
  "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"
    }
  ]
}
```

例 - すべてのマーケットプレイス追加を無効にする：

```
{
  "strictKnownMarketplaces": []
}
```

例：内部 git サーバーからすべてのマーケットプレイスを許可：


```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

完全一致要件：

マーケットプレイスソースはユーザーの追加を許可するために**正確に一致**する必要があります。git ベースのソース（github および git）の場合、これはすべてのオプションフィールドを含みます：

- repo または url は正確に一致する必要があります
- ref フィールドは正確に一致する必要があります（または両方が未定義）
- path フィールドは正確に一致する必要があります（または両方が未定義）

一致しないソースの例：

```
// これらは異なるソースです：
{ "source": "github", "repo": "acme-corp/plugins" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main" }

// これらも異なります：
{ "source": "github", "repo": "acme-corp/plugins", "path": "marketplace" }
{ "source": "github", "repo": "acme-corp/plugins" }
```

extraKnownMarketplaces との比較：

側面	strictKnownMarketplaces	extraKnownMarketplaces
目的	組織ポリシーの実施	チームの利便性
設定ファイル	managed-settings.json のみ	任意の設定ファイル
動作	許可されていない追加をブロック	不足しているマーケットプレイスを自動インストール

側面	<code>strictKnownMarketplaces</code>	<code>extraKnownMarketplaces</code>
実施時期	ネットワーク/ファイルシステム操作の前	ユーザー信頼プロンプト後
オーバーライド可能か	いいえ（最高優先度）	はい（高優先度設定による）
ソース形式	直接ソースオブジェクト	名前付きマーケットプレイスとネストされたソース
ユースケース	コンプライアンス、セキュリティ制限	オンボーディング、標準化

形式の違い：

`strictKnownMarketplaces` は直接ソースオブジェクトを使用します：

```
{
  "strictKnownMarketplaces": [
    { "source": "github", "repo": "acme-corp/plugins" }
  ]
}
```

`extraKnownMarketplaces` は名前付きマーケットプレイスが必要です：

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": { "source": "github", "repo": "acme-corp/plugins" }
    }
  }
}
```

重要な注意：

- ・制限はネットワークリクエストまたはファイルシステム操作の前にチェックされます
- ・ブロックされた場合、ユーザーは Managed ポリシーによってソースがブロックされていることを示す明確なエラーメッセージを表示します

- ・制限は新しいマーケットプレイスの追加にのみ適用されます。以前にインストールされたマーケットプレイスはアクセス可能なままです
- ・Managed 設定は最高優先度を持ち、オーバーライドできません

ユーザー向けドキュメントについては、[Managed マーケットプレイス制限](#)を参照してください。

プラグインの管理

`/plugin` コマンドを使用してプラグインを対話的に管理します：

- ・マーケットプレイスから利用可能なプラグインを参照
- ・プラグインをインストール/アンインストール
- ・プラグインを有効/無効にする
- ・プラグインの詳細を表示（提供されるコマンド、エージェント、hooks）
- ・マーケットプレイスを追加/削除

プラグインシステムの詳細については、[プラグインドキュメント](#)を参照してください。

環境変数

Claude Code は、その動作を制御するために以下の環境変数をサポートしています：

Note:

すべての環境変数は `settings.json` でも構成できます。これは、各セッションの環境変数を自動的に設定するか、チーム全体または組織全体に環境変数のセットをロールアウトする方法として役立ちます。

変数	目的	
<code>ANTHROPIC_API_KEY</code>	<code>X-Api-Key</code> ヘッダーとして送信される API キー。通常は Claude SDK 用（インタラクティブ使用の場合は <code>/login</code> を実行）	
<code>ANTHROPIC_AUTH_TOKEN</code>	<code>Authorization</code> ヘッダーのカスタム値（ここで設定した値には <code>Bearer</code> が付加されます）	

変数	目的	
<code>ANTHROPIC_CUSTOM_HEADERS</code>	リクエストに追加するカスタムヘッダー（ <code>Name: Value</code> 形式、複数のヘッダーの場合は newline で区切られます）	
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL</code>	モデル構成 を参照	
<code>ANTHROPIC_DEFAULT_OPUS_MODEL</code>	モデル構成 を参照	
<code>ANTHROPIC_DEFAULT_SONNET_MODEL</code>	モデル構成 を参照	
<code>ANTHROPIC_FOUNDRY_API_KEY</code>	Microsoft Foundry 認証用の API キー（ Microsoft Foundry を参照）	
<code>ANTHROPIC_FOUNDRY_BASE_URL</code>	Foundry リソースの完全なベース URL（例： <code>https://my-resource.services.ai.azure.com/anthropic</code> ）。 <code>ANTHROPIC_FOUNDRY_RESOURCE</code> の代替（ Microsoft Foundry を参照）	
<code>ANTHROPIC_FOUNDRY_RESOURCE</code>	Foundry リソース名（例： <code>my-resource</code> ）。 <code>ANTHROPIC_FOUNDRY_BASE_URL</code> が設定されていない場合は必須（ Microsoft Foundry を参照）	
<code>ANTHROPIC_MODEL</code>	使用するモデル設定の名前（ モデル構成 を参照）	
<code>ANTHROPIC_SMALL_FAST_MODEL</code>	[非推奨] バックグラウンドタスク用の Haiku クラスモデル の名前	
<code>ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION</code>	Bedrock を使用する場合、Haiku クラスモデルの AWS リージョンをオーバーライド	
<code>AWS_BEARER_TOKEN_BEDROCK</code>	Bedrock 認証用の API キー（ Bedrock API キー を参照）	

変数	目的	
<code>BASH_DEFAULT_TIMEOUT_MS</code>	長時間実行される bash コマンドのデフォルトタイムアウト	
<code>BASH_MAX_OUTPUT_LENGTH</code>	中央で切り詰められる前の bash 出力の最大文字数	
<code>BASH_MAX_TIMEOUT_MS</code>	モデルが長時間実行される bash コマンドに設定できる最大タイムアウト	
<code>CLAUDE_AUTOCOMPACT_PERCENT_OVERRIDE</code>	オートコンパクションがトリガーされるコンテキスト容量の割合（1～100）を設定します。デフォルトでは、オートコンパクションは約 95% の容量でトリガーされます。<code>50</code> などの低い値を使用して早期にコンパクトします。デフォルトしきい値より高い値は効果がありません。メイン会話と subagent の両方に適用されます。この割合は status line で利用可能な <code>context_window.used_percentage</code> フィールドと一致します	
<code>CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR</code>	各 Bash コマンド後に元の作業ディレクトリに戻ります	
<code>CLAUDE_CODE_ACCOUNT_UUID</code>	認証されたユーザーのアカウント UUID。SDK 呼び出し元が同期的にアカウント情報を提供するために使用され、初期のテレメトリイベントがアカウントメタデータを欠落する競合状態を回避します。 <code>CLAUDE_CODE_USER_EMAIL</code> および <code>CLAUDE_CODE_ORGANIZATION_UUID</code> も設定する必要があります	
<code>CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD</code>	<code>--add-dir</code> で指定されたディレクトリから CLAUDE.md ファイルを読み込むには 1 に設定します。デフォルトでは、追加ディレクトリはメモリファイルを読み込みません	1

変数	目的	
<code>CLAUDE_CODE_API_KEY_HELPER_TTL_MS</code>	認証情報をリフレッシュする間隔（ミリ秒）（ <code>apiKeyHelper</code> を使用する場合）	
<code>CLAUDE_CODE_CLIENT_CERT</code>	mTLS 認証用のクライアント証明書ファイルへのパス	
<code>CLAUDE_CODE_CLIENT_KEY</code>	mTLS 認証用のクライアント秘密鍵ファイルへのパス	
<code>CLAUDE_CODE_CLIENT_KEY_PASSPHRASE</code>	暗号化された <code>CLAUDE_CODE_CLIENT_KEY</code> のパスフレーズ（オプション）	
<code>CLAUDE_CODE_DISABLE_1M_CONTEXT</code>	1M コンテキストウィンドウサポート を無効にするには <code>1</code> に設定します。設定されている場合、1M モデルバリエーションはモデルピッカーで利用できません。コンプライアンス要件を持つエンタープライズ環境に役立ちます	
<code>CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING</code>	Opus 4.6 および Sonnet 4.6 の 適応推論 を無効にするには <code>1</code> に設定します。無効にされた場合、これらのモデルは <code>MAX_THINKING_TOKENS</code> で制御される固定思考予算にフォールバックします	
<code>CLAUDE_CODE_DISABLE_AUTOMEMORY</code>	自動メモリ を無効にするには <code>1</code> に設定します。段階的なロールアウト中に自動メモリを強制的にオンにするには <code>0</code> に設定します。無効にされた場合、Claude は自動メモリファイルを作成または読み込みません	

変数	目的	
<code>CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS</code>	Claude のシステムプロンプトから組み込みコミットおよび PR ワークフロー命令を削除するには 1 に設定します。独自の git ワークフロースキルを使用する場合に役立ちます。設定されている場合、 <code>includeGitInstructions</code> 設定よりも優先されます	
<code>CLAUDE_CODE_DISABLE_BACKGROUND_TASKS</code>	Bash および subagent ツールの <code>run_in_background</code> パラメーター、自動バックグラウンド化、Ctrl+B ショートカットを含むすべてのバックグラウンドタスク機能を無効にするには 1 に設定します	
<code>CLAUDE_CODE_DISABLE_CRON</code>	スケジュール済みタスク を無効にするには 1 に設定します。 <code>/loop</code> スキルと cron ツールが利用できなくなり、既にスケジュール済みのタスク（実行中のタスクを含む）が停止します	
<code>CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS</code>	Anthropic API 固有の <code>anthropic-beta</code> ヘッダーを無効にするには 1 に設定します。サードパーティプロバイダーを持つ LLM ゲートウェイを使用する場合、「Unexpected value(s) for the <code>anthropic-beta</code> header」などの問題が発生している場合に使用します	
<code>CLAUDE_CODE_DISABLE_FAST_MODE</code>	高速モード を無効にするには 1 に設定します	
<code>CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY</code>	「Claude はどのように機能していますか？」セッション品質調査を無効にするには 1 に設定します。サードパーティプロバイダーを使用する場合またはテレメトリが無効な場合も無効になります。 セッション品質調査 を参照してください	

変数	目的	
<code>CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC</code>	<code>DISABLE_AUTOUPDATER</code> 、 <code>DISABLE_BUG_COMMAND</code> 、 <code>DISABLE_ERROR_REPORTING</code> 、および <code>DISABLE_TELEMETRY</code> を設定するのと同様	
<code>CLAUDE_CODE_DISABLE_TERMINAL_TITLE</code>	会話コンテキストに基づいて自動ターミナルタイトル更新を無効にするには <code>1</code> に設定します	
<code>CLAUDE_CODE_EFFORT_LEVEL</code>	サポートされているモデルの努力レベルを設定します。値: <code>low</code> 、 <code>medium</code> 、 <code>high</code> 。低い努力はより速く、より安く、高い努力はより深い推論を提供します。Opus 4.6 および Sonnet 4.6 でサポートされています。 努力レベルを調整 を参照してください	
<code>CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION</code>	プロンプト提案を無効にするには <code>false</code> に設定します (<code>/config</code> の「プロンプト提案」トグル)。これらは Claude が応答した後、プロンプト入力に表示される灰色の予測です。 プロンプト提案 を参照してください	
<code>CLAUDE_CODE_ENABLE_TASKS</code>	以前の TODO リストに一時的に戻すには <code>false</code> に設定します。デフォルト: <code>true</code> 。 タスクリスト を参照してください	
<code>CLAUDE_CODE_ENABLE_TELEMETRY</code>	メトリクスとログの OpenTelemetry データ収集を有効にするには <code>1</code> に設定します。OTel エクスポートを構成する前に必須です。 監視 を参照してください	

変数	目的	
<code>CLAUDE_CODE_EXIT_AFTER_STOP_DELAY</code>	クエリループがアイドル状態になった後、自動的に終了するまで待機する時間（ミリ秒）。自動化されたワークフローおよび SDK モードを使用するスクリプトに役立ちます	
<code>CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS</code>	エージェントチーム を有効にするには <code>1</code> に設定します。エージェントチームは実験的で、デフォルトで無効です	
<code>CLAUDE_CODE_FILE_READ_MAX_OUTPUT_TOKENS</code>	ファイル読み取りのデフォルトトークン制限をオーバーライドします。より大きなファイルを完全に読み取る必要がある場合に役立ちます	
<code>CLAUDE_CODE_HIDE_ACCOUNT_INFO</code>	メールアドレスと組織名を Claude Code UI から非表示にするには <code>1</code> に設定します。ストリーミングまたは記録時に役立ちます	
<code>CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL</code>	IDE 拡張機能の自動インストールをスキップ	
<code>CLAUDE_CODE_MAX_OUTPUT_TOKENS</code>	ほとんどのリクエストの最大出力トークン数を設定します。デフォルト：32,000。最大：64,000。この値を増やすと、 オートコンパクション がトリガーされる前に利用可能な有効なコンテキストウィンドウが削減されます。	
<code>CLAUDE_CODE_ORGANIZATION_UUID</code>	認証されたユーザーの組織 UUID。SDK 呼び出し元が同期的にアカウント情報を提供するために使用されます。 <code>CLAUDE_CODE_ACCOUNT_UUID</code> および <code>CLAUDE_CODE_USER_EMAIL</code> も設定する必要があります	
<code>CLAUDE_CODE_OTEL_HELPERS_DEBOUNCE_MS</code>	動的 OpenTelemetry ヘッダーをリフレッシュする間隔（ミリ秒）（デフォルト：1740000 / 29 分）。 動的ヘッダー を参照してください	

変数	目的	
<code>CLAUDE_CODE_PLAN_MODE_REQUIRED</code>	エージェントチーム チームメイトがプラン承認を必要とする場合、<code>true</code> に自動設定されます。読み取り専用：チームメイトをスポンするときに Claude Code によって設定されます。チームメイトのプラン承認を要求を参照してください	
<code>CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS</code>	プラグインをインストールまたは更新するときの git 操作のタイムアウト（ミリ秒）（デフォルト：120000）。大規模なリポジトリまたは遅いネットワーク接続の場合、この値を増やします。Git 操作がタイムアウトを参照してください	
<code>CLAUDE_CODE_PROXY_RESOLVES_HOSTS</code>	プロキシが呼び出し元の代わりに DNS 解決を実行できるようにするには <code>true</code> に設定します。プロキシがホスト名解決を処理する必要がある環境のオプション	
<code>CLAUDE_CODE_SHELL</code>	自動シェル検出をオーバーライドします。ログインシェルが優先作業シェルと異なる場合に役立ちます（例：<code>bash</code> vs <code>zsh</code>）	
<code>CLAUDE_CODE_SHELL_PREFIX</code>	すべての bash コマンドをラップするコマンドプレフィックス（例：ログまたは監査用）。例：<code>/path/to/logger.sh</code> は <code>/path/to/logger.sh <command></code> を実行します	
<code>CLAUDE_CODE_SIMPLE</code>	最小限のシステムプロンプトと Bash、ファイル読み取り、ファイル編集ツールのみで実行するには <code>1</code> に設定します。MCP ツール、添付ファイル、hooks、および CLAUDE.md ファイルを無効にします	

変数	目的	
<code>CLAUDE_CODE_SKIP_BEDROCK_AUTH</code>	Bedrock の AWS 認証をスキップします (例: LLM ゲートウェイを使用する場合)	
<code>CLAUDE_CODE_SKIP_FOUNDRY_AUTH</code>	Microsoft Foundry の Azure 認証をスキップします (例: LLM ゲートウェイを使用する場合)	
<code>CLAUDE_CODE_SKIP_VERTEX_AUTH</code>	Vertex の Google 認証をスキップします (例: LLM ゲートウェイを使用する場合)	
<code>CLAUDE_CODE_SUBAGENT_MODEL</code>	モデル構成 を参照	
<code>CLAUDE_CODE_TASK_LIST_ID</code>	セッション全体でタスクリストを共有します。複数の Claude Code インスタンスで同じ ID を設定して、共有タスクリストで調整します。 タスクリスト を参照してください	
<code>CLAUDE_CODE_TEAM_NAME</code>	このチームメイトが属するエージェントチームの名前。 エージェントチーム メンバーで自動的に設定されます	
<code>CLAUDE_CODE_TMPDIR</code>	内部一時ファイルに使用される一時ディレクトリをオーバーライドします。Claude Code はこのパスに <code>/claude/</code> を追加します。デフォルト: Unix/macOS では <code>/tmp</code> 、Windows では <code>os.tmpdir()</code>	
<code>CLAUDE_CODE_USER_EMAIL</code>	認証されたユーザーのメールアドレス。SDK 呼び出し元が同期的にアカウント情報を提供するために使用されます。 <code>CLAUDE_CODE_ACCOUNT_UUID</code> および <code>CLAUDE_CODE_ORGANIZATION_UUID</code> も設定する必要があります	

変数	目的	
<code>CLAUDE_CODE_USE_BEDROCK</code>	Bedrock を使用	
<code>CLAUDE_CODE_USE_FOUNDRY</code>	Microsoft Foundry を使用	
<code>CLAUDE_CODE_USE_VERTEX</code>	Vertex を使用	
<code>CLAUDE_CONFIG_DIR</code>	Claude Code が構成とデータファイルを保存する場所をカスタマイズ	
<code>DISABLE_AUTOUPDATER</code>	自動更新を無効にするには 1 に設定します。	
<code>DISABLE_BUG_COMMAND</code>	<code>/bug</code> コマンドを無効にするには 1 に設定します	
<code>DISABLE_COST_WARNINGS</code>	コスト警告メッセージを無効にするには 1 に設定します	
<code>DISABLE_ERROR_REPORTING</code>	Sentry エラーレポートをオプトアウトするには 1 に設定します	
<code>DISABLE_INSTALLATION_CHECKS</code>	インストール警告を無効にするには 1 に設定します。インストール場所を手動で管理する場合のみ使用してください。標準インストールの問題をマスクできます	
<code>DISABLE_NON_ESSENTIAL_MODEL_CALLS</code>	フレーバーテキストなどの非重要パスのモデル呼び出しを無効にするには 1 に設定します	
<code>DISABLE_PROMPT_CACHING</code>	すべてのモデルのプロンプトキャッシングを無効にするには 1 に設定します（モデルごとの設定よりも優先）	
<code>DISABLE_PROMPT_CACHING_HAIKU</code>	Haiku モデルのプロンプトキャッシングを無効にするには 1 に設定します	

変数	目的	
<code>DISABLE_PROMPT_CACHING_OPUS</code>	Opus モデルのプロンプトキャッシュを無効にするには <code>1</code> に設定します	
<code>DISABLE_PROMPT_CACHING_SONNET</code>	Sonnet モデルのプロンプトキャッシュを無効にするには <code>1</code> に設定します	
<code>DISABLE_TELEMETRY</code>	Statsig テレメトリをオプトアウトするには <code>1</code> に設定します (Statsig イベントにはコード、ファイルパス、bash コマンドなどのユーザーデータが含まれないことに注意)	
<code>ENABLE_CLAUDEAI_MCP_SERVERS</code>	Claude Code で claude.ai MCP サーバー を無効にするには <code>false</code> に設定します。ログインしているユーザーに対してデフォルトで有効	
<code>ENABLE_TOOL_SEARCH</code>	MCP ツール検索 を制御します。値： <code>auto</code> (デフォルト、10% コンテキストで有効)、 <code>auto:N</code> (カスタムしきい値、例：5% の場合は <code>auto:5</code>)、 <code>true</code> (常にオン)、 <code>false</code> (無効)	
<code>FORCE_AUTOUPDATE_PLUGINS</code>	メイン自動アップdater が <code>DISABLE_AUTOUPDATER</code> を通じて無効になっている場合でも、プラグイン自動更新を強制するには <code>true</code> に設定します	
<code>HTTP_PROXY</code>	ネットワーク接続用の HTTP プロキシサーバーを指定	
<code>HTTPS_PROXY</code>	ネットワーク接続用の HTTPS プロキシサーバーを指定	

変数	目的	
<code>IS_DEMO</code>	デモモードを有効にするには <code>true</code> に設定します：UI からメールと組織を非表示にし、オンボーディングをスキップし、内部コマンドを非表示にします。ストリーミングまたは記録セッションに役立ちます	
<code>MAX_MCP_OUTPUT_TOKENS</code>	MCP ツール応答で許可される最大トークン数。出力が 10,000 トークンを超える場合、Claude Code は警告を表示します（デフォルト：25000）	
<code>MAX_THINKING_TOKENS</code>	拡張思考 トークン予算をオーバーライドします。思考はデフォルトで最大予算（31,999 トークン）で有効になります。予算を制限するには（例： <code>MAX_THINKING_TOKENS=10000</code> ） または思考を完全に無効にするには（ <code>MAX_THINKING_TOKENS=0</code> ）これを使用します。Opus 4.6 の場合、思考の深さは 努力レベル で制御され、 <code>0</code> に設定して思考を無効にしない限り、この変数は無視されます。	
<code>MCP_CLIENT_SECRET</code>	事前構成された認証情報 を必要とする MCP サーバーの OAuth クライアントシークレット。 <code>--client-secret</code> でサーバーを追加するときにインタラクティブプロンプトを回避します	
<code>MCP_OAUTH_CALLBACK_PORT</code>	事前構成された認証情報 で MCP サーバーを追加する場合、 <code>--callback-port</code> の代替として OAuth リダイレクトコールバック用の固定ポート	
<code>MCP_TIMEOUT</code>	MCP サーバー起動のタイムアウト（ミリ秒）	
<code>MCP_TOOL_TIMEOUT</code>	MCP ツール実行のタイムアウト（ミリ秒）	

変数	目的	
<code>NO_PROXY</code>	プロキシをバイパスして直接発行されるリクエストのドメインと IP のリスト	
<code>SLASH_COMMAND_TOOL_CHARACTER_BUDGET</code>	Skill ツール に表示されるスキルメタデータの文字予算をオーバーライドします。予算はコンテキストウィンドウの 2% で動的にスケーリングされ、16,000 文字のフォールバックがあります。後方互換性のために古い名前が保持されます	
<code>USE_BUILTIN_RIPGREP</code>	Claude Code に含まれる <code>rg</code> の代わりにシステムインストール済みの <code>rg</code> を使用するには <code>0</code> に設定します	
<code>VERTEX_REGION_CLAUDE_3_5_HAIKU</code>	Vertex AI を使用する場合、Claude 3.5 Haiku のリージョンをオーバーライド	
<code>VERTEX_REGION_CLAUDE_3_7_SONNET</code>	Vertex AI を使用する場合、Claude 3.7 Sonnet のリージョンをオーバーライド	
<code>VERTEX_REGION_CLAUDE_4_0_OPUS</code>	Vertex AI を使用する場合、Claude 4.0 Opus のリージョンをオーバーライド	
<code>VERTEX_REGION_CLAUDE_4_0_SONNET</code>	Vertex AI を使用する場合、Claude 4.0 Sonnet のリージョンをオーバーライド	
<code>VERTEX_REGION_CLAUDE_4_1_OPUS</code>	Vertex AI を使用する場合、Claude 4.1 Opus のリージョンをオーバーライド	

Claude が利用できるツール

Claude Code は、コードベースを理解および変更するのに役立つ強力なツールセットにアクセスできます：

ツール	説明	権限が必要
Agent	タスクを処理するための独自のコンテキストウィンドウを持つ subagent をスポン	いいえ
AskUserQuestion	要件を収集または曖昧さを明確にするための複数選択質問を尋ねます	いいえ
Bash	環境でシェルコマンドを実行します。 Bash ツール動作 を参照	はい
CronCreate	現在のセッション内で定期的または1回限りのプロンプトをスケジュールします（Claude が終了すると消えます）。 スケジュール済みタスク を参照	いいえ
CronDelete	ID でスケジュール済みタスクをキャンセル	いいえ
CronList	セッション内のすべてのスケジュール済みタスクをリスト	いいえ
Edit	特定のファイルに対して対象を絞った編集を行う	はい
EnterPlanMode	コーディング前にアプローチを設計するためにプランモードに切り替え	いいえ
EnterWorktree	分離された git worktree を作成してそれに切り替え	いいえ
ExitPlanMode	承認用にプランを提示してプランモードを終了	はい
ExitWorktree	worktree セッションを終了して元のディレクトリに戻る	いいえ
Glob	パターンマッチングに基づいてファイルを検索	いいえ
Grep	ファイルコンテンツ内のパターンを検索	いいえ

ツール	説明	権限が必要
ListMcpResourcesTool	接続された MCP サーバー によって公開されたリソースをリスト	いいえ
LSP	言語サーバー経由のコード知能。ファイル編集後に型エラーと警告を自動的に報告します。ナビゲーション操作もサポート：定義にジャンプ、参照を検索、型情報を取得、シンボルをリスト、実装を検索、呼び出し階層をトレース。 コード知能プラグイン とその言語サーバーバイナリが必要	いいえ
NotebookEdit	Jupyter ノートブックセルを変更	はい
Read	ファイルの内容を読み取る	いいえ
ReadMcpResourceTool	URI で特定の MCP リソースを読み取る	いいえ
Skill	メイン会話内で skill を実行	はい
TaskCreate	タスクリストに新しいタスクを作成	いいえ
TaskGet	特定のタスクの完全な詳細を取得	いいえ
TaskList	現在のステータスを持つすべてのタスクをリスト	いいえ
TaskOutput	バックグラウンドタスクから出力を取得	いいえ
TaskStop	ID で実行中のバックグラウンドタスクを強制終了	いいえ
TaskUpdate	タスクステータス、依存関係、詳細を更新するか、タスクを削除	いいえ

ツール	説明	権限が必要
TodoWrite	セッションタスクチェックリストを管理します。非インタラクティブモードおよび Agent SDK で利用可能。インタラクティブセッションは代わりに TaskCreate、TaskGet、TaskList、および TaskUpdate を使用	いいえ
ToolSearch	ツール検索 が有効な場合、遅延ツールを検索して読み込む	いいえ
WebFetch	指定された URL からコンテンツをフェッチ	はい
WebSearch	ウェブ検索を実行	はい
Write	ファイルを作成または上書き	はい

権限ルールは `/allowed-tools` を使用するか、[権限設定](#) で構成できます。[ツール固有の権限ルール](#) も参照してください。

Bash ツール動作

Bash ツールは以下の永続性動作でシェルコマンドを実行します：

- ・ **作業ディレクトリは永続化**：Claude が作業ディレクトリを変更する場合（例：`cd /path/to/dir`）、後続の Bash コマンドはそのディレクトリで実行されます。`CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR=1` を使用して、各コマンド後にプロジェクトディレクトリにリセットできます。
- ・ **環境変数は永続化しません**：1 つの Bash コマンドで設定された環境変数（例：`export MY_VAR=value`）は、後続の Bash コマンドでは**利用できません**。各 Bash コマンドは新しいシェル環境で実行されます。

Bash コマンドで環境変数を利用可能にするには、**3 つのオプション**があります：

オプション 1：Claude Code を開始する前に環境をアクティブ化（最も簡単なアプローチ）

Claude Code を起動する前に、ターミナルで仮想環境をアクティブ化します：

```
conda activate myenv
## または : source /path/to/venv/bin/activate
claude
```

これはシェル環境で機能しますが、Claude の Bash コマンド内で設定された環境変数はコマンド間で永続化されません。

オプション 2 : Claude Code を開始する前に CLAUDE_ENV_FILE を設定（永続的な環境セットアップ）

環境セットアップを含むシェルスクリプトへのパスをエクスポート：

```
export CLAUDE_ENV_FILE=/path/to/env-setup.sh
claude
```

ここで `/path/to/env-setup.sh` には以下が含まれます：

```
conda activate myenv
## または : source /path/to/venv/bin/activate
## または : export MY_VAR=value
```

Claude Code は各 Bash コマンド前にこのファイルをソースし、すべてのコマンド全体で環境を永続化します。

オプション 3 : SessionStart hook を使用（プロジェクト固有の構成）

`.claude/settings.json` で構成：

```
{
  "hooks": {
    "SessionStart": [{
      "matcher": "startup",
      "hooks": [{
        "type": "command",
        "command": "echo 'conda activate myenv' >> \"${CLAUDE_ENV_FILE}\""
      }]
    }]
  }
}
```

hook は `$CLAUDE_ENV_FILE` に書き込み、その後各 Bash コマンド前にソースされます。これはチーム共有プロジェクト構成に最適です。

[SessionStart hooks](#) を参照してください。オプション 3 の詳細については。

hooks でツールを拡張

[Claude Code hooks](#) を使用して、任意のツール実行の前後にカスタムコマンドを実行できます。

たとえば、Claude が Python ファイルを変更した後に Python フォーマッターを自動的に実行するか、特定のパスへの Write 操作をブロックして本番構成ファイルの変更を防ぐことができます。

関連項目

- [権限](#)：権限システム、ルール構文、ツール固有パターン、および Managed ポリシー
- [認証](#)：Claude Code へのユーザーアクセスをセットアップ
- [トラブルシューティング](#)：一般的な構成問題の解決策

モデル設定

Claude Code のモデル設定について学習します。opusplan などのモデルエイリアスを含みます。

利用可能なモデル

Claude Code の `model` 設定では、以下のいずれかを設定できます。

- ・ モデルエイリアス
- ・ モデル名
 - Anthropic API：完全な [モデル名](#)
 - Bedrock：推論プロファイル ARN
 - Foundry：デプロイメント名
 - Vertex：バージョン名

モデルエイリアス

モデルエイリアスは、正確なバージョン番号を覚えることなくモデル設定を選択するための便利な方法を提供します。

モデルエイリアス	動作
<code>default</code>	アカウントタイプに応じた推奨モデル設定
<code>sonnet</code>	日常的なコーディングタスク用に最新の Sonnet モデル（現在は Sonnet 4.6）を使用します
<code>opus</code>	複雑な推論タスク用に最新の Opus モデル（現在は Opus 4.6）を使用します
<code>haiku</code>	シンプルなタスク用に高速で効率的な Haiku モデルを使用します
<code>sonnet[1m]</code>	長いセッション用に 100 万トークンのコンテキストウィンドウ を備えた Sonnet を使用します
<code>opusplan</code>	Plan Mode 中は <code>opus</code> を使用し、実行中は <code>sonnet</code> に切り替わる特別なモード

エイリアスは常に最新バージョンを指します。特定のバージョンに固定するには、完全なモデル名（例： `claude-opus-4-6` ）を使用するか、 `ANTHROPIC_DEFAULT_OPUS_MODEL` のような対応する環境変数を設定します。

モデルの設定

モデルは複数の方法で設定できます。優先順位順に以下の通りです。

1. **セッション中** - `/model <alias|name>` を使用してセッション中にモデルを切り替えます
2. **起動時** - `claude --model <alias|name>` で起動します
3. **環境変数** - `ANTHROPIC_MODEL=<alias|name>` を設定します
4. **設定** - `model` フィールドを使用して設定ファイルで永続的に設定します。

使用例：

```
## Opus で開始
claude --model opus

## セッション中に Sonnet に切り替え
/model sonnet
```

設定ファイルの例：

```
{
  "permissions": {
    ...
  },
  "model": "opus"
}
```

モデル選択の制限

エンタープライズ管理者は、[管理設定またはポリシー設定](#) で `availableModels` を使用して、ユーザーが選択できるモデルを制限できます。

`availableModels` が設定されている場合、ユーザーは `/model`、`--model` フラグ、Config ツール、または `ANTHROPIC_MODEL` 環境変数を使用してリスト内にはないモデルに切り替えることはできません。

```
{
  "availableModels": ["sonnet", "haiku"]
}
```

デフォルトモデルの動作

モデルピッカーの Default オプションは `availableModels` の影響を受けません。常にご利用可能であり、[ユーザーのサブスクリプションティアに基づいた](#) システムのランタイム デフォルトを表します。

`availableModels: []` の場合でも、ユーザーはそのティア用の Default モデルで Claude Code を使用できます。

ユーザーが実行するモデルを制御する

モデル体験を完全に制御するには、`availableModels` と `model` 設定を一緒に使用します。

- **availableModels** : ユーザーが切り替えられるものを制限します
- **model** : 明示的なモデルオーバーライドを設定し、Default より優先されます

この例は、すべてのユーザーが Sonnet 4.6 を実行し、Sonnet と Haiku のみを選択できることを保証します。

```
{
  "model": "sonnet",
  "availableModels": ["sonnet", "haiku"]
}
```

マージ動作

`availableModels` がユーザー設定とプロジェクト設定など複数のレベルで設定されている場合、配列はマージされ、重複排除されます。厳密なアローストを適用するには、最優先度を持つ管理設定またはポリシー設定で `availableModels` を設定します。

特別なモデルの動作

`default` モデル設定

`default` の動作はアカウントタイプによって異なります。

- **Max と Team Premium** : Opus 4.6 がデフォルト

- **Pro と Team Standard** : Sonnet 4.6 がデフォルト
- **Enterprise** : Opus 4.6 は利用可能ですがデフォルトではありません

Claude Code は、Opus の使用量閾値に達した場合、自動的に Sonnet にフォールバックする可能性があります。

opuslan モデル設定

opuslan モデルエイリアスは、自動化されたハイブリッドアプローチを提供します。

- **Plan Mode 中** - 複雑な推論とアーキテクチャの決定用に **opus** を使用します
- **実行モード中** - コード生成と実装用に自動的に **sonnet** に切り替わります

これにより、両方の長所が得られます。計画用の Opus の優れた推論と、実行用の Sonnet の効率性です。

努力レベルの調整

努力レベル は適応的推論を制御し、タスクの複雑さに基づいて思考を動的に割り当てます。低い努力はシンプルなタスクではより高速で安価ですが、高い努力は複雑な問題に対してより深い推論を提供します。

3つのレベルが利用可能です。**low**、**medium**、**high**。Opus 4.6 は Max と Team サブスクリバターのデフォルトで medium 努力です。

努力レベルの設定：

- **/model 内**：モデルを選択する際に左右矢印キーを使用して努力スライダーを調整します
- **環境変数**： **CLAUDE_CODE_EFFORT_LEVEL=low|medium|high** を設定します
- **設定**：設定ファイルで **effortLevel** を設定します

努力は Opus 4.6 と Sonnet 4.6 でサポートされています。サポートされているモデルが選択されている場合、努力スライダーが **/model** に表示されます。現在の努力レベルはロゴとスピナーの横にも表示されます（例：「with low effort」）。**/model** を開かなくても、どの設定がアクティブであるかを確認できます。

Opus 4.6 と Sonnet 4.6 で適応的推論を無効にし、以前の固定思考予算に戻すには、**CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1** を設定します。無効にされた場合、これらのモデルは **MAX_THINKING_TOKENS** で制御される固定予算を使用します。**環境変数** を参照してください。

拡張コンテキスト

Opus 4.6 と Sonnet 4.6 は、大規模なコードベースを持つ長いセッション用に [100万トークンのコンテキストウィンドウ](#) をサポートしています。

Note:

1M コンテキストウィンドウは現在ベータ版です。機能、価格、および可用性は変更される可能性があります。

拡張コンテキストは以下で利用可能です。

- **API と従量課金ユーザー**：1M コンテキストへのフルアクセス
- **Pro、Max、Teams、および Enterprise サブスクリャー**：[追加使用](#) が有効な場合に利用可能

1M コンテキストを完全に無効にするには、`CLAUDE_CODE_DISABLE_1M_CONTEXT=1` を設定します。これにより、1M モデルバリエーションがモデルピッカーから削除されます。[環境変数](#)を参照してください。

1M モデルを選択しても、請求はすぐには変わりません。セッションはコンテキストが 200K トークンを超えるまで標準レートを使用します。200K トークンを超えると、リクエストは [長コンテキスト価格](#) で専用 [レート制限](#) で請求されます。サブスクリャーの場合、200K を超えるトークンはサブスクリプションではなく追加使用として請求されます。

アカウントが 1M コンテキストをサポートしている場合、オプションは Claude Code の最新バージョンのモデルピッカー（`/model`）に表示されます。表示されない場合は、セッションを再起動してみてください。

モデルエイリアスまたは完全なモデル名で `[1m]` サフィックスを使用することもできます。

```
## sonnet[1m] エイリアスを使用
/model sonnet[1m]

## または完全なモデル名に [1m] を追加
/model claude-sonnet-4-6[1m]
```

現在のモデルの確認

現在使用しているモデルは複数の方法で確認できます。

1. [ステータスライン](#) 内（設定されている場合）
2. `/status` 内。アカウント情報も表示されます。

環境変数

以下の環境変数を使用できます。これらはエイリアスがマップするモデル名を制御するために、完全な **モデル名**（または API プロバイダーの同等のもの）である必要があります。

環境変数	説明
<code>ANTHROPIC_DEFAULT_OPUS_MODEL</code>	<code>opus</code> に使用するモデル、または Plan Mode がアクティブな場合の <code>opusplan</code> に使用するモデル。
<code>ANTHROPIC_DEFAULT_SONNET_MODEL</code>	<code>sonnet</code> に使用するモデル、または Plan Mode がアクティブでない場合の <code>opusplan</code> に使用するモデル。
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL</code>	<code>haiku</code> に使用するモデル、または バックグラウンド機能 に使用するモデル
<code>CLAUDE_CODE_SUBAGENT_MODEL</code>	<code>subagents</code> に使用するモデル

注： `ANTHROPIC_SMALL_FAST_MODEL` は `ANTHROPIC_DEFAULT_HAIKU_MODEL` の代わりに非推奨です。

サードパーティデプロイメント用のモデルのピン留め

[Bedrock](#)、[Vertex AI](#)、または [Foundry](#) を通じて Claude Code をデプロイする場合、ユーザーへのロールアウト前にモデルバージョンをピン留めします。

ピン留めなしでは、Claude Code はモデルエイリアス（`sonnet`、`opus`、`haiku`）を使用し、最新バージョンに解決されます。Anthropic が新しいモデルをリリースすると、新しいバージョンが有効になっていないアカウントを持つユーザーは静かに破損します。

Warning:

初期セットアップの一部として、3つのモデル環境変数すべてを特定のバージョン ID に設定します。このステップをスキップすると、Claude Code の更新があなたのアクションなしでユーザーを破損させる可能性があります。

プロバイダー用のバージョン固有のモデル ID で以下の環境変数を使用します。

プロバイダー	例
Bedrock	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-6-v1'</code>

プロバイダー	例
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>

`ANTHROPIC_DEFAULT_SONNET_MODEL` と `ANTHROPIC_DEFAULT_HAIKU_MODEL` に同じパターンを適用します。すべてのプロバイダー全体の現在および従来のモデル ID については、[モデル概要](#)を参照してください。ユーザーを新しいモデルバージョンにアップグレードするには、これらの環境変数を更新して再デプロイします。

Note:

サードパーティプロバイダーを使用する場合、`settings.availableModels` アローリストは引き続き適用されます。フィルタリングはプロバイダー固有のモデル ID ではなく、モデルエイリアス（`opus`、`sonnet`、`haiku`）で一致します。

バージョンごとのモデル ID のオーバーライド

上記のファミリーレベルの環境変数は、ファミリーエイリアスごとに 1 つのモデル ID を設定します。同じファミリー内の複数のバージョンを異なるプロバイダー ID にマップする必要がある場合は、代わりに `modelOverrides` 設定を使用します。

`modelOverrides` は個別の Anthropic モデル ID をプロバイダー固有の文字列にマップします。Claude Code はこれらをプロバイダーの API に送信します。ユーザーが `/model` ピッカーでマップされたモデルを選択すると、Claude Code は組み込みのデフォルトの代わりに設定された値を使用します。

これにより、エンタープライズ管理者は、ガバナンス、コスト配分、または地域的ルーティング用に、各モデルバージョンを特定の Bedrock 推論プロファイル ARN、Vertex AI バージョン名、または Foundry デプロイメント名にルーティングできます。

[設定ファイル](#) で `modelOverrides` を設定します。

```
{
  "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"
  }
}
```

キーは [モデル概要](#) にリストされている Anthropic モデル ID である必要があります。日付付きモデル ID の場合、そこに表示されるとおりに日付サフィックスを含めます。不明なキーは無視されます。

オーバーライドは、`/model` ピッカーの各エントリをサポートする組み込みモデル ID を置き換えます。Bedrock では、オーバーライドは Claude Code が起動時に自動的に検出する推論プロファイルより優先されます。`ANTHROPIC_MODEL`、`--model`、または `ANTHROPIC_DEFAULT_*_MODEL` 環境変数を通じて直接提供する値は、プロバイダーにそのまま渡され、`modelOverrides` によって変換されません。

`modelOverrides` は `availableModels` と一緒に機能します。アローリストは Anthropic モデル ID に対して評価され、オーバーライド値に対してではないため、`availableModels` の `"opus"` のようなエントリは、Opus バージョンが ARN にマップされている場合でも一致し続けます。

プロンプトキャッシング設定

Claude Code は [プロンプトキャッシング](#) を自動的に使用してパフォーマンスを最適化し、コストを削減します。プロンプトキャッシングをグローバルに、または特定のモデルティア用に無効にできます。

環境変数	説明
<code>DISABLE_PROMPT_CACHING</code>	すべてのモデルのプロンプトキャッシングを無効にするには 1 に設定します（モデル固有の設定より優先されます）
<code>DISABLE_PROMPT_CACHING_HAIKU</code>	Haiku モデルのみのプロンプトキャッシングを無効にするには 1 に設定します

環境変数	説明
<code>DISABLE_PROMPT_CACHING_SONNET</code>	Sonnet モデルのみのプロンプトキャッシングを無効にするには <code>1</code> に設定します
<code>DISABLE_PROMPT_CACHING_OPUS</code>	Opus モデルのみのプロンプトキャッシングを無効にするには <code>1</code> に設定します

これらの環境変数は、プロンプトキャッシング動作に対する細かい制御を提供します。グローバル `DISABLE_PROMPT_CACHING` 設定はモデル固有の設定より優先され、必要に応じてすべてのキャッシングをすばやく無効にできます。モデル固有の設定は、特定のモデルをデバッグする場合や、異なるキャッシング実装を持つ可能性があるクラウドプロバイダーと連携する場合など、選択的な制御に役立ちます。

出力スタイル

ソフトウェアエンジニアリング以外の用途に合わせて Claude Code をカスタマイズする

出力スタイルを使用すると、Claude Code をあらゆるタイプのエージェントとして使用できます。ローカルスクリプトの実行、ファイルの読み書き、TODO の追跡など、コアの機能を保持したままです。

組み込み出力スタイル

Claude Code の **Default** 出力スタイルは既存のシステムプロンプトであり、ソフトウェアエンジニアリングタスクを効率的に完了するのに役立つように設計されています。

コードベースと Claude の動作方法を教えることに焦点を当てた、2 つの追加の組み込み出力スタイルがあります。

- **Explanatory:** ソフトウェアエンジニアリングタスクの完了を支援しながら、教育的な「Insights」を提供します。実装の選択肢とコードベースのパターンを理解するのに役立ちます。
- **Learning:** 協調的な学習型モードです。Claude はコーディング中に「Insights」を共有するだけでなく、小さな戦略的なコードの一部を自分で実装するよう求めます。Claude Code はコード内に **TODO(human)** マーカーを追加して、実装すべき箇所を示します。

出力スタイルの仕組み

出力スタイルは Claude Code のシステムプロンプトを直接変更します。

- すべての出力スタイルは、効率的な出力のための指示（簡潔に回答するなど）を除外します。
- カスタム出力スタイルは、**keep-coding-instructions** が true でない限り、コーディングのための指示（テストでコードを検証するなど）を除外します。
- すべての出力スタイルは、システムプロンプトの最後に独自のカスタム指示が追加されます。
- すべての出力スタイルは、会話中に出力スタイルの指示に従うよう Claude に思い出させるリマインダーをトリガーします。

出力スタイルを変更する

`/config` を実行し、**Output style** を選択してメニューからスタイルを選択します。選択内容は [ローカルプロジェクトレベル](#) の `.claude/settings.local.json` に保存されます。

メニューなしでスタイルを設定するには、設定ファイルの `outputStyle` フィールドを直接編集します。

```
{
  "outputStyle": "Explanatory"
}
```

出力スタイルはセッション開始時にシステムプロンプトで設定されるため、変更は新しいセッションを開始した次回に有効になります。これにより、会話全体を通じてシステムプロンプトが安定し、プロンプトキャッシングがレイテンシとコストを削減できます。

カスタム出力スタイルを作成する

カスタム出力スタイルは、`frontmatter` とシステムプロンプトに追加されるテキストを含む Markdown ファイルです。

```
---
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...]
```

これらのファイルはユーザーレベル（`~/.claude/output-styles`）またはプロジェクトレベル（`.claude/output-styles`）に保存できます。

Frontmatter

出力スタイルファイルは、メタデータを指定するための frontmatter をサポートしています。

Frontmatter	目的	デフォルト
<code>name</code>	出力スタイルの名前（ファイル名でない場合）	ファイル名から継承
<code>description</code>	<code>/config</code> ピッカーに表示される出力スタイルの説明	なし
<code>keep-coding-instructions</code>	Claude Code のシステムプロンプトのコーディング関連の部分を保持するかどうか	false

関連機能との比較

出力スタイル vs. CLAUDE.md vs. `--append-system-prompt`

出力スタイルは、ソフトウェアエンジニアリング固有の Claude Code のデフォルトシステムプロンプトの部分を完全に「オフ」にします。CLAUDE.md も `--append-system-prompt` も Claude Code のデフォルトシステムプロンプトを編集しません。CLAUDE.md は内容をユーザーメッセージとして Claude Code のデフォルトシステムプロンプトの「後に」追加します。`--append-system-prompt` はコンテンツをシステムプロンプトに追加します。

出力スタイル vs. [Agents](#)

出力スタイルはメインエージェントループに直接影響し、システムプロンプトのみに影響します。エージェントは特定のタスクを処理するために呼び出され、使用するモデル、利用可能なツール、エージェントをいつ使用するかに関するコンテキストなどの追加設定を含めることができます。

出力スタイル vs. [Skills](#)

出力スタイルは Claude の応答方法（フォーマット、トーン、構造）を変更し、選択されると常にアクティブです。Skills はタスク固有のプロンプトであり、`/skill-name` で呼び出すか、関連する場合に Claude が自動的に読み込みます。一貫したフォーマット設定を使用する場合は出力スタイルを使用します。再利用可能なワークフローとタスクの場合は Skills を使用します。

高速モードでレスポンスを高速化

Claude Code で高速モードを切り替えて、Opus 4.6 のレスポンスを高速化します。

Note:

高速モードは[リサーチプレビュー](#)段階です。機能、価格、および利用可能性はフィードバックに基づいて変更される可能性があります。

高速モードは Claude Opus 4.6 の高速構成で、モデルを 2.5 倍高速化しますが、トークンあたりのコストが高くなります。迅速な反復やライブデバッグなどのインタラクティブな作業で速度が必要な場合は `/fast` でオンにし、コストがレイテンシーより重要な場合はオフにします。

高速モードは異なるモデルではありません。同じ Opus 4.6 を使用していますが、コスト効率よりも速度を優先する異なる API 構成です。同じ品質と機能が得られ、レスポンスが高速化されるだけです。

Note:

高速モードには Claude Code v2.1.36 以降が必要です。 `claude --version` でバージョンを確認してください。

知っておくべきこと：

- Claude Code CLI で `/fast` を使用して高速モードをオンにします。Claude Code VS Code Extension でも `/fast` で利用可能です。
- Opus 4.6 の高速モード価格は \$30/150 MTok から始まります。高速モードは 2 月 16 日午後 11 時 59 分 PT まですべてのプランで 50% 割引で利用可能です。
- すべての Claude Code ユーザーがサブスクリプションプラン（Pro/Max/Team/Enterprise）および Claude Console で利用可能です。
- サブスクリプションプラン（Pro/Max/Team/Enterprise）の Claude Code ユーザーの場合、高速モードは追加使用量のみで利用可能であり、サブスクリプションレート制限に含まれていません。

このページでは、[高速モードの切り替え](#)、その[コストのトレードオフ](#)、[使用時期の判断](#)、[要件](#)、[セッションごとのオプトイン](#)、および[レート制限の処理](#)について説明します。

高速モードの切り替え

次のいずれかの方法で高速モードを切り替えます：

- ・ `/fast` と入力して Tab キーを押してオンまたはオフに切り替える
- ・ ユーザー設定ファイルで `"fastMode": true` を設定する (</ja/settings>)

デフォルトでは、高速モードはセッション間で保持されます。管理者は高速モードを各セッションでリセットするように構成できます。詳細は[セッションごとのオプトインが必要](#)を参照してください。

最適なコスト効率を得るには、会話の途中で切り替えるのではなく、セッションの開始時に高速モードを有効にしてください。詳細は[コストのトレードオフを理解する](#)を参照してください。

高速モードを有効にすると：

- ・ 別のモデルを使用している場合、Claude Code は自動的に Opus 4.6 に切り替わります
- ・ 確認メッセージが表示されます：「Fast mode ON」
- ・ 高速モードがアクティブな間、プロンプトの横に小さい  アイコンが表示されます
- ・ いつでも `/fast` を再度実行して、高速モードがオンかオフかを確認できます

`/fast` を再度実行して高速モードを無効にすると、Opus 4.6 に留まります。モデルは以前のモデルに戻りません。別のモデルに切り替えるには、`/model` を使用してください。

コストのトレードオフを理解する

高速モードは標準 Opus 4.6 よりもトークンあたりの価格が高くなります：

モード	入力 (MTok)	出力 (MTok)
Opus 4.6 の高速モード (<200K)	\$30	\$150
Opus 4.6 の高速モード (>200K)	\$60	\$225

高速モードは 1M トークン拡張コンテキストウィンドウと互換性があります。

会話の途中で高速モードに切り替えると、会話コンテキスト全体に対して完全な高速モードキャッシュなし入力トークン価格を支払います。これは最初から高速モードを有効にした場合よりもコストがかかります。

高速モードの使用時期を判断する

高速モードはレスポンスレイテンシーがコストより重要なインタラクティブな作業に最適です：

- ・コード変更の迅速な反復
- ・ライブデバッグセッション
- ・厳しい期限を持つ時間に敏感な作業

標準モードは以下に適しています：

- ・速度がそれほど重要でない長い自動タスク
- ・バッチ処理またはCI/CD パイプライン
- ・コストに敏感なワークロード

高速モードと努力レベル

高速モードと努力レベルはどちらもレスポンス速度に影響しますが、異なる方法で影響します：

設定	効果
高速モード	同じモデル品質、低レイテンシー、高コスト
低い努力レベル	思考時間が短い、レスポンスが高速、複雑なタスクでは品質が低下する可能性

両方を組み合わせることができます：単純なタスクで最大速度を得るために、低い[努力レベル](#)で高速モードを使用します。

要件

高速モードには以下のすべてが必要です：

- ・**サードパーティクラウドプロバイダーでは利用不可**：高速モードは Amazon Bedrock、Google Vertex AI、または Microsoft Azure Foundry では利用できません。高速モードは Anthropic Console API および追加使用量を使用する Claude サブスクリプションプランで利用可能です。
- ・**追加使用量が有効**：アカウントで追加使用量が有効になっている必要があります。これにより、プランに含まれる使用量を超えて請求できます。個人アカウントの場合、[Console 請求設定](#)で有効にしてください。Teams および Enterprise の場合、管理者が組織の追加使用量を有効にする必要があります。

Note:

高速モード使用量は、プランに残りの使用量がある場合でも、追加使用量に直接請求されます。これは、高速モードトークンがプランに含まれる使用量にカウントされず、最初のトークンから高速モード料金で請求されることを意味します。

- **Teams および Enterprise の管理者による有効化**：高速モードは Teams および Enterprise 組織ではデフォルトで無効になっています。ユーザーがアクセスできるようにするには、管理者が明示的に[高速モードを有効にする](#)必要があります。

Note:

管理者が組織の高速モードを有効にしていない場合、`/fast` コマンドは「Fast mode has been disabled by your organization.」と表示されます。

組織の高速モードを有効にする

管理者は以下で高速モードを有効にできます：

- **Console**（API カスタマー）：[Claude Code 設定](#)
- **Claude AI**（Teams および Enterprise）：[Admin Settings > Claude Code](#)

高速モードを完全に無効にするもう 1 つのオプションは、

`CLAUDE_CODE_DISABLE_FAST_MODE=1` を設定することです。[環境変数](#)を参照してください。

セッションごとのオプトインが必要

デフォルトでは、高速モードはセッション間で保持されます：ユーザーが高速モードを有効にすると、将来のセッションでもオンのままです。[Teams](#) または [Enterprise](#) プランの管理者は、[管理設定](#)または[サーバー管理設定](#)で `fastModePerSessionOptIn` を `true` に設定することでこれを防ぐことができます。これにより、各セッションは高速モードがオフで開始され、ユーザーが `/fast` で明示的に有効にする必要があります。

```
{
  "fastModePerSessionOptIn": true
}
```

これは、ユーザーが複数の同時セッションを実行する組織でコストを制御するのに役立ちます。ユーザーは速度が必要な場合でも `/fast` で高速モードを有効にできますが、新しいセッションの開始時にリセットされます。ユーザーの高速モード設定は保存されたままなので、この設定を削除するとデフォルトの永続的な動作が復元されます。

レート制限の処理

高速モードは標準 Opus 4.6 とは別のレート制限があります。高速モードレート制限に達するか、追加使用量クレジットが不足した場合：

1. 高速モードは自動的に標準 Opus 4.6 にフォールバックします
2.  アイコンがグレーに変わってクールダウンを示します
3. 標準速度と価格で作業を続けます
4. クールダウンが終了すると、高速モードは自動的に再度有効になります

クールダウンを待つ代わりに高速モードを手動で無効にするには、`/fast` を再度実行してください。

リサーチプレビュー

高速モードはリサーチプレビュー機能です。これは以下を意味します：

- ・ 機能はフィードバックに基づいて変更される可能性があります
- ・ 利用可能性と価格は変更される可能性があります
- ・ 基盤となる API 構成は進化する可能性があります

通常の Anthropic サポートチャネルを通じて問題またはフィードバックを報告してください。

関連項目

- ・ [モデル構成](#)：モデルを切り替えて努力レベルを調整する
- ・ [コストを効果的に管理する](#)：トークン使用量を追跡してコストを削減する
- ・ [ステータスラインの構成](#)：モデルとコンテキスト情報を表示する

コストを効果的に管理する

トークン使用量を追跡し、チームの支出制限を設定し、コンテキスト管理、モデル選択、拡張思考設定、前処理フックを使用して Claude Code のコストを削減します。

Claude Code は各インタラクションでトークンを消費します。コストはコードベースのサイズ、クエリの複雑さ、会話の長さに基づいて異なります。平均コストは開発者 1 人あたり 1 日 6 ドルで、90% のユーザーの日次コストは 12 ドル以下です。

チーム使用の場合、Claude Code は API トークン消費によって課金されます。平均的には、Claude Code は Sonnet 4.6 で開発者 1 人あたり月額約 100~200 ドルのコストがかかりますが、ユーザーが実行しているインスタンスの数と自動化で使用しているかどうかに応じて大きなばらつきがあります。

このページでは、[コストを追跡する方法](#)、[チームのコストを管理する方法](#)、および [トークン使用量を削減する方法](#) について説明します。

コストを追跡する

`/cost` コマンドを使用する

Note:

`/cost` コマンドは API トークン使用量を表示し、API ユーザーを対象としています。Claude Max および Pro サブスクリバードはサブスクリプションに使用量が含まれているため、`/cost` データは請求目的では関連がありません。サブスクリバードは `/stats` を使用して使用パターンを表示できます。

`/cost` コマンドは現在のセッションの詳細なトークン使用統計を提供します。

```
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
```

チームのコストを管理する

Claude API を使用する場合、Claude Code ワークスペース支出の合計に対して [ワークスペース支出制限を設定](#) できます。管理者は Console で [コストと使用状況レポートを表示](#) できます。

Note:

Claude Code を Claude Console アカウントで初めて認証すると、「Claude Code」というワークスペースが自動的に作成されます。このワークスペースは、組織内のすべての Claude Code 使用量の一元化されたコスト追跡と管理を提供します。このワークスペースの API キーを作成することはできません。これは Claude Code 認証と使用量専用です。

Bedrock、Vertex、および Foundry では、Claude Code はクラウドからメトリクスを送信しません。コストメトリクスを取得するために、複数の大規模企業は [LiteLLM](#) を使用していると報告しており、これは企業が [キーごとに支出を追跡](#) するのに役立つオープンソースツールです。このプロジェクトは Anthropic と提携していないため、セキュリティについて監査されていません。

レート制限の推奨事項

チーム向けに Claude Code を設定する場合、組織のサイズに基づいて、これらのユーザーあたりのトークン/分（TPM）およびリクエスト/分（RPM）の推奨事項を検討してください。

チームサイズ	ユーザーあたり TPM	ユーザーあたり RPM
1～5 ユーザー	200k～300k	5～7
5～20 ユーザー	100k～150k	2.5～3.5
20～50 ユーザー	50k～75k	1.25～1.75
50～100 ユーザー	25k～35k	0.62～0.87
100～500 ユーザー	15k～20k	0.37～0.47
500 ユーザー以上	10k～15k	0.25～0.35

たとえば、200 ユーザーがいる場合、各ユーザーに 20k TPM をリクエストするか、合計 400 万 TPM（ $200 \times 20,000 = 400$ 万）をリクエストできます。

チームサイズが大きくなるにつれて、ユーザーあたりの TPM は減少します。これは、より大きな組織では Claude Code を同時に使用するユーザーが少ない傾向があるためです。これらのレート制限は個別ユーザーレベルではなく組織レベルで適用されます。つまり、他のユーザーが積極的にサービスを使用していない場合、個別ユーザーは計算された共有量を一時的に超えて消費できます。

Note:

大規模グループとのライブトレーニングセッションなど、異常に高い同時使用シナリオが予想される場合は、ユーザーあたりのより高い TPM 割り当てが必要になる場合があります。

エージェントチームのトークンコスト

[エージェントチーム](#) は複数の Claude Code インスタンスを生成し、各インスタンスは独自のコンテキストウィンドウを持ちます。トークン使用量はアクティブなチームメイトの数と各チームメイトが実行される期間に応じてスケーリングされます。

エージェントチームのコストを管理可能に保つには、以下を実行してください。

- ・チームメイトに Sonnet を使用します。これは調整タスクの機能とコストのバランスを取ります。
- ・チームを小さく保ちます。各チームメイトは独自のコンテキストウィンドウを実行するため、トークン使用量はおよそチームサイズに比例します。
- ・スポーンプロンプトを焦点を絞ったものにします。チームメイトは CLAUDE.md、MCP サーバー、およびスキルを自動的に読み込みますが、スポーンプロンプト内のすべてが最初からコンテキストに追加されます。
- ・作業が完了したらチームをクリーンアップします。アクティブなチームメイトはアイドル状態でもトークンを消費し続けます。
- ・エージェントチームはデフォルトで無効になっています。[settings.json](#) または環境で `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` を設定して有効にします。[エージェントチームを有効にする](#) を参照してください。

トークン使用量を削減する

トークンコストはコンテキストサイズに応じてスケーリングされます。Claude が処理するコンテキストが多いほど、より多くのトークンを使用します。Claude Code はプロンプトキャッシング（システムプロンプトなどの繰り返されるコンテンツのコストを削減）と自動コンパクション（コンテキスト制限に近づくとき会話履歴を要約）を通じてコストを自動的に最適化します。

以下の戦略は、コンテキストを小さく保ち、メッセージあたりのコストを削減するのに役立ちます。

コンテキストを積極的に管理する

`/cost` を使用して現在のトークン使用量を確認するか、[ステータスラインを設定](#) してそれを継続的に表示します。

- **タスク間でクリアする:** 関連のない作業に切り替える場合は `/clear` を使用して新しく開始します。古いコンテキストは後続のすべてのメッセージでトークンを浪費します。クリアする前に `/rename` を使用してセッションに名前を付けると、後で簡単に付けることができます。その後、`/resume` を使用して戻ります。
- **カスタムコンパクション指示を追加する:** `/compact Focus on code samples and API usage` は、要約中に保持する内容を Claude に指示します。

CLAUDE.md でコンパクション動作をカスタマイズすることもできます。

```
## Compact instructions
```

```
When you are using compact, please focus on test output and code changes
```

適切なモデルを選択する

Sonnet はほとんどのコーディングタスクをうまく処理し、Opus よりもコストが低くなります。複雑なアーキテクチャの決定または複数ステップの推論のために Opus を予約します。`/model` を使用してセッション中にモデルを切り替えるか、`/config` でデフォルトを設定します。単純な subagent タスクの場合、[subagent 設定](#) で `model: haiku` を指定します。

MCP サーバーのオーバーヘッドを削減する

各 MCP サーバーはアイドル状態でもツール定義をコンテキストに追加します。`/context` を実行して、何がスペースを消費しているかを確認します。

- **利用可能な場合は CLI ツールを優先する:** `gh`、`aws`、`gcloud`、`sentry-cli`などのツールは、永続的なツール定義を追加しないため、MCP サーバーよりもコンテキスト効率が高くなります。Claude はオーバーヘッドなしで CLI コマンドを直接実行できます。
- **未使用のサーバーを無効にする:** `/mcp` を実行して設定されたサーバーを確認し、積極的に使用していないサーバーを無効にします。
- **ツール検索は自動です:** MCP ツール説明がコンテキストウィンドウの 10% を超える場合、Claude Code は自動的にそれらを遅延させ、[ツール検索](#) を介してオンデマンド

でツールを読み込みます。遅延されたツールは実際に使用される場合にのみコンテキストに入るため、より低いしきい値は、スペースを消費するアイドルツール定義が少ないことを意味します。 `ENABLE_TOOL_SEARCH=auto:<N>` でより低いしきい値を設定します（たとえば、 `auto:5` はツールがコンテキストウィンドウの 5% を超える場合にトリガーされます）。

型付き言語用のコードインテリジェンスプラグインをインストールする

[コードインテリジェンスプラグイン](#) は Claude にテキストベースの検索の代わりに正確なシンボルナビゲーションを提供し、不慣れなコードを探索する際の不要なファイル読み取りを削減します。単一の「定義に移動」呼び出しは、`grep` の後に複数の候補ファイルを読み取る必要があるものを置き換えます。インストールされた言語サーバーは編集後に型エラーを自動的に報告するため、Claude はコンパイラを実行せずにエラーをキャッチします。

フックとスキルに処理をオフロードする

カスタム [フック](#) は Claude がそれを見る前にデータを前処理できます。Claude が 10,000 行のログファイルを読んでエラーを見つける代わりに、フックは `ERROR` に対して `grep` を実行し、一致する行のみを返すことができ、コンテキストを数万トークンから数百に削減します。

[スキル](#) は Claude にドメイン知識を与えることができるため、探索する必要がありません。たとえば、「codebase-overview」スキルはプロジェクトのアーキテクチャ、主要なディレクトリ、および命名規則を説明できます。Claude がスキルを呼び出すと、構造を理解するために複数のファイルを読むトークンを費やす代わりに、このコンテキストが即座に取得されます。

たとえば、この `PreToolUse` フックはテスト出力をフィルタリングして失敗のみを表示します。

settings.json

これを [settings.json](#) に追加して、すべての Bash コマンドの前にフックを実行します。

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/.claude/hooks/filter-test-output.sh"
          }
        ]
      }
    ]
  }
}
```

filter-test-output.sh

フックはこのスクリプトを呼び出し、コマンドがテストランナーであるかどうかを確認し、失敗のみを表示するように変更します。

```
#!/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
```

CLAUDE.md から スキルに指示を移動する

[CLAUDE.md](#) ファイルはセッション開始時にコンテキストに読み込まれます。PR レビューやデータベース移行などの特定のワークフロー用の詳細な指示が含まれている場合、関連のない作業を行っている場合でもそれらのトークンが存在します。[スキル](#) はオ

ンデマンドでのみ呼び出されたときに読み込まれるため、特殊な指示をスキルに移動することで、ベースコンテキストを小さく保ちます。CLAUDE.md を約 500 行以下に保つことを目指し、必須のみを含めます。

拡張思考を調整する

拡張思考はデフォルトで有効になっており、予算は 31,999 トークンです。これは複雑な計画と推論タスクのパフォーマンスを大幅に向上させるためです。ただし、思考トークンは出力トークンとして課金されるため、深い推論が必要ない単純なタスクの場合、`/model` で Opus 4.6 の [努力レベル](#) を低下させるか、`/config` で思考を無効にするか、予算を低下させることでコストを削減できます（たとえば、`MAX_THINKING_TOKENS=8000`）。

詳細な操作を subagent に委任する

テストの実行、ドキュメントの取得、またはログファイルの処理は、かなりのコンテキストを消費できます。これらを [subagent](#) に委任して、詳細な出力が subagent のコンテキストに留まり、メインの会話に戻るのはサマリーのみです。

エージェントチームのコストを管理する

エージェントチームは、チームメイトがプランモードで実行される場合、標準セッションよりも約 7 倍多くのトークンを使用します。これは、各チームメイトが独自のコンテキストウィンドウを維持し、別の Claude インスタンスとして実行されるためです。チームメイトあたりのトークン使用量を制限するために、チームタスクを小さく自己完結させておきます。詳細については、[エージェントチーム](#) を参照してください。

具体的なプロンプトを作成する

「このコードベースを改善する」のような曖昧なリクエストは、広範なスキャンをトリガーします。「auth.ts のログイン関数に入力検証を追加する」のような具体的なリクエストにより、Claude は最小限のファイル読み取りで効率的に作業できます。

複雑なタスクで効率的に作業する

より長いまたはより複雑な作業の場合、これらの習慣は間違った方向に進むことからの無駄なトークンを回避するのに役立ちます。

- **複雑なタスクにはプランモードを使用する:** Shift+Tab を押して、実装の前に [プランモード](#) に入ります。Claude はコードベースを探索し、承認のためのアプローチを提案し、初期方向が間違っている場合の高価な再作業を防ぎます。
- **早期に方向を修正する:** Claude が間違った方向に向かい始めた場合は、Escape を押して直ちに停止します。`/rewind` を使用するか、Escape をダブルタップして、会話とコードを前のチェックポイントに復元します。

- ・ **検証ターゲットを指定する:** テストケースを含めるか、スクリーンショットを貼り付けるか、プロンプトで予想される出力を定義します。Claude が独自の作業を検証できる場合、修正をリクエストする必要がある前に問題をキャッチします。
- ・ **段階的にテストする:** 1 つのファイルを作成し、テストしてから続行します。これは、修正が安い場合に早期に問題をキャッチします。

バックグラウンドトークン使用量

Claude Code はアイドル状態でも、バックグラウンド機能にトークンを使用します。

- ・ **会話要約:** `claude --resume` 機能の前の会話を要約するバックグラウンドジョブ
- ・ **コマンド処理:** `/cost` などの一部のコマンドは、ステータスを確認するためにリクエストを生成する場合があります

これらのバックグラウンドプロセスは、アクティブなインタラクションがなくても、少量のトークン（通常はセッションあたり 0.04 ドル未満）を消費します。

Claude Code の動作の変更を理解する

Claude Code は、コスト報告を含む機能の動作方法を変更する可能性のある定期的な更新を受け取ります。 `claude --version` を実行して現在のバージョンを確認します。特定の請求に関する質問については、[Console アカウント](#) を通じて Anthropic サポートに連絡してください。チーム展開の場合は、より広い展開の前に小規模なパイロットグループで使用パターンを確立することから始めます。

Part 5: Extensibility

Hooks リファレンス

Claude Code のフック イベント、設定スキーマ、JSON 入出力形式、終了コード、非同期フック、HTTP フック、プロンプト フック、MCP ツール フックのリファレンス。

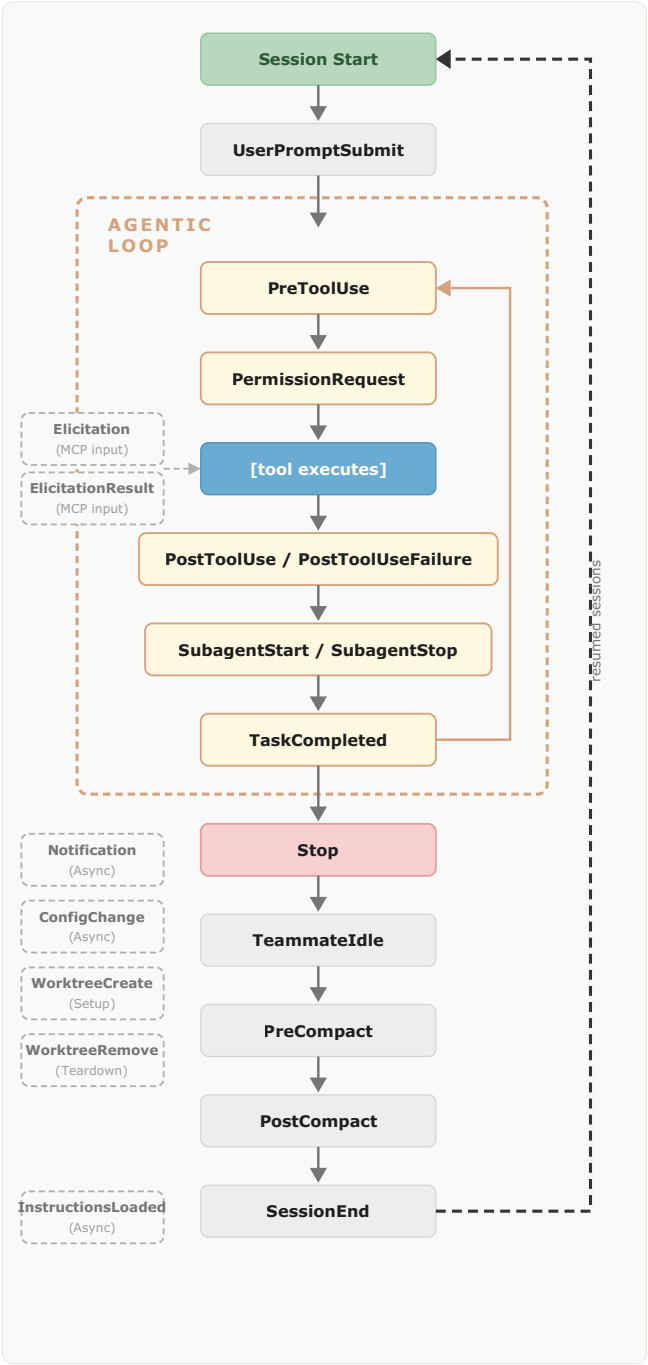
Tip:

例を含むクイックスタート ガイドについては、[ワークフローをフックで自動化する](#)を参照してください。

フックは、Claude Code のライフサイクル内の特定のポイントで自動的に実行されるユーザー定義のシェル コマンド、HTTP エンドポイント、または LLM プロンプトです。このリファレンスを使用して、イベント スキーマ、設定オプション、JSON 入出力形式、および非同期フック、HTTP フック、MCP ツール フックなどの高度な機能を検索してください。初めてフックを設定する場合は、代わりに[ガイド](#)から始めてください。

フック ライフサイクル

フックは Claude Code セッション中の特定のポイントで発火します。イベントが発火し、マッチャーがマッチすると、Claude Code はイベントに関する JSON コンテキストをフック ハンドラーに渡します。コマンド フックの場合、入力 は stdin で到着します。HTTP フックの場合、POST リクエスト本体として到着します。ハンドラーは入力を検査し、アクションを実行し、オプションで決定を返すことができます。一部のイベントはセッションごとに 1 回発火しますが、他のイベントは agentic ループ内で繰り返し発火します。



SessionStart から agentic ループを経由して SessionEnd までのフックのシーケンスを示すフック ライフサイクル図。WorktreeCreate、WorktreeRemove、InstructionsLoaded はスタンドアロン非同期イベント

以下の表は、各イベントがいつ発火するかをまとめています。[フック イベントセクション](#)では、各イベントの完全な入力スキーマと決定制御オプションについて説明しています。

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

フックがどのように解決されるか

これらの部分がどのように組み合わせるかを確認するために、破壊的なシェル コマンドをブロックする `PreToolUse` フックを考えてみましょう。フックは Bash ツール呼び出しの前に `block-rm.sh` を実行します。

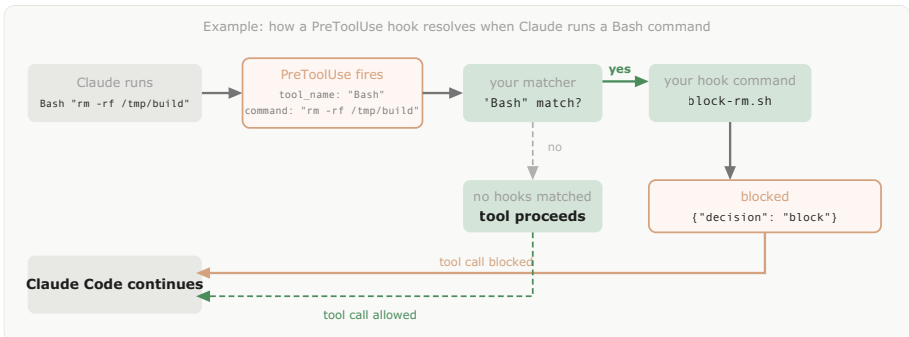
```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": ".claude/hooks/block-rm.sh"
          }
        ]
      }
    ]
  }
}
```

スクリプトは stdin から JSON 入力を読み取り、コマンドを抽出し、`rm -rf` が含まれている場合は `permissionDecision` の `"deny"` を返します。

```
#!/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
```

ここで Claude Code が `Bash "rm -rf /tmp/build"` を実行することを決定したとします。以下が起こります。



フック解決フロー: PreToolUse イベントが発火し、マッチャーが Bash マッチをチェックし、フックハンドラーが実行され、結果が Claude Code に返される

Step 1: イベントが発火

`PreToolUse` イベントが発火します。Claude Code はツール入力を JSON として stdin でフックに送信します。

```
{ "tool_name": "Bash", "tool_input": { "command": "rm -rf /tmp/build" }, ... }
```

Step 2: マッチャーがチェック

マッチャー `"Bash"` がツール名にマッチするため、`block-rm.sh` が実行されます。マッチャーを省略するか `"*"` を使用すると、フックはイベントのすべての出現で実行されます。マッチャーが定義されていてマッチしない場合にのみ、フックはスキップされます。

Step 3: フック ハンドラーが実行

スクリプトは入力から `"rm -rf /tmp/build"` を抽出し、`rm -rf` を見つけるため、決定を stdout に出力します。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Destructive command blocked by hook"
  }
}
```

コマンドが安全だった場合（`npm test` など）、スクリプトは代わりに `exit 0` に到達し、これは Claude Code にツール呼び出しを許可するよう指示します。

Step 4: Claude Code が結果に基づいて行動

Claude Code は JSON 決定を読み取り、ツール呼び出しをブロックし、Claude に理由を表示します。

以下の [設定](#) セクションでは完全なスキーマについて説明し、各 [フック イベント](#) セクションでは、コマンドが受け取る入力と返すことができる出力について説明しています。

設定

フックは JSON 設定ファイルで定義されます。設定には 3 つのネストレベルがあります。

1. 応答する [フック イベント](#) を選択します（`PreToolUse` や `Stop` など）
2. 発火するタイミングをフィルタリングする [マッチャーグループ](#)を追加します（`'Bash ツールのみ'` など）
3. マッチしたときに実行する 1 つ以上の [フック ハンドラー](#) を定義します

完全なウォークスルーと注釈付きの例については、上記の [フックがどのように解決されるか](#) を参照してください。

Note:

このページでは各レベルに特定の用語を使用しています。ライフサイクル ポイントの場合は **フック イベント**、フィルターの場合は **マッチャー グループ**、実行されるシェル コマンド、HTTP エンドポイント、プロンプト、またはエージェントの場合は **フック ハンドラー** です。'フック' 単独は一般的な機能を指します。

フック位置

フックを定義する場所によって、そのスコープが決まります。

位置	スコープ	共有可能
<code>~/.claude/settings.json</code>	すべてのプロジェクト	いいえ、マシンローカル
<code>.claude/settings.json</code>	単一プロジェクト	はい、リポジトリにコミット可能
<code>.claude/settings.local.json</code>	単一プロジェクト	いいえ、gitignored
管理ポリシー設定	組織全体	はい、管理者制御
プラグイン <code>hooks/hooks.json</code>	プラグインが有効な場合	はい、プラグインにバンドル
スキル または エージェント フロントマター	コンポーネントがアクティブな場合	はい、コンポーネントファイルで定義

設定ファイル解決の詳細については、[設定](#) を参照してください。エンタープライズ管理者は `allowManagedHooksOnly` を使用して、ユーザー、プロジェクト、プラグイン フックをブロックできます。[フック設定](#) を参照してください。

マッチャー パターン

`matcher` フィールドは、フックが発火するタイミングをフィルタリングする正規表現文字列です。すべての出現にマッチするには、`"*"`、`""`、または `matcher` を完全に省略します。各イベント タイプは異なるフィールドでマッチします。

イベント	マッチャーがフィルタリングするもの	マッチャー値の例
<code>PreToolUse</code> 、 <code>PostToolUse</code> 、 <code>PostToolUseFailure</code> 、 <code>PermissionRequest</code>	ツール名	<code>Bash</code> 、 <code>Edit Write</code> 、 <code>mc__.*</code>
<code>SessionStart</code>	セッションの開始方法	<code>startup</code> 、 <code>resume</code> 、 <code>clear</code> 、 <code>compact</code>
<code>SessionEnd</code>	セッションが終了した理由	<code>clear</code> 、 <code>logout</code> 、 <code>prompt_input_exit</code> 、 <code>bypass_permissions_disabled</code> 、 <code>other</code>
<code>Notification</code>	通知タイプ	<code>permission_prompt</code> 、 <code>idle_prompt</code> 、 <code>auth_success</code> 、 <code>elicitation_dialog</code>
<code>SubagentStart</code>	エージェント タイプ	<code>Bash</code> 、 <code>Explore</code> 、 <code>Plan</code> 、またはカスタム エージェント名
<code>PreCompact</code>	コンパクションをトリガーしたもの	<code>manual</code> 、 <code>auto</code>
<code>SubagentStop</code>	エージェント タイプ	<code>SubagentStart</code> と同じ 値
<code>ConfigChange</code>	設定ソース	<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> 、 <code>InstructionsLoaded</code>	マッチャー サポートなし	すべての出現で常に発火

マッチャーは正規表現であるため、`Edit|Write` は両方のツールにマッチし、`Notebook.*` は Notebook で始まるツールにマッチします。マッチャーは、Claude Code がフックに stdin で送信する [JSON 入力](#) のフィールドに対して実行されます。ツール イベントの場合、そのフィールドは `tool_name` です。各 [フックイベント](#) セクションでは、マッチャー値の完全なセットとそのイベントの入力スキーマをリストしています。

この例は、Claude がファイルを書き込むまたは編集するときのみリントスクリプトを実行します。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/lint-check.sh"
          }
        ]
      }
    ]
  }
}
```

`UserPromptSubmit`、`Stop`、`TeammateIdle`、`TaskCompleted`、`WorktreeCreate`、`WorktreeRemove`、`InstructionsLoaded` はマッチャーをサポートせず、すべての出現で常に発火します。これらのイベントに `matcher` フィールドを追加すると、サイレントに無視されます。

MCP ツールをマッチ

[MCP](#) サーバー ツールはツール イベント（`PreToolUse`、`PostToolUse`、`PostToolUseFailure`、`PermissionRequest`）で通常のツールとして表示されるため、他のツール名と同じ方法でマッチできます。

MCP ツールは `mcp__<server>__<tool>` という命名パターンに従います。例えば、

- `mcp__memory__create_entities` : Memory サーバーの create entities ツール
- `mcp__filesystem__read_file` : Filesystem サーバーの read file ツール
- `mcp__github__search_repositories` : GitHub サーバーの search ツール

正規表現パターンを使用して、特定の MCP ツールまたはツール グループをターゲットにします。

- `mcp__memory__.*` は `memory` サーバーのすべてのツールにマッチします
- `mcp__.*__write.*` はいずれかのサーバーから「write」を含むツールにマッチします

この例は、すべてのメモリ サーバー操作をログに記録し、任意の MCP サーバーからの書き込み操作を検証します。

```
{
  "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"
          }
        ]
      }
    ]
  }
}
```

フック ハンドラー フィールド

内側の `hooks` 配列の各オブジェクトはフック ハンドラーです。マッチャーがマッチしたときに実行されるシェル コマンド、HTTP エンドポイント、LLM プロンプト、または エージェントです。4つのタイプがあります。

- **コマンド フック** (`type: "command"`) : シェル コマンドを実行します。スクリプトはイベントの [JSON 入力](#) を stdin で受け取り、終了コードと stdout を通じて結果を通信します。
- **HTTP フック** (`type: "http"`) : イベントの JSON 入力を HTTP POST リクエストとして URL に送信します。エンドポイントは、コマンド フックと同じ [JSON 出力形式](#) を使用して、レスポンス本体を通じて結果を通信します。
- **プロンプト フック** (`type: "prompt"`) : Claude モデルにプロンプトを送信して、単一ターン評価を行います。モデルは yes/no 決定を JSON として返します。 [プロンプトベースのフック](#) を参照してください。
- **エージェント フック** (`type: "agent"`) : Read、Grep、Glob などのツールを使用して条件を検証してから決定を返すことができるサブエージェントを生成します。 [エージェントベースのフック](#) を参照してください。

共通フィールド

これらのフィールドはすべてのフック タイプに適用されます。

フィールド	必須	説明
<code>type</code>	はい	<code>"command"</code> 、 <code>"http"</code> 、 <code>"prompt"</code> 、または <code>"agent"</code>
<code>timeout</code>	いいえ	キャンセルするまでの秒数。デフォルト: コマンドは 600、プロンプトは 30、エージェントは 60
<code>statusMessage</code>	いいえ	フックの実行中に表示されるカスタム スピナー メッセージ

フィールド	必須	説明
<code>once</code>	いいえ	<code>true</code> の場合、セッションごとに1回だけ実行してから削除されます。スキルのみ、エージェントではありません。 スキルとエージェントのフック を参照してください

コマンド フック フィールド

[共通フィールド](#)に加えて、コマンド フックはこれらのフィールドを受け入れます。

フィールド	必須	説明
<code>command</code>	はい	実行するシェル コマンド
<code>async</code>	いいえ	<code>true</code> の場合、ブロックせずにバックグラウンドで実行されます。 バックグラウンドでフックを実行 を参照してください

HTTP フック フィールド

[共通フィールド](#)に加えて、HTTP フックはこれらのフィールドを受け入れます。

フィールド	必須	説明
<code>url</code>	はい	POST リクエストを送信する URL
<code>headers</code>	いいえ	キー値ペアとしての追加 HTTP ヘッダー。値は <code>\$VAR_NAME</code> または <code>\${VAR_NAME}</code> 構文を使用した環境変数補間をサポートします。 <code>allowedEnvVars</code> にリストされている変数のみが解決されます

フィールド	必須	説明
<code>allowedEnvVars</code>	いいえ	ヘッダー値に補間される可能性のある環境変数名のリスト。リストされていない変数への参照は空の文字列に置き換えられます。環境変数補間が機能するために必須です

Claude Code はフックの[JSON 入力](#)を `Content-Type: application/json` の POST リクエスト本体として送信します。レスポンス本体はコマンド フックと同じ[JSON 出力形式](#)を使用します。

エラー処理はコマンド フックと異なります。2xx 以外のレスポンス、接続失敗、タイムアウトはすべて、実行を続行できる非ブロッキング エラーを生成します。ツール呼び出しをブロックまたは権限を拒否するには、`decision: "block"` または `permissionDecision: "deny"` を含む JSON 本体を持つ 2xx レスポンスを返します。

この例は、`PreToolUse` イベントをローカル検証サービスに送信し、`MY_TOKEN` 環境変数からのトークンで認証します。

```
{
  "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 フックは設定 JSON を直接編集して設定する必要があります。 `/hooks` インタラクティブメニューはコマンドフックの追加のみをサポートしています。

プロンプトフックとエージェントフックフィールド

[共通フィールド](#)に加えて、プロンプトフックとエージェントフックはこれらのフィールドを受け入れます。

フィールド	必須	説明
<code>prompt</code>	はい	モデルに送信するプロンプトテキスト。フック入力 JSON のプレースホルダーとして <code>\$ARGUMENTS</code> を使用します

フィールド	必須	説明
<code>model</code>	いいえ	評価に使用するモデル。デフォルトは高速モデル

マッチするすべてのフックは並列で実行され、同一のハンドラーは自動的に重複排除されます。コマンド フックはコマンド文字列で重複排除され、HTTP フックは URL で重複排除されます。ハンドラーは Claude Code の環境で現在のディレクトリで実行されます。`$CLAUDE_CODE_REMOTE` 環境変数はリモート Web 環境で `"true"` に設定され、ローカル CLI では設定されません。

パスでスクリプトを参照

環境変数を使用して、フック スクリプトをプロジェクトまたはプラグイン ルートを基準に参照し、フックが実行されるときに作業ディレクトリに関係なく参照します。

- `$CLAUDE_PROJECT_DIR`: プロジェクト ルート。スペースを含むパスを処理するために引用符で囲みます。
- `${CLAUDE_PLUGIN_ROOT}`: プラグインのルート ディレクトリ。[プラグイン](#) にバンドルされたスクリプト用です。

プロジェクトスクリプト

この例は `$CLAUDE_PROJECT_DIR` を使用して、`Write` または `Edit` ツール呼び出しの後、プロジェクトの `.claude/hooks/` ディレクトリからスタイル チェッカーを実行します。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/check-style.sh"
          }
        ]
      }
    ]
  }
}
```

プラグイン スクリプト

`hooks/hooks.json` でプラグインフックを定義し、オプションのトップレベル `description` フィールドを含めます。プラグインが有効な場合、そのフックはユーザーおよびプロジェクトフックとマージされます。

この例は、プラグインにバンドルされたフォーマットスクリプトを実行します。

```

{
  "description": "Automatic code formatting",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format.sh",
            "timeout": 30
          }
        ]
      }
    ]
  }
}

```

プラグイン フックの作成の詳細については、[プラグイン コンポーネント リファレンス](#)を参照してください。

スキルとエージェントのフック

設定ファイルとプラグインに加えて、フックはフロントマターを使用して[スキル](#)と[サブエージェント](#)で直接定義できます。これらのフックはコンポーネントのライフサイクルにスコープされ、そのコンポーネントがアクティブな場合にのみ実行されます。

すべてのフック イベントがサポートされています。サブエージェントの場合、**Stop** フックは自動的に **SubagentStop** に変換されます。これはサブエージェントが完了したときに発火するイベントです。

フックは設定ベースのフックと同じ設定形式を使用しますが、コンポーネントの有効期間にスコープされ、完了時にクリーンアップされます。

このスキルは、各 **Bash** コマンドの前にセキュリティ検証スクリプトを実行する **PreToolUse** フックを定義します。

```

---
name: secure-operations
description: Perform operations with security checks
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/security-check.sh"
---

```

エージェントはYAML フロントマターで同じ形式を使用します。

`/hooks` メニュー

Claude Code で `/hooks` と入力して、インタラクティブフック マネージャーを開きます。ここで、設定ファイルを直接編集せずにフックを表示、追加、削除できます。ステップバイステップのワークスルーについては、ガイドの[最初のフックを設定](#)を参照してください。

メニュー内の各フックには、ソースを示すブラケット プレフィックスがラベル付けされています。

- `[User]`: `~/.claude/settings.json` から
- `[Project]`: `.claude/settings.json` から
- `[Local]`: `.claude/settings.local.json` から
- `[Plugin]`: プラグインの `hooks/hooks.json` から、読み取り専用

フックを無効化または削除

フックを削除するには、設定 JSON ファイルからそのエントリを削除するか、`/hooks` メニューを使用してフックを選択して削除します。

すべてのフックを削除せずに一時的に無効化するには、設定ファイルで

`"disableAllHooks": true` を設定するか、`/hooks` メニューのトグルを使用します。個別のフックを設定に保持したまま無効化する方法はありません。

`disableAllHooks` 設定は管理設定階層を尊重します。管理者が管理ポリシー設定を通じてフックを設定している場合、ユーザー、プロジェクト、またはローカル設定で設定された `disableAllHooks` はそれらの管理フックを無効化できません。管理設定レベルで設定された `disableAllHooks` のみが管理フックを無効化できます。

設定ファイルのフックへの直接編集は即座には有効になりません。Claude Code はスタートアップ時にフックのスナップショットをキャプチャし、セッション全体で使用します。これにより、悪意のある、または偶発的なフック変更がセッション中盤に有効になるのを防ぎます。フックが外部で変更された場合、Claude Code は警告を表示し、変更が適用される前に `/hooks` メニューでレビューが必要です。

フック入出力

コマンド フックは stdin 経由で JSON データを受け取り、終了コード、stdout、stderr を通じて結果を通信します。HTTP フックは同じ JSON を POST リクエスト本体として受け取り、HTTP レスポンス本体を通じて結果を通信します。このセクションでは、すべてのイベントに共通するフィールドと動作について説明します。[フック イベント](#)の各セクションには、その特定の入力スキーマと決定制御オプションが含まれています。

共通入力フィールド

すべてのフック イベントは、各[フック イベント](#)セクションで説明されているイベント固有のフィールドに加えて、これらのフィールドを JSON として受け取ります。コマンド フックの場合、この JSON は stdin 経由で到着します。HTTP フックの場合、POST リクエスト本体として到着します。

フィールド	説明
<code>session_id</code>	現在のセッション識別子
<code>transcript_path</code>	会話 JSON へのパス
<code>cwd</code>	フックが呼び出されときの現在の作業ディレクトリ
<code>permission_mode</code>	現在の 権限モード : <code>"default"</code> 、 <code>"plan"</code> 、 <code>"acceptEdits"</code> 、 <code>"dontAsk"</code> 、または <code>"bypassPermissions"</code>
<code>hook_event_name</code>	発火したイベントの名前

`--agent` で実行するか、サブエージェント内で実行する場合、2 つの追加フィールドが含まれます。

フィールド	説明
<code>agent_id</code>	サブエージェントの一意の識別子。フックがサブエージェント呼び出し内で発火する場合にのみ存在します。サブエージェント フック呼び出しをメイン スレッド呼び出しから区別するために使用します。

フィールド	説明
<code>agent_type</code>	エージェント名（例えば、 <code>"Explore"</code> または <code>"security-reviewer"</code> ）。セッションが <code>--agent</code> を使用するか、フックがサブエージェント内で発火する場合に存在します。サブエージェントの場合、サブエージェントのタイプがセッションの <code>--agent</code> 値よりも優先されます。

例えば、Bash コマンドの `PreToolUse` フックは stdin で以下を受け取ります。

```
{
  "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"
  }
}
```

`tool_name` と `tool_input` フィールドはイベント固有です。各[フック イベント](#)セクションでは、そのイベントの追加フィールドについて説明しています。

終了コード出力

フック コマンドからの終了コードは、Claude Code にアクションを続行するか、ブロックするか、無視するかを指示します。

終了 0 は成功を意味します。Claude Code は stdout を [JSON 出力フィールド](#) で解析します。JSON 出力は終了 0 のみ処理されます。ほとんどのイベントでは、stdout は詳細モード（`Ctrl+0`）でのみ表示されます。例外は `UserPromptSubmit` と `SessionStart` で、stdout は Claude が見て行動できるコンテキストとして追加されます。

終了 2 はブロッキング エラーを意味します。Claude Code は stdout と JSON を無視します。代わりに、stderr テキストが Claude にエラー メッセージとしてフィードバックされます。効果はイベントに依存します。`PreToolUse` はツール呼び出しをブロックし、`UserPromptSubmit` はプロンプトを拒否します。詳細については、[終了コード 2 動作](#)を参照してください。

その他の終了コード は非ブロッキング エラーです。stderr は詳細モード（`Ctrl+0`）で表示され、実行は続行されます。

例えば、危険な Bash コマンドをブロックするフック コマンド スクリプト。

```
#!/bin/bash
## stdin から JSON 入力を読み取り、コマンドをチェック
command=$(jq -r '.tool_input.command' < /dev/stdin)

if [[ "$command" == rm* ]]; then
    echo "Blocked: rm commands are not allowed" >&2
    exit 2 # ブロッキング エラー: ツール呼び出しが防止される
fi

exit 0 # 成功: ツール呼び出しが続行される
```

イベントごとの終了コード 2 動作

終了コード 2 は、フックが「停止、これをしないでください」と通知する方法です。効果はイベントに依存します。一部のイベントはブロック可能なアクション（まだ発生していないツール呼び出しなど）を表し、他のイベントはすでに発生したか防止できないことを表します。

フック イベント	ブロック可能？	終了 2 で何が起こるか
PreToolUse	はい	ツール呼び出しをブロック
PermissionRequest	はい	権限を拒否
UserPromptSubmit	はい	プロンプト処理をブロックしてプロンプトを消去
Stop	はい	Claude が停止するのを防ぎ、会話を続行
SubagentStop	はい	サブエージェントが停止するのを防止
TeammateIdle	はい	チームメイトがアイドル状態になるのを防止（チームメイトが作業を続行）
TaskCompleted	はい	タスクが完了としてマークされるのを防止

フック イベント	ブロック可能？	終了 2 で何が起こるか
<code>ConfigChange</code>	はい	設定変更が有効になるのをブロック (<code>policy_settings</code> を除く)
<code>PostToolUse</code>	いいえ	Claude に stderr を表示 (ツールはすでに実行)
<code>PostToolUseFailure</code>	いいえ	Claude に stderr を表示 (ツールはすでに失敗)
<code>Notification</code>	いいえ	ユーザーのみに stderr を表示
<code>SubagentStart</code>	いいえ	ユーザーのみに stderr を表示
<code>SessionStart</code>	いいえ	ユーザーのみに stderr を表示
<code>SessionEnd</code>	いいえ	ユーザーのみに stderr を表示
<code>PreCompact</code>	いいえ	ユーザーのみに stderr を表示
<code>WorktreeCreate</code>	はい	0 以外の終了コードはワークツリー作成を失敗させる
<code>WorktreeRemove</code>	いいえ	失敗はデバッグ モードでのみログに記録
<code>InstructionsLoaded</code>	いいえ	終了コードは無視される

HTTP レスポンス処理

HTTP フックは終了コードと stdout の代わりに HTTP ステータス コードとレスポンス本体を使用します。

- **2xx で空の本体:** 成功、終了コード 0 で出力なしと同等
- **2xx でプレーン テキスト本体:** 成功、テキストがコンテキストとして追加される
- **2xx で JSON 本体:** 成功、コマンド フックと同じ [JSON 出力](#) スキーマを使用して解析
- **2xx 以外のステータス:** 非ブロッキング エラー、実行は続行

- ・ **接続失敗またはタイムアウト**: 非ブロッキング エラー、実行は続行

コマンド フックとは異なり、HTTP フックはステータス コードのみでブロッキング エラーを通知できません。ツール呼び出しをブロックまたは権限を拒否するには、適切な決定フィールドを含む JSON 本体を持つ 2xx レスポンスを返します。

JSON 出力

終了コードでは許可またはブロックできますが、JSON 出力はより細かい制御を提供します。終了コード 2 でブロックする代わりに、終了 0 して JSON オブジェクトを stdout に出力します。Claude Code はその JSON から特定のフィールドを読み取り、ブロック、許可、またはユーザーへのエスカレーションを含む動作を制御します。

Note:

フックごとに 1 つのアプローチを選択する必要があります。両方ではありません。終了コードのみでシグナリングするか、終了 0 して構造化制御のために JSON を出力するかです。Claude Code は終了 0 でのみ JSON を処理します。終了 2 の場合、JSON は無視されます。

フックの stdout には JSON オブジェクトのみが含まれている必要があります。シェル プロファイルがスタートアップ時にテキストを出力する場合、JSON 解析に干渉する可能性があります。トラブルシューティング ガイドの[JSON 検証に失敗](#)を参照してください。

JSON オブジェクトは 3 種類のフィールドをサポートしています。

- ・ **continue** などのユニバーサル フィールド はすべてのイベントで機能します。これらは以下の表にリストされています。
- ・ トップレベルの **decision** と **reason** は一部のイベントで使用され、ブロックまたはフィードバックを提供します。
- ・ **hookSpecificOutput** はより豊かな制御が必要なイベント用のネストされたオブジェクトです。イベント名に設定された **hookEventName** フィールドが必要です。

フィールド	デフォルト	説明
continue	true	false の場合、フック実行後 Claude は完全に処理を停止します。イベント固有の決定フィールドよりも優先されます

フィールド	デフォルト	説明
<code>stopReason</code>	なし	<code>continue</code> が <code>false</code> のときにユーザーに表示されるメッセージ。Claude には表示されません
<code>suppressOutput</code>	<code>false</code>	<code>true</code> の場合、詳細モード出力から stdout を非表示にします
<code>systemMessage</code>	なし	ユーザーに表示される警告メッセージ

Claude を完全に停止するには、イベント タイプに関係なく。

```
{ "continue": false, "stopReason": "Build failed, fix errors before continuing" }
```

決定制御

すべてのイベントが JSON を通じたブロッキングまたは動作制御をサポートしているわけではありません。サポートするイベントは、その決定を表現するために異なるフィールドセットを使用します。フックを書く前に、このテーブルをクイック リファレンスとして使用してください。

イベント	決定パターン	キー フィールド
UserPromptSubmit、 PostToolUse、 PostToolUseFailure、 Stop、SubagentStop、 ConfigChange	トップレベル <code>decision</code>	<code>decision: "block"</code> 、 <code>reason</code>
TeammateIdle、 TaskCompleted	終了コードまたは <code>continue: false</code>	終了コード 2 はアクションをブロックし、stderr フィードバックを提供します。JSON <code>{"continue": false, "stopReason": "...}"</code> はチームメイト全体を停止し、 <code>Stop</code> フック動作と一致します

イベント	決定パターン	キー フィールド
PreToolUse	<code>hookSpecificOutput</code>	<code>permissionDecision</code> (allow/deny/ask) 、 <code>permissionDecisionReason</code>
PermissionRequest	<code>hookSpecificOutput</code>	<code>decision.behavior</code> (allow/deny)
WorktreeCreate	stdout パス	フックは作成されたワークツリーへの絶対パスを出力します。0 以外の終了は作成を失敗させます
WorktreeRemove 、 Notification 、 SessionEnd 、 PreCompact 、 InstructionsLoaded	なし	決定制御なし。ログやク リーンアップなどの副作用 に使用

各パターンの実行例を以下に示します。

トップレベル決定

`UserPromptSubmit` 、 `PostToolUse` 、 `PostToolUseFailure` 、 `Stop` 、 `SubagentStop` 、 `ConfigChange` で使用されます。唯一の値は `"block"` です。アクションを続行させるには、JSON から `decision` を省略するか、JSON なしで終了 0 で終了します。

```
{
  "decision": "block",
  "reason": "Test suite must pass before proceeding"
}
```

PreToolUse

より豊かな制御のために `hookSpecificOutput` を使用します。許可、拒否、またはユーザーへのエスカレーション。ツール入力を実行前に変更したり、Claude 用に追加コンテキストを注入することもできます。オプションの完全なセットについては、[PreToolUse 決定制御](#)を参照してください。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Database writes are not allowed"
  }
}
```

PermissionRequest

`hookSpecificOutput` を使用して、ユーザーに代わって権限リクエストを許可または拒否します。許可する場合、ツールの入力を変更したり、権限ルールを適用して、ユーザーが再度プロンプトされないようにすることもできます。オプションの完全なセットについては、[PermissionRequest 決定制御](#)を参照してください。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

Bash コマンド検証、プロンプトフィルタリング、自動承認スクリプトを含む拡張例については、ガイドの[自動化できること](#)と[Bash コマンドバリデーター リファレンス実装](#)を参照してください。

フック イベント

各イベントは Claude Code のライフサイクル内のポイントに対応し、フックが実行できます。以下のセクションはライフサイクルに一致するように順序付けられています。セッションセットアップから agentic ループを経由してセッション終了まで。各セクションでは、イベントがいつ発火するか、サポートするマッチャー、受け取る JSON 入力、出力を通じた動作制御方法について説明しています。

SessionStart

新しいセッションを開始するか、既存のセッションを再開するときに実行されます。既存の問題や最近のコードベース変更など、開発コンテキストを読み込んだり、環境変数を設定したりするのに便利です。静的コンテキストでスクリプトが不要な場合は、代わりに [CLAUDE.md](#) を使用してください。

SessionStart はすべてのセッションで実行されるため、これらのフックを高速に保ちます。 `type: "command"` フックのみがサポートされています。

マッチャー値はセッションがどのように開始されたかに対応します。

マッチャー	いつ発火するか
<code>startup</code>	新しいセッション
<code>resume</code>	<code>--resume</code> 、 <code>--continue</code> 、または <code>/resume</code>
<code>clear</code>	<code>/clear</code>
<code>compact</code>	自動またはマニュアル コンパクション

SessionStart 入力

[共通入力フィールド](#)に加えて、SessionStart フックは `source`、`model`、およびオプションで `agent_type` を受け取ります。 `source` フィールドはセッションがどのように開始されたかを示します。新しいセッションの場合は `"startup"`、再開されたセッションの場合は `"resume"`、`/clear` の後は `"clear"`、コンパクション後は `"compact"`。 `model` フィールドはモデル識別子を含みます。 `claude --agent <name>` で Claude Code を開始する場合、`agent_type` フィールドはエージェント名を含みます。

```
{
  "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 決定制御

フック スクリプトが stdout に出力するテキストは Claude のコンテキストとして追加されます。すべてのフックで利用可能な [JSON 出力フィールド](#) に加えて、これらのイベント固有のフィールドを返すことができます。

フィールド	説明
<code>additionalContext</code>	Claude のコンテキストに追加される文字列。複数のフックの値は連結されます

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "My additional context here"
  }
}
```

環境変数を永続化

SessionStart フックは `CLAUDE_ENV_FILE` 環境変数にアクセスでき、後続の Bash コマンド用に環境変数を永続化できるファイル パスを提供します。

個別の環境変数を設定するには、`CLAUDE_ENV_FILE` に `export` ステートメントを書き込みます。他のフックで設定された変数を保持するには、追加 (`>>`) を使用します。

```
#!/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
```

セットアップ コマンドからのすべての環境変更をキャプチャするには、前後でエクスポートされた変数を比較します。

```
#!/bin/bash

ENV_BEFORE=$(export -p | sort)

## 環境を変更するセットアップ コマンドを実行
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
```

このファイルに書き込まれた変数は、セッション中に Claude Code が実行するすべての後続の Bash コマンドで利用可能になります。

Note:

`CLAUDE_ENV_FILE` は SessionStart フックで利用可能です。他のフック タイプはこの変数にアクセスできません。

InstructionsLoaded

`CLAUDE.md` または `.claude/rules/*.md` ファイルがコンテキストに読み込まれるときに発火します。このイベントはセッション開始時に熱心に読み込まれたファイルに対して発火し、後で遅延読み込みされたときに再度発火します。例えば、Claude がネストされた `CLAUDE.md` を含むサブディレクトリにアクセスするか、`paths:` フロントマターを持つ条件付きルールがマッチするとき。フックはブロッキングまたは決定制御をサポートしていません。観測可能性の目的で非同期に実行されます。

InstructionsLoaded はマッチャーをサポートせず、すべての読み込み出現で発火します。

InstructionsLoaded 入力

[共通入力フィールド](#)に加えて、InstructionsLoaded フックはこれらのフィールドを受け取ります。

フィールド	説明
<code>file_path</code>	読み込まれた命令ファイルへの絶対パス
<code>memory_type</code>	ファイルのスコープ: <code>"User"</code> 、 <code>"Project"</code> 、 <code>"Local"</code> 、または <code>"Managed"</code>
<code>load_reason</code>	ファイルが読み込まれた理由: <code>"session_start"</code> 、 <code>"nested_traversal"</code> 、 <code>"path_glob_match"</code> 、または <code>"include"</code>
<code>globs</code>	ファイルの <code>paths</code> : フロントマターからのパス グロブ パターン（存在する場合）。 <code>path_glob_match</code> 読み込みの場合のみ存在
<code>trigger_file_path</code>	遅延読み込みの場合、この読み込みをトリガーしたファイルへのパス
<code>parent_file_path</code>	<code>include</code> 読み込みの場合、このファイルを含む親命令ファイルへのパス

```
{
  "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 決定制御

InstructionsLoaded フックは決定制御がありません。命令読み込みをブロックまたは変更できません。このイベントを監査ログ、コンプライアンス追跡、または観測可能性に使用します。

UserPromptSubmit

ユーザーがプロンプトを送信するときに実行されます。Claude がそれを処理する前です。これにより、プロンプト/会話に基づいて追加コンテキストを追加したり、プロンプトを検証したり、特定のタイプのプロンプトをブロックしたりできます。

UserPromptSubmit 入力

[共通入力フィールド](#)に加えて、UserPromptSubmit フックはユーザーが送信したテキストを含む `prompt` フィールドを受け取ります。

```
{
  "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 決定制御

`UserPromptSubmit` フックはユーザー プロンプトが処理されるかどうかを制御し、コンテキストを追加できます。すべての[JSON 出力フィールド](#)が利用可能です。

終了コード 0 で会話にコンテキストを追加する 2 つの方法があります。

- **プレーン テキスト stdout:** stdout に書き込まれた非 JSON テキストはコンテキストとして追加されます
- **`additionalContext` を含む JSON:** より多くの制御のために以下の JSON 形式を使用します。`additionalContext` フィールドはコンテキストとして追加されます

プレーン stdout はトランスクリプトでフック出力として表示されます。

`additionalContext` フィールドはより控えめに追加されます。

プロンプトをブロックするには、`decision` を `"block"` に設定した JSON オブジェクトを返します。

フィールド	説明
<code>decision</code>	<code>"block"</code> はプロンプトが処理されるのを防ぎ、コンテキストから消去します。許可するには省略します
<code>reason</code>	<code>decision</code> が <code>"block"</code> のときにユーザーに表示されます。コンテキストに追加されません
<code>additionalContext</code>	Claude のコンテキストに追加される文字列

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "My additional context here"
  }
}
```

Note:

JSON 形式は単純なユースケースでは必須ではありません。コンテキストを追加するには、終了コード 0 で stdout にプレーン テキストを出力できます。プロンプトをブロックしたい場合、またはより構造化された制御が必要な場合は JSON を使用します。

PreToolUse

Claude がツール パラメーターを作成した後、ツール呼び出しを処理する前に実行されます。ツール名でマッチします。 `Bash`、`Edit`、`Write`、`Read`、`Glob`、`Grep`、`Agent`、`WebFetch`、`WebSearch`、および任意の [MCP ツール名](#)。

[PreToolUse 決定制御](#)を使用して、ツールの使用を許可、拒否、または許可を求めます。

PreToolUse 入力

[共通入力フィールド](#)に加えて、PreToolUse フックは `tool_name`、`tool_input`、`tool_use_id` を受け取ります。`tool_input` フィールドはツールに依存します。

Bash

シェル コマンドを実行します。

フィールド	タイプ	例	説明
<code>command</code>	文字列	<code>"npm test"</code>	実行するシェル コマンド
<code>description</code>	文字列	<code>"Run test suite"</code>	コマンドが何を するかのオプション 説明

フィールド	タイプ	例	説明
<code>timeout</code>	数値	<code>120000</code>	ミリ秒単位のオプション タイムアウト
<code>run_in_background</code>	ブール値	<code>false</code>	コマンドをバックグラウンドで実行するかどうか

Write

ファイルを作成または上書きします。

フィールド	タイプ	例	説明
<code>file_path</code>	文字列	<code>"/path/to/file.txt"</code>	書き込むファイルへの絶対パス
<code>content</code>	文字列	<code>"file content"</code>	ファイルに書き込むコンテンツ

Edit

既存ファイル内の文字列を置換します。

フィールド	タイプ	例	説明
<code>file_path</code>	文字列	<code>"/path/to/file.txt"</code>	編集するファイルへの絶対パス
<code>old_string</code>	文字列	<code>"original text"</code>	検索して置換するテキスト
<code>new_string</code>	文字列	<code>"replacement text"</code>	置換テキスト
<code>replace_all</code>	ブール値	<code>false</code>	すべての出現を置換するかどうか

Read

ファイル コンテンツを読み取ります。

フィールド	タイプ	例	説明
<code>file_path</code>	文字列	<code>"/path/to/file.txt"</code>	読み取るファイルへの絶対パス
<code>offset</code>	数値	<code>10</code>	読み取りを開始する行番号のオプション
<code>limit</code>	数値	<code>50</code>	読み取る行数のオプション

Glob

グロブ パターンに一致するファイルを検索します。

フィールド	タイプ	例	説明
<code>pattern</code>	文字列	<code>"**/*.ts"</code>	ファイルに対してマッチするグロブ パターン
<code>path</code>	文字列	<code>"/path/to/dir"</code>	検索するオプションディレクトリ。デフォルトは現在の作業ディレクトリ

Grep

正規表現でファイル コンテンツを検索します。

フィールド	タイプ	例	説明
<code>pattern</code>	文字列	<code>"TODO.*fix"</code>	検索する正規表現パターン
<code>path</code>	文字列	<code>"/path/to/dir"</code>	検索するオプションファイルまたはディレクトリ
<code>glob</code>	文字列	<code>"*.ts"</code>	ファイルをフィルタリングするオプショングロブ パターン

フィールド	タイプ	例	説明
<code>output_mode</code>	文字列	<code>"content"</code>	<code>"content"</code> 、 <code>"files_with_matches"</code> 、または <code>"count"</code> 。デフォルトは <code>"files_with_matches"</code>
<code>-i</code>	ブール値	<code>true</code>	大文字小文字を区別しない検索
<code>multiline</code>	ブール値	<code>false</code>	マルチライン マッチングを有効化

WebFetch

Web コンテンツを取得して処理します。

フィールド	タイプ	例	説明
<code>url</code>	文字列	<code>"https://example.com/api"</code>	コンテンツを取得する URL
<code>prompt</code>	文字列	<code>"Extract the API endpoints"</code>	取得したコンテンツで実行するプロンプト

WebSearch

Web を検索します。

フィールド	タイプ	例	説明
<code>query</code>	文字列	<code>"react hooks best practices"</code>	検索クエリ
<code>allowed_domains</code>	配列	<code>["docs.example.com"]</code>	オプション: これらのドメインからのみ結果を含める
<code>blocked_domains</code>	配列	<code>["spam.example.com"]</code>	オプション: これらのドメインからの結果を除外

Agent

[サブエージェント](#)を生成します。

フィールド	タイプ	例	説明
<code>prompt</code>	文字列	<code>"Find all API endpoints"</code>	エージェントが実行するタスク
<code>description</code>	文字列	<code>"Find API endpoints"</code>	タスクの短い説明
<code>subagent_type</code>	文字列	<code>"Explore"</code>	使用する特殊エージェントのタイプ
<code>model</code>	文字列	<code>"sonnet"</code>	デフォルトをオーバーライドするオプション モデル エイリアス

PreToolUse 決定制御

`PreToolUse` フックはツール呼び出しが続行されるかどうかを制御できます。トップレベル `decision` フィールドを使用する他のフックとは異なり、`PreToolUse` は `hookSpecificOutput` オブジェクト内に決定を返します。これにより、より豊かな制御が可能になります。3つの結果（許可、拒否、または質問）と、実行前にツール入力を変更する機能。

フィールド	説明
<code>permissionDecision</code>	<code>"allow"</code> は権限システムをバイパスし、 <code>"deny"</code> はツール呼び出しを防ぎ、 <code>"ask"</code> はユーザーに確認を促します
<code>permissionDecisionReason</code>	<code>"allow"</code> と <code>"ask"</code> の場合、ユーザーに表示されますが Claude には表示されません。 <code>"deny"</code> の場合、Claude に表示されます
<code>updatedInput</code>	実行前にツールの入力パラメーターを変更します。 <code>"allow"</code> と組み合わせて自動承認するか、 <code>"ask"</code> と組み合わせて変更された入力をユーザーに表示します
<code>additionalContext</code>	ツール実行前に Claude のコンテキストに追加される文字列

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow",
    "permissionDecisionReason": "My reason here",
    "updatedInput": {
      "field_to_modify": "new value"
    },
    "additionalContext": "Current environment: production. Proceed with caution."
  }
}
```

Note:

PreToolUse は以前、トップレベル `decision` と `reason` フィールドを使用していました
が、このイベントでは非推奨です。代わりに `hookSpecificOutput.permissionDecision`
と `hookSpecificOutput.permissionDecisionReason` を使用してください。非推奨の値
"approve" と "block" は "allow" と "deny" にマップされます。PostToolUse と Stop
などの他のイベントは、現在の形式としてトップレベル `decision` と `reason` を使用し続け
ます。

PermissionRequest

ユーザーに権限ダイアログが表示されるときに実行されます。 [PermissionRequest 決定制御](#)を使用して、ユーザーに代わって許可または拒否します。

ツール名でマッチします。PreToolUse と同じ値。

PermissionRequest 入力

PermissionRequest フックは PreToolUse フックのような `tool_name` と `tool_input`
フィールドを受け取りますが、`tool_use_id` はありません。オプションの
`permission_suggestions` 配列には、ユーザーが通常権限ダイアログで見る「常に許可」
オプションが含まれています。違いはフックが発火するタイミングです。

PermissionRequest フックはユーザーに権限ダイアログが表示されようとしているとき
に実行され、PreToolUse フックは権限ステータスに関係なくツール実行前に実行されま
す。

```
{
  "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 決定制御

`PermissionRequest` フックは権限リクエストを許可または拒否できます。すべてのフックで利用可能な[JSON 出力フィールド](#)に加えて、フック スクリプトはこれらのイベント固有のフィールドを持つ `decision` オブジェクトを返すことができます。

フィールド	説明
<code>behavior</code>	<code>"allow"</code> は権限を付与し、 <code>"deny"</code> はそれを拒否します
<code>updatedInput</code>	<code>"allow"</code> のみ: 実行前にツールの入力パラメーターを変更します
<code>updatedPermissions</code>	<code>"allow"</code> のみ: 権限ルール更新を適用します。ユーザーが「常に許可」オプションを選択するのと同様です
<code>message</code>	<code>"deny"</code> のみ: 権限が拒否された理由を Claude に伝えます
<code>interrupt</code>	<code>"deny"</code> のみ: <code>true</code> の場合、Claude を停止します

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

PostToolUse

ツールが正常に完了した直後に実行されます。

ツール名でマッチします。PreToolUse と同じ値。

PostToolUse 入力

PostToolUse フックはツールがすでに正常に実行された後に発火します。入力には、ツールに送信された引数である `tool_input` と、それが返した結果である `tool_response` の両方が含まれます。両方の正確なスキーマはツールに依存します。

```
{
  "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 決定制御

PostToolUse フックはツール実行後に Claude にフィードバックを提供できます。すべてのフックで利用可能な **JSON 出力フィールド**に加えて、フック スクリプトはこれらのイベント固有のフィールドを返すことができます。

フィールド	説明
<code>decision</code>	<code>"block"</code> は Claude に <code>reason</code> でプロンプトを表示します。許可するには省略します
<code>reason</code>	<code>decision</code> が <code>"block"</code> のときに Claude に表示される説明
<code>additionalContext</code>	Claude が考慮する追加コンテキスト
<code>updatedMCPToolOutput</code>	MCP ツール のみ: ツールの出力を提供された値に置き換えます

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "additionalContext": "Additional information for Claude"
  }
}
```

PostToolUseFailure

ツール実行が失敗するときに実行されます。このイベントはエラーをスローするか失敗結果を返すツール呼び出しに対して発火します。これを使用して失敗をログに記録したり、アラートを送信したり、Claude に是正フィードバックを提供したりします。

ツール名でマッチします。PreToolUse と同じ値。

PostToolUseFailure 入力

PostToolUseFailure フックは PostToolUse と同じ `tool_name` と `tool_input` フィールドを受け取り、エラー情報をトップレベル フィールドとして受け取ります。

```
{
  "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
}
```

フィールド	説明
<code>error</code>	何が悪かったかを説明する文字列
<code>is_interrupt</code>	失敗がユーザー割り込みによって引き起こされたかどうかを示すオプション ブール値

PostToolUseFailure 決定制御

`PostToolUseFailure` フックはツール失敗後に Claude にコンテキストを提供できます。すべてのフックで利用可能な [JSON 出力フィールド](#) に加えて、フック スクリプトはこれらのイベント固有のフィールドを返すことができます。

フィールド	説明
<code>additionalContext</code>	Claude がエラーと一緒に考慮する追加情報

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUseFailure",
    "additionalContext": "Additional information about the failure for Claude"
  }
}
```

Notification

Claude Code が通知を送信するときに実行されます。通知タイプでマッチします。

`permission_prompt`、`idle_prompt`、`auth_success`、`elicitation_dialog`。すべての通知タイプのフックを実行するには、マッチャーを省略します。

別のマッチャーを使用して、通知タイプに応じて異なるハンドラーを実行します。この設定は、Claude が権限承認を必要とするときに権限固有のアラート スクリプトをトリガーし、Claude がアイドル状態になったときに異なる通知をトリガーします。

```
{
  "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 入力

[共通入力フィールド](#)に加えて、Notification フックは通知テキストを含む `message`、オブジェクトの `title`、発火した種類を示す `notification_type` を受け取ります。


```
{
  "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 フックは通知をブロックまたは変更できません。すべてのフックで利用可能な[JSON 出力フィールド](#)に加えて、`additionalContext` を返して会話にコンテキストを追加できます。

フィールド	説明
<code>additionalContext</code>	Claude のコンテキストに追加される文字列

SubagentStart

Agent ツール経由で Claude Code サブエージェントが生成されるときに実行されます。エージェント タイプ名でフィルタリングするマッチャーをサポートします（`Bash`、`Explore`、`Plan` などの組み込みエージェント、または `.claude/agents/` からのカスタム エージェント名）。

SubagentStart 入力

[共通入力フィールド](#)に加えて、SubagentStart フックはサブエージェントの一意の識別子を含む `agent_id` とエージェント名を含む `agent_type`（`"Bash"`、`"Explore"`、`"Plan"` などの組み込みエージェント、またはカスタム エージェント名）を受け取ります。

```
{
  "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 フックはサブエージェント作成をブロックできませんが、サブエージェントにコンテキストを注入できます。すべてのフックで利用可能な[JSON 出力フィールド](#)に加えて、以下を返すことができます。

フィールド	説明
<code>additionalContext</code>	サブエージェントのコンテキストに追加される文字列

```
{
  "hookSpecificOutput": {
    "hookEventName": "SubagentStart",
    "additionalContext": "Follow security guidelines for this task"
  }
}
```

SubagentStop

Claude Code サブエージェントが応答を終了したときに実行されます。エージェントタイプでマッチします。SubagentStart と同じ値。

SubagentStop 入力

[共通入力フィールド](#)に加えて、SubagentStop フックは `stop_hook_active`、`agent_id`、`agent_type`、`agent_transcript_path`、`last_assistant_message` を受け取ります。`agent_type` フィールドはマッチャー フィルタリングに使用される値です。`transcript_path` はメイン セッションのトランスクリプトで、`agent_transcript_path` はネストされた `subagents/` フォルダに保存されるサブエージェント独自のトランスクリ

プトです。 `last_assistant_message` フィールドはサブエージェントの最終応答のテキスト コンテンツを含むため、フックはトランスクリプト ファイルを解析せずにアクセスできます。

```
{
  "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 フックは[Stop フック](#)と同じ決定制御形式を使用します。

Stop

メイン Claude Code エージェントが応答を終了したときに実行されます。ユーザー割り込みが原因で停止が発生した場合は実行されません。

Stop 入力

[共通入力フィールド](#)に加えて、Stop フックは `stop_hook_active` と `last_assistant_message` を受け取ります。 `stop_hook_active` フィールドは、Claude Code がすでに stop フックの結果として続行している場合は `true` です。この値をチェックするか、トランスクリプトを処理して Claude Code が無限に実行されるのを防ぎます。 `last_assistant_message` フィールドは Claude の最終応答のテキスト コンテンツを含むため、フックはトランスクリプト ファイルを解析せずにアクセスできます。

```
{
  "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 決定制御

Stop と **SubagentStop** フックは Claude が続行するかどうかを制御できます。すべてのフックで利用可能な **JSON 出力フィールド** に加えて、フック スクリプトはこれらのイベント固有のフィールドを返すことができます。

フィールド	説明
decision	"block" は Claude が停止するのを防ぎます。Claude を停止させるには省略します
reason	decision が "block" のときに必須。Claude が続行すべき理由を伝えます

```
{
  "decision": "block",
  "reason": "Must be provided when Claude is blocked from stopping"
}
```

TeammateIdle

エージェント チーム チームメイトがターンを終了した後、アイドル状態になろうとしているときに実行されます。これを使用して、チームメイトが作業を停止する前に品質ゲートを実施します。例えば、リント チェックの合格を要求したり、出力ファイルが存在することを確認したりします。

TeammateIdle フックが終了コード 2 で終了すると、チームメイトは stderr メッセージをフィードバックとして受け取り、アイドル状態になる代わりに作業を続行します。チームメイトを再実行する代わりに完全に停止するには、`{"continue": false, "stopReason": "..."}` を含む JSON を返します。TeammateIdle フックはマッチャーをサポートせず、すべての出現で発火します。

TeammateIdle 入力

[共通入力フィールド](#)に加えて、TeammateIdle フックは `teammate_name` と `team_name` を受け取ります。

```
{
  "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"
}
```

フィールド	説明
<code>teammate_name</code>	アイドル状態になろうとしているチームメイトの名前
<code>team_name</code>	チームの名前

TeammateIdle 決定制御

TeammateIdle フックはチームメイト動作を制御する 2 つの方法をサポートしています。

- ・ **終了コード 2:** チームメイトは stderr メッセージをフィードバックとして受け取り、アイドル状態になる代わりに作業を続行します。
- ・ **JSON `{"continue": false, "stopReason": "..."}` :** チームメイトを完全に停止します。 `Stop` フック動作と一致します。 `stopReason` はユーザーに表示されます。

この例は、チームメイトがアイドル状態になることを許可する前に、ビルド アーティファクトが存在することをチェックします。

```
#!/bin/bash

if [ ! -f "./dist/output.js" ]; then
    echo "Build artifact missing. Run the build before stopping." >&2
    exit 2
fi

exit 0
```

TaskCompleted

タスクが完了としてマークされるときに実行されます。これは2つの状況で発火します。任意のエージェントがTaskUpdate ツール経由でタスクを明示的に完了としてマークするか、[エージェント チーム](#) チームメイトが進行中のタスクでターンを終了するとき。これを使用して、テストの合格やリント チェックなどの完了基準を実施します。

TaskCompleted フックが終了コード 2 で終了すると、タスクは完了としてマークされず、stderr メッセージはモデルにフィードバックとしてフィードバックされます。チームメイトを再実行する代わりに完全に停止するには、`{"continue": false, "stopReason": "..."}` を含む JSON を返します。TaskCompleted フックはマッチャーをサポートせず、すべての出現で発火します。

TaskCompleted 入力

[共通入力フィールド](#)に加えて、TaskCompleted フックは `task_id`、`task_subject`、およびオプションで `task_description`、`teammate_name`、`team_name` を受け取ります。

```
{
  "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"
}
```

フィールド	説明
<code>task_id</code>	完了しているタスクの識別子
<code>task_subject</code>	タスクのタイトル
<code>task_description</code>	タスクの詳細説明。存在しない場合があります
<code>teammate_name</code>	タスクを完了しているチームメイトの名前。存在しない場合があります
<code>team_name</code>	チームの名前。存在しない場合があります

TaskCompleted 決定制御

TaskCompleted フックはタスク完了を制御する 2 つの方法をサポートしています。

- **終了コード 2:** タスクは完了としてマークされず、stderr メッセージはモデルにフィードバックとしてフィードバックされます。
- **JSON `{"continue": false, "stopReason": "...}"`:** チームメイトを完全に停止します。`Stop` フック動作と一致します。`stopReason` はユーザーに表示されます。

この例はテストを実行し、失敗した場合はタスク完了をブロックします。

```
#!/bin/bash
INPUT=$(cat)
TASK_SUBJECT=$(echo "$INPUT" | jq -r '.task_subject')

## テストスイートを実行
if ! npm test 2>&1; then
    echo "Tests not passing. Fix failing tests before completing: $TASK_SUBJECT" >&2
    exit 2
fi

exit 0
```

ConfigChange

セッション中に設定ファイルが変更されるときに実行されます。設定変更を監査したり、セキュリティ ポリシーを実施したり、設定ファイルへの不正な変更をブロックしたりするために使用します。

ConfigChange フックは設定ファイル、管理ポリシー設定、スキル ファイルの変更に對して発火します。入力の `source` フィールドは、どのタイプの設定が変更されたかを示し、オプションの `file_path` フィールドは変更されたファイルへのパスを提供します。

マッチャーは設定ソースでフィルタリングします。

マッチャー	いつ発火するか
<code>user_settings</code>	<code>~/.claude/settings.json</code> が変更
<code>project_settings</code>	<code>.claude/settings.json</code> が変更
<code>local_settings</code>	<code>.claude/settings.local.json</code> が変更
<code>policy_settings</code>	管理ポリシー設定が変更
<code>skills</code>	<code>.claude/skills/</code> のスキル ファイルが変更

この例はセキュリティ 監査のためにすべての設定変更をログに記録します。


```
{
  "hooks": {
    "ConfigChange": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/audit-config-
change.sh"
          }
        ]
      }
    ]
  }
}
```

ConfigChange 入力

[共通入力フィールド](#)に加えて、ConfigChange フックは `source` とオプションで `file_path` を受け取ります。 `source` フィールドは変更された設定タイプを示し、 `file_path` は変更されたファイルへのパスを提供します。

```
{
  "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 決定制御

ConfigChange フックは設定変更が有効になるのをブロックできます。終了コード 2 または JSON `decision` を使用して変更を防ぎます。ブロックされた場合、新しい設定は実行中のセッションに適用されません。

フィールド	説明
<code>decision</code>	<code>"block"</code> は設定変更が適用されるのを防ぎます。変更を許可するには省略します
<code>reason</code>	<code>decision</code> が <code>"block"</code> のときにユーザーに表示される説明

```
{
  "decision": "block",
  "reason": "Configuration changes to project settings require admin approval"
}
```

`policy_settings` の変更はブロックできません。フックは `policy_settings` ソースに対して発火するため、監査ログに使用できますが、ブロッキング決定は無視されます。これにより、エンタープライズ管理設定が常に有効になることが保証されます。

WorktreeCreate

`claude --worktree` を実行するか、サブエージェントが `isolation: "worktree"` を使用するとき、Claude Code は `git worktree` を使用して分離されたワーキング コピーを作成します。WorktreeCreate フックを設定する場合、デフォルトの git 動作を置き換え、SVN、Perforce、Mercurial などの別のバージョン管理システムを使用できます。

フックは作成されたワークツリー ディレクトリへの絶対パスを stdout に出力する必要があります。Claude Code はこのパスを分離されたセッションの作業ディレクトリとして使用します。

この例は SVN ワーキング コピーを作成し、Claude Code が使用するパスを出力します。リポジトリ URL を自分のものに置き換えます。

```
{
  "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\\\"'"
          }
        ]
      }
    ]
  }
}
```

フックは stdin の JSON 入力からワークツリー `name` を読み取り、新しいディレクトリに新しいコピーをチェックアウトし、ディレクトリ パスを出力します。最後の行の `echo` は Claude Code が読み取るワークツリー パスです。他の出力を stderr にリダイレクトして、パスに干渉しないようにします。

WorktreeCreate 入力

[共通入力フィールド](#)に加えて、WorktreeCreate フックは `name` フィールドを受け取ります。これは新しいワークツリーのスラッグ識別子で、ユーザーが指定するか自動生成されます（例えば、`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 出力

フックは作成されたワークツリー ディレクトリへの絶対パスを stdout に出力する必要があります。フックが失敗するか出力を生成しない場合、ワークツリー作成はエラーで失敗します。

WorktreeCreate フックは標準的な許可/ブロック決定モデルを使用しません。代わりに、フックの成功または失敗が結果を決定します。 `type: "command"` フックのみがサポートされています。

WorktreeRemove

[WorktreeCreate](#) のクリーンアップ対応。このフックはワークツリーが削除されるときに発火します。 `--worktree` セッションを終了して削除を選択するか、 `isolation: "worktree"` を持つサブエージェントが完了するとき。git ベースのワークツリーの場合、Claude は `git worktree remove` で自動的にクリーンアップを処理します。非 git バージョン管理システムの WorktreeCreate フックを設定した場合、クリーンアップを処理するために WorktreeRemove フックとペアにします。なければ、ワークツリー ディレクトリはディスク上に残ります。

Claude Code は WorktreeCreate が stdout に出力したパスを `worktree_path` としてフック入力に渡します。この例はそのパスを読み取り、ディレクトリを削除します。

```
{
  "hooks": {
    "WorktreeRemove": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'jq -r .worktree_path | xargs rm -rf'"
          }
        ]
      }
    ]
  }
}
```

WorktreeRemove 入力

[共通入力フィールド](#)に加えて、WorktreeRemove フックは削除されるワークツリーへの絶対パスである `worktree_path` フィールドを受け取ります。

```
{
  "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 フックは決定制御がありません。ワークツリー削除をブロックできませんが、バージョン管理状態の削除やアーカイブ変更などのクリーンアップタスクを実行できます。フック失敗はデバッグモードでのみログに記録されます。

`type: "command"` フックのみがサポートされています。

PreCompact

Claude Code がコンパクト操作を実行しようとしている前に実行されます。

マッチャー値はコンパクションが手動でトリガーされたか自動でトリガーされたかを示します。

マッチャー	いつ発火するか
<code>manual</code>	<code>/compact</code>
<code>auto</code>	コンテキストウィンドウが満杯のときの自動コンパクション

PreCompact 入力

[共通入力フィールド](#)に加えて、PreCompact フックは `trigger` と `custom_instructions` を受け取ります。 `manual` の場合、`custom_instructions` はユーザーが `/compact` に渡すものを含みます。 `auto` の場合、`custom_instructions` は空です。

```
{
  "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

Claude Code セッションが終了するときに実行されます。クリーンアップ タスク、ログ セッション統計、またはセッション状態の保存に便利です。終了理由でフィルタリングするマッチャーをサポートします。

フック入力の `reason` フィールドはセッションが終了した理由を示します。

理由	説明
<code>clear</code>	<code>/clear</code> コマンドでセッションをクリア
<code>logout</code>	ユーザーがログアウト
<code>prompt_input_exit</code>	プロンプト入力が表示されている間にユーザーが終了
<code>bypass_permissions_disabled</code>	バイパス権限モードが無効化
<code>other</code>	その他の終了理由

SessionEnd 入力

[共通入力フィールド](#)に加えて、SessionEnd フックはセッションが終了した理由を示す `reason` フィールドを受け取ります。上記の[理由テーブル](#)をすべての値について参照してください。

```
{
  "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 フックは決定制御がありません。セッション終了をブロックできませんが、クリーンアップタスクを実行できます。

プロンプトベースのフック

コマンドフックとHTTPフックに加えて、Claude Code はプロンプトベースのフック（`type: "prompt"`）をサポートしており、LLM を使用してアクションを許可またはブロックするかを評価し、エージェントフック（`type: "agent"`）はツールアクセスを持つ agentic ベリファイアーを生成します。すべてのイベントがすべてのフックタイプをサポートしているわけではありません。

4 つのフックタイプ（`command`、`http`、`prompt`、`agent`）すべてをサポートするイベント。

- `PermissionRequest`
- `PostToolUse`
- `PostToolUseFailure`
- `PreToolUse`
- `Stop`
- `SubagentStop`
- `TaskCompleted`
- `UserPromptSubmit`

`type: "command"` フックのみをサポートするイベント。

- `ConfigChange`
- `InstructionsLoaded`
- `Notification`
- `PreCompact`

- `SessionEnd`
- `SessionStart`
- `SubagentStart`
- `TeammateIdle`
- `WorktreeCreate`
- `WorktreeRemove`

プロンプトベースのフックがどのように機能するか

Bash コマンドを実行する代わりに、プロンプトベースのフック。

1. フック入力とプロンプトを Claude モデル（デフォルトは Haiku）に送信
2. LLM は決定を含む構造化 JSON で応答
3. Claude Code は決定を自動的に処理

プロンプトフック設定

`type` を `"prompt"` に設定し、`command` の代わりに `prompt` 文字列を提供します。

`$ARGUMENTS` プレースホルダーを使用して、フックの JSON 入力データをプロンプトテキストに注入します。Claude Code は結合されたプロンプトと入力を高速 Claude モデルに送信し、JSON 決定を返します。

この `Stop` フックは、Claude が終了する前にすべてのタスクが完了しているかどうかを評価するよう LLM に求めます。


```

{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if Claude should stop: $ARGUMENTS. Check if all
tasks are complete."
          }
        ]
      }
    ]
  }
}

```

フィールド	必須	説明
<code>type</code>	はい	<code>"prompt"</code> である必要があります
<code>prompt</code>	はい	LLM に送信するプロンプトテキスト。フック入力 JSON のプレースホルダーとして <code>\$ARGUMENTS</code> を使用します。 <code>\$ARGUMENTS</code> が存在しない場合、入力 JSON はプロンプトに追加されます
<code>model</code>	いいえ	評価に使用するモデル。デフォルトは高速モデル
<code>timeout</code>	いいえ	秒単位のタイムアウト。デフォルト: 30

レスポンス スキーマ

LLM は以下を含む JSON で応答する必要があります。

```
{
  "ok": true | false,
  "reason": "Explanation for the decision"
}
```

フィールド	説明
<code>ok</code>	<code>true</code> はアクションを許可し、 <code>false</code> はそれを防ぎます
<code>reason</code>	<code>ok</code> が <code>false</code> のときに必須。Claude に表示される説明

例: マルチ基準 Stop フック

この `Stop` フックは詳細なプロンプトを使用して、Claude が停止することを許可する前に 3 つの条件をチェックします。 `"ok"` が `false` の場合、Claude は提供された理由を次の指示として受け取り、作業を続行します。 `SubagentStop` フックは同じ形式を使用して、[サブエージェント](#)が停止すべきかどうかを評価します。

```
{
  "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
          }
        ]
      }
    ]
  }
}
```

エージェントベースのフック

エージェントベースのフック（`type: "agent"`）はプロンプトベースのフックのようですが、マルチターンツールアクセスを備えています。単一の LLM 呼び出しの代わりに、エージェントフックはプロンプトとフックの JSON 入力を持つサブエージェントを生成します。サブエージェントはファイルを読み取り、コードを検索し、コードベースを検査して条件を検証できます。エージェントフックはプロンプトベースのフックと同じイベントをサポートしています。

エージェントフックがどのように機能するか

エージェントフックが発火するとき。

1. Claude Code はプロンプトとフックの JSON 入力を持つサブエージェントを生成
2. サブエージェントは Read、Grep、Glob などのツールを使用して調査できます
3. 最大 50 ターン後、サブエージェントは構造化 `{ "ok": true/false }` 決定を返します
4. Claude Code はプロンプトフックと同じ方法で決定を処理

エージェントフックは、フック入力データのみを評価するのではなく、実際のファイルを検査またはテスト出力を検査する必要がある検証に便利です。

エージェントフック設定

`type` を `"agent"` に設定し、`prompt` 文字列を提供します。設定フィールドは[プロンプトフック](#)と同じで、より長いデフォルトタイムアウト。

フィールド	必須	説明
<code>type</code>	はい	<code>"agent"</code> である必要があります
<code>prompt</code>	はい	検証する内容を説明するプロンプト。フック入力 JSON のプレースホルダーとして <code>\$ARGUMENTS</code> を使用します
<code>model</code>	いいえ	使用するモデル。デフォルトは高速モデル
<code>timeout</code>	いいえ	秒単位のタイムアウト。デフォルト: 60

レスポンス スキーマはプロンプト フックと同じです。許可するには `{ "ok": true }` またはブロックするには `{ "ok": false, "reason": "..." }`。

この `Stop` フックは、Claude が終了することを許可する前にすべてのユニット テストが合格することを検証します。

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

バックグラウンドでフックを実行

デフォルトでは、フックは完了するまで Claude の実行をブロックします。デプロイメント、テストスイート、外部 API 呼び出しなどの長時間実行タスクの場合、`"async": true` を設定して、Claude が作業を続行している間、フックをバックグラウンドで実行します。非同期フックはブロックまたは Claude の動作を制御できません。`decision`、`permissionDecision`、`continue` などのレスポンス フィールドは、制御しようとしたアクションがすでに完了しているため、効果がありません。

非同期フックを設定

コマンド フックの設定に `"async": true` を追加して、Claude をブロックせずにバックグラウンドで実行します。このフィールドは `type: "command"` フックでのみ利用可能です。

このフックは `Write` ツール呼び出しの後、テスト スクリプトを実行します。Claude は `run-tests.sh` が最大 120 秒間実行されている間、すぐに作業を続行します。スクリプトが完了すると、その出力は次の会話ターンで配信されます。

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/run-tests.sh",
            "async": true,
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

timeout フィールドはバックグラウンド プロセスの最大時間を秒単位で設定します。指定されない場合、非同期フックは同期フックと同じ 10 分のデフォルトを使用します。

非同期フックがどのように実行されるか

非同期フックが発火すると、Claude Code はフック プロセスを開始し、完了を待たずにすぐに続行します。フックは同期フックと同じ JSON 入力を stdin 経由で受け取ります。

バックグラウンド プロセスが終了した後、フックが **systemMessage** または **additionalContext** フィールドを含む JSON レスポンスを生成した場合、そのコンテンツは次の会話ターンで Claude にコンテキストとして配信されます。

例: ファイル変更後にテストを実行

このフックは Claude がファイルを書き込むたびにバックグラウンドでテストスイートを開始し、テストが完了したときに結果を Claude に報告します。このスクリプトをプロジェクトの **.claude/hooks/run-tests-async.sh** に保存し、**chmod +x** で実行可能にします。

```
#!/bin/bash

## run-tests-async.sh

## stdin からフック入力を読み取る
INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

## ソース ファイルのみテストを実行
if [[ "$FILE_PATH" != *.ts && "$FILE_PATH" != *.js ]]; then
    exit 0
fi

## テストを実行し、systemMessage 経由で結果を報告
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
```

次に、プロジェクトルートの `.claude/settings.json` にこの設定を追加します。 `async: true` フラグにより、Claude はテストの実行中に作業を続行できます。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/run-tests-async.sh",
            "async": true,
            "timeout": 300
          }
        ]
      }
    ]
  }
}
```

制限事項

非同期フックは同期フックと比べていくつかの制約があります。

- `async` をサポートするのは `type: "command"` フックのみです。プロンプト ベースのフックは非同期で実行できません。
- 非同期フックはツール呼び出しをブロックまたは決定を返すことができません。フックが完了するまでに、トリガーするアクションはすでに進行しています。
- フック出力は次の会話ターンで配信されます。セッションがアイドル状態の場合、レスポンスは次のユーザー操作まで待機します。
- 各実行は個別のバックグラウンド プロセスを作成します。同じ非同期フックの複数の発火間での重複排除はありません。

セキュリティに関する考慮事項

免責事項

コマンド フックはシステム ユーザーの完全な権限で実行されます。

Warning:

コマンド フックはユーザー アカウントの完全な権限でシェル コマンドを実行します。ユーザー アカウントがアクセスできるファイルを変更、削除、またはアクセスできます。フック コマンドを設定に追加する前に、すべてのフック コマンドを確認してテストしてください。

セキュリティ ベスト プラクティス

フックを書くときは、これらのプラクティスに留意してください。

- **入力を検証およびサニタイズ:** 入力データを盲目的に信頼しないでください
- **常にシェル変数を引用:** `$VAR` ではなく `"$VAR"` を使用
- **パス トラバーサルをブロック:** ファイル パスで `..` をチェック
- **絶対パスを使用:** スクリプトの完全なパスを指定し、プロジェクト ルートに `"$CLAUDE_PROJECT_DIR"` を使用
- **機密ファイルをスキップ:** `.env`、`.git/`、キーなどを避ける

フックをデバッグ

`claude --debug` を実行して、フック実行の詳細を確認します。マッチしたフック、終了コード、出力を含みます。 `Ctrl+O` で詳細モードを切り替えて、トランスクリプトでフック進行状況を確認します。

```
[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>
```

フックが発火しない、無限 Stop フック ループ、設定エラーなどの一般的な問題のトラブルシューティングについては、ガイドの[制限事項とトラブルシューティング](#)を参照してください。

hooks でワークフローを自動化する

Claude Code がファイルを編集したり、タスクを完了したり、入力が必要な場合に、シェルコマンドを自動的に実行します。コードをフォーマットし、通知を送信し、コマンドを検証し、プロジェクトルールを適用します。

Hooks は Claude Code のライフサイクルの特定のポイントで実行されるユーザー定義のシェルコマンドです。これらは Claude Code の動作に対して決定論的な制御を提供し、LLM が実行を選択するのに依存するのではなく、特定のアクションが常に発生することを保証します。Hooks を使用して、プロジェクトルールを適用し、反復的なタスクを自動化し、Claude Code を既存のツールと統合します。

判断が必要な決定については、決定論的なルールではなく、Claude モデルを使用して条件を評価する [プロンプトベースの hooks](#) または [エージェントベースの hooks](#) を使用することもできます。

Claude Code を拡張する他の方法については、Claude に追加の指示と実行可能なコマンドを与えるための [skills](#)、分離されたコンテキストでタスクを実行するための [subagents](#)、プロジェクト全体で共有する拡張機能をパッケージ化するための [plugins](#) を参照してください。

Tip:

このガイドでは一般的なユースケースと始め方をカバーしています。完全なイベントスキーマ、JSON 入力/出力形式、非同期 hooks や MCP ツール hooks などの高度な機能については、[Hooks リファレンス](#) を参照してください。

最初の hook をセットアップする

Hook を作成する最速の方法は、Claude Code の `/hooks` インタラクティブメニューを使用することです。このチュートリアルでは、デスクトップ通知 hook を作成するため、Claude がターミナルを監視する代わりに入力を待っているときにアラートを受け取ります。

Step 1: hooks メニューを開く

Claude Code CLI で `/hooks` と入力します。利用可能なすべての hook イベントのリストと、すべての hooks を無効にするオプションが表示されます。各イベントは Claude のライフサイクルのカスタムコードを実行できるポイントに対応しています。 `Notification` を選択して、Claude があなたの注意が必要な場合に発火する hook を作成します。

Step 2: マッチャーを設定する

メニューはマッチャーのリストを表示します。マッチャーは hook が発火するタイミングをフィルタリングします。マッチャーを `*` に設定して、すべての通知タイプで発火するようにします。後で `permission_prompt` や `idle_prompt` などの特定の値にマッチャーを変更して、絞り込むことができます。

Step 3: コマンドを追加する

`+ Add new hook...` を選択します。メニューはイベントが発火したときに実行するシェルコマンドを求めます。Hooks は提供するすべてのシェルコマンドを実行するため、プラットフォームの組み込み通知ツールを使用できます。OS のコマンドをコピーします：

macOS

AppleScript を通じてネイティブ macOS 通知をトリガーするために `osascript` を使用します：

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

Linux

ほとんどの Linux デスクトップに通知デーモンがプリインストールされている `notify-send` を使用します：

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

Windows (PowerShell)

PowerShell を使用して .NET の Windows Forms を通じてネイティブメッセージボックスを表示します：

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

Step 4: ストレージの場所を選択する

メニューは hook 設定を保存する場所を尋ねます。 `User settings` を選択して `~/.claude/settings.json` に保存します。これはすべてのプロジェクトに hook を適用します。現在のプロジェクトにスコープするために `Project settings` を選択することもできます。利用可能なすべてのスコープについては、[Configure hook location](#) を参照してください。

Step 5: hook をテストする

`Esc` を押して CLI に戻ります。Claude に許可が必要な何かを実行するよう依頼し、ターミナルから切り替えます。デスクトップ通知を受け取るはずでです。

自動化できるもの

Hooks を使用すると、Claude Code のライフサイクルの重要なポイントでコードを実行できます：編集後にファイルをフォーマットし、実行前にコマンドをブロックし、Claude が入力が必要とするときに通知を送信し、セッション開始時にコンテキストを注入するなど。Hook イベントの完全なリストについては、[Hooks リファレンス](#) を参照してください。

各例には、[設定ファイル](#) に追加する準備ができた設定ブロックが含まれています。最も一般的なパターン：

- [Claude が入力が必要とするときに通知を受け取る](#)
- [編集後にコードを自動フォーマットする](#)
- [保護されたファイルへの編集をブロックする](#)
- [圧縮後にコンテキストを再注入する](#)
- [設定変更を監査する](#)

Claude が入力が必要とするときに通知を受け取る

Claude が作業を完了して入力が必要とするときはいつでもデスクトップ通知を受け取り、ターミナルをチェックせずに他のタスクに切り替えることができます。

この hook は `Notification` イベントを使用します。これは Claude が入力または許可を待っているときに発火します。以下の各タブはプラットフォームのネイティブ通知コマンドを使用します。これを `~/.claude/settings.json` に追加するか、上記の [インタラクティブチュートリアル](#) を使用して `/hooks` で設定します：

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')\"";
          }
        ]
      }
    ]
  }
}
```

編集後にコードを自動フォーマットする

Claude が編集するすべてのファイルで [Prettier](#) を自動的に実行し、手動操作なしでフォーマットを一貫性のあるものにします。

この hook は `PostToolUse` イベントを `Edit|Write` マッチャーで使用するため、ファイル編集ツール後にのみ実行されます。コマンドは `jq` で編集されたファイルパスを抽出し、Prettier に渡します。これをプロジェクトルートの `.claude/settings.json` に追加します：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --
write"
          }
        ]
      }
    ]
  }
}
```

Note:

このページの Bash の例は JSON 解析に `jq` を使用しています。 `brew install jq` (macOS) 、 `apt-get install jq` (Debian/Ubuntu) でインストールするか、 [jq ダウンロード](#) を参照してください。

保護されたファイルへの編集をブロックする

Claude が `.env` 、 `package-lock.json` 、 `.git/` 内のものなどの機密ファイルを変更するのを防ぎます。Claude は編集がブロックされた理由を説明するフィードバックを受け取るため、アプローチを調整できます。

この例は hook が呼び出す別のスクリプトファイルを使用します。スクリプトはターゲットファイルパスを保護されたパターンのリストに対してチェックし、終了コード 2 で編集をブロックします。

Step 1: hook スクリプトを作成する

これを `.claude/hooks/protect-files.sh` に保存します：

```
#!/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: スクリプトを実行可能にする (macOS/Linux)

Hook スクリプトは Claude Code が実行するために実行可能である必要があります：

```
chmod +x .claude/hooks/protect-files.sh
```

Step 3: hook を登録する

`PreToolUse` hook を `.claude/settings.json` に追加して、`Edit` または `Write` ツール呼び出しの前にスクリプトを実行します：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/protect-files.sh"
          }
        ]
      }
    ]
  }
}
```

圧縮後にコンテキストを再注入する

Claude のコンテキストウィンドウがいっぱいになると、圧縮は会話を要約してスペースを解放します。これは重要な詳細を失う可能性があります。 `SessionStart` hook を `compact` マッチャーで使用して、すべての圧縮後に重要なコンテキストを再注入します。

コマンドが stdout に書き込むテキストは Claude のコンテキストに追加されます。この例は Claude にプロジェクト規約と最近の作業を思い出させます。これをプロジェクトルートの `.claude/settings.json` に追加します：


```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Reminder: use Bun, not npm. Run bun test before committing. Current sprint: auth refactor.'"
          }
        ]
      }
    ]
  }
}
```

`echo` を `git log --oneline -5` などの動的出力を生成するコマンドに置き換えることができます。セッション開始時にコンテキストを注入する場合は、代わりに [CLAUDE.md](#) を使用することを検討してください。環境変数については、リファレンスの [CLAUDE_ENV_FILE](#) を参照してください。

設定変更を監査する

セッション中に設定またはスキルファイルが変更されるときを追跡します。

`ConfigChange` イベントは外部プロセスまたはエディタが設定ファイルを変更するときに発火するため、コンプライアンスのために変更をログに記録したり、不正な変更をブロックしたりできます。

この例は各変更を監査ログに追加します。これを `~/.claude/settings.json` に追加します：

```
{
  "hooks": {
    "ConfigChange": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "jq -c '{timestamp: now | todate, source: .source,
file: .file_path}' >> ~/claude-config-audit.log"
          }
        ]
      }
    ]
  }
}
```

マッチャーは設定タイプでフィルタリングします: `user_settings`、`project_settings`、`local_settings`、`policy_settings`、または `skills`。変更が有効になるのをブロックするには、終了コード 2 で終了するか、`{"decision": "block"}` を返します。完全な入力スキーマについては、[ConfigChange リファレンス](#) を参照してください。

hooks の仕組み

Hook イベントは Claude Code のライフサイクルの特定のポイントで発火します。イベントが発火すると、すべてのマッチングする hooks が並列で実行され、同一の hook コマンドは自動的に重複排除されます。以下の表は各イベントとそれがトリガーされることを示しています：

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

Event	When it fires
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
PostCompact	After context compaction completes
Elicitation	When an MCP server requests user input during a tool call
ElicitationResult	After a user responds to an MCP elicitation, before the response is sent back to the server
SessionEnd	When a session terminates

各 hook には、実行方法を決定する `type` があります。ほとんどの hooks は `"type": "command"` を使用し、シェルコマンドを実行します。他に3つのタイプが利用可能です：

- `"type": "http"` : イベントデータを URL に POST します。[HTTP hooks](#) を参照してください。

- `"type": "prompt"`: シングルターン LLM 評価。 [プロンプトベースの hooks](#) を参照してください。
- `"type": "agent"`: ツールアクセス付きマルチターン検証。 [エージェントベースの hooks](#) を参照してください。

入力を読み取り、出力を返す

Hooks は stdin、stdout、stderr、および終了コードを通じて Claude Code と通信します。イベントが発火すると、Claude Code はイベント固有のデータを JSON としてスクリプトの stdin に渡します。スクリプトはそのデータを読み取り、作業を行い、終了コードを通じて Claude Code に次に何をするかを伝えます。

Hook 入力

すべてのイベントには `session_id` と `cwd` などの共通フィールドが含まれていますが、各イベントタイプは異なるデータを追加します。たとえば、Claude が Bash コマンドを実行するとき、`PreToolUse` hook は stdin で次のようなものを受け取ります：

```
{
  "session_id": "abc123",           // このセッションの一意の ID
  "cwd": "/Users/sarah/myproject", // イベントが発火したときの作業ディレクトリ
  "hook_event_name": "PreToolUse", // この hook をトリガーしたイベント
  "tool_name": "Bash",             // Claude が使用しようとしているツール
  "tool_input": {                  // Claude がツールに渡した引数
    "command": "npm test"         // Bash の場合、これはシェルコマンド
  }
}
```

スクリプトはその JSON を解析し、これらのフィールドのいずれかに基づいて動作できます。`UserPromptSubmit` hooks は `prompt` テキストを取得し、`SessionStart` hooks は `source` (startup、resume、clear、compact) を取得するなど。リファレンスの [共通入力フィールド](#) で共有フィールドを参照し、各イベントのセクションでイベント固有のスキーマを参照してください。

Hook 出力

スクリプトは stdout または stderr に書き込み、特定のコードで終了することで、Claude Code に次に何をするかを伝えます。たとえば、コマンドをブロックしたい `PreToolUse` hook：

```
#!/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 は Claude のフィード
    バックになります
    exit 2 # exit 2 = アクションをブロック
fi

exit 0 # exit 0 = 続行を許可
```

終了コードは次に何が起きるかを決定します：

- ・ **終了 0**：アクションが続行されます。 `UserPromptSubmit` および `SessionStart` hooks の場合、stdout に書き込むすべてのものが Claude のコンテキストに追加されます。
- ・ **終了 2**：アクションがブロックされます。stderr に理由を書き込み、Claude はそれをフィードバックとして受け取り、調整できます。
- ・ **その他の終了コード**：アクションが続行されます。Stderr はログに記録されますが、Claude には表示されません。 `Ctrl+O` で詳細モードを切り替えて、トランスクリプトでこれらのメッセージを確認してください。

構造化 JSON 出力

終了コードは 2 つのオプションを提供します：許可またはブロック。より多くの制御のために、終了 0 して stdout に JSON オブジェクトを出力します。

Note:

終了 2 で stderr メッセージでブロックするか、終了 0 で JSON で構造化制御を使用します。混在させないでください：Claude Code は終了 2 のときに JSON を無視します。

たとえば、`PreToolUse` hook はツール呼び出しを拒否して理由を Claude に伝えるか、ユーザーの承認のためにエスカレートできます：

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Use rg instead of grep for better performance"
  }
}
```

Claude Code は `permissionDecision` を読み取り、ツール呼び出しをキャンセルし、`permissionDecisionReason` を Claude にフィードバックとして返します。これら 3 つのオプションは `PreToolUse` に固有です：

- `"allow"`：許可プロンプトを表示せずに続行
- `"deny"`：ツール呼び出しをキャンセルし、理由を Claude に送信
- `"ask"`：通常どおりユーザーに許可プロンプトを表示

他のイベントは異なる決定パターンを使用します。たとえば、`PostToolUse` および `Stop` hooks はトップレベルの `decision: "block"` フィールドを使用し、`PermissionRequest` は `hookSpecificOutput.decision.behavior` を使用します。リファレンスの [サマリーテーブル](#) でイベント別の完全な内訳を参照してください。

`UserPromptSubmit` hooks の場合、代わりに `additionalContext` を使用して Claude のコンテキストにテキストを注入します。プロンプトベースの hooks (`type: "prompt"`) は出力を異なる方法で処理します：[プロンプトベースの hooks](#) を参照してください。

マッチャーで hooks をフィルタリングする

マッチャーがない場合、hook はそのイベントのすべての発生で発火します。マッチャーを使用すると、それを絞り込むことができます。たとえば、すべてのツール呼び出しの後ではなく、ファイル編集の後にのみフォーマッターを実行したい場合は、`PostToolUse` hook にマッチャーを追加します：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "prettier --write ..." }
        ]
      }
    ]
  }
}
```

"Edit|Write" マッチャーはツール名にマッチする正規表現パターンです。Hook は Claude が Edit または Write ツールを使用するときのみ発火し、 Bash 、 Read 、または他のツールを使用するときには発火しません。

各イベントタイプは特定のフィールドでマッチします。マッチャーは正確な文字列と正規表現パターンをサポートしています：

イベント	マッチャーがフィルタリングするもの	マッチャー値の例
PreToolUse 、 PostToolUse 、 PostToolUseFailure 、 PermissionRequest	ツール名	Bash 、 Edit Write 、 mcp_.*
SessionStart	セッションの開始方法	startup 、 resume 、 clear 、 compact
SessionEnd	セッションが終了した理由	clear 、 logout 、 prompt_input_exit 、 bypass_permissions_disabled 、 other
Notification	通知タイプ	permission_prompt 、 idle_prompt 、 auth_success 、 elicitation_dialog

イベント	マッチャーがフィルタリングするもの	マッチャー値の例
<code>SubagentStart</code>	エージェントタイプ	<code>Bash</code> 、 <code>Explore</code> 、 <code>Plan</code> 、またはカスタムエージェント名
<code>PreCompact</code>	圧縮をトリガーしたもの	<code>manual</code> 、 <code>auto</code>
<code>SubagentStop</code>	エージェントタイプ	<code>SubagentStart</code> と同じ値
<code>ConfigChange</code>	設定ソース	<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>	マッチャーサポートなし	すべての発生で常に発火

異なるイベントタイプのマッチャーを示すいくつかの例：

すべての Bash コマンドをログに記録する

`Bash` ツール呼び出しのみをマッチさせ、各コマンドをファイルにログに記録します。
`PostToolUse` イベントはコマンドが完了した後に発火するため、`tool_input.command` は実行されたものを含みます。Hook は stdin で JSON としてイベントデータを受け取り、`jq -r '.tool_input.command'` はコマンド文字列のみを抽出し、`>>` はログファイルに追加します：


```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.command' >> ~/.claude/command-log.txt"
          }
        ]
      }
    ]
  }
}
```

MCP ツールをマッチさせる

MCP ツールは組み込みツールとは異なる命名規則を使用します：

`mcp__<server>__<tool>`。ここで `<server>` は MCP サーバー名で、`<tool>` はそれが提供するツールです。たとえば、`mcp__github__search_repositories` または `mcp__filesystem__read_file`。特定のサーバーからすべてのツールをターゲットするために正規表現マッチャーを使用するか、`mcp__.*__write.*` のようなパターンでサーバー全体でマッチさせます。完全な例のリストについては、リファレンスの [MCP ツールをマッチさせる](#) を参照してください。

以下のコマンドは hook の JSON 入力から `jq` でツール名を抽出し、詳細モード（`Ctrl+O`）で表示される `stderr` に書き込みます：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp_github_.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"GitHub tool called: $(jq -r '.tool_name')\" >&2"
          }
        ]
      }
    ]
  }
}
```

セッション終了時にクリーンアップする

`SessionEnd` イベントはセッションが終了した理由のマッチャーをサポートしています。この hook は `clear` (`/clear` を実行するとき) でのみ発火し、通常の終了では発火しません：

```
{
  "hooks": {
    "SessionEnd": [
      {
        "matcher": "clear",
        "hooks": [
          {
            "type": "command",
            "command": "rm -f /tmp/claude-scratch-*.txt"
          }
        ]
      }
    ]
  }
}
```

完全なマッチャー構文については、[Hooks リファレンス](#)を参照してください。

hook の場所を設定する

Hook を追加する場所はそのスコープを決定します：

場所	スコープ	共有可能
<code>~/.claude/ settings.json</code>	すべてのプロジェクト	いいえ、マシンにローカル
<code>.claude/ settings.json</code>	単一プロジェクト	はい、リポジトリにコミット可能
<code>.claude/ settings.local.json</code>	単一プロジェクト	いいえ、gitignored
管理ポリシー設定	組織全体	はい、管理者制御
<code>Plugin hooks/ hooks.json</code>	プラグインが有効な場合	はい、プラグインにバンドル
<code>Skill</code> または <code>agent frontmatter</code>	スキルまたはエージェントがアクティブな場合	はい、コンポーネントファイルで定義

Claude Code の `/hooks` [メニュー](#) を使用して、hooks をインタラクティブに追加、削除、表示することもできます。すべての hooks を一度に無効にするには、`/hooks` メニューの下部にあるトグルを使用するか、設定ファイルで `"disableAllHooks": true` を設定します。

`/hooks` メニューを通じて追加された Hooks は直ちに有効になります。Claude Code が実行中に設定ファイルを直接編集する場合、変更は `/hooks` メニューで確認するか、セッションを再開するまで有効になりません。

プロンプトベースの hooks

決定論的なルールではなく判断が必要な決定については、`type: "prompt"` hooks を使用します。シェルコマンドを実行する代わりに、Claude Code はプロンプトと hook の入力データを Claude モデル（デフォルトでは Haiku）に送信して決定を下します。より多くの機能が必要な場合は、`model` フィールドで異なるモデルを指定できます。

モデルの唯一の仕事は、yes/no 決定を JSON として返すことです：

- `"ok": true` : アクションが続行されます

- `"ok": false` : アクションがブロックされます。モデルの `"reason"` は Claude にフィードバックとして返されるため、調整できます。

この例は `Stop` hook を使用して、要求されたすべてのタスクが完了しているかどうかをモデルに尋ねます。モデルが `"ok": false` を返す場合、Claude は作業を続け、`reason` を次の指示として使用します：

```
{
  "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\"}."
          }
        ]
      }
    ]
  }
}
```

完全な設定オプションについては、リファレンスの [プロンプトベースの hooks](#) を参照してください。

エージェントベースの hooks

検証がファイルの検査またはコマンドの実行を必要とする場合は、`type: "agent"` hooks を使用します。プロンプト hooks は単一の LLM 呼び出しを行います。エージェント hooks は、条件を返す前にファイルを読み取り、コードを検索し、他のツールを使用できるサブエージェントを生成します。

エージェント hooks はプロンプト hooks と同じ `"ok"` / `"reason"` 応答形式を使用しますが、デフォルトのタイムアウトが 60 秒で、最大 50 ツール使用ターンです。

この例は Claude が停止することを許可する前にテストが合格することを検証します：

```

{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

Hook 入力データだけで決定を下すのに十分な場合は、プロンプト hooks を使用します。コードベースの実際の状態に対して何かを検証する必要がある場合は、エージェント hooks を使用します。

完全な設定オプションについては、リファレンスの [エージェントベースの hooks](#) を参照してください。

HTTP hooks

`type: "http"` hooks を使用して、シェルコマンドを実行する代わりに、イベントデータを HTTP エンドポイントに POST します。エンドポイントはコマンド hook が stdin で受け取るのと同じ JSON を受け取り、HTTP レスポンスボディを使用して同じ JSON 形式で結果を返します。

HTTP hooks は、Web サーバー、クラウド関数、または外部サービスに hook ロジックを処理させたい場合に便利です。たとえば、チーム全体のツール使用イベントをログに記録する共有監査サービスです。

この例はすべてのツール使用をローカルログサービスに POST します：

```

{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/tool-use",
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}

```

エンドポイントは、コマンド hooks と同じ [出力形式](#) を使用して JSON レスポンスボディを返す必要があります。ツール呼び出しをブロックするには、適切な `hookSpecificOutput` フィールドで 2xx レスポンスを返します。HTTP ステータスコードだけではアクションをブロックできません。

ヘッダー値は `$VAR_NAME` または `${VAR_NAME}` 構文を使用した環境変数補間をサポートしています。 `allowedEnvVars` 配列にリストされている変数のみが解決されます。他のすべての `$VAR` 参照は空のままです。

Note:

HTTP hooks は設定 JSON を直接編集することで設定する必要があります。 `/hooks` インタラクティブメニューはコマンド hooks の追加のみをサポートしています。

完全な設定オプションとレスポンス処理については、リファレンスの [HTTP hooks](#) を参照してください。

制限事項とトラブルシューティング

制限事項

- コマンド hooks は stdout、stderr、および終了コードを通じてのみ通信します。コマンドまたはツール呼び出しを直接トリガーすることはできません。HTTP hooks はレスポンスボディを通じて通信します。
- Hook タイムアウトはデフォルトで 10 分で、hook ごとに `timeout` フィールド（秒単位）で設定可能です。
- `PostToolUse` hooks はツールが既に実行されているため、アクションを元に戻すことはできません。
- `PermissionRequest` hooks は **非インタラクティブモード**（`-p`）では発火しません。自動化された許可決定には `PreToolUse` hooks を使用します。
- `Stop` hooks はタスク完了時だけでなく、Claude が応答を完了するたびに発火します。ユーザーの割り込みでは発火しません。

Hook が発火しない

Hook は設定されていますが、実行されません。

- `/hooks` を実行し、hook が正しいイベントの下に表示されることを確認します
- マッチャーパターンがツール名と正確にマッチすることを確認します（マッチャーは大文字と小文字を区別します）
- 正しいイベントタイプをトリガーしていることを確認します（例： `PreToolUse` はツール実行前に発火し、 `PostToolUse` は後に発火します）
- 非インタラクティブモード（`-p`）で `PermissionRequest` hooks を使用している場合は、代わりに `PreToolUse` に切り替えます

Hook エラーが出力に表示される

トランスクリプトに「PreToolUse hook error: ...」というメッセージが表示されます。

- スクリプトが予期せずゼロ以外のコードで終了しました。サンプル JSON をパイプして手動でテストします：

```
echo '{"tool_name":"Bash","tool_input":{"command":"ls"}}' | ./my-hook.sh
echo $? # 終了コードを確認
```

- 「command not found」が表示される場合は、絶対パスを使用するか、スクリプトを参照するために `$CLAUDE_PROJECT_DIR` を使用します

- ・「jq: command not found」が表示される場合は、jq をインストールするか、JSON 解析に Python/Node.js を使用します
- ・スクリプトがまったく実行されていない場合は、実行可能にします：`chmod +x ./my-hook.sh`

`/hooks` に設定された hooks が表示されない

設定ファイルを編集しましたが、hooks がメニューに表示されません。

- ・セッションを再開するか、`/hooks` を開いてリロードします。`/hooks` メニューを通じて追加された Hooks は直ちに有効になりますが、手動ファイル編集はリロードが必要です。
- ・JSON が有効であることを確認します（末尾のコンマとコメントは許可されていません）
- ・設定ファイルが正しい場所にあることを確認します：プロジェクト hooks の場合は `.claude/settings.json`、グローバル hooks の場合は `~/.claude/settings.json`

Stop hook が永遠に実行される

Claude は無限ループで作業を続けます。

Stop hook スクリプトは、既にトリガーされたかどうかを確認する必要があります。

JSON 入力から `stop_hook_active` フィールドを解析し、`true` の場合は早期に終了します：

```
#!/bin/bash
INPUT=$(cat)
if [ "$(echo "$INPUT" | jq -r '.stop_hook_active')" = "true" ]; then
    exit 0 # Claude が停止することを許可
fi
## ... hook ロジックの残り
```

JSON 検証に失敗した

Hook スクリプトが有効な JSON を出力しているにもかかわらず、Claude Code は JSON 解析エラーを表示します。

Claude Code が hook を実行するとき、プロファイル（`~/.zshrc` または `~/.bashrc`）をソースするシェルを生成します。プロファイルに無条件の `echo` ステートメントが含まれている場合、その出力は hook の JSON の前に付加されます：


```
Shell ready on arm64
{"decision": "block", "reason": "Not allowed"}
```

Claude Code はこれを JSON として解析しようとして失敗します。これを修正するには、シェルスクリプトの echo ステートメントをラップして、インタラクティブシェルでのみ実行するようにします：

```
## ~/.zshrc または ~/.bashrc 内
if [[ $- == *i* ]]; then
  echo "Shell ready"
fi
```

`$-` 変数はシェルフラグを含み、`i` はインタラクティブを意味します。Hooks は非インタラクティブシェルで実行されるため、echo はスキップされます。

デバッグ技術

`Ctrl+O` で詳細モードを切り替えてトランスクリプトで hook 出力を確認するか、`claude --debug` を実行して、どの hooks がマッチしたか、その終了コードなど、完全な実行詳細を確認します。

さらに詳しく

- [Hooks リファレンス](#)：完全なイベントスキーマ、JSON 出力形式、非同期 hooks、および MCP ツール hooks
- [セキュリティに関する考慮事項](#)：共有環境または本番環境に hooks をデプロイする前に確認してください
- [Bash コマンドバリデータの例](#)：完全なリファレンス実装

MCP を使用して Claude Code をツールに接続する

Model Context Protocol を使用して Claude Code をツールに接続する方法を学びます。

Claude Code は、AI ツール統合のためのオープンソース標準である [Model Context Protocol \(MCP\)](#) を通じて、数百の外部ツールとデータソースに接続できます。MCP サーバーは Claude Code にツール、データベース、API へのアクセスを提供します。

MCP でできること

MCP サーバーが接続されている場合、Claude Code に以下のことを依頼できます：

- **課題追跡ツールから機能を実装する**：「JIRA の課題 ENG-4521 に記載されている機能を追加し、GitHub に PR を作成してください。」
- **監視データを分析する**：「Sentry と Statsig をチェックして、ENG-4521 に記載されている機能の使用状況を確認してください。」
- **データベースをクエリする**：「PostgreSQL データベースに基づいて、ENG-4521 機能を使用した 10 人のランダムなユーザーのメールアドレスを検索してください。」
- **デザインを統合する**：「Slack に投稿された新しい Figma デザインに基づいて、標準メールテンプレートを更新してください。」
- **ワークフローを自動化する**：「新機能に関するフィードバックセッションに招待する 10 人のユーザーに Gmail ドラフトを作成してください。」

人気のある MCP サーバー

Claude Code に接続できる一般的に使用される MCP サーバーをいくつか紹介します：

Warning:

サードパーティの MCP サーバーは自己責任で使用してください。Anthropic はこれらすべてのサーバーの正確性またはセキュリティを検証していません。インストールする MCP サーバーを信頼していることを確認してください。信頼できないコンテンツを取得する可能性のある MCP サーバーを使用する場合は特に注意してください。これらはプロンプトインジェクションのリスクにさらされる可能性があります。

Server	Description	Command
Notion	Connect your Notion workspace to search, update, and power workflows across tools	<pre>claude mcp add -- transport http notion https:// mcp.notion.com/mcp</pre>
Canva	Search, create, autofill, and export Canva designs	<pre>claude mcp add -- transport http canva https:// mcp.canva.com/mcp</pre>
Figma	Generate diagrams and better code from Figma context	<pre>claude mcp add -- transport http figma- remote-mcp https:// mcp.figma.com/mcp</pre>
Atlassian	Access Jira & Confluence from Claude	<pre>claude mcp add -- transport http atlassian https:// mcp.atlassian.com/v1/ mcp</pre>
Linear	Manage issues, projects & team workflows in Linear	<pre>claude mcp add -- transport http linear https:// mcp.linear.app/mcp</pre>
monday.com	Manage projects, boards, and workflows in monday.com	<pre>claude mcp add -- transport http monday https:// mcp.monday.com/mcp</pre>
Intercom	Access to Intercom data for better customer insights	<pre>claude mcp add -- transport http intercom https:// mcp.intercom.com/mcp</pre>
Vercel	Analyze, debug, and manage projects and deployments	<pre>claude mcp add -- transport http vercel https:// mcp.vercel.com</pre>

Server	Description	Command
Granola	The AI notepad for meetings	<pre>claude mcp add -- transport http granola https:// mcp.granola.ai/mcp</pre>
Asana	Connect to Asana to coordinate tasks, projects, and goals	<pre>claude mcp add -- transport streamable- http asana https:// mcp.asana.com/v2/mcp</pre>
Miro	Access and create new content on Miro boards	<pre>claude mcp add -- transport http miro https:// mcp.miro.com/</pre>
Sentry	Search, query, and debug errors intelligently	<pre>claude mcp add -- transport http sentry https:// mcp.sentry.dev/mcp</pre>
Supabase	Manage databases, authentication, and storage	<pre>claude mcp add -- transport http supabase https:// mcp.supabase.com/mcp</pre>
Hugging Face	Access the Hugging Face Hub and thousands of Gradio Apps	<pre>claude mcp add -- transport http hugging-face https:// huggingface.co/mcp</pre>
Context7	Up-to-date docs for LLMs and AI code editors	<pre>claude mcp add -- transport http context7 https:// mcp.context7.com/mcp</pre>
Stripe	Payment processing and financial infrastructure tools	<pre>claude mcp add -- transport http stripe https:// mcp.stripe.com</pre>

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:

特定の統合が必要ですか？ [GitHub](#) で数百以上の MCP サーバーを検索するか、[MCP SDK](#) を使用して独自のサーバーを構築してください。

MCP サーバーのインストール

MCP サーバーは、ニーズに応じて 3 つの異なる方法で設定できます：

オプション 1：リモート HTTP サーバーを追加する

HTTP サーバーはリモート MCP サーバーに接続するための推奨オプションです。これはクラウドベースのサービスに最も広くサポートされているトランスポートです。

基本的な構文

```
claude mcp add --transport http <name> <url>
```

実際の例：Notion に接続する

```
claude mcp add --transport http notion https://mcp.notion.com/mcp
```

Bearer トークンを使用した例

```
claude mcp add --transport http secure-api https://api.example.com/mcp \
  --header "Authorization: Bearer your-token"
```

オプション 2：リモート SSE サーバーを追加する

Warning:

SSE (Server-Sent Events) トランスポートは非推奨です。利用可能な場合は HTTP サーバーを使用してください。

基本的な構文

```
claude mcp add --transport sse <name> <url>
```

実際の例: Asana に接続する

```
claude mcp add --transport sse asana https://mcp.asana.com/sse
```

認証ヘッダーを使用した例

```
claude mcp add --transport sse private-api https://api.company.com/sse \  
  --header "X-API-Key: your-key-here"
```

オプション 3: ローカル stdio サーバーを追加する

Stdio サーバーはマシン上でローカルプロセスとして実行されます。システムへの直接アクセスやカスタムスクリプトが必要なツールに最適です。

基本的な構文

```
claude mcp add [options] <name> -- <command> [args...]
```

実際の例: Airtable サーバーを追加する

```
claude mcp add --transport stdio --env AIRTABLE_API_KEY=YOUR_KEY airtable \  
  -- npx -y airtable-mcp-server
```

Note:

重要: オプションの順序

すべてのオプション (`--transport` 、 `--env` 、 `--scope` 、 `--header`) はサーバー名の前に来る必要があります。 `--` (ダブルダッシュ) はサーバー名を MCP サーバーに渡されるコマンドと引数から分離します。

例:

- `claude mcp add --transport stdio myserver -- npx server` → `npx server` を実行します
- `claude mcp add --transport stdio --env KEY=value myserver -- python server.py --port 8080` → 環境に `KEY=value` を設定して `python server.py --port 8080` を実行します

これにより、Claude のフラグとサーバーのフラグの間の競合を防ぎます。

サーバーの管理

設定後、これらのコマンドで MCP サーバーを管理できます：

```
## すべての設定済みサーバーをリストする
claude mcp list

## 特定のサーバーの詳細を取得する
claude mcp get github

## サーバーを削除する
claude mcp remove github

## (Claude Code 内) サーバーのステータスを確認する
/mcp
```

動的ツール更新

Claude Code は MCP `list_changed` 通知をサポートしており、MCP サーバーが切断して再接続することなく、利用可能なツール、プロンプト、リソースを動的に更新できます。MCP サーバーが `list_changed` 通知を送信すると、Claude Code はそのサーバーから利用可能な機能を自動的に更新します。

Tip:

ヒント：

- `--scope` フラグを使用して、設定が保存される場所を指定します：
- `local` (デフォルト)：現在のプロジェクトでのみ利用可能 (古いバージョンでは `project` と呼ばれていました)
- `project`：`.mcp.json` ファイルを通じてプロジェクト内のすべてのユーザーと共有
- `user`：すべてのプロジェクト全体で利用可能 (古いバージョンでは `global` と呼ばれていました)
- `--env` フラグで環境変数を設定します (例： `--env KEY=value`)
- `MCP_TIMEOUT` 環境変数を使用して MCP サーバーのスタートアップタイムアウトを設定します (例： `MCP_TIMEOUT=10000 claude` は 10 秒のタイムアウトを設定します)
- Claude Code は MCP ツール出力が 10,000 トークンを超えると警告を表示します。この制限を増やすには、`MAX_MCP_OUTPUT_TOKENS` 環境変数を設定します (例：
`MAX_MCP_OUTPUT_TOKENS=50000`)
- `/mcp` を使用して、OAuth 2.0 認証が必要なリモートサーバーで認証します

Warning:

Windows ユーザー向け：ネイティブ Windows (WSL ではない) では、`npx` を使用するローカル MCP サーバーは適切な実行を確保するために `cmd /c` ラッパーが必要です。

```
## これにより、Windows が実行できる command="cmd" が作成されます
claude mcp add --transport stdio my-server -- cmd /c npx -y @some/package
```

`cmd /c` ラッパーがないと、Windows は `npx` を直接実行できないため「Connection closed」エラーが発生します。（`--` パラメータの説明については上記のメモを参照してください。）

プラグイン提供の MCP サーバー

[プラグイン](#) は MCP サーバーをバンドルでき、プラグインが有効になると自動的にツールと統合を提供します。プラグイン MCP サーバーはユーザーが設定したサーバーと同じように機能します。

プラグイン MCP サーバーの仕組み：

- プラグインはプラグインルートの `.mcp.json` または `plugin.json` 内でインラインで MCP サーバーを定義します
- プラグインが有効になると、その MCP サーバーが自動的に起動します
- プラグイン MCP ツールは手動で設定された MCP ツールと一緒に表示されます
- プラグインサーバーはプラグインのインストールを通じて管理されます（`/mcp` コマンドではありません）

プラグイン MCP 設定の例：

プラグインルートの `.mcp.json` 内：

```
{
  "database-tools": {
    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
    "env": {
      "DB_URL": "${DB_URL}"
    }
  }
}
```

または `plugin.json` 内でインライン：

```
{
  "name": "my-plugin",
  "mcpServers": {
    "plugin-api": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/api-server",
      "args": ["--port", "8080"]
    }
  }
}
```

プラグイン MCP 機能：

- ・ **自動ライフサイクル**：プラグインが有効になるとサーバーが起動しますが、MCP サーバーの変更（有効化または無効化）を適用するには Claude Code を再起動する必要があります
- ・ **環境変数**：プラグイン相対パスに `${CLAUDE_PLUGIN_ROOT}` を使用します
- ・ **ユーザー環境アクセス**：手動で設定されたサーバーと同じ環境変数へのアクセス
- ・ **複数のトランスポートタイプ**：stdio、SSE、HTTP トランスポートをサポート（トランスポートサポートはサーバーによって異なる場合があります）

プラグイン MCP サーバーの表示：

```
## Claude Code 内で、プラグインのものを含むすべての MCP サーバーを表示
/mcp
```

プラグインサーバーはプラグインから来ていることを示すインジケータ付きでリストに表示されます。

プラグイン MCP サーバーの利点：

- ・ **バンドル配布**：ツールとサーバーが一緒にパッケージ化されます
- ・ **自動セットアップ**：手動の MCP 設定は不要です
- ・ **チーム一貫性**：プラグインがインストールされると、すべてのユーザーが同じツールを取得します

プラグインで MCP サーバーをバンドルする詳細については、[プラグインコンポーネントリファレンス](#)を参照してください。

MCP インストールスコープ

MCP サーバーは3つの異なるスコープレベルで設定でき、それぞれサーバーのアクセス可能性と共有を管理するための異なる目的を果たします。これらのスコープを理解することで、特定のニーズに合わせてサーバーを設定する最適な方法を決定するのに役立ちます。

ローカルスコープ

ローカルスコープのサーバーはデフォルトの設定レベルを表し、プロジェクトのパスの下での `~/.claude.json` に保存されます。これらのサーバーはプライベートなままで、現在のプロジェクトディレクトリ内で作業する場合にのみアクセス可能です。このスコープは、個人開発サーバー、実験的な設定、または共有すべきでない機密認証情報を含むサーバーに最適です。

Note:

MCP サーバーの「ローカルスコープ」という用語は、一般的なローカル設定とは異なります。MCP ローカルスコープのサーバーは `~/.claude.json`（ホームディレクトリ）に保存されますが、一般的なローカル設定は `.claude/settings.local.json`（プロジェクトディレクトリ内）を使用します。設定ファイルの場所の詳細については、[設定](#)を参照してください。

```
## ローカルスコープのサーバーを追加する（デフォルト）
```

```
claude mcp add --transport http stripe https://mcp.stripe.com
```

```
## ローカルスコープを明示的に指定する
```

```
claude mcp add --transport http stripe --scope local https://mcp.stripe.com
```

プロジェクトスコープ

プロジェクトスコープのサーバーは、プロジェクトのルートディレクトリの `.mcp.json` ファイルに設定を保存することで、チーム間のコラボレーションを可能にします。このファイルはバージョン管理にチェックインするように設計されており、すべてのチームメンバーが同じ MCP ツールとサービスにアクセスできることを保証します。プロジェクトスコープのサーバーを追加すると、Claude Code は自動的にこのファイルを作成または更新して、適切な設定構造を使用します。

```
## プロジェクトスコープのサーバーを追加する
```

```
claude mcp add --transport http paypal --scope project https://mcp.paypal.com/mcp
```

結果の `.mcp.json` ファイルは標準化された形式に従います：

```
{
  "mcpServers": {
    "shared-server": {
      "command": "/path/to/server",
      "args": [],
      "env": {}
    }
  }
}
```

セキュリティ上の理由から、Claude Code は `.mcp.json` ファイルからプロジェクトスコープのサーバーを使用する前に承認を求めます。これらの承認選択をリセットする必要がある場合は、`claude mcp reset-project-choices` コマンドを使用してください。

ユーザースコープ

ユーザースコープのサーバーは `~/.claude.json` に保存され、クロスプロジェクトのアクセス可能性を提供し、マシン上のすべてのプロジェクト全体で利用可能になりながら、ユーザーアカウントにプライベートなままです。このスコープは、個人的なユーティリティサーバー、開発ツール、または異なるプロジェクト全体で頻繁に使用するサービスに適しています。

ユーザーサーバーを追加する

```
claude mcp add --transport http hubspot --scope user https://mcp.hubspot.com/anthropic
```

適切なスコープの選択

以下に基づいてスコープを選択してください：

- **ローカルスコープ**：個人サーバー、実験的な設定、または1つのプロジェクトに固有の機密認証情報
- **プロジェクトスコープ**：チーム共有サーバー、プロジェクト固有のツール、またはコラボレーションに必要なサービス
- **ユーザースコープ**：複数のプロジェクト全体で必要な個人的なユーティリティ、開発ツール、または頻繁に使用されるサービス

Note:

MCP サーバーはどこに保存されていますか？

- **ユーザーおよびローカルスコープ**： `~/.claude.json`（`mcpServers` フィールドまたはプロジェクトパスの下）
- **プロジェクトスコープ**：プロジェクトルートの `.mcp.json`（ソース管理にチェックイン）
- **管理対象**：システムディレクトリの `managed-mcp.json`（[管理対象 MCP 設定](#)を参照）

スコープの階層と優先順位

MCP サーバー設定は明確な優先順位の階層に従います。同じ名前のサーバーが複数のスコープに存在する場合、システムはローカルスコープのサーバーを最初に優先し、次にプロジェクトスコープのサーバー、最後にユーザースコープのサーバーを優先することで競合を解決します。この設計により、必要に応じて個人設定が共有設定をオーバーライドできることが保証されます。

`.mcp.json` での環境変数の展開

Claude Code は `.mcp.json` ファイルの環境変数の展開をサポートしており、チームが設定を共有しながら、マシン固有のパスと API キーなどの機密値の柔軟性を維持できます。

サポートされている構文：

- `${VAR}` - 環境変数 `VAR` の値に展開されます
- `${VAR:-default}` - `VAR` が設定されている場合は `VAR` に展開され、そうでない場合はデフォルトを使用します

展開場所： 環境変数は以下で展開できます：

- `command` - サーバー実行可能ファイルのパス
- `args` - コマンドライン引数
- `env` - サーバーに渡される環境変数
- `url` - HTTP サーバータイプの場合
- `headers` - HTTP サーバー認証の場合

変数展開を使用した例：

```
{
  "mcpServers": {
    "api-server": {
      "type": "http",
      "url": "${API_BASE_URL:-https://api.example.com}/mcp",
      "headers": {
        "Authorization": "Bearer ${API_KEY}"
      }
    }
  }
}
```

必要な環境変数が設定されておらず、デフォルト値がない場合、Claude Code は設定の解析に失敗します。

実践的な例

`{/* ### 例：Playwright でブラウザテストを自動化する`

```
claude mcp add --transport stdio playwright -- npx -y @playwright/mcp@latest
```

その後、ブラウザテストを作成して実行します：

```
test@example.com でログインフローが機能するかテストしてください
```

モバイルでチェックアウトページのスクリーンショットを撮ってください

検索機能が結果を返すことを確認してください

```
``` */}
```

`#### 例：Sentry でエラーを監視する`

```
```bash theme={null}
```

```
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```


Sentry アカウントで認証します：

```
/mcp
```

その後、本番環境の問題をデバッグします：

過去 24 時間で最も一般的なエラーは何ですか？

エラー ID abc123 のスタックトレースを表示してください

どのデプロイメントがこれらの新しいエラーを導入しましたか？

例：コードレビューのために GitHub に接続する

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

必要に応じて GitHub を選択して認証します：

```
/mcp
```

その後、GitHub で作業します：

PR #456 をレビューして改善を提案してください

見つけたバグの新しい課題を作成してください

自分に割り当てられているすべてのオープン PR を表示してください

例：PostgreSQL データベースをクエリする

```
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \  
--dsn "postgresql://readonly:pass@prod.db.com:5432/analytics"
```

その後、データベースを自然に照会します：

今月の総収益はいくらですか？

orders テーブルのスキーマを表示してください

過去 90 日間に購入していない顧客を検索してください

リモート MCP サーバーで認証する

多くのクラウドベースの MCP サーバーは認証が必要です。Claude Code は安全な接続のために OAuth 2.0 をサポートしています。

Step 1: 認証が必要なサーバーを追加する

例：

```
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Step 2: Claude Code 内で /mcp コマンドを使用する

Claude Code で、コマンドを使用します：

/mcp

その後、ブラウザでログインするための手順に従ってください。

Tip:

ヒント：

- ・ 認証トークンは安全に保存され、自動的に更新されます
- ・ `/mcp` メニューで「Clear authentication」を使用してアクセスを取り消します

- ブラウザが自動的に開かない場合は、提供された URL をコピーして手動で開いてください
- ブラウザのリダイレクトが認証後に接続エラーで失敗する場合は、ブラウザのアドレスバーから完全なコールバック URL を Claude Code に表示される URL プロンプトに貼り付けてください
- OAuth 認証は HTTP サーバーで機能します

固定 OAuth コールバックポートを使用する

一部の MCP サーバーは、事前に登録された特定のリダイレクト URI が必要です。デフォルトでは、Claude Code は OAuth コールバック用にランダムに利用可能なポートを選択します。 `--callback-port` を使用してポートを固定し、 `http://localhost:PORT/callback` の形式の事前登録されたリダイレクト URI と一致させます。

`--callback-port` を単独で使用できます（動的クライアント登録を使用）、または `--client-id` と一緒に使用できます（事前設定された認証情報を使用）。

動的クライアント登録を使用した固定コールバックポート

```
claude mcp add --transport http \
  --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

事前設定された OAuth 認証情報を使用する

一部の MCP サーバーは自動 OAuth セットアップをサポートしていません。

「Incompatible auth server: does not support dynamic client registration」のようなエラーが表示される場合、サーバーは事前設定された認証情報が必要です。まずサーバーの開発者ポータルを通じて OAuth アプリを登録し、サーバーを追加するときに認証情報を提供してください。

Step 1: サーバーで OAuth アプリを登録する

サーバーの開発者ポータルを通じてアプリを作成し、クライアント ID とクライアントシークレットをメモしてください。

多くのサーバーはリダイレクト URI も必要とします。その場合は、ポートを選択し、 `http://localhost:PORT/callback` の形式でリダイレクト URI を登録してください。次のステップで `--callback-port` と同じポートを使用してください。

Step 2: 認証情報を使用してサーバーを追加する

次のいずれかの方法を選択してください。 `--callback-port` に使用されるポートは、利用可能な任意のポートにすることができます。前のステップで登録したリダイレクト URI と一致する必要があります。

claude mcp add

`--client-id` を使用してアプリのクライアント ID を渡します。 `--client-secret` フラグはマスクされた入力でシークレットを求めます：

```
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

JSON 設定に `oauth` オブジェクトを含め、 `--client-secret` を別のフラグとして渡します：

```
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（コールバックポートのみ）

動的クライアント登録を使用しながらポートを固定するには、クライアント ID なしで `--callback-port` を使用します：

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":
{"callbackPort":8080}}'
```

CI / env var

環境変数を通じてシークレットを設定して、対話的なプロンプトをスキップします：

```
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: Claude Code で認証する

Claude Code で `/mcp` を実行し、ブラウザのログインフローに従ってください。

Tip:

ヒント:

- ・クライアントシークレットはシステムキーチェーン（macOS）または認証情報ファイルに安全に保存され、設定には保存されません
- ・サーバーがシークレットなしのパブリック OAuth クライアントを使用する場合は、`--client-secret` なしで `--client-id` のみを使用してください
- ・`--callback-port` は `--client-id` の有無にかかわらず使用できます
- ・これらのフラグは HTTP および SSE トランスポートにのみ適用されます。stdio サーバーには影響しません
- ・`claude mcp get <name>` を使用して、OAuth 認証情報がサーバーに設定されていることを確認してください

OAuth メタデータ検出をオーバーライドする

MCP サーバーが標準 OAuth メタデータエンドポイント（`/.well-known/oauth-authorization-server`）でエラーを返すが、動作する OIDC エンドポイントを公開している場合、Claude Code に指定した URL から OAuth メタデータを直接取得するように指示でき、標準検出チェーンをバイパスできます。

`.mcp.json` のサーバー設定の `oauth` オブジェクトに `authServerMetadataUrl` を設定します：

```
{
  "mcpServers": {
    "my-server": {
      "type": "http",
      "url": "https://mcp.example.com/mcp",
      "oauth": {
        "authServerMetadataUrl": "https://auth.example.com/.well-known/openid-configuration"
      }
    }
  }
}
```

URL は `https://` を使用する必要があります。このオプションには Claude Code v2.1.64 以降が必要です。

JSON 設定から MCP サーバーを追加する

MCP サーバーの JSON 設定がある場合は、直接追加できます：

Step 1: JSON から MCP サーバーを追加する

基本的な構文

```
claude mcp add-json <name> '<json>'
```

例：JSON 設定を使用して HTTP サーバーを追加する

```
claude mcp add-json weather-api '{"type":"http","url":"https://api.weather.com/mcp","headers":{"Authorization":"Bearer token"}}'
```

例：JSON 設定を使用して stdio サーバーを追加する

```
claude mcp add-json local-weather '{"type":"stdio","command":"/path/to/weather-cli","args":["--api-key","abc123"],"env":{"CACHE_DIR":"/tmp"}}'
```

例：事前設定された OAuth 認証情報を使用して HTTP サーバーを追加する

```
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: サーバーが追加されたことを確認する

```
claude mcp get weather-api
```

Tip:

ヒント：

- JSON がシェルで適切にエスケープされていることを確認してください
- JSON は MCP サーバー設定スキーマに準拠する必要があります
- `--scope user` を使用して、プロジェクト固有のサーバーの代わりにユーザー設定にサーバーを追加できます

Claude Desktop から MCP サーバーをインポートする

Claude Desktop で MCP サーバーを既に設定している場合は、それらをインポートできます：

Step 1: Claude Desktop からサーバーをインポートする

基本的な構文

```
claude mcp add-from-claude-desktop
```

Step 2: インポートするサーバーを選択する

コマンドを実行した後、インポートするサーバーを選択できる対話的なダイアログが表示されます。

Step 3: サーバーがインポートされたことを確認する

```
claude mcp list
```

Tip:

ヒント：

- この機能は macOS と Windows Subsystem for Linux (WSL) でのみ機能します
- これらのプラットフォームの標準的な場所から Claude Desktop 設定ファイルを読み取ります
- `--scope user` フラグを使用してサーバーをユーザー設定に追加します
- インポートされたサーバーは Claude Desktop と同じ名前を持ちます
- 同じ名前のサーバーが既に存在する場合、数値サフィックスが付与されます（例：`server_1`）

Claude.ai から MCP サーバーを使用する

[Claude.ai](#) アカウントで Claude Code にログインしている場合、Claude.ai で追加した MCP サーバーは Claude Code で自動的に利用可能です：

Step 1: Claude.ai で MCP サーバーを設定する

[claude.ai/settings/connectors](#) でサーバーを追加します。Team および Enterprise プランでは、管理者のみがサーバーを追加できます。

Step 2: MCP サーバーを認証する

Claude.ai で必要な認証ステップを完了します。

Step 3: Claude Code でサーバーを表示および管理する

Claude Code で、コマンドを使用します：

```
/mcp
```

Claude.ai サーバーはリストに表示され、Claude.ai から来ていることを示すインジケータが付きます。

Claude Code で claude.ai MCP サーバーを無効にするには、

`ENABLE_CLAUDEAI_MCP_SERVERS` 環境変数を `false` に設定します：

```
ENABLE_CLAUDEAI_MCP_SERVERS=false claude
```

Claude Code を MCP サーバーとして使用する

Claude Code 自体を MCP サーバーとして使用でき、他のアプリケーションが接続できます：

```
## Claude を stdio MCP サーバーとして起動する
claude mcp serve
```

これを Claude Desktop で使用するには、この設定を `claude_desktop_config.json` に追加します：

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Warning:

実行可能ファイルパスの設定： `command` フィールドは Claude Code 実行可能ファイルを参照する必要があります。 `claude` コマンドがシステムの PATH にはない場合は、実行可能ファイルへの完全なパスを指定する必要があります。

完全なパスを見つけるには：

```
which claude
```

その後、設定で完全なパスを使用します：

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "/full/path/to/claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

正しい実行可能ファイルパスがないと、`spawn claude ENOENT` のようなエラーが発生します。

Tip:

ヒント：

- ・サーバーは View、Edit、LS などの Claude のツールへのアクセスを提供します
- ・Claude Desktop で、Claude にディレクトリ内のファイルを読み取り、編集などを行うよう依頼してみてください。
- ・この MCP サーバーは Claude Code のツールのみを MCP クライアントに公開しているため、独自のクライアントは個々のツール呼び出しのユーザー確認を実装する責任があります。

MCP 出力制限と警告

MCP ツールが大きな出力を生成する場合、Claude Code はトークン使用量を管理して、会話コンテキストが圧倒されるのを防ぐのに役立ちます：

- ・**出力警告閾値**：Claude Code は MCP ツール出力が 10,000 トークンを超えると警告を表示します
- ・**設定可能な制限**：`MAX_MCP_OUTPUT_TOKENS` 環境変数を使用して、許可される最大 MCP 出力トークンを調整できます
- ・**デフォルト制限**：デフォルトの最大値は 25,000 トークンです

大きな出力を生成するツールの制限を増やすには：

```
## MCP ツール出力の制限を高くする
export MAX_MCP_OUTPUT_TOKENS=50000
claude
```

これは特に以下を行う MCP サーバーで役立ちます：

- ・ 大規模なデータセットまたはデータベースをクエリする
- ・ 詳細なレポートまたはドキュメントを生成する
- ・ 広範なログファイルまたはデバッグ情報を処理する

Warning:

特定の MCP サーバーで出力警告が頻繁に発生する場合は、制限を増やすか、サーバーをページネーションまたはフィルタリング応答するように設定することを検討してください。

MCP リソースを使用する

MCP サーバーはリソースを公開でき、ファイルを参照する方法と同様に @ メンションを使用して参照できます。

MCP リソースを参照する

Step 1: 利用可能なリソースをリストする

プロンプトで @ を入力して、接続されているすべての MCP サーバーから利用可能なリソースを表示します。リソースはオートコンプリートメニューのファイルと一緒に表示されます。

Step 2: 特定のリソースを参照する

@server:protocol://resource/path の形式を使用してリソースを参照します：

```
@github:issue://123 を分析して修正を提案できますか？
```

```
@docs:file://api/authentication のAPI ドキュメントをレビューしてください
```

Step 3: 複数のリソース参照

1 つのプロンプトで複数のリソースを参照できます：

@postgres:schema://users と @docs:file://database/user-model を比較してください

Tip:

ヒント：

- ・ リソースは参照されると自動的に取得され、添付ファイルとして含まれます
- ・ リソースパスは @ メンションオートコンプリートでファジー検索可能です
- ・ Claude Code はサーバーがサポートしている場合、MCP リソースをリストおよび読み取るツールを自動的に提供します
- ・ リソースには、MCP サーバーが提供するあらゆるタイプのコンテンツ（テキスト、JSON、構造化データなど）を含めることができます

MCP ツール検索でスケーリングする

多くの MCP サーバーを設定している場合、ツール定義はコンテキストウィンドウの大部分を消費する可能性があります。MCP ツール検索は、すべてをプリロードする代わりに、オンデマンドでツールを動的にロードすることでこれを解決します。

仕組み

Claude Code は、MCP ツール説明がコンテキストウィンドウの 10% 以上を消費する場合、ツール検索を自動的に有効にします。[このしきい値を調整](#)するか、ツール検索を完全に無効にすることができます。トリガーされると：

1. MCP ツールは事前にコンテキストにロードされるのではなく、遅延されます
2. Claude は必要に応じて検索ツールを使用して関連する MCP ツールを検出します
3. Claude が実際に必要とするツールのみがコンテキストにロードされます
4. MCP ツールは引き続き、あなたの視点からは以前と同じように機能します

MCP サーバー作成者向け

MCP サーバーを構築している場合、ツール検索が有効になっているとサーバー命令フィールドがより有用になります。サーバー命令は、[スキル](#)の仕組みと同様に、Claude がいつサーバーのツールを検索するかを理解するのに役立ちます。

明確で説明的なサーバー命令を追加して、以下を説明します：

- ・ ツールが処理するタスクのカテゴリ
- ・ Claude がツールを検索すべき場合
- ・ サーバーが提供する主な機能

ツール検索を設定する

ツール検索はデフォルトで自動モードで実行されます。つまり、MCP ツール定義がコンテキストしきい値を超える場合にのみアクティブになります。ツールが少ない場合は、ツール検索なしで通常にロードされます。この機能には、`tool_reference` ブロックをサポートするモデルが必要です：Sonnet 4 以降、または Opus 4 以降。Haiku モデルはツール検索をサポートしていません。

`ENABLE_TOOL_SEARCH` 環境変数でツール検索の動作を制御します：

値	動作
<code>auto</code>	MCP ツールがコンテキストの 10% を超える場合にアクティブになります（デフォルト）
<code>auto:<N></code>	カスタムしきい値でアクティブになります。 <code><N></code> はパーセンテージです（例：5% の場合は <code>auto:5</code> ）
<code>true</code>	常に有効
<code>false</code>	無効、すべての MCP ツールが事前にロードされます

```
## カスタム 5% しきい値を使用する
ENABLE_TOOL_SEARCH=auto:5 claude

## ツール検索を完全に無効にする
ENABLE_TOOL_SEARCH=false claude
```

または、`settings.json` `env` フィールドで値を設定します。

`disallowedTools` 設定を使用して MCPSearch ツールを特別に無効にすることもできます：

```
{
  "permissions": {
    "deny": ["MCPSearch"]
  }
}
```

MCP プロンプトをコマンドとして使用する

MCP サーバーはプロンプトを公開でき、Claude Code でコマンドとして利用可能になります。

MCP プロンプトを実行する

Step 1: 利用可能なプロンプトを検出する

`/` を入力して、MCP サーバーからのプロンプトを含むすべての利用可能なコマンドを表示します。MCP プロンプトは `/mcp__servername__promptname` の形式で表示されます。

Step 2: 引数なしでプロンプトを実行する

```
/mcp__github__list_prs
```

Step 3: 引数を使用してプロンプトを実行する

多くのプロンプトは引数を受け入れます。コマンドの後にスペース区切りで渡します：

```
/mcp__github__pr_review 456
```

```
/mcp__jira__create_issue "ログインフローのバグ" high
```

Tip:

ヒント：

- MCP プロンプトは接続されているサーバーから動的に検出されます
- 引数はプロンプトの定義されたパラメータに基づいて解析されます
- プロンプト結果は会話に直接注入されます
- サーバーとプロンプト名は正規化されます（スペースはアンダースコアになります）

管理対象 MCP 設定

MCP サーバーの集中管理が必要な組織の場合、Claude Code は 2 つの設定オプションをサポートしています：

1. **managed-mcp.json** による排他的制御：ユーザーが変更または拡張できない固定の MCP サーバーセットをデプロイします

2. 許可リスト/拒否リストによるポリシーベースの制御：ユーザーが独自のサーバーを追加できるようにしますが、許可されているサーバーを制限します

これらのオプションにより、IT 管理者は以下を実行できます：

- **従業員がアクセスできる MCP サーバーを制御する**：組織全体で承認された MCP サーバーの標準化されたセットをデプロイします
- **不正な MCP サーバーを防止する**：ユーザーが未承認の MCP サーバーを追加するのを制限します
- **MCP を完全に無効にする**：必要に応じて MCP 機能を完全に削除します

オプション 1： `managed-mcp.json` による排他的制御

`managed-mcp.json` ファイルをデプロイすると、すべての MCP サーバーに対して**排他的な制御**が行われます。ユーザーはこのファイルで定義されているもの以外の MCP サーバーを追加、変更、または使用することはできません。これは、完全な制御を望む組織にとって最も単純なアプローチです。

システム管理者は、設定ファイルをシステム全体のディレクトリにデプロイします：

- macOS： `/Library/Application Support/ClaudeCode/managed-mcp.json`
- Linux および WSL： `/etc/claude-code/managed-mcp.json`
- Windows： `C:\Program Files\ClaudeCode\managed-mcp.json`

Note:

これらはシステム全体のパス（`~/Library/...` のようなユーザーホームディレクトリではない）であり、管理者権限が必要です。IT 管理者によってデプロイされるように設計されています。

`managed-mcp.json` ファイルは標準的な `.mcp.json` ファイルと同じ形式を使用します：

```
{
  "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"
      }
    }
  }
}
```

オプション 2：許可リストと拒否リストによるポリシーベースの制御

排他的な制御を行う代わりに、管理者はユーザーが独自の MCP サーバーを設定できるようにしながら、許可されているサーバーに制限を適用できます。このアプローチは、[管理対象設定ファイル](#)の `allowedMcpServers` と `deniedMcpServers` を使用します。

Note:

オプションの選択： 固定のサーバーセットをデプロイしてユーザーのカスタマイズを行わない場合はオプション 1（`managed-mcp.json`）を使用します。ユーザーがポリシー制約内で独自のサーバーを追加できるようにする場合はオプション 2（許可リスト/拒否リスト）を使用します。

制限オプション

許可リストまたは拒否リストの各エントリは、3 つの方法でサーバーを制限できます：

1. **サーバー名による**（`serverName`）：設定されたサーバーの名前と一致します

2. **コマンドによる** (`serverCommand`) : stdio サーバーを起動するために使用される正確なコマンドと引数と一致します
3. **URL パターンによる** (`serverUrl`) : ワイルドカードサポート付きのリモートサーバー URL と一致します

重要: 各エントリは `serverName` 、 `serverCommand` 、または `serverUrl` のいずれか 1 つだけを持つ必要があります。

設定例

```
{
  "allowedMcpServers": [
    // サーバー名で許可
    { "serverName": "github" },
    { "serverName": "sentry" },

    // 正確なコマンドで許可 (stdio サーバーの場合)
    { "serverCommand": ["npx", "-y", "@modelcontextprotocol/server-filesystem"] },
    { "serverCommand": ["python", "/usr/local/bin/approved-server.py"] },

    // URL パターンで許可 (リモートサーバーの場合)
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ],
  "deniedMcpServers": [
    // サーバー名でブロック
    { "serverName": "dangerous-server" },

    // 正確なコマンドでブロック (stdio サーバーの場合)
    { "serverCommand": ["npx", "-y", "unapproved-package"] },

    // URL パターンでブロック (リモートサーバーの場合)
    { "serverUrl": "https://*.untrusted.com/*" }
  ]
}
```


コマンドベースの制限の仕組み

完全一致：

- ・ コマンド配列は**完全**に一致する必要があります。コマンドと正しい順序のすべての引数
- ・ 例：`["npx", "-y", "server"]` は `["npx", "server"]` または `["npx", "-y", "server", "--flag"]` と一致しません

Stdio サーバーの動作：

- ・ 許可リストに**任意**の `serverCommand` エントリが含まれている場合、stdio サーバーはそれらのコマンドの1つと一致する必要があります
- ・ Stdio サーバーはコマンド制限が存在する場合、名前だけでは通過できません
- ・ これにより、管理者は実行が許可されているコマンドを強制できます

非 stdio サーバーの動作：

- ・ リモートサーバー（HTTP、SSE、WebSocket）は、許可リストに `serverUrl` エントリが存在する場合、URL ベースのマッチングを使用します
- ・ URL エントリが存在しない場合、リモートサーバーは名前ベースのマッチングにフォールバックします
- ・ コマンド制限はリモートサーバーには適用されません

URL ベースの制限の仕組み

URL パターンは `*` を使用してワイルドカードをサポートし、任意の文字シーケンスと一致します。これはドメイン全体またはサブドメイン全体を許可するのに役立ちます。

ワイルドカード例：

- ・ `https://mcp.company.com/*` - 特定のドメイン上のすべてのパスを許可
- ・ `https://*.example.com/*` - example.com の任意のサブドメインを許可
- ・ `http://localhost:*/*` - localhost 上の任意のポートを許可

リモートサーバーの動作：

- ・ 許可リストに**任意**の `serverUrl` エントリが含まれている場合、リモートサーバーはそれらの URL パターンの1つと一致する必要があります
- ・ リモートサーバーは URL 制限が存在する場合、名前だけでは通過できません
- ・ これにより、管理者は許可されているリモートエンドポイントを強制できます

```
{
  "allowedMcpServers": [
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ]
}
```

結果：

- `https://mcp.company.com/api` の HTTP サーバー：✅ 許可（URL パターンと一致）
- `https://api.internal.corp/mcp` の HTTP サーバー：✅ 許可（ワイルドカードサブドメインと一致）
- `https://external.com/mcp` の HTTP サーバー：❌ ブロック（URL パターンと一致しない）
- 任意のコマンドの Stdio サーバー：❌ ブロック（一致する名前またはコマンドエントリがない）

```
{
  "allowedMcpServers": [
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

結果：

- `["npx", "-y", "approved-package"]` の Stdio サーバー：✅ 許可（コマンドと一致）
- `["node", "server.js"]` の Stdio サーバー：❌ ブロック（コマンドと一致しない）
- 「my-api」という名前の HTTP サーバー：❌ ブロック（一致する名前エントリがない）

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

結果：

- 「local-tool」という名前で `["npx", "-y", "approved-package"]` の Stdio サーバー：✅ 許可（コマンドと一致）
- 「local-tool」という名前で `["node", "server.js"]` の Stdio サーバー：❌ ブロック（コマンドエントリが存在しますが一致しない）
- 「github」という名前で `["node", "server.js"]` の Stdio サーバー：❌ ブロック（stdio サーバーはコマンドエントリが存在する場合、コマンドと一致する必要があります）
- 「github」という名前の HTTP サーバー：✅ 許可（名前と一致）
- 「other-api」という名前の HTTP サーバー：❌ ブロック（名前と一致しない）

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverName": "internal-tool" }
  ]
}
```

結果：

- 任意のコマンドで「github」という名前の Stdio サーバー：✅ 許可（コマンド制限なし）
- 任意のコマンドで「internal-tool」という名前の Stdio サーバー：✅ 許可（コマンド制限なし）
- 「github」という名前の HTTP サーバー：✅ 許可（名前と一致）
- 「other」という名前のサーバー：❌ ブロック（名前と一致しない）

許可リストの動作（`allowedMcpServers`）

- `undefined`（デフォルト）：制限なし。ユーザーは任意の MCP サーバーを設定できます
- 空の配列 `[]`：完全なロックダウン。ユーザーは MCP サーバーを設定できません
- エントリのリスト：ユーザーは名前、コマンド、または URL パターンで一致するサーバーのみを設定できます

拒否リストの動作（`deniedMcpServers`）

- `undefined`（デフォルト）：サーバーはブロックされません
- 空の配列 `[]`：サーバーはブロックされません
- エントリのリスト：指定されたサーバーはすべてのスコープ全体で明示的にブロックされます

重要な注意事項

- オプション 1 とオプション 2 を組み合わせることができます：`managed-mcp.json` が存在する場合、排他的な制御があり、ユーザーはサーバーを追加できません。許可リスト/拒否リストは管理対象サーバー自体に引き続き適用されます。
- 拒否リストは絶対的な優先順位を持ちます：サーバーが拒否リストエントリ（名前、コマンド、または URL による）と一致する場合、許可リストに含まれていても、ブロックされます
- 名前ベース、コマンドベース、URL ベースの制限は一緒に機能します：サーバーは名前エントリ、コマンドエントリ、または URL パターンのいずれかと一致する場合に通過します（拒否リストでブロックされていない限り）

Note:

`managed-mcp.json` を使用する場合：ユーザーは `claude mcp add` または設定ファイルを通じて MCP サーバーを追加できません。`allowedMcpServers` と `deniedMcpServers` の設定は、実際にロードされる管理対象サーバーをフィルタリングするために引き続き適用されます。

スキルで Claude を拡張する

Claude Code でスキルを作成、管理、共有して Claude の機能を拡張します。カスタムコマンドとバンドルされたスキルが含まれます。

スキルは Claude ができることを拡張します。 `SKILL.md` ファイルに指示を記述すると、Claude はそれをツールキットに追加します。Claude は関連する場合にスキルを使用するか、 `/skill-name` で直接呼び出すことができます。

Note:

`/help` や `/compact` などの組み込みコマンドについては、[インタラクティブモード](#)を参照してください。

カスタムコマンドはスキルにマージされました。 `.claude/commands/deploy.md` のファイルと `.claude/skills/deploy/SKILL.md` のスキルの両方が `/deploy` を作成し、同じように機能します。既存の `.claude/commands/` ファイルは引き続き機能します。スキルはオプション機能を追加します。サポートファイル用のディレクトリ、[スキルを呼び出すユーザーを制御するためのフロントマター](#)、および Claude が関連する場合に自動的にスキルを読み込む機能です。

Claude Code スキルは [Agent Skills](#) オープンスタンダードに従い、複数の AI ツール全体で機能します。Claude Code は [呼び出し制御](#)、[サブエージェント実行](#)、[動的コンテキスト注入](#)などの追加機能でスタンダードを拡張します。

バンドルされたスキル

バンドルされたスキルは Claude Code に付属し、すべてのセッションで利用可能です。固定ロジックを直接実行する[組み込みコマンド](#)とは異なり、バンドルされたスキルはプロンプトベースです。Claude に詳細なプレイブックを提供し、ツールを使用して作業を調整させます。つまり、バンドルされたスキルは並列エージェントを生成し、ファイルを読み取り、コードベースに適応できます。

バンドルされたスキルは他のスキルと同じ方法で呼び出します。 `/` の後にスキル名を入力します。

- `/simplify`: 最近変更されたファイルをコード再利用、品質、効率の問題についてレビューし、修正します。機能またはバグ修正を実装した後に実行して、作業をクリーンアップします。3つのレビューエージェント（コード再利用、コード品質、効率）

を並列で生成し、その結果を集約して修正を適用します。オプションのテキストを渡して特定の懸念事項に焦点を当てます。 `/simplify focus on memory efficiency` 。

- `/batch <instruction>`: コードベース全体で大規模な変更を並列で調整します。変更の説明を提供すると、`/batch` はコードベースを調査し、作業を 5～30 の独立したユニットに分解し、承認用のプランを提示します。承認されると、各ユニットごとに 1 つのバックグラウンドエージェントを生成し、各エージェントは分離された `git worktree` で実行されます。各エージェントはそのユニットを実装し、テストを実行し、プルリクエストを開きます。git リポジトリが必要です。例: `/batch migrate src/ from Solid to React` 。
- `/debug [description]`: セッションデバッグログを読み取ることで、現在の Claude Code セッションをトラブルシューティングします。オプションで問題を説明して分析に焦点を当てます。
- `/loop [interval] <prompt>`: セッションが開いている間、プロンプトを間隔で繰り返し実行します。Claude は間隔を解析し、定期的な cron タスクをスケジュールし、ケイデンスを確認します。デプロイをポーリングしたり、PR を監視したり、別のスキルを定期的に再実行したりするのに便利です。例: `/loop 5m check if the deploy finished` 。
- `/claude-api`: プロジェクトの言語 (Python、TypeScript、Java、Go、Ruby、C#、PHP、または cURL) の Claude API リファレンス資料と Python および TypeScript の Agent SDK リファレンスを読み込みます。ツール使用、ストリーミング、バッチ、構造化出力、および一般的な落とし穴をカバーしています。また、コードが `anthropic`、`@anthropic-ai/sdk`、または `claude_agent_sdk` をインポートする場合は自動的にアクティブになります。

はじめに

最初のスキルを作成する

この例は、Claude にビジュアルダイアグラムと類推を使用してコードを説明するように教えるスキルを作成します。デフォルトのフロントマターを使用するため、何かの仕組みを尋ねるときに Claude が自動的にスキルを読み込むか、`/explain-code` で直接呼び出すことができます。

Step 1: スキルディレクトリを作成する

個人スキルフォルダにスキル用のディレクトリを作成します。個人スキルはすべてのプロジェクト全体で利用可能です。

```
mkdir -p ~/.claude/skills/explain-code
```

Step 2: SKILL.md を記述する

すべてのスキルには **SKILL.md** ファイルが必要です。2つの部分があります。YAML フロントマター（`---` マーカー間）は Claude にスキルをいつ使用するかを伝え、マークダウンコンテンツはスキルが呼び出されるときに Claude が従う指示です。**name** フィールドは `/slash-command` になり、**description** は Claude がスキルを自動的に読み込むかどうかを決定するのに役立ちます。

`~/.claude/skills/explain-code/SKILL.md` を作成します。

```
---
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: スキルをテストする

2つの方法でテストできます。

説明に一致するものを尋ねることで Claude に自動的に呼び出させます。

```
How does this code work?
```

またはスキル名で直接呼び出します。

```
/explain-code src/auth/login.ts
```

どちらの方法でも、Claude の説明に類推と ASCII ダイアグラムが含まれるはずです。

スキルが存在する場所

スキルを保存する場所によって、誰がそれを使用できるかが決まります。

場所	パス	適用対象
Enterprise	管理設定 を参照	組織内のすべてのユーザー
Personal	<code>~/.claude/skills/<skill-name>/SKILL.md</code>	すべてのプロジェクト
Project	<code>.claude/skills/<skill-name>/SKILL.md</code>	このプロジェクトのみ
Plugin	<code><plugin>/skills/<skill-name>/SKILL.md</code>	プラグインが有効な場所

スキルがレベル全体で同じ名前を共有する場合、優先度の高い場所が優先されます。
enterprise > personal > project。プラグインスキルは `plugin-name:skill-name` 名前空間を使用するため、他のレベルと競合することはできません。`.claude/commands/` にファイルがある場合、それらは同じように機能しますが、スキルとコマンドが同じ名前を共有する場合、スキルが優先されます。

ネストされたディレクトリからの自動検出

サブディレクトリ内のファイル进行操作する場合、Claude Code はネストされた `.claude/skills/` ディレクトリからスキルを自動的に検出します。たとえば、`packages/frontend/` 内のファイルを編集している場合、Claude Code は `packages/frontend/.claude/skills/` でもスキルを探します。これはパッケージが独自のスキルを持つモノレポセットアップをサポートしています。

各スキルは `SKILL.md` をエントリポイントとするディレクトリです。

```
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
```


SKILL.md はメイン指示を含み、必須です。他のファイルはオプションで、より強力なスキルを構築できます。Claude が入力するテンプレート、期待される形式を示す出力例、Claude が実行できるスクリプト、または詳細なリファレンスドキュメント。**SKILL.md** からこれらのファイルを参照して、Claude がそれらの内容と読み込むタイミングを知ることができます。詳細については、[サポートファイルを追加する](#)を参照してください。

Note:

.claude/commands/ 内のファイルは引き続き機能し、同じ**フロントマター**をサポートしています。スキルはサポートファイルなどの追加機能をサポートするため、推奨されます。

追加ディレクトリからのスキル

--add-dir 経由で追加されたディレクトリ内の **.claude/skills/** で定義されたスキルは自動的に読み込まれ、ライブ変更検出によって取得されるため、セッション中に再起動せずに編集できます。

Note:

--add-dir ディレクトリの **CLAUDE.md** ファイルはデフォルトでは読み込まれません。読み込むには、**CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1** を設定します。[追加ディレクトリから読み込む](#)を参照してください。

スキルを設定する

スキルは **SKILL.md** の上部の YAML フロントマターと、その後続くマークダウンコンテンツを通じて設定されます。

スキルコンテンツのタイプ

スキルファイルには任意の指示を含めることができますが、それらを呼び出す方法を考えることは、含める内容をガイドするのに役立ちます。

リファレンスコンテンツ は Claude が現在の作業に適用する知識を追加します。規約、パターン、スタイルガイド、ドメイン知識。このコンテンツはインラインで実行されるため、Claude は会話コンテキストと一緒にそれを使用できます。

```

---
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

```

タスクコンテンツ は Claude に、デプロイ、コミット、コード生成などの特定のアクション用のステップバイステップの指示を提供します。これらは多くの場合、Claude が実行するかどうかを決定させるのではなく、`/skill-name` で直接呼び出したいアクションです。`disable-model-invocation: true` を追加して、Claude が自動的にトリガーするのを防ぎます。

```

---
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

```

SKILL.md には何でも含めることができますが、スキルを呼び出す方法（ユーザーが呼び出すか、Claude が呼び出すか、またはその両方）と実行場所（インラインまたはサブエージェント）を考えることは、含める内容をガイドするのに役立ちます。複雑なスキルの場合、[サポートファイルを追加する](#)ことで、メインスキルに焦点を当てることもできます。

フロントマターリファレンス

マークダウンコンテンツを超えて、**SKILL.md** ファイルの上部の `---` マーカー間の YAML フロントマターフィールドを使用してスキルの動作を設定できます。

```
---
name: my-skill
description: What this skill does
disable-model-invocation: true
allowed-tools: Read, Grep
---

Your skill instructions here...
```

すべてのフィールドはオプションです。Claude がスキルをいつ使用するかを知るために、`description` のみが推奨されます。

フィールド	必須	説明
<code>name</code>	いいえ	スキルの表示名。省略した場合、ディレクトリ名を使用します。小文字、数字、ハイフンのみ（最大 64 文字）。
<code>description</code>	推奨	スキルが何をするか、いつ使用するか。Claude はこれを使用してスキルを適用するかどうかを決定します。省略した場合、マークダウンコンテンツの最初の段落を使用します。
<code>argument-hint</code>	いいえ	予想される引数を示すためにオートコンプリート中に表示されるヒント。例： <code>[issue-number]</code> または <code>[filename] [format]</code> 。
<code>disable-model-invocation</code>	いいえ	Claude がこのスキルを自動的に読み込むのを防ぐには <code>true</code> に設定します。 <code>/name</code> で手動でトリガーしたいワークフロー用に使用します。デフォルト： <code>false</code> 。

フィールド	必須	説明
<code>user-invocable</code>	いいえ	<code>/</code> メニューから非表示にするには <code>false</code> に設定します。ユーザーが直接呼び出すべきではないバックグラウンド知識用に使用します。デフォルト: <code>true</code> 。
<code>allowed-tools</code>	いいえ	このスキルがアクティブな場合、Claude が許可を求めずに使用できるツール。
<code>model</code>	いいえ	このスキルがアクティブな場合に使用するモデル。
<code>context</code>	いいえ	フォークされたサブエージェントコンテキストで実行するには <code>fork</code> に設定します。
<code>agent</code>	いいえ	<code>context: fork</code> が設定されている場合に使用するサブエージェントタイプ。
<code>hooks</code>	いいえ	このスキルのライフサイクルにスコープされたフック。設定形式については、 スキルとエージェントのフック を参照してください。

利用可能な文字列置換

スキルはスキルコンテンツの動的値の文字列置換をサポートしています。

変数	説明
<code>\$ARGUMENTS</code>	スキルを呼び出すときに渡されたすべての引数。 <code>\$ARGUMENTS</code> がコンテンツに存在しない場合、引数は <code>ARGUMENTS: <value></code> として追加されます。
<code>\$ARGUMENTS[N]</code>	0 ベースのインデックスで特定の引数にアクセスします。例: 最初の引数の場合は <code>\$ARGUMENTS[0]</code> 。

変数	説明
<code>\$N</code>	<code>\$ARGUMENTS[N]</code> の短縮形。例：最初の引数の場合は <code>\$0</code> 、2 番目の引数の場合は <code>\$1</code> 。
<code>\$ {CLAUDE_SESSION_ID }</code>	現在のセッション ID。ログ、セッション固有のファイルの作成、またはスキル出力とセッションの相関付けに便利です。
<code>\$ {CLAUDE_SKILL_DIR}</code>	スキルの <code>SKILL.md</code> ファイルを含むディレクトリ。プラグインスキルの場合、これはプラグインルートではなく、プラグイン内のスキルのサブディレクトリです。現在の作業ディレクトリに関係なく、スキルにバンドルされたスクリプトまたはファイルを参照するために <code>bash</code> インジェクションコマンドで使います。

置換を使用した例：

```
---
name: session-logger
description: Log activity for this session
---

Log the following to logs/${CLAUDE_SESSION_ID}.log:

$ARGUMENTS
```

サポートファイルを追加する

スキルはディレクトリ内に複数のファイルを含めることができます。これにより、`SKILL.md` は本質的なものに焦点を当てながら、Claude は必要な場合にのみ詳細なリファレンス資料にアクセスできます。大規模なリファレンスドキュメント、API 仕様、または例のコレクションは、スキルが実行されるたびにコンテキストに読み込む必要はありません。

```
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)
```

SKILL.md からサポートファイルを参照して、Claude が各ファイルの内容と読み込むタイミングを知るようにします。

Additional resources

- For complete API details, see `[reference.md](reference.md)`
- For usage examples, see `[examples.md](examples.md)`

Tip:

SKILL.md を 500 行以下に保ちます。詳細なリファレンス資料を別のファイルに移動します。

スキルを呼び出すユーザーを制御する

デフォルトでは、ユーザーと Claude の両方がスキルを呼び出すことができます。 `/skill-name` を入力して直接呼び出すことができ、Claude は会話に関連する場合に自動的にスキルを読み込むことができます。2つのフロントマターフィールドでこれを制限できます。

- **disable-model-invocation: true**: ユーザーのみがスキルを呼び出すことができます。副作用があるワークフロー、または `/commit`、`/deploy`、`/send-slack-message` などのタイミングを制御したいワークフロー用に使用します。コードが準備完了に見えるため、Claude がデプロイすることを望みません。
- **user-invocable: false**: Claude のみがスキルを呼び出すことができます。アクションできないバックグラウンド知識用に使用します。 `legacy-system-context` スキルは古いシステムの仕組みを説明しています。Claude は関連する場合にこれを知るべきですが、 `/legacy-system-context` はユーザーが実行する意味のあるアクションではありません。

この例は、ユーザーのみがトリガーできるデプロイスキルを作成します。

disable-model-invocation: true フィールドは Claude が自動的に実行するのを防ぎます。

```
---
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
```

2つのフィールドが呼び出しとコンテキスト読み込みにどのように影響するかは次のとおりです。

フロントマ ター	ユーザーが呼び出せる	Claude が呼び出せる	コンテキストに読み 込まれるタイミング
(デフォ ルト)	はい	はい	説明は常にコンテ キストに含まれ、呼び 出されるとフルスキ ルが読み込まれます
<code>disable- model- invocatio n: true</code>	はい	いいえ	説明はコンテキスト に含まれず、ユー ザーが呼び出すとフ ルスキルが読み込ま れます
<code>user- invocable : false</code>	いいえ	はい	説明は常にコンテ キストに含まれ、呼び 出されるとフルスキ ルが読み込まれます

Note:

通常のセッションでは、スキル説明がコンテキストに読み込まれるため、Claude は利用可能なものを知っていますが、フルスキルコンテンツは呼び出されるときにのみ読み込まれます。[プリロードされたスキルを持つサブエージェント](#)は異なります。フルスキルコンテンツはスタートアップで注入されます。

ツールアクセスを制限する

`allowed-tools` フィールドを使用して、スキルがアクティブな場合に Claude が使用できるツールを制限します。このスキルは、Claude がファイルを探索できるが変更できない読み取り専用モードを作成します。

```
---
name: safe-reader
description: Read files without making changes
allowed-tools: Read, Grep, Glob
---
```

スキルに引数を渡す

ユーザーと Claude の両方がスキルを呼び出すときに引数を渡すことができます。引数は `$ARGUMENTS` プレースホルダーを通じて利用可能です。

このスキルは GitHub の問題を番号で修正します。 `$ARGUMENTS` プレースホルダーはスキル名の後に続くものに置き換えられます。

```
---
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

`/fix-issue 123` を実行すると、Claude は `'Fix GitHub issue 123 following our coding standards...'` を受け取ります。

スキルに引数を指定して呼び出しても、スキルに `$ARGUMENTS` が含まれていない場合、Claude Code は `ARGUMENTS: <your input>` をスキルコンテンツの最後に追加するため、Claude は入力したものを引き続き見ることができます。

位置で個別の引数にアクセスするには、`$ARGUMENTS[N]` または短い `$N` を使用します。

```
---
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.
```

`/migrate-component SearchBar React Vue` を実行すると、`$ARGUMENTS[0]` が `SearchBar` に、`$ARGUMENTS[1]` が `React` に、`$ARGUMENTS[2]` が `Vue` に置き換えられます。`$N` 短縮形を使用する同じスキル。

```
---
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.
```

高度なパターン

動的コンテキストを注入する

`!`command`` 構文はスキルコンテンツが Claude に送信される前にシェルコマンドを実行します。コマンド出力はプレースホルダーを置き換えるため、Claude はコマンド自体ではなく実際のデータを受け取ります。

このスキルは GitHub CLI でライブ PR データを取得することで、プルリクエストを要約します。`!`gh pr diff`` およびその他のコマンドが最初に実行され、その出力がプロンプトに挿入されます。

```
---
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...
```

このスキルが実行されるとき：

1. 各 `!`command`` が直ちに実行されます（Claude が何かを見る前に）
2. 出力はスキルコンテンツのプレースホルダーを置き換えます
3. Claude は実際の PR データを含む完全にレンダリングされたプロンプトを受け取ります

これは前処理であり、Claude が実行するものではありません。Claude は最終結果のみを見ます。

Tip:

スキルで [拡張思考](#) を有効にするには、スキルコンテンツのどこかに `'ultrathink'` という単語を含めます。

スキルをサブエージェントで実行する

スキルを分離して実行したい場合は、フロントマターに `context: fork` を追加します。スキルコンテンツはサブエージェントを駆動するプロンプトになります。会話履歴にアクセスできません。

Warning:

context: fork は明示的な指示を含むスキルにのみ意味があります。スキルにタスクなしで 'これらの API 規約を使用する' などのガイドラインが含まれている場合、サブエージェントはガイドラインを受け取りますが、実行可能なプロンプトはなく、意味のある出力なしで返されます。

スキルと [サブエージェント](#) は 2 つの方向で連携します。

アプローチ	システムプロンプト	タスク	また読み込む
context: fork を持つスキル	エージェントタイプから (Explore 、 Plan など)	SKILL.md コンテンツ	CLAUDE.md
skills フィールドを持つサブエージェント	サブエージェントのマークダウン本体	Claude の委任メッセージ	プリロードされたスキル + CLAUDE.md

context: fork を使用すると、スキルにタスクを記述し、実行するエージェントタイプを選択します。逆の場合（スキルをリファレンス資料として使用するカスタムサブエージェントを定義する）については、 [サブエージェント](#) を参照してください。

例：Explore エージェントを使用した研究スキル

このスキルはフォークされた Explore エージェントで研究を実行します。スキルコンテンツはタスクになり、エージェントはコードベース探索に最適化された読み取り専用ツールを提供します。

```
---
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
```

このスキルが実行されるとき：

1. 新しい分離されたコンテキストが作成されます
2. サブエージェントはスキルコンテンツをプロンプト（‘\$ARGUMENTS を徹底的に調査…’）として受け取ります
3. `agent` フィールドは実行環境（モデル、ツール、権限）を決定します
4. 結果は要約され、メイン会話に返されます

`agent` フィールドは使用するサブエージェント設定を指定します。オプションには、組み込みエージェント（`Explore`、`Plan`、`general-purpose`）または `.claude/agents/` からのカスタムサブエージェントが含まれます。省略した場合、`general-purpose` を使用します。

Claude のスキルアクセスを制限する

デフォルトでは、Claude は `disable-model-invocation: true` が設定されていないスキルを呼び出すことができます。 `allowed-tools` を定義するスキルは、スキルがアクティブな場合、ユーザーごとの承認なしに Claude にこれらのツールへのアクセスを付与します。権限設定は引き続き、他のすべてのツールのベースライン承認動作を管理します。 `/compact` や `/init` などの組み込みコマンドは、Skill ツールを通じて利用できません。

Claude が呼び出すことができるスキルを制御する 3 つの方法。

`/permissions` で Skill ツールを拒否することで、すべてのスキルを無効にします。

```
## Add to deny rules:  
Skill
```

権限ルールを使用して特定のスキルを許可または拒否します。

```
## Allow only specific skills  
Skill(commit)  
Skill(review-pr *)  
  
## Deny specific skills  
Skill(deploy *)
```

権限構文：完全一致の場合は `Skill(name)`、任意の引数を含むプレフィックス一致の場合は `Skill(name *)`。

フロントマターに `disable-model-invocation: true` を追加することで、個別のスキルを非表示にします。これにより、Claude のコンテキストからスキルが完全に削除されます。

Note:

`user-invocable` フィールドはメニューの可視性のみを制御し、Skill ツールアクセスは制御しません。プログラムによる呼び出しをブロックするには `disable-model-invocation: true` を使用します。

スキルを共有する

スキルはオーディエンスに応じて異なるスコープで配布できます。

- ・ **プロジェクトスキル**: `.claude/skills/` をバージョン管理にコミットします
- ・ **プラグイン**: **プラグイン**に `skills/` ディレクトリを作成します
- ・ **管理**: **管理設定**を通じて組織全体にデプロイします

ビジュアル出力を生成する

スキルは任意の言語でスクリプトをバンドルして実行でき、Claude に単一のプロンプトで可能なもの以上の機能を提供します。1 つの強力なパターンはビジュアル出力を生成することです。データの探索、デバッグ、またはレポート作成用にブラウザで開くインタラクティブな HTML ファイル。

この例はコードベースエクスプローラーを作成します。ディレクトリを展開および折りたたむことができるインタラクティブなツリービュー、ファイルサイズを一目で確認、ファイルタイプを色で識別します。

スキルディレクトリを作成します。

```
mkdir -p ~/.claude/skills/codebase-visualizer/scripts
```

`~/.claude/skills/codebase-visualizer/SKILL.md` を作成します。説明は Claude にこのスキルをいつアクティブにするかを伝え、指示は Claude にバンドルされたスクリプトを実行するよう伝えます。

```
---
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 .
```

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
```

`~/.claude/skills/codebase-visualizer/scripts/visualize.py` を作成します。このスクリプトはディレクトリツリーをスキャンし、以下を含む自己完結型の HTML ファイルを生成します。

- ファイル数、ディレクトリ数、合計サイズ、ファイルタイプ数を表示する**サマリーサイドバー**
- コードベースをファイルタイプ別に分類する**棒グラフ**（サイズ別トップ8）

- ディレクトリを展開および折りたたむことができる**折りたたみ可能なツリー**。色分けされたファイルタイプインジケータ付き

スクリプトは Python が必要ですが、組み込みライブラリのみを使用するため、インストールするパッケージはありません。


```
#!/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][: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="211 358 228 371"/> 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="211 638 228 651"/> {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()}')
```

テストするには、任意のプロジェクトで Claude Code を開き、「Visualize this codebase」と尋ねます。Claude はスクリプトを実行し、`codebase-map.html` を生成し、ブラウザで開きます。

このパターンは任意のビジュアル出力に機能します。依存関係グラフ、テストカバレッジレポート、API ドキュメント、またはデータベーススキーマの可視化。バンドルされたスクリプトが重い処理を行い、Claude がツールを使用して作業を調整します。

トラブルシューティング

スキルがトリガーされない

Claude がスキルを期待どおりに使用しない場合：

1. 説明にユーザーが自然に言うキーワードが含まれていることを確認します
2. スキルが「利用可能なスキルは何ですか？」に表示されることを確認します
3. 説明により密接に一致するようにリクエストを言い換えてみます
4. スキルがユーザーが呼び出し可能な場合は、`/skill-name` で直接呼び出してみます

スキルが頻繁にトリガーされる

Claude がスキルを望まないときに使用する場合：

1. 説明をより具体的にします
2. スキルを手動で呼び出したい場合のみ `disable-model-invocation: true` を追加します

Claude がすべてのスキルを見ない

スキル説明がコンテキストに読み込まれるため、Claude は利用可能なものを知っています。多くのスキルがある場合、文字予算を超える可能性があります。予算は動的にコンテキストウィンドウの 2% でスケーリングされ、16,000 文字のフォールバックがあります。`/context` を実行して、除外されたスキルに関する警告を確認します。

制限をオーバーライドするには、`SLASH_COMMAND_TOOL_CHAR_BUDGET` 環境変数を設定します。

関連リソース

- [サブエージェント](#): 特化したエージェントにタスクを委任する
- [プラグイン](#): スキルと他の拡張機能をパッケージ化して配布する
- [フック](#): ツールイベント周辺のワークフローを自動化する
- [メモリ](#): 永続的なコンテキスト用に CLAUDE.md ファイルを管理する
- [インタラクティブモード](#): 組み込みコマンドとショートカット
- [権限](#): ツールとスキルアクセスを制御する

プラグインを作成する

スキル、エージェント、フック、MCP サーバーで Claude Code を拡張するカスタムプラグインを作成します。

プラグインを使用すると、Claude Code をカスタム機能で拡張でき、プロジェクトとチーム全体で共有できます。このガイドでは、スキル、エージェント、フック、MCP サーバーを使用して独自のプラグインを作成する方法について説明します。

既存のプラグインをインストールしたいですか？ [プラグインを検出してインストールする](#) を参照してください。完全な技術仕様については、[プラグインリファレンス](#) を参照してください。

プラグインとスタンドアロン設定を使い分ける

Claude Code では、カスタムスキル、エージェント、フックを追加する 2 つの方法をサポートしています。

アプローチ	スキル名	最適な用途
スタンドアロン (<code>.claude/</code> ディレクトリ)	<code>/hello</code>	個人的なワークフロー、プロジェクト固有のカスタマイズ、クイック実験
プラグイン (<code>.claude-plugin/plugin.json</code> を含むディレクトリ)	<code>/plugin-name:hello</code>	チームメンバーとの共有、コミュニティへの配布、バージョン管理されたりリリース、プロジェクト全体で再利用可能

スタンドアロン設定を使用する場合：

- ・単一のプロジェクト用に Claude Code をカスタマイズしている
- ・設定が個人的で共有する必要がない
- ・スキルやフックをパッケージ化する前に実験している
- ・`/hello` や `/deploy` のような短いスキル名が必要

プラグインを使用する場合：

- ・機能をチームまたはコミュニティと共有したい

- 複数のプロジェクト全体で同じスキル/エージェントが必要
- 拡張機能のバージョン管理と簡単な更新が必要
- マーケットプレイスを通じて配布している
- `/my-plugin:hello` のような名前空間付きスキルで問題ない（名前空間はプラグイン間の競合を防ぎます）

Tip:

`.claude/` でスタンドアロン設定を使用してクイック反復を行い、共有する準備ができれば
[既存の設定をプラグインに変換](#)してください。

クイックスタート

このクイックスタートでは、カスタムスキルを使用してプラグインを作成する手順を説明します。マニフェスト（プラグインを定義する設定ファイル）を作成し、スキルを追加して、`--plugin-dir` フラグを使用してローカルでテストします。

前提条件

- Claude Code [インストール済みで認証済み](#)
- Claude Code バージョン 1.0.33 以降（`claude --version` で確認）

Note:

`/plugin` コマンドが表示されない場合は、Claude Code を最新バージョンに更新してください。アップグレード手順については、[トラブルシューティング](#)を参照してください。

最初のプラグインを作成する

Step 1: プラグインディレクトリを作成する

すべてのプラグインは、マニフェストとスキル、エージェント、またはフックを含む独自のディレクトリに存在します。今すぐ作成してください。

```
mkdir my-first-plugin
```

Step 2: プラグインマニフェストを作成する

`.claude-plugin/plugin.json` のマニフェストファイルは、プラグインの ID（名前、説明、バージョン）を定義します。Claude Code はこのメタデータを使用して、プラグインマネージャーにプラグインを表示します。

プラグインフォルダ内に `.claude-plugin` ディレクトリを作成します。

```
mkdir my-first-plugin/.claude-plugin
```

次に、このコンテンツで `my-first-plugin/.claude-plugin/plugin.json` を作成します。

```
{
  "name": "my-first-plugin",
  "description": "A greeting plugin to learn the basics",
  "version": "1.0.0",
  "author": {
    "name": "Your Name"
  }
}
```

フィールド	目的
<code>name</code>	一意の識別子とスキル名前空間。スキルにはこれが接頭辞として付きます（例： <code>/my-first-plugin:hello</code> ）。
<code>description</code>	プラグインマネージャーでプラグインを参照またはインストールするときに表示されます。
<code>version</code>	セマンティックバージョニング を使用してリリースを追跡します。
<code>author</code>	オプション。属性に役立ちます。

`homepage` 、 `repository` 、 `license` などの追加フィールドについては、[完全なマニフェストスキーマ](#)を参照してください。

Step 3: スキルを追加する

スキルは `skills/` ディレクトリに存在します。各スキルは `SKILL.md` ファイルを含むフォルダです。フォルダ名がスキル名になり、プラグインの名前空間が接頭辞として付きます（ `my-first-plugin` という名前のプラグイン内の `hello/` は `/my-first-plugin:hello` を作成します ）。

プラグインフォルダ内にスキルディレクトリを作成します。

```
mkdir -p my-first-plugin/skills/hello
```


次に、このコンテンツで `my-first-plugin/skills/hello/SKILL.md` を作成します。

```
---
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: プラグインをテストする

`--plugin-dir` フラグを使用して Claude Code を実行し、プラグインを読み込みます。

```
claude --plugin-dir ./my-first-plugin
```

Claude Code が起動したら、新しいスキルを試してください。

```
/my-first-plugin:hello
```

Claude がグリーティングで応答します。`/help` を実行して、プラグイン名前空間の下にリストされたスキルを確認してください。

Note:

名前空間を使う理由は？ プラグインスキルは常に名前空間が付きます（`/greet:hello` など）。複数のプラグインが同じ名前のスキルを持つ場合の競合を防ぐためです。

名前空間プレフィックスを変更するには、`plugin.json` の `name` フィールドを更新してください。

Step 5: スキル引数を追加する

`$ARGUMENTS` プレースホルダーを使用してユーザー入力をキャプチャすることで、スキルを動的にします。

`SKILL.md` ファイルを更新します。

```
---  
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.
```

`/reload-plugins` を実行して変更を反映させ、スキルを名前で試してください。

```
/my-first-plugin:hello Alex
```

Claude があなたを名前で挨拶します。スキルに引数を渡す方法の詳細については、[スキル](#)を参照してください。

これらの主要なコンポーネントを使用してプラグインを正常に作成およびテストしました。

- **プラグインマニフェスト**（`.claude-plugin/plugin.json`）：プラグインのメタデータを説明します
- **スキルディレクトリ**（`skills/`）：カスタムスキルを含みます
- **スキル引数**（`$ARGUMENTS`）：動的な動作のためにユーザー入力をキャプチャします

Tip:

`--plugin-dir` フラグは開発とテストに役立ちます。プラグインを他のユーザーと共有する準備ができたなら、[プラグインマーケットプレイスを作成して配布する](#)を参照してください。

プラグイン構造の概要

スキルを使用してプラグインを作成しましたが、プラグインにはさらに多くの機能を含めることができます。カスタムエージェント、フック、MCP サーバー、LSP サーバーです。

Warning:

よくある間違い： `commands/`、`agents/`、`skills/`、`hooks/` を `.claude-plugin/` ディレクトリ内に配置しないでください。 `plugin.json` のみが `.claude-plugin/` 内に入ります。他のすべてのディレクトリはプラグインルートレベルにある必要があります。

ディレクトリ	場所	目的
<code>.claude-plugin/</code>	プラグインルート	<code>plugin.json</code> マニフェストを含みます（コンポーネントがデフォルトの場所を使用する場合はオプション）
<code>commands/</code>	プラグインルート	Markdown ファイルとしてのスキル
<code>agents/</code>	プラグインルート	カスタムエージェント定義
<code>skills/</code>	プラグインルート	<code>SKILL.md</code> ファイルを含むエージェントスキル
<code>hooks/</code>	プラグインルート	<code>hooks.json</code> のイベントハンドラー
<code>.mcp.json</code>	プラグインルート	MCP サーバー設定
<code>.lsp.json</code>	プラグインルート	コードインテリジェンス用の LSP サーバー設定
<code>settings.json</code>	プラグインルート	プラグインが有効になったときに適用されるデフォルト 設定

Note:

次のステップ：さらに多くの機能を追加する準備ができましたか？ [より複雑なプラグインを開発する](#)にジャンプして、エージェント、フック、MCP サーバー、LSP サーバーを追加してください。すべてのプラグインコンポーネントの完全な技術仕様については、[プラグインリファレンス](#)を参照してください。

より複雑なプラグインを開発する

基本的なプラグインに慣れたら、より高度な拡張機能を作成できます。

プラグインにスキルを追加する

プラグインには、Claude の機能を拡張する [エージェントスキル](#) を含めることができます。スキルはモデル呼び出し型です。Claude はタスクコンテキストに基づいて自動的にそれらを使用します。

プラグインルートに `skills/` ディレクトリを追加し、`SKILL.md` ファイルを含むスキルフォルダを追加します。

```
my-plugin/  
├── .claude-plugin/  
│   └── plugin.json  
└── skills/  
    └── code-review/  
        └── SKILL.md
```

各 `SKILL.md` には、`name` と `description` フィールドを含むフロントマターが必要で、その後に指示が続きます。

```
---  
  
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
```

プラグインをインストールした後、`/reload-plugins` を実行してスキルを読み込みます。段階的な開示とツール制限を含む完全なスキル作成ガイドンスについては、[エージェントスキル](#) を参照してください。

プラグインに LSP サーバーを追加する

Tip:

TypeScript、Python、Rust などの一般的な言語については、公式マーケットプレイスから事前構築された LSP プラグインをインストールしてください。既に対応されていない言語のサポートが必要な場合にのみ、カスタム LSP プラグインを作成してください。

LSP (Language Server Protocol) プラグインは Claude にリアルタイムコード インテリジェンスを提供します。公式 LSP プラグインがない言語をサポートする必要がある場合は、プラグインに `.lsp.json` ファイルを追加することで、独自のプラグインを作成できます。

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

プラグインをインストールするユーザーは、言語サーバーバイナリをマシンにインストールしておく必要があります。

完全な LSP 設定オプションについては、[LSP サーバー](#)を参照してください。

プラグインでデフォルト設定を配布する

プラグインは、プラグインルートに `settings.json` ファイルを含めて、プラグインが有効になったときにデフォルト設定を適用できます。現在、`agent` キーのみがサポートされています。

`agent` を設定すると、プラグインの[カスタムエージェント](#)の1つがメインスレッドとしてアクティブになり、そのシステムプロンプト、ツール制限、モデルが適用されます。これにより、プラグインは有効になったときに Claude Code の動作方法をデフォルトで変更できます。

```
{
  "agent": "security-reviewer"
}
```

この例は、プラグインの `agents/` ディレクトリで定義された `security-reviewer` エージェントをアクティブにします。`settings.json` の設定は、`plugin.json` で宣言された `settings` よりも優先されます。不明なキーは無視されます。

複雑なプラグインを整理する

多くのコンポーネントを持つプラグインの場合、ディレクトリ構造を機能別に整理してください。完全なディレクトリレイアウトと整理パターンについては、[プラグインディレクトリ構造](#)を参照してください。

プラグインをローカルでテストする

開発中にプラグインをテストするには、`--plugin-dir` フラグを使用してください。これにより、インストールを必要とせずにプラグインが直接読み込まれます。

```
claude --plugin-dir ./my-plugin
```

プラグインに変更を加えると、`/reload-plugins` を実行して再起動せずに更新を反映させます。LSP サーバー設定への変更には、完全な再起動が必要です。プラグインコンポーネントをテストします。

- `/plugin-name:skill-name` でスキルを試す
- `/agents` でエージェントが表示されることを確認する
- フックが期待どおりに機能することを確認する

Tip:

フラグを複数回指定することで、複数のプラグインを一度に読み込むことができます。

```
claude --plugin-dir ./plugin-one --plugin-dir ./plugin-two
```

プラグインの問題をデバッグする

プラグインが期待どおりに機能しない場合：

1. **構造を確認する**：ディレクトリが `.claude-plugin/` 内ではなく、プラグインルートにあることを確認してください
2. **コンポーネントを個別にテストする**：各コマンド、エージェント、フックを個別に確認してください
3. **検証とデバッグツールを使用する**：CLI コマンドとトラブルシューティング技術については、[デバッグと開発ツール](#)を参照してください

プラグインを共有する

プラグインを共有する準備ができれば：

1. **ドキュメントを追加する**：インストールと使用方法の指示を含む `README.md` を含めます
2. **プラグインをバージョン管理する**：`plugin.json` で [セマンティックバージョンニング](#) を使用します
3. **マーケットプレイスを作成または使用する**：[プラグインマーケットプレイス](#)を通じて配布してインストールします
4. **他のユーザーでテストする**：より広い配布の前に、チームメンバーにプラグインをテストしてもらいます

プラグインがマーケットプレイ스에登録されたら、他のユーザーは[プラグインを検出してインストールする](#)の指示を使用してインストールできます。

プラグインを公式マーケットプレイスに送信する

プラグインを公式 Anthropic マーケットプレイスに送信するには、アプリ内送信フォームの1つを使用してください。

- **Claude.ai**：claude.ai/settings/plugins/submit
- **Console**：platform.claude.com/plugins/submit

Note:

完全な技術仕様、デバッグ技術、配布戦略については、[プラグインリファレンス](#)を参照してください。

既存の設定をプラグインに変換する

`.claude/` ディレクトリにスキルまたはフックが既にある場合は、それらをプラグインに変換して、より簡単に共有および配布できます。

移行手順

Step 1: プラグイン構造を作成する

新しいプラグインディレクトリを作成します。

```
mkdir -p my-plugin/.claude-plugin
```

`my-plugin/.claude-plugin/plugin.json` にマニフェストファイルを作成します。

```
{
  "name": "my-plugin",
  "description": "Migrated from standalone configuration",
  "version": "1.0.0"
}
```

Step 2: 既存のファイルをコピーする

既存の設定をプラグインディレクトリにコピーします。

```
## 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: フックを移行する

設定にフックがある場合は、フックディレクトリを作成します。

```
mkdir my-plugin/hooks
```

`my-plugin/hooks/hooks.json` をフック設定で作成します。`.claude/settings.json` または `settings.local.json` から `hooks` オブジェクトをコピーします。形式は同じです。コマンドはフック入力を stdin で JSON として受け取るため、`jq` を使用してファイルパスを抽出します。


```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [{ "type": "command", "command":
"jq -r '.tool_input.file_path' | xargs npm run lint:fix" }]
      }
    ]
  }
}
```

Step 4: 移行したプラグインをテストする

プラグインを読み込んで、すべてが機能することを確認します。

```
claude --plugin-dir ./my-plugin
```

各コンポーネントをテストします。コマンドを実行し、`/agents` にエージェントが表示されることを確認し、フックが正しくトリガーされることを確認します。

移行時の変更点

スタンドアロン（ <code>.claude/</code> ）	プラグイン
1つのプロジェクトでのみ利用可能	マーケットプレイス経由で共有可能
<code>.claude/commands/</code> 内のファイル	<code>plugin-name/commands/</code> 内のファイル
<code>settings.json</code> のフック	<code>hooks/hooks.json</code> のフック
共有するには手動でコピーする必要がある	<code>/plugin install</code> でインストール

Note:

移行後、重複を避けるために `.claude/` から元のファイルを削除できます。読み込まれたときは、プラグインバージョンが優先されます。

次のステップ

Claude Code のプラグインシステムを理解したので、異なる目標のための推奨パスを以下に示します。

プラグインユーザー向け

- [プラグインを検出してインストールする](#)：マーケットプレイスを参照してプラグインをインストール
- [チームマーケットプレイスを設定する](#)：チーム用のリポジトリレベルプラグインを設定

プラグイン開発者向け

- [マーケットプレイスを作成して配布する](#)：プラグインをパッケージ化して共有
- [プラグインリファレンス](#)：完全な技術仕様
- 特定のプラグインコンポーネントをさらに詳しく調べる：
 - [スキル](#)：スキル開発の詳細
 - [サブエージェント](#)：エージェント設定と機能
 - [フック](#)：イベント処理と自動化
 - [MCP](#)：外部ツール統合

プラグインリファレンス

Claude Code プラグインシステムの完全な技術リファレンス。スキーマ、CLI コマンド、コンポーネント仕様を含みます。

Tip:

プラグインをインストールしたいですか？ [プラグインの検出とインストール](#)を参照してください。プラグインの作成については、[プラグイン](#)を参照してください。プラグインの配布については、[プラグインマーケットプレイス](#)を参照してください。

このリファレンスは、Claude Code プラグインシステムの完全な技術仕様を提供します。コンポーネントスキーマ、CLI コマンド、開発ツールを含みます。

プラグインは、Claude Code をカスタム機能で拡張する自己完結型のコンポーネントディレクトリです。プラグインコンポーネントには、skills、agents、hooks、MCP servers、LSP servers が含まれます。

プラグインコンポーネントリファレンス

Skills

プラグインは Claude Code に skills を追加し、`/name` ショートカットを作成します。これらは、あなたまたは Claude が呼び出すことができます。

場所: プラグインルートの `skills/` または `commands/` ディレクトリ

ファイル形式: Skills はディレクトリで `SKILL.md` を含みます。commands はシンプルなマークダウンファイルです。

Skill 構造:

```
skills/
├── pdf-processor/
│   ├── SKILL.md
│   ├── reference.md (optional)
│   └── scripts/ (optional)
└── code-reviewer/
    └── SKILL.md
```

統合動作:

- Skills と commands はプラグインがインストールされると自動的に検出されます
- Claude はタスクコンテキストに基づいて自動的にそれら呼び出すことができます
- Skills は SKILL.md の横にサポートファイルを含めることができます

詳細については、[Skills](#)を参照してください。

Agents

プラグインは、特定のタスク用の特化した subagents を提供できます。Claude は必要に応じて自動的にそれら呼び出すことができます。

場所: プラグインルート `agents/` ディレクトリ

ファイル形式: エージェント機能を説明するマークダウンファイル

Agent 構造:

```
---
name: agent-name
description: このエージェントが専門とする内容と、Claude がそれら呼び出すべき時期
---
```

エージェントの役割、専門知識、動作を説明する詳細なシステムプロンプト。

統合ポイント:

- Agents は `/agents` インターフェイスに表示されます
- Claude はタスクコンテキストに基づいて自動的にエージェントを呼び出すことができます
- Agents はユーザーが手動で呼び出すことができます
- プラグインエージェントは組み込みの Claude エージェントと一緒に動作します

詳細については、[Subagents](#)を参照してください。

Hooks

プラグインは Claude Code イベントに自動的に応答するイベントハンドラーを提供できます。

場所: プラグインルートの `hooks/hooks.json`、または `plugin.json` 内のインライン

形式: イベントマッチャーとアクションを含む JSON 設定

Hook 設定:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format-code.sh"
          }
        ]
      }
    ]
  }
}
```

利用可能なイベント:

- `PreToolUse`: Claude がツールを使用する前
- `PostToolUse`: Claude がツールを正常に使用した後
- `PostToolUseFailure`: Claude ツール実行が失敗した後
- `PermissionRequest`: パーミッションダイアログが表示される時
- `UserPromptSubmit`: ユーザーがプロンプトを送信するとき
- `Notification`: Claude Code が通知を送信するとき
- `Stop`: Claude が停止を試みるとき
- `SubagentStart`: subagent が開始される時
- `SubagentStop`: subagent が停止を試みるとき

- `SessionStart`: セッションの開始時
- `SessionEnd`: セッションの終了時
- `TeammateIdle`: エージェントチームのチームメイトがアイドル状態になろうとしているとき
- `TaskCompleted`: タスクが完了としてマークされているとき
- `PreCompact`: 会話履歴がコンパクト化される前

Hook タイプ:

- `command`: シェルコマンドまたはスクリプトを実行
- `prompt`: LLM でプロンプトを評価（コンテキストの `$ARGUMENTS` プレースホルダーを使用）
- `agent`: 複雑な検証タスク用のツール付き agentic verifier を実行

MCP servers

プラグインは Model Context Protocol (MCP) servers をバンドルして、Claude Code を外部ツールおよびサービスに接続できます。

場所: プラグインルートの `.mcp.json`、または `plugin.json` 内のインライン

形式: 標準 MCP サーバー設定

MCP サーバー設定:

```
{
  "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}"
    }
  }
}
```

統合動作:

- プラグイン MCP servers はプラグインが有効になると自動的に開始されます
- Servers は Claude のツールキットに標準 MCP ツールとして表示されます
- サーバー機能は Claude の既存ツールとシームレスに統合されます
- プラグインサーバーはユーザー MCP servers とは独立して設定できます

LSP servers

Tip:

LSP プラグインを使用したいですか？公式マーケットプレイスからインストールしてください。 [/plugin Discover](#) タブで「lsp」を検索してください。このセクションでは、公式マーケットプレイスでカバーされていない言語用の LSP プラグインを作成する方法を説明しています。

プラグインは [Language Server Protocol](#) (LSP) servers を提供して、Claude がコードベースで作業する際にリアルタイムコード インテリジェンスを得ることができます。

LSP 統合は以下を提供します:

- **即座の診断:** Claude は各編集後すぐにエラーと警告を確認できます
- **コードナビゲーション:** 定義へのジャンプ、参照の検索、ホバー情報

- **言語認識:** コードシンボルの型情報とドキュメント

場所: プラグインルートでの `.lsp.json`、または `plugin.json` 内のインライン

形式: 言語サーバー名をその設定にマップする JSON 設定

`.lsp.json` ファイル形式:

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

`plugin.json` 内のインライン:

```
{
  "name": "my-plugin",
  "lspServers": {
    "go": {
      "command": "gopls",
      "args": ["serve"],
      "extensionToLanguage": {
        ".go": "go"
      }
    }
  }
}
```

必須フィールド:

フィールド	説明
<code>command</code>	実行する LSP バイナリ (PATH に含まれている必要があります)

フィールド	説明
<code>extensionToLanguage</code>	ファイル拡張子を言語識別子にマップ

オプションフィールド:

フィールド	説明
<code>args</code>	LSP サーバーのコマンドライン引数
<code>transport</code>	通信トランスポート: <code>stdio</code> (デフォルト) または <code>socket</code>
<code>env</code>	サーバー起動時に設定する環境変数
<code>initializationOptions</code>	初期化中にサーバーに渡されるオプション
<code>settings</code>	<code>workspace/didChangeConfiguration</code> 経由で渡される設定
<code>workspaceFolder</code>	サーバーのワークスペースフォルダーパス
<code>startupTimeout</code>	サーバー起動を待つ最大時間 (ミリ秒)
<code>shutdownTimeout</code>	グレースフルシャットダウンを待つ最大時間 (ミリ秒)
<code>restartOnCrash</code>	サーバーがクラッシュした場合に自動的に再起動するかどうか
<code>maxRestarts</code>	諦める前の最大再起動試行回数

Warning:

言語サーバーバイナリを別途インストールする必要があります。LSP プラグインは Claude Code が言語サーバーに接続する方法を設定しますが、サーバー自体は含まれていません。 / `plugin Errors` タブに `Executable not found in $PATH` が表示される場合は、言語に必要なバイナリをインストールしてください。

利用可能な LSP プラグイン:

プラグイン	言語サーバー	インストールコマンド
<code>pyright-lsp</code>	Pyright (Python)	<code>pip install pyright</code> または <code>npm install -g pyright</code>

プラグイン	言語サーバー	インストールコマンド
<code>typescript-lsp</code>	TypeScript Language Server	<code>npm install -g typescript-language-server typescript</code>
<code>rust-lsp</code>	rust-analyzer	rust-analyzer インストールを参照

言語サーバーをまずインストールしてから、マーケットプレイスからプラグインをインストールしてください。

プラグインインストールスコープ

プラグインをインストールするときは、プラグインが利用可能な場所と他のユーザーが使用できるかどうかを決定する**スコープ**を選択します。

スコープ	設定ファイル	ユースケース
<code>user</code>	<code>~/.claude/settings.json</code>	すべてのプロジェクト全体で利用可能な個人プラグイン（デフォルト）
<code>project</code>	<code>.claude/settings.json</code>	バージョン管理経由で共有されるチームプラグイン
<code>local</code>	<code>.claude/settings.local.json</code>	プロジェクト固有のプラグイン、gitignored
<code>managed</code>	管理設定	管理プラグイン（読み取り専用、更新のみ）

プラグインは他の Claude Code 設定と同じスコープシステムを使用します。インストール手順とスコープフラグについては、[プラグインのインストール](#)を参照してください。スコープの完全な説明については、[設定スコープ](#)を参照してください。

プラグインマニフェストスキーマ

`.claude-plugin/plugin.json` ファイルはプラグインのメタデータと設定を定義します。このセクションでは、サポートされているすべてのフィールドとオプションを説明しています。

マニフェストはオプションです。省略された場合、Claude Code は [デフォルト場所](#) のコンポーネントを自動検出し、ディレクトリ名からプラグイン名を導出します。メタデータを提供するか、カスタムコンポーネントパスが必要な場合はマニフェストを使用してください。

完全なスキーマ

```
{
  "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"
}
```

必須フィールド

マニフェストを含める場合、`name` は唯一の必須フィールドです。

フィールド	型	説明	例
<code>name</code>	string	一意の識別子 (kebab-case、スペースなし)	<code>"deployment-tools"</code>

この名前はコンポーネントの名前空間に使用されます。たとえば、UI では、名前が `plugin-dev` のプラグインのエージェント `agent-creator` は `plugin-dev:agent-creator` として表示されます。

メタデータフィールド

フィールド	型	説明	例
<code>version</code>	string	セマンティックバージョン。マーケットプレイスエントリにも設定されている場合、 <code>plugin.json</code> が優先されます。1つの場所に設定するだけで済みます。	<code>"2.1.0"</code>
<code>description</code>	string	プラグインの目的の簡潔な説明	<code>"Deployment automation tools"</code>
<code>author</code>	object	著者情報	<code>{ "name": "Dev Team", "email": "dev@company.com" }</code>
<code>homepage</code>	string	ドキュメント URL	<code>"https://docs.example.com"</code>
<code>repository</code>	string	ソースコード URL	<code>"https://github.com/user/plugin"</code>
<code>license</code>	string	ライセンス識別子	<code>"MIT" 、 "Apache-2.0"</code>
<code>keywords</code>	array	検出タグ	<code>["deployment", "ci-cd"]</code>

コンポーネントパスフィールド

フィールド	型	説明	例
<code>commands</code>	string array	追加のコマンドファイル/ディレクトリ	<code>"/custom/cmd.md"</code> または <code>["./cmd1.md"]</code>
<code>agents</code>	string array	追加のエージェントファイル	<code>"/custom/agents/reviewer.md"</code>
<code>skills</code>	string array	追加のスキルディレクトリ	<code>"/custom/skills/"</code>
<code>hooks</code>	string array object	Hook 設定パスまたはインライン設定	<code>"/my-extra-hooks.json"</code>
<code>mcpServers</code>	string array object	MCP 設定パスまたはインライン設定	<code>"/my-extra-mcp-config.json"</code>
<code>outputStyles</code>	string array	追加の出力スタイルファイル/ディレクトリ	<code>"/styles/"</code>
<code>lspServers</code>	string array object	Language Server Protocol コードインテリジェンス用の設定（定義へのジャンプ、参照の検索など）	<code>"/.lsp.json"</code>

パス動作ルール

重要: カスタムパスはデフォルトディレクトリを置き換えるのではなく、補足します。

- `commands/` が存在する場合、カスタムコマンドパスに加えてロードされます
- すべてのパスはプラグインルートに相対的で、`./` で始まる必要があります
- カスタムパスからのコマンドは同じ命名と名前空間ルールを使用します
- 複数のパスを配列として指定して柔軟性を確保できます

パスの例:

```
{
  "commands": [
    "./specialized/deploy.md",
    "./utilities/batch-process.md"
  ],
  "agents": [
    "./custom-agents/reviewer.md",
    "./custom-agents/tester.md"
  ]
}
```

環境変数

`${CLAUDE_PLUGIN_ROOT}`: プラグインディレクトリへの絶対パスを含みます。hooks、MCP servers、スクリプトでこれを使用して、インストール場所に関係なく正しいパスを確保してください。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/process.sh"
          }
        ]
      }
    ]
  }
}
```

プラグインキャッシングとファイル解決

プラグインは2つの方法で指定されます:

- `claude --plugin-dir` を通じて、セッションの期間。
- マーケットプレイスを通じて、将来のセッション用にインストール。

セキュリティと検証の目的で、Claude Code は_マーケットプレイス_プラグインをユーザーのローカル**プラグインキャッシュ**（`~/.claude/plugins/cache`）にコピーします。これらを所定の場所で使用するのではなく、この動作を理解することは、外部ファイルを参照するプラグインを開発する際に重要です。

パストラバーサル制限

インストールされたプラグインはディレクトリの外側のファイルを参照できません。プラグインルートの外側をトラバースするパス（`../shared-utils` など）は、これらの外部ファイルがキャッシュにコピーされないため、インストール後は機能しません。

外部依存関係の操作

プラグインがディレクトリの外側のファイルにアクセスする必要がある場合、プラグインディレクトリ内の外部ファイルへのシンボリックリンクを作成できます。シンボリックリンクはコピープロセス中に尊重されます:

```
## プラグインディレクトリ内
ln -s /path/to/shared-utils ./shared-utils
```

シンボリックリンクされたコンテンツはプラグインキャッシュにコピーされます。これはキャッシングシステムのセキュリティ上の利点を維持しながら柔軟性を提供します。

プラグインディレクトリ構造

標準プラグインレイアウト

完全なプラグインは次の構造に従います:

```

enterprise-plugin/
├── .claude-plugin/          # メタデータディレクトリ (オプション)
│   ├── plugin.json         # プラグインマニフェスト
│   └── commands/          # デフォルトコマンド場所
│       ├── status.md
│       └── logs.md
├── agents/                 # デフォルトエージェント場所
│   ├── security-reviewer.md
│   ├── performance-tester.md
│   └── compliance-checker.md
├── skills/                 # エージェント Skills
│   ├── code-reviewer/
│   │   └── SKILL.md
│   └── pdf-processor/
│       ├── SKILL.md
│       └── scripts/
├── hooks/                  # Hook 設定
│   ├── hooks.json          # メイン hook 設定
│   └── security-hooks.json # 追加 hooks
├── settings.json           # プラグインのデフォルト設定
├── .mcp.json               # MCP サーバー定義
├── .lsp.json               # LSP サーバー設定
├── scripts/                # Hook とユーティリティスクリプト
│   ├── security-scan.sh
│   ├── format-code.py
│   └── deploy.js
├── LICENSE                 # ライセンスファイル
└── CHANGELOG.md            # バージョン履歴

```

Warning:

`.claude-plugin/` ディレクトリは `plugin.json` ファイルを含みます。他のすべてのディレクトリ (`commands/`、`agents/`、`skills/`、`hooks/`) は `.claude-plugin/` 内ではなく、プラグインルートにある必要があります。

ファイル場所リファレンス

コンポーネント	デフォルト場所	目的
マニフェスト	<code>.claude-plugin/plugin.json</code>	プラグインメタデータと設定（オプション）
コマンド	<code>commands/</code>	Skill マークダウンファイル（レガシー。新しい skills には <code>skills/</code> を使用）
エージェント	<code>agents/</code>	Subagent マークダウンファイル
Skills	<code>skills/</code>	<code><name>/SKILL.md</code> 構造の Skills
Hooks	<code>hooks/hooks.json</code>	Hook 設定
MCP servers	<code>.mcp.json</code>	MCP サーバー定義
LSP servers	<code>.lsp.json</code>	言語サーバー設定
設定	<code>settings.json</code>	プラグインが有効になったときに適用されるデフォルト設定。現在、 <code>agent</code> 設定のみがサポートされています

CLI コマンドリファレンス

Claude Code は非対話的なプラグイン管理用の CLI コマンドを提供します。スクリプトと自動化に役立ちます。

plugin install

利用可能なマーケットプレイスからプラグインをインストールします。

```
claude plugin install <plugin> [options]
```

引数:

- `<plugin>`: プラグイン名または特定のマーケットプレイス用の `plugin-name@marketplace-name`

オプション:

オプション	説明	デフォルト
<code>-s, --scope <scope></code>	インストールスコープ: <code>user</code> 、 <code>project</code> 、または <code>local</code>	<code>user</code>
<code>-h, --help</code>	コマンドのヘルプを表示	

スコープはインストールされたプラグインが追加される設定ファイルを決定します。たとえば、`--scope project` は `.claude/settings.json` の `enabledPlugins` に書き込み、プロジェクトリポジトリをクローンした全員がプラグインを利用できるようにします。

例:

```
## ユーザースコープにインストール (デフォルト)
claude plugin install formatter@my-marketplace

## プロジェクトスコープにインストール (チームと共有)
claude plugin install formatter@my-marketplace --scope project

## ローカルスコープにインストール (gitignored)
claude plugin install formatter@my-marketplace --scope local
```

plugin uninstall

インストール済みプラグインを削除します。

```
claude plugin uninstall <plugin> [options]
```

引数:

- `<plugin>`: プラグイン名または `plugin-name@marketplace-name`

オプション:

オプション	説明	デフォルト
<code>-s, --scope <scope></code>	スコープからアンインストール: <code>user</code> 、 <code>project</code> 、または <code>local</code>	<code>user</code>
<code>-h, --help</code>	コマンドのヘルプを表示	

エイリアス: `remove`、`rm`

plugin enable

無効なプラグインを有効にします。

```
claude plugin enable <plugin> [options]
```

引数:

- `<plugin>`: プラグイン名または `plugin-name@marketplace-name`

オプション:

オプション	説明	デフォルト
<code>-s, --scope <scope></code>	有効にするスコープ: <code>user</code> 、 <code>project</code> 、または <code>local</code>	<code>user</code>
<code>-h, --help</code>	コマンドのヘルプを表示	

plugin disable

プラグインをアンインストールせずに無効にします。

```
claude plugin disable <plugin> [options]
```

引数:

- `<plugin>`: プラグイン名または `plugin-name@marketplace-name`

オプション:

オプション	説明	デフォルト
<code>-s, --scope <scope></code>	無効にするスコープ: <code>user</code> 、 <code>project</code> 、または <code>local</code>	<code>user</code>

オプション	説明	デフォルト
<code>-h, --help</code>	コマンドのヘルプを表示	

plugin update

プラグインを最新バージョンに更新します。

```
claude plugin update <plugin> [options]
```

引数:

- `<plugin>`: プラグイン名または `plugin-name@marketplace-name`

オプション:

オプション	説明	デフォルト
<code>-s, --scope <scope></code>	更新するスコープ: <code>user</code> 、 <code>project</code> 、 <code>local</code> 、または <code>managed</code>	<code>user</code>
<code>-h, --help</code>	コマンドのヘルプを表示	

デバッグと開発ツール

デバッグコマンド

`claude --debug`（または TUI 内の `/debug`）を使用してプラグイン読み込みの詳細を確認します:

これは以下を表示します:

- どのプラグインが読み込まれているか
- プラグインマニフェストのエラー
- コマンド、エージェント、hook 登録
- MCP サーバー初期化

一般的な問題

問題	原因	解決策
プラグインが読み込まれない	無効な <code>plugin.json</code>	<code>claude plugin validate</code> または <code>/plugin validate</code> で JSON 構文を検証
コマンドが表示されない	ディレクトリ構造が間違っている	<code>commands/</code> がルートにあることを確認。 <code>.claude-plugin/</code> 内ではない
Hooks が発火しない	スクリプトが実行可能でない	<code>chmod +x script.sh</code> を実行
MCP サーバーが失敗	<code>\${CLAUDE_PLUGIN_ROOT}</code> が不足	すべてのプラグインパスに変数を使用
パスエラー	絶対パスが使用されている	すべてのパスは相対的で <code>.</code> <code>/</code> で始まる必要があります
LSP Executable not found in \$PATH	言語サーバーがインストールされていない	バイナリをインストール (例: <code>npm install -g typescript-language-server typescript</code>)

エラーメッセージの例

マニフェスト検証エラー:

- `Invalid JSON syntax: Unexpected token }` in JSON at position 142: コンマの欠落、余分なコンマ、またはクォートされていない文字列を確認
- `Plugin has an invalid manifest file at .claude-plugin/plugin.json. Validation errors: name: Required:` 必須フィールドが不足
- `Plugin has a corrupt manifest file at .claude-plugin/plugin.json. JSON parse error: ...:` JSON 構文エラー

プラグイン読み込みエラー:

- `Warning: No commands found in plugin my-plugin custom directory: ./cmds. Expected .md files or SKILL.md in subdirectories.`: コマンドパスが存在するが有効なコマンドファイルが含まれていない

- Plugin directory not found at path: `./plugins/my-plugin`. Check that the marketplace entry has the correct path.: marketplace.json の `source` パスが存在しないディレクトリを指している
- Plugin my-plugin has conflicting manifests: both plugin.json and marketplace entry specify components.: 重複するコンポーネント定義を削除するか、marketplace エントリから `strict: false` を削除

Hook トラブルシューティング

Hook スクリプトが実行されない:

1. スクリプトが実行可能であることを確認: `chmod +x ./scripts/your-script.sh`
2. shebang 行を確認: 最初の行は `#!/bin/bash` または `#!/usr/bin/env bash` である必要があります
3. パスが `${CLAUDE_PLUGIN_ROOT}` を使用していることを確認: `"command": "${CLAUDE_PLUGIN_ROOT}/scripts/your-script.sh"`
4. スクリプトを手動でテスト: `./scripts/your-script.sh`

Hook が予期されたイベントでトリガーされない:

1. イベント名が正しいことを確認 (大文字小文字を区別): `PostToolUse`、`postToolUse` ではない
2. マッチャーパターンがツールと一致することを確認: ファイル操作の場合 `"matcher": "Write|Edit"`
3. hook タイプが有効であることを確認: `command`、`prompt`、または `agent`

MCP サーバートラブルシューティング

サーバーが起動しない:

1. コマンドが存在し、実行可能であることを確認
2. すべてのパスが `${CLAUDE_PLUGIN_ROOT}` 変数を使用していることを確認
3. MCP サーバログを確認: `claude --debug` は初期化エラーを表示
4. Claude Code の外部でサーバーを手動でテスト

サーバーツールが表示されない:

1. サーバーが `.mcp.json` または `plugin.json` で正しく設定されていることを確認
2. サーバーが MCP プロトコルを正しく実装していることを確認
3. デバッグ出力で接続タイムアウトを確認

ディレクトリ構造の違い

症状: プラグインは読み込まれるがコンポーネント（コマンド、エージェント、hooks）が不足している。

正しい構造: コンポーネントはプラグインルートにある必要があり、`.claude-plugin/` 内ではありません。`.claude-plugin/` には `plugin.json` のみが属します。

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json    ← マニフェストのみここ
├── commands/         ← ルートレベル
├── agents/            ← ルートレベル
└── hooks/             ← ルートレベル
```

コンポーネントが `.claude-plugin/` 内にある場合は、プラグインルートに移動してください。

デバッグチェックリスト:

1. `claude --debug` を実行し、「loading plugin」メッセージを探す
2. 各コンポーネントディレクトリがデバッグ出力にリストされていることを確認
3. プラグインファイルを読み取ることができるファイルパーミッションを確認

配布とバージョン管理リファレンス

バージョン管理

プラグインリリースにはセマンティックバージョンングに従ってください:

```
{
  "name": "my-plugin",
  "version": "2.1.0"
}
```

バージョン形式: MAJOR.MINOR.PATCH

- **MAJOR:** 破壊的変更（互換性のない API 変更）
- **MINOR:** 新機能（後方互換性のある追加）
- **PATCH:** バグ修正（後方互換性のある修正）

ベストプラクティス:

- 最初の安定リリースは `1.0.0` から開始
- 変更を配布する前に `plugin.json` のバージョンを更新
- `CHANGELOG.md` ファイルで変更を文書化
- テスト用に `2.0.0-beta.1` のようなプレリリースバージョンを使用

Warning:

Claude Code はバージョンを使用してプラグインを更新するかどうかを判断します。プラグインのコードを変更しても `plugin.json` のバージョンをバンプしない場合、キャッシングのため既存ユーザーは変更を確認できません。

プラグインが [マーケットプレイス](#) ディレクトリ内にある場合、代わりに `marketplace.json` を通じてバージョンを管理でき、`plugin.json` から `version` フィールドを省略できます。

関連項目

- [プラグイン](#) - チュートリアルと実践的な使用法
- [プラグインマーケットプレイス](#) - マーケットプレイスの作成と管理
- [Skills](#) - Skill 開発の詳細
- [Subagents](#) - エージェント設定と機能
- [Hooks](#) - イベント処理と自動化
- [MCP](#) - 外部ツール統合
- [設定](#) - プラグインの設定オプション

プラグインマーケットプレイスの作成と配布

Claude Code 拡張機能を配布するためのプラグインマーケットプレイスを構築およびホストします。

プラグインマーケットプレイスは、他のユーザーにプラグインを配布できるカタログです。マーケットプレイスは、一元化された検出、バージョン追跡、自動更新、および複数のソースタイプ（Git リポジトリ、ローカルパス、その他）のサポートを提供します。このガイドでは、チームやコミュニティとプラグインを共有するための独自のマーケットプレイスを作成する方法を説明します。

既存のマーケットプレイスからプラグインをインストールしたいですか？ [既成プラグインの検出とインストール](#)を参照してください。

概要

マーケットプレイスの作成と配布には、以下が含まれます。

1. **プラグインの作成**：コマンド、エージェント、hooks、MCP サーバー、または LSP サーバーを使用して 1 つ以上のプラグインを構築します。このガイドでは、配布するプラグインが既にあることを前提としています。プラグインの作成方法の詳細については、[プラグインの作成](#)を参照してください。
2. **マーケットプレイスファイルの作成**：プラグインとその場所を一覧表示する `marketplace.json` を定義します（[マーケットプレイスファイルの作成](#)を参照）。
3. **マーケットプレイスのホスト**：GitHub、GitLab、または別の Git ホストにプッシュします（[マーケットプレイスのホストと配布](#)を参照）。
4. **ユーザーと共有**：ユーザーが `/plugin marketplace add` でマーケットプレイスを追加し、個別のプラグインをインストールします（[プラグインの検出とインストール](#)を参照）。

マーケットプレイスがライブになったら、リポジトリに変更をプッシュして更新できます。ユーザーは `/plugin marketplace update` でローカルコピーを更新します。

チュートリアル：ローカルマーケットプレイスの作成

この例では、1 つのプラグイン（コードレビュー用の `/quality-review` スキル）を含むマーケットプレイスを作成します。ディレクトリ構造を作成し、スキルを追加し、プラグインマニフェストとマーケットプレイスカatalogを作成してから、インストールしてテストします。

Step 1: ディレクトリ構造の作成

```
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: スキルの作成

`/quality-review` スキルが何をするかを定義する `SKILL.md` ファイルを作成します。

```
---
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: プラグインマニフェストの作成

プラグインを説明する `plugin.json` ファイルを作成します。マニフェストは `.claude-plugin/` ディレクトリに配置されます。

```
{
  "name": "quality-review-plugin",
  "description": "Adds a /quality-review skill for quick code reviews",
  "version": "1.0.0"
}
```

Step 4: マーケットプレイスファイルの作成

プラグインを一覧表示するマーケットプレイスカタログを作成します。

```
{
  "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: 追加とインストール

マーケットプレイスを追加し、プラグインをインストールします。

```
/plugin marketplace add ./my-marketplace
/plugin install quality-review-plugin@my-plugins
```

Step 6: 試してみる

エディタでコードを選択し、新しいコマンドを実行します。

```
/review
```

プラグインが実行できることの詳細（hooks、エージェント、MCP サーバー、LSP サーバーを含む）については、[プラグイン](#)を参照してください。

Note:

プラグインのインストール方法：ユーザーがプラグインをインストールすると、Claude Code はプラグインディレクトリをキャッシュロケーションにコピーします。これは、`../shared-utils` のようなパスを使用してプラグインディレクトリの外部のファイルを参照できないことを意味します。これらのファイルはコピーされないためです。

プラグイン間でファイルを共有する必要がある場合は、シンボリックリンクを使用します（コピー中にフォローされます）。詳細については、[プラグインキャッシングとファイル解決](#)を参照してください。

マーケットプレイスファイルの作成

リポジトリルートに `.claude-plugin/marketplace.json` を作成します。このファイルは、マーケットプレイスの名前、所有者情報、およびソースを含むプラグインのリストを定義します。

各プラグインエントリには、最低限 `name` と `source`（取得元）が必要です。利用可能なすべてのフィールドについては、以下の[完全なスキーマ](#)を参照してください。

```
{
  "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"
    }
  ]
}
```

マーケットプレイススキーマ

必須フィールド

フィールド	タイプ	説明	例
<code>name</code>	string	マーケットプレイス識別子（ケバブケース、スペースなし）。これは公開向けです。ユーザーはプラグインをインストールするときに表示されます（例： <code>/plugin install my-tool@your-marketplace</code> ）。	<code>"acme-tools"</code>
<code>owner</code>	object	マーケットプレイスメンテナー情報（ 以下のフィールドを参照 ）	
<code>plugins</code>	array	利用可能なプラグインのリスト	以下を参照

Note:

予約名：以下のマーケットプレイス名は Anthropic の公式使用のために予約されており、サードパーティのマーケットプレイスでは使用できません。 `claude-code-marketplace` 、 `claude-code-plugins` 、 `claude-plugins-official` 、 `anthropic-marketplace` 、 `anthropic-plugins` 、 `agent-skills` 、 `life-sciences` 。公式マーケットプレイスになりますしている名前（ `official-claude-plugins` や `anthropic-tools-v2` など）もブロックされています。

所有者フィールド

フィールド	タイプ	必須	説明
<code>name</code>	string	はい	メンテナーまたはチームの名前
<code>email</code>	string	いいえ	メンテナーの連絡先メール

オプションメタデータ

フィールド	タイプ	説明
<code>metadata.description</code>	string	マーケットプレイスの簡潔な説明
<code>metadata.version</code>	string	マーケットプレイスバージョン
<code>metadata.pluginRoot</code>	string	相対プラグインソースパスの前に付加される基本ディレクトリ（例： <code>./plugins</code> ）を使用すると、 <code>"source": "./plugins/formatter"</code> の代わりに <code>"source": "formatter"</code> と記述できます）

プラグインエントリ

`plugins` 配列内の各プラグインエントリは、プラグインとその場所を説明します。[プラグインマニフェストスキーマ](#)のフィールド（`description`、`version`、`author`、`commands`、`hooks` など）を含めることができます。さらに、これらのマーケットプレイス固有のフィールド：`source`、`category`、`tags`、`strict` があります。

必須フィールド

フィールド	タイプ	説明
<code>name</code>	string	プラグイン識別子（ケバブケース、スペースなし）。これは公開向けです。ユーザーはインストール時に表示されます (例： <code>/plugin install my-plugin@marketplace</code>)。
<code>source</code>	string object	プラグインを取得する場所（以下の プラグインソース を参照）

オプションプラグインフィールド

標準メタデータフィールド：

フィールド	タイプ	説明
<code>description</code>	string	プラグインの簡潔な説明
<code>version</code>	string	プラグインバージョン
<code>author</code>	object	プラグイン作成者情報（ <code>name</code> は必須、 <code>email</code> はオプション）
<code>homepage</code>	string	プラグインホームページまたはドキュメント URL
<code>repository</code>	string	ソースコードリポジトリ URL
<code>license</code>	string	SPDX ライセンス識別子（例：MIT、Apache-2.0）
<code>keywords</code>	array	プラグイン検出と分類用のタグ
<code>category</code>	string	整理用のプラグインカテゴリ
<code>tags</code>	array	検索可能性用のタグ
<code>strict</code>	boolean	<code>plugin.json</code> がコンポーネント定義の権限であるかどうかを制御します（デフォルト：true）。以下の 厳密モード を参照してください。

コンポーネント設定フィールド：

フィールド	タイプ	説明
<code>commands</code>	string array	コマンドファイルまたはディレクトリへのカスタムパス

フィールド	タイプ	説明
<code>agents</code>	string array	エージェントファイルへのカスタムパス
<code>hooks</code>	string object	カスタム hooks 設定または hooks ファイルへのパス
<code>mcpServers</code>	string object	MCP サーバー設定または MCP 設定ファイルへのパス
<code>lspServers</code>	string object	LSP サーバー設定または LSP 設定ファイルへのパス

プラグインソース

プラグインソースは、マーケットプレイスに一覧表示されている各個別プラグインを取得する場所を Claude Code に指示します。これらは `marketplace.json` 内の各プラグインエントリの `source` フィールドで設定されます。

プラグインがローカルマシンにクローンまたはコピーされると、`~/.claude/plugins/cache` のローカルバージョン管理プラグインキャッシュにコピーされます。

ソース	タイプ	フィールド	注記
相対パス	<code>string</code> (例: <code>"./my-plugin"</code>)	—	マーケットプレイスリポジトリ内のローカルディレクトリ。 <code>./</code> で始まる必要があります
<code>github</code>	object	<code>repo</code> 、 <code>ref?</code> 、 <code>sha?</code>	
<code>url</code>	object	<code>url</code> (<code>.git</code> で終わる必要があります)、 <code>ref?</code> 、 <code>sha?</code>	Git URL ソース

ソース	タイプ	フィールド	注記
<code>git-subdir</code>	object	<code>url</code> 、 <code>path</code> 、 <code>ref?</code> 、 <code>sha?</code>	Git リポジトリ内のサブディレクトリ。帯域幅を最小化するためにスパースクローンします
<code>npm</code>	object	<code>package</code> 、 <code>version?</code> 、 <code>registry?</code>	<code>npm install</code> でインストール
<code>pip</code>	object	<code>package</code> 、 <code>version?</code> 、 <code>registry?</code>	pip でインストール

Note:

マーケットプレイスソースとプラグインソース：これらは異なる概念で、異なるものを制御します。

- ・ **マーケットプレイスソース** — `marketplace.json` カタログ自体を取得する場所。ユーザーが `/plugin marketplace add` を実行するか、`extraKnownMarketplaces` 設定で設定されます。`ref` (ブランチ/タグ) をサポートしますが、`sha` はサポートしません。
- ・ **プラグインソース** — マーケットプレイスに一覧表示されている個別プラグインを取得する場所。`marketplace.json` 内の各プラグインエントリの `source` フィールドで設定されます。`ref` (ブランチ/タグ) と `sha` (正確なコミット) の両方をサポートします。

例えば、`acme-corp/plugin-catalog` (マーケットプレイスソース) でホストされているマーケットプレイスは、`acme-corp/code-formatter` (プラグインソース) から取得されたプラグインを一覧表示できます。マーケットプレイスソースとプラグインソースは異なるリポジトリを指し、独立して固定されます。

相対パス

同じリポジトリ内のプラグインの場合：

```
{
  "name": "my-plugin",
  "source": "../plugins/my-plugin"
}
```

Note:

相対パスは、ユーザーが Git（GitHub、GitLab、または Git URL）経由でマーケットプレイスを追加する場合にのみ機能します。ユーザーが `marketplace.json` ファイルへの直接 URL でマーケットプレイスを追加する場合、相対パスは正しく解決されません。URL ベースの配布の場合は、GitHub、npm、または Git URL ソースを使用してください。詳細については、[トラブルシューティング](#)を参照してください。

GitHub リポジトリ

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo"
  }
}
```

特定のブランチ、タグ、またはコミットに固定できます。

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

フィールド	タイプ	説明
<code>repo</code>	string	必須。 <code>owner/repo</code> 形式の GitHub リポジトリ
<code>ref</code>	string	オプション。Git ブランチまたはタグ（デフォルトはリポジトリのデフォルトブランチ）

フィールド	タイプ	説明
<code>sha</code>	string	オプション。完全な 40 文字の Git コミット SHA で正確なバージョンに固定

Git リポジトリ

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git"
  }
}
```

特定のブランチ、タグ、またはコミットに固定できます。

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git",
    "ref": "main",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

フィールド	タイプ	説明
<code>url</code>	string	必須。完全な Git リポジトリ URL (<code>.git</code> で終わる必要があります)
<code>ref</code>	string	オプション。Git ブランチまたはタグ (デフォルトはリポジトリのデフォルトブランチ)

フィールド	タイプ	説明
<code>sha</code>	string	オプション。完全な 40 文字の Git コミット SHA で正確なバージョンに固定

Git サブディレクトリ

`git-subdir` を使用して、Git リポジトリのサブディレクトリ内に存在するプラグインを指します。Claude Code はスパースな部分クローンを使用してサブディレクトリのみを取得し、大規模なモノレポの帯域幅を最小化します。

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin"
  }
}
```

特定のブランチ、タグ、またはコミットに固定できます。

```
{
  "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"
  }
}
```

`url` フィールドは、GitHub ショートハンド（`owner/repo`）または SSH URL（`git@github.com:owner/repo.git`）も受け入れます。

フィールド	タイプ	説明
<code>url</code>	string	必須。Git リポジトリ URL、GitHub <code>owner/repo</code> ショートハンド、または SSH URL
<code>path</code>	string	必須。プラグインを含むリポジトリ内のサブディレクトリパス（例： <code>"tools/claude-plugin"</code> ）
<code>ref</code>	string	オプション。Git ブランチまたはタグ（デフォルトはリポジトリのデフォルトブランチ）
<code>sha</code>	string	オプション。完全な 40 文字の Git コミット SHA で正確なバージョンに固定

npm パッケージ

npm パッケージとして配布されるプラグインは、`npm install` を使用してインストールされます。これは、公開 npm レジストリまたはチームがホストするプライベートレジストリ上の任意のパッケージで機能します。

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin"
  }
}
```

特定のバージョンに固定するには、`version` フィールドを追加します。

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "2.1.0"
  }
}
```

プライベートまたは内部レジストリからインストールするには、`registry` フィールドを追加します。

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "^2.0.0",
    "registry": "https://npm.example.com"
  }
}
```

フィールド	タイプ	説明
<code>package</code>	string	必須。パッケージ名またはスコープ付きパッケージ（例： <code>@org/plugin</code> ）
<code>version</code>	string	オプション。バージョンまたはバージョン範囲（例： <code>2.1.0</code> 、 <code>^2.0.0</code> 、 <code>~1.5.0</code> ）
<code>registry</code>	string	オプション。カスタム npm レジストリ URL。デフォルトはシステム npm レジストリ（通常は <code>npmjs.org</code> ）

高度なプラグインエントリ

この例は、コマンド、エージェント、hooks、MCP サーバーのカスタムパスを含む、多くのオプションフィールドを使用するプラグインエントリを示しています。


```

{
  "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
}
```

注目すべき重要な点：

- **commands** と **agents**：複数のディレクトリまたは個別のファイルを指定できます。パスはプラグインルートに相対的です。
- **\${CLAUDE_PLUGIN_ROOT}**：hooks と MCP サーバー設定でこの変数を使用して、プラグインのインストールディレクトリ内のファイルを参照します。プラグインはインストール時にキャッシュロケーションにコピーされるため、これは必要です。
- **strict: false**：これが false に設定されているため、プラグインは独自の **plugin.json** を必要としません。マーケットプレイスエントリがすべてを定義します。以下の[厳密モード](#)を参照してください。

厳密モード

strict フィールドは、**plugin.json** がコンポーネント定義（コマンド、エージェント、hooks、スキル、MCP サーバー、出力スタイル）の権限であるかどうかを制御します。

値	動作
true (デフォルト)	plugin.json が権限です。マーケットプレイスエントリは追加のコンポーネントで補足でき、両方のソースがマージされます。
false	マーケットプレイスエントリが完全な定義です。プラグインに plugin.json があってコンポーネントを宣言している場合、それは競合であり、プラグインは読み込みに失敗します。

各モードを使用する場合：

- **strict: true**：プラグインは独自の **plugin.json** を持ち、独自のコンポーネントを管理します。マーケットプレイスエントリは上に追加のコマンドまたは hooks を追加できます。これはデフォルトで、ほとんどのプラグインで機能します。
- **strict: false**：マーケットプレイスオペレーターが完全に制御したい場合。プラグインリポジトリは生ファイルを提供し、マーケットプレイスエントリはそれらのファイルのどれがコマンド、エージェント、hooks などとして公開されるかを定義しま

す。マーケットプレイスがプラグイン作成者の意図と異なる方法でプラグインのコンポーネントを再構成またはキュレートする場合に便利です。

マーケットプレイスのホストと配布

GitHub でホスト（推奨）

GitHub は最も簡単な配布方法を提供します。

1. **リポジトリを作成**：マーケットプレイス用の新しいリポジトリを設定します
2. **マーケットプレイスファイルを追加**：プラグイン定義を含む `.claude-plugin/marketplace.json` を作成します
3. **チームと共有**：ユーザーが `/plugin marketplace add owner/repo` でマーケットプレイスを追加します

メリット：組み込みバージョン管理、問題追跡、チームコラボレーション機能。

他の Git サービスでホスト

GitLab、Bitbucket、自己ホスト型サーバーなど、任意の Git ホスティングサービスが機能します。ユーザーは完全なリポジトリ URL で追加します。

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

プライベートリポジトリ

Claude Code は、プライベートリポジトリからプラグインをインストールすることをサポートしています。手動インストールと更新の場合、Claude Code は既存の Git 認証情報ヘルパーを使用します。ターミナルでプライベートリポジトリに対して `git clone` が機能する場合、Claude Code でも機能します。一般的な認証情報ヘルパーには、GitHub の `gh auth login`、macOS キーチェーン、`git-credential-store` が含まれます。

バックグラウンド自動更新は、認証情報ヘルパーなしで起動時に実行されます。これは、対話的なプロンプトが Claude Code の起動をブロックするためです。プライベートマーケットプレイスの自動更新を有効にするには、環境に適切な認証トークンを設定します。

プロバイダー	環境変数	注記
GitHub	<code>GITHUB_TOKEN</code> または <code>GH_TOKEN</code>	個人用アクセストークンまたは GitHub App トークン
GitLab	<code>GITLAB_TOKEN</code> または <code>GL_TOKEN</code>	個人用アクセストークンまたはプロジェクトトークン

プロバイダー	環境変数	注記
Bitbucket	<code>BITBUCKET_TOKEN</code>	アプリパスワードまたはリポジトリアクセストークン

シェル設定（例： `.bashrc` 、 `.zshrc` ）でトークンを設定するか、 Claude Code を実行するときに渡します。

```
export GITHUB_TOKEN=ghp_XXXXXXXXXXXXXXXXXXXX
```

Note:

CI/CD 環境の場合、トークンをシークレット環境変数として設定します。GitHub Actions は同じ組織内のリポジトリに対して `GITHUB_TOKEN` を自動的に提供します。

配布前にローカルでテスト

共有する前にマーケットプレイスをローカルでテストします。

```
/plugin marketplace add ./my-local-marketplace
/plugin install test-plugin@my-local-marketplace
```

add コマンドの完全な範囲（GitHub、Git URL、ローカルパス、リモート URL）については、 [マーケットプレイスの追加](#) を参照してください。

チーム向けマーケットプレイスの要求

リポジトリを設定して、チームメンバーがプロジェクトフォルダを信頼するときにマーケットプレイスをインストールするよう自動的に促されるようにできます。マーケットプレイスを `.claude/settings.json` に追加します。

```
{
  "extraKnownMarketplaces": {
    "company-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

デフォルトで有効にするプラグインを指定することもできます。

```
{
  "enabledPlugins": {
    "code-formatter@company-tools": true,
    "deployment-tools@company-tools": true
  }
}
```

完全な設定オプションについては、[プラグイン設定](#)を参照してください。

管理マーケットプレイスの制限

プラグインソースを厳密に制御する必要がある組織の場合、管理者は管理設定の `strictKnownMarketplaces` 設定を使用して、ユーザーが追加できるプラグインマーケットプレイスを制限できます。

`strictKnownMarketplaces` が管理設定で設定されている場合、制限動作は値によって異なります。

値	動作
未定義（デフォルト）	制限なし。ユーザーは任意のマーケットプレイスを追加できます
空配列 <code>[]</code>	完全なロックダウン。ユーザーは新しいマーケットプレイスを追加できません

値	動作
ソースのリスト	ユーザーはホワイトリストと正確に一致するマーケットプレイスのみを追加できます

一般的な設定

すべてのマーケットプレイス追加を無効にする：

```
{
  "strictKnownMarketplaces": []
}
```

特定のマーケットプレイスのみを許可する：

```
{
  "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"
    }
  ]
}
```

ホストの正規表現パターンマッチングを使用して、内部 Git サーバーからのすべてのマーケットプレイスを許可する：

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

パスの正規表現パターンマッチングを使用して、特定のディレクトリからのファイルシステムベースのマーケットプレイスを許可する：

```
{
  "strictKnownMarketplaces": [
    {
      "source": "pathPattern",
      "pathPattern": "^/opt/approved/"
    }
  ]
}
```

`pathPattern` として `".*"` を使用して、ネットワークソースを `hostPattern` で制御しながら、任意のファイルシステムパスを許可します。

制限の仕組み

制限はプラグインインストールプロセスの早い段階で検証され、ネットワークリクエストやファイルシステム操作が発生する前に検証されます。これにより、不正なマーケットプレイスアクセスの試みを防ぎます。

ホワイトリストはほとんどのソースタイプに対して正確なマッチングを使用します。マーケットプレイスが許可されるには、指定されたすべてのフィールドが正確に一致する必要があります。

- GitHub ソースの場合： `repo` は必須で、ホワイトリストで指定されている場合は `ref` または `path` も一致する必要があります
- URL ソースの場合：完全な URL が正確に一致する必要があります
- `hostPattern` ソースの場合：マーケットプレイスホストが正規表現パターンと照合されます

- `pathPattern` ソースの場合：マーケットプレイスのファイルシステムパスが正規表現パターンと照合されます

`strictKnownMarketplaces` は管理設定で設定されるため、個別のユーザーとプロジェクト設定はこれらの制限をオーバーライドできません。

完全な設定詳細（サポートされているすべてのソースタイプと `extraKnownMarketplaces` との比較を含む）については、[strictKnownMarketplaces リファレンス](#)を参照してください。

バージョン解決とリリースチャネル

プラグインバージョンはキャッシュパスと更新検出を決定します。プラグインマニフェスト（`plugin.json`）またはマーケットプレイスエントリ（`marketplace.json`）でバージョンを指定できます。

Warning:

可能な限り、両方の場所でバージョンを設定することを避けてください。プラグインマニフェストは常に無言で優先されるため、マーケットプレイスバージョンが無視される可能性があります。相対パスプラグインの場合、マーケットプレイスエントリでバージョンを設定します。他のすべてのプラグインソースの場合、プラグインマニフェストで設定します。

リリースチャネルの設定

プラグインの「安定」と「最新」リリースチャネルをサポートするには、同じリポジトリの異なる ref または SHA を指す2つのマーケットプレイスを設定できます。その後、[管理設定](#)を通じて2つのマーケットプレイスを異なるユーザーグループに割り当てることができます。

Warning:

プラグインの `plugin.json` は、各固定 ref またはコミットで異なる `version` を宣言する必要があります。2つの ref またはコミットが同じマニフェストバージョンを持つ場合、Claude Code はそれらを同一として扱い、更新をスキップします。

例

```
{
  "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"
      }
    }
  ]
}
```

チャンネルをユーザーグループに割り当てる

管理設定を通じて各マーケットプレイスを適切なユーザーグループに割り当てます。例えば、安定グループは以下を受け取ります。

```
{
  "extraKnownMarketplaces": {
    "stable-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/stable-tools"
      }
    }
  }
}
```

早期アクセスグループは代わりに `latest-tools` を受け取ります。

```
{
  "extraKnownMarketplaces": {
    "latest-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/latest-tools"
      }
    }
  }
}
```

検証とテスト

共有する前にマーケットプレイスをテストします。

マーケットプレイス JSON 構文を検証します。

```
claude plugin validate .
```

または Claude Code 内から：

```
/plugin validate .
```

テスト用にマーケットプレイスを追加します。

```
/plugin marketplace add ./path/to/marketplace
```

すべてが機能することを確認するためにテストプラグインをインストールします。

```
/plugin install test-plugin@marketplace-name
```

完全なプラグインテストワークフローについては、[プラグインをローカルでテスト](#)を参照してください。技術的なトラブルシューティングについては、[プラグインリファレンス](#)を参照してください。

トラブルシューティング

マーケットプレイスが読み込まれない

症状：マーケットプレイスを追加できない、またはそこからプラグインが表示されない

解決策：

- ・マーケットプレイス URL がアクセス可能であることを確認します
- ・`.claude-plugin/marketplace.json` が指定されたパスに存在することを確認します
- ・`claude plugin validate` または `/plugin validate` を使用して JSON 構文が有効であることを確認します
- ・プライベートリポジトリの場合、アクセス権限があることを確認します

マーケットプレイス検証エラー

マーケットプレイスディレクトリから `claude plugin validate .` または `/plugin validate .` を実行して、問題をチェックします。一般的なエラー：

エラー	原因	解決策
<code>File not found: .claude-plugin/marketplace.json</code>	マニフェストが見つかりません	必須フィールドを含む <code>.claude-plugin/marketplace.json</code> を作成します
<code>Invalid JSON syntax: Unexpected token...</code>	JSON 構文エラー	コンマの欠落、余分なコンマ、または引用符なしの文字列をチェックします

エラー	原因	解決策
<code>Duplicate plugin name "x" found in marketplace</code>	2つのプラグインが同じ名前を共有しています	各プラグインに一意的な <code>name</code> 値を指定します
<code>plugins[0].source: Path traversal not allowed</code>	ソースパスに <code>..</code> が含まれています	マーケットプレイスルートに相対的なパスを使用し、 <code>..</code> なしで使します

警告（ブロックなし）：

- `Marketplace has no plugins defined` : `plugins` 配列に少なくとも1つのプラグインを追加します
- `No marketplace description provided` : ユーザーがマーケットプレイスを理解するのに役立つように `metadata.description` を追加します

プラグインインストール失敗

症状：マーケットプレイスは表示されますが、プラグインインストールが失敗します

解決策：

- プラグインソース URL がアクセス可能であることを確認します
- プラグインディレクトリに必須ファイルが含まれていることを確認します
- GitHub ソースの場合、リポジトリが公開されているか、アクセス権限があることを確認します
- プラグインソースを手動でクローン/ダウンロードしてテストします

プライベートリポジトリ認証が失敗する

症状：プライベートリポジトリからプラグインをインストールするときに認証エラーが発生します

解決策：

手動インストールと更新の場合：

- Git プロバイダーで認証されていることを確認します（例：GitHub の場合は `gh auth status` を実行）
- 認証情報ヘルパーが正しく設定されていることを確認します：`git config --global credential.helper`
- リポジトリを手動でクローンして、認証情報が機能することを確認します

バックグラウンド自動更新の場合：

- ・環境でトークンが設定されていることを確認します：`echo $GITHUB_TOKEN`
- ・トークンに必要な権限があることを確認します（リポジトリへの読み取りアクセス）
- ・GitHub の場合、トークンがプライベートリポジトリの `repo` スcopeを持つことを確認します
- ・GitLab の場合、トークンが少なくとも `read_repository` スcopeを持つことを確認します
- ・トークンが期限切れになっていないことを確認します

Git 操作がタイムアウトする

症状：プラグインインストールまたはマーケットプレイス更新が「Git clone timed out after 120s」または「Git pull timed out after 120s」などのタイムアウトエラーで失敗します。

原因：Claude Code は、プラグインリポジトリのクローンやマーケットプレイス更新のプルを含む、すべての Git 操作に 120 秒のタイムアウトを使用します。大規模なリポジトリまたは遅いネットワーク接続がこの制限を超える可能性があります。

解決策：`CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS` 環境変数を使用してタイムアウトを増やします。値はミリ秒単位です。

```
export CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS=300000 # 5 minutes
```

相対パスを持つプラグインが URL ベースのマーケットプレイスで失敗する

症状：URL（`https://example.com/marketplace.json` など）経由でマーケットプレイスを追加しましたが、`./plugins/my-plugin` のような相対パスソースを持つプラグインが「path not found」エラーでインストールに失敗します。

原因：URL ベースのマーケットプレイスは `marketplace.json` ファイル自体のみをダウンロードします。サーバーからプラグインファイルをダウンロードしません。マーケットプレイスエントリの相対パスは、ダウンロードされなかったリモートサーバー上のファイルを参照します。

解決策：

- ・**外部ソースを使用：**プラグインエントリを相対パスの代わりに GitHub、npm、または Git URL ソースを使用するように変更します。

```
{ "name": "my-plugin", "source": { "source": "github", "repo": "owner/repo" } }
```

- **Git ベースのマーケットプレイスを使用**：マーケットプレイスを Git リポジトリでホストし、Git URL で追加します。Git ベースのマーケットプレイスはリポジトリ全体をクローンするため、相対パスが正しく機能します。

インストール後にファイルが見つからない

症状：プラグインはインストールされますが、ファイルへの参照が失敗します。特に、プラグインディレクトリの外部のファイル

原因：プラグインはインプレイスで使用されるのではなく、キャッシュディレクトリにコピーされます。プラグインディレクトリの外部のファイルを参照するパス（`../shared-utils` など）は、それらのファイルがコピーされないため機能しません。

解決策：シンボリックリンクとディレクトリ再構成を含む回避策については、[プラグインキャッシングとファイル解決](#)を参照してください。

追加のデバッグツールと一般的な問題については、[デバッグと開発ツール](#)を参照してください。

関連項目

- [既存プラグインの検出とインストール](#) - 既存のマーケットプレイスからプラグインをインストール
- [プラグイン](#) - 独自のプラグインの作成
- [プラグインリファレンス](#) - 完全な技術仕様とスキーマ
- [プラグイン設定](#) - プラグイン設定オプション
- [strictKnownMarketplaces リファレンス](#) - 管理マーケットプレイス制限

マーケットプレイスから事前構築されたプラグインを発見してインストールする

マーケットプレイスからプラグインを検索してインストールし、Claude Code を新しいコマンド、エージェント、機能で拡張します。

プラグインは Claude Code をスキル、エージェント、フック、MCP サーバーで拡張します。プラグインマーケットプレイスは、これらの拡張機能を自分で構築することなく発見してインストールするのに役立つカタログです。

独自のマーケットプレイスを作成して配布したいですか？ [プラグインマーケットプレイスを作成して配布する](#) を参照してください。

マーケットプレイスの仕組み

マーケットプレイスは、他の誰かが作成して共有したプラグインのカタログです。マーケットプレイスを使用するのは 2 段階のプロセスです。

Step 1: マーケットプレイスを追加する

これにより、カタログが Claude Code に登録され、利用可能なものを参照できるようになります。プラグインはまだインストールされていません。

Step 2: 個別のプラグインをインストールする

カタログを参照して、必要なプラグインをインストールします。

アプリストアを追加するようなものと考えてください。ストアを追加するとそのコレクションを参照できるようになりますが、どのアプリをダウンロードするかは個別に選択します。

公式 Anthropic マーケットプレイス

公式 Anthropic マーケットプレイス（`claude-plugins-official`）は Claude Code を起動すると自動的に利用可能になります。`/plugin` を実行して **Discover** タブに移動し、利用可能なものを参照してください。

公式マーケットプレイスからプラグインをインストールするには：

```
/plugin install plugin-name@claude-plugins-official
```

Note:

公式マーケットプレイスは Anthropic によって管理されています。公式マーケットプレイスにプラグインを送信するには、アプリ内送信フォームのいずれかを使用してください。

- **Claude.ai:** claude.ai/settings/plugins/submit
- **Console:** platform.claude.com/plugins/submit

プラグインを独立して配布するには、[独自のマーケットプレイスを作成](#)してユーザーと共有してください。

公式マーケットプレイスには、プラグインのいくつかのカテゴリが含まれています。

コードインテリジェンス

コードインテリジェンスプラグインは Claude Code の組み込み LSP ツールを有効にし、Claude が定義にジャンプしたり、参照を見つけたり、編集直後に型エラーを確認したりできるようにします。これらのプラグインは [Language Server Protocol](#) 接続を構成します。これは VS Code のコードインテリジェンスを強化する同じテクノロジーです。

これらのプラグインでは、言語サーバーバイナリがシステムにインストールされている必要があります。言語サーバーが既にインストールされている場合、プロジェクトを開くと Claude は対応するプラグインをインストールするよう促す場合があります。

言語	プラグイン	必要なバイナリ
C/C++	<code>clangd-lsp</code>	<code>clangd</code>
C#	<code>csharp-lsp</code>	<code>csharp-ls</code>
Go	<code>gopls-lsp</code>	<code>gopls</code>
Java	<code>jdtls-lsp</code>	<code>jdtls</code>
Kotlin	<code>kotlin-lsp</code>	<code>kotlin-language-server</code>
Lua	<code>lua-lsp</code>	<code>lua-language-server</code>
PHP	<code>php-lsp</code>	<code>intelephense</code>
Python	<code>pyright-lsp</code>	<code>pyright-langserver</code>

言語	プラグイン	必要なバイナリ
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>

[他の言語用に独自の LSP プラグインを作成](#)することもできます。

Note:

プラグインをインストール後に `/plugin` Errors タブに `Executable not found in $PATH` が表示される場合は、上記の表から必要なバイナリをインストールしてください。

コード インテリジェンス プラグインから Claude が得られるもの

コード インテリジェンス プラグインがインストールされ、その言語サーバーバイナリが利用可能になると、Claude は 2 つの機能を得られます。

- **自動診断:** Claude が行うすべてのファイル編集後、言語サーバーは変更を分析し、エラーと警告を自動的に報告します。Claude はコンパイラやリンターを実行することなく、型エラー、不足しているインポート、構文の問題を確認します。Claude がエラーを導入した場合、それに気付いて同じターンで問題を修正します。これはプラグインをインストール以外の設定は必要ありません。「diagnostics found」インジケータが表示されたときに **Ctrl+O** を押すと、診断をインラインで確認できます。
- **コードナビゲーション:** Claude は言語サーバーを使用して定義にジャンプしたり、参照を見つけたり、ホバーで型情報を取得したり、シンボルをリストしたり、実装を見つけたり、呼び出し階層をトレースしたりできます。これらの操作により、Claude は grep ベースの検索よりも正確なナビゲーションが可能になりますが、言語と環境によって可用性が異なる場合があります。

問題が発生した場合は、[コード インテリジェンスのトラブルシューティング](#)を参照してください。

外部統合

これらのプラグインは事前構成された [MCP サーバー](#) をバンドルしているため、手動セットアップなしで Claude を外部サービスに接続できます。

- **ソース管理:** `github`、`gitlab`
- **プロジェクト管理:** `atlassian` (Jira/Confluence)、`asana`、`linear`、`notion`

マーケットプレイスから事前構築されたプラグインを発見してインストールする

- **デザイン:** `figma`
- **インフラストラクチャ:** `vercel`、`firebase`、`supabase`
- **コミュニケーション:** `slack`
- **監視:** `sentry`

開発ワークフロー

一般的な開発タスク用のコマンドとエージェントを追加するプラグイン。

- **commit-commands:** コミット、プッシュ、PR 作成を含む Git コミット ワークフロー
- **pr-review-toolkit:** プルリクエストをレビューするための特化したエージェント
- **agent-sdk-dev:** Claude Agent SDK で構築するためのツール
- **plugin-dev:** 独自のプラグインを作成するためのツールキット

出力スタイル

Claude の応答方法をカスタマイズします。

- **explanatory-output-style:** 実装の選択に関する教育的な洞察
- **learning-output-style:** スキル構築のためのインタラクティブな学習モード

試してみる: デモマーケットプレイスを追加する

Anthropic は、プラグインシステムで何が可能かを示す例プラグインを含む [デモプラグインマーケットプレイス](#) (`claude-code-plugins`) も管理しています。公式マーケットプレイスとは異なり、このマーケットプレイスは手動で追加する必要があります。

Step 1: マーケットプレイスを追加する

Claude Code 内から、`anthropics/claude-code` マーケットプレイスの `plugin marketplace add` コマンドを実行します。

```
/plugin marketplace add anthropics/claude-code
```

これにより、マーケットプレイス カタログがダウンロードされ、そのプラグインが利用可能になります。

Step 2: 利用可能なプラグインを参照する

`/plugin` を実行してプラグイン マネージャーを開きます。これにより、**Tab**（または後方に移動するには **Shift+Tab**）を使用して循環できる 4 つのタブを持つタブ付きインターフェイスが開きます。

- **Discover:** すべてのマーケットプレイスから利用可能なプラグインを参照
- **Installed:** インストール済みプラグインを表示および管理
- **Marketplaces:** 追加したマーケットプレイスを追加、削除、または更新
- **Errors:** プラグイン読み込みエラーを表示

Discover タブに移動して、追加したばかりのマーケットプレイスからプラグインを確認してください。

Step 3: プラグインをインストールする

プラグインを選択してその詳細を表示し、インストール スコープを選択します。

- **User scope:** すべてのプロジェクト全体で自分用にインストール
- **Project scope:** このリポジトリのすべてのコラボレーター用にインストール
- **Local scope:** このリポジトリ内で自分用にのみインストール

たとえば、**commit-commands**（git ワークフロー コマンドを追加するプラグイン）を選択して、ユーザー スコープにインストールします。

コマンドラインから直接インストールすることもできます。

```
/plugin install commit-commands@anthropics-claude-code
```

スコープの詳細については、[構成スコープ](#)を参照してください。

Step 4: 新しいプラグインを使用する

インストール後、プラグインのコマンドはすぐに利用可能になります。プラグイン コマンドはプラグイン名でネームスペース化されているため、**commit-commands** は `/commit-commands:commit` のようなコマンドを提供します。

ファイルに変更を加えて、以下を実行して試してみてください。

```
/commit-commands:commit
```

これにより、変更がステージされ、コミット メッセージが生成され、コミットが作成されます。

各プラグインは異なる方法で機能します。**Discover** タブのプラグインの説明またはそのホームページをチェックして、提供されるコマンドと機能を確認してください。

このガイドの残りの部分では、マーケットプレイスを追加し、プラグインをインストールし、構成を管理するすべての方法について説明します。

マーケットプレイスを追加する

`/plugin marketplace add` コマンドを使用して、異なるソースからマーケットプレイスを追加します。

Tip:

ショートカット: `/plugin marketplace` の代わりに `/plugin market` を使用でき、`remove` の代わりに `rm` を使用できます。

- **GitHub リポジトリ:** `owner/repo` 形式 (例: `anthropics/claude-code`)
- **Git URL:** 任意の git リポジトリ URL (GitLab、Bitbucket、自己ホスト)
- **ローカルパス:** ディレクトリまたは `marketplace.json` ファイルへの直接パス
- **リモート URL:** ホストされた `marketplace.json` ファイルへの直接 URL

GitHub から追加する

`.claude-plugin/marketplace.json` ファイルを含む GitHub リポジトリを `owner/repo` 形式を使用して追加します。ここで `owner` は GitHub ユーザー名または組織で、`repo` はリポジトリ名です。

たとえば、`anthropics/claude-code` は `anthropics` が所有する `claude-code` リポジトリを指します。

```
/plugin marketplace add anthropics/claude-code
```

他の Git ホストから追加する

完全な URL を提供することで、任意の git リポジトリを追加します。これは GitLab、Bitbucket、自己ホスト サーバーを含む任意の Git ホストで機能します。

HTTPS を使用する場合：

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

マーケットプレイスから事前構築されたプラグインを発見してインストールする

SSH を使用する場合：

```
/plugin marketplace add git@gitlab.com:company/plugins.git
```

特定のブランチまたはタグを追加するには、**#** の後に ref を追加します。

```
/plugin marketplace add https://gitlab.com/company/plugins.git#v1.0.0
```

ローカルパスから追加する

.claude-plugin/marketplace.json ファイルを含むローカル ディレクトリを追加します。

```
/plugin marketplace add ./my-marketplace
```

marketplace.json ファイルへの直接パスを追加することもできます。

```
/plugin marketplace add ./path/to/marketplace.json
```

リモート URL から追加する

URL 経由でリモート **marketplace.json** ファイルを追加します。

```
/plugin marketplace add https://example.com/marketplace.json
```

Note:

URL ベースのマーケットプレイスは、Git ベースのマーケットプレイスと比べていくつかの制限があります。プラグインをインストールするときに「path not found」エラーが発生した場合は、[トラブルシューティング](#)を参照してください。

プラグインをインストールする

マーケットプレイスを追加したら、プラグインを直接インストールできます（デフォルトではユーザー スコープにインストール）。

```
/plugin install plugin-name@marketplace-name
```

別の[インストール スコープ](#)を選択するには、インタラクティブ UI を使用します。 `/plugin` を実行して **Discover** タブに移動し、プラグインで **Enter** を押します。以下のオプションが表示されます。

- **User scope** (デフォルト) : すべてのプロジェクト全体で自分用にインストール
- **Project scope**: このリポジトリのすべてのコラボレーター用にインストール (`.claude/settings.json` に追加)
- **Local scope**: このリポジトリ内で自分用にのみインストール (コラボレーターと共有されない)

managed スコープのプラグインも表示される場合があります。これらは管理者が[管理設定](#)経由でインストールしたもので、変更することはできません。

`/plugin` を実行して **Installed** タブに移動し、スコープでグループ化されたプラグインを確認してください。

Warning:

プラグインをインストールする前に、それを信頼していることを確認してください。Anthropic はプラグインに含まれる MCP サーバー、ファイル、またはその他のソフトウェアを制御せず、意図したとおりに機能することを確認できません。詳細については、各プラグインのホームページを確認してください。

インストール済みプラグインを管理する

`/plugin` を実行して **Installed** タブに移動し、プラグインを表示、有効化、無効化、またはアンインストールします。プラグイン名または説明でリストをフィルタリングするには、入力します。

直接コマンドでプラグインを管理することもできます。

プラグインをアンインストールせずに無効化します。

```
/plugin disable plugin-name@marketplace-name
```

無効化されたプラグインを再度有効化します。

```
/plugin enable plugin-name@marketplace-name
```

プラグインを完全に削除します。

```
/plugin uninstall plugin-name@marketplace-name
```

`--scope` オプションを使用すると、CLI コマンドで特定のスコープをターゲットにできます。

```
claude plugin install formatter@your-org --scope project
claude plugin uninstall formatter@your-org --scope project
```

プラグインの変更をリスタートなしで適用する

セッション中にプラグインをインストール、有効化、または無効化すると、一部の変更（新しいコマンドとフックなど）はすぐに有効になります。LSP サーバーの更新を含む他の変更には、リスタートが必要です。

すべての保留中のプラグイン変更をリスタートなしで有効化するには、以下を実行します。

```
/reload-plugins
```

Claude Code はすべてのアクティブなプラグインをリロードし、読み込まれたものを報告します。LSP サーバーが追加または更新された場合、それらが有効になるにはリスタートが必要であることが通知されます。

マーケットプレイスを管理する

インタラクティブな `/plugin` インターフェースまたは CLI コマンドを使用してマーケットプレイスを管理できます。

インタラクティブ インターフェースを使用する

`/plugin` を実行して **Marketplaces** タブに移動して、以下を実行します。

- 追加したすべてのマーケットプレイスをそのソースとステータスで表示
- 新しいマーケットプレイスを追加
- マーケットプレイス リストを更新して最新のプラグインを取得

- ・ 不要になったマーケットプレイスを削除

CLI コマンドを使用する

直接コマンドでマーケットプレイスを管理することもできます。

構成されたすべてのマーケットプレイスをリストします。

```
/plugin marketplace list
```

マーケットプレイスからプラグイン リストを更新します。

```
/plugin marketplace update marketplace-name
```

マーケットプレイスを削除します。

```
/plugin marketplace remove marketplace-name
```

Warning:

マーケットプレイスを削除すると、そこからインストールしたプラグインがアンインストールされます。

自動更新を構成する

Claude Code はスタートアップ時にマーケットプレイスとそのインストール済みプラグインを自動的に更新できます。マーケットプレイスで自動更新が有効になっている場合、Claude Code はマーケットプレイス データを更新し、インストール済みプラグインを最新バージョンに更新します。プラグインが更新された場合、`/reload-plugins` を実行するよう促すメッセージが表示されます。

UI を通じて個別のマーケットプレイスの自動更新を切り替えます。

1. `/plugin` を実行してプラグイン マネージャーを開く
2. **Marketplaces** を選択
3. リストからマーケットプレイスを選択
4. **Enable auto-update** または **Disable auto-update** を選択

公式 Anthropic マーケットプレイスはデフォルトで自動更新が有効になっています。サードパーティおよびローカル開発マーケットプレイスはデフォルトで自動更新が無効になっています。

Claude Code とすべてのプラグインの両方のすべての自動更新を完全に無効化するには、`DISABLE_AUTOUPDATER` 環境変数を設定します。詳細については、[自動更新](#)を参照してください。

Claude Code の自動更新を無効化しながらプラグイン自動更新を有効化したままにするには、`DISABLE_AUTOUPDATER` と共に `FORCE_AUTOUPDATE_PLUGINS=true` を設定します。

```
export DISABLE_AUTOUPDATER=true
export FORCE_AUTOUPDATE_PLUGINS=true
```

これは Claude Code の更新を手動で管理したいが、プラグイン更新を自動的に受け取りたい場合に便利です。

チーム マーケットプレイスを構成する

チーム管理者は、`.claude/settings.json` にマーケットプレイス構成を追加することで、プロジェクトの自動マーケットプレイス インストールを設定できます。チームメンバーがリポジトリ フォルダを信頼すると、Claude Code はこれらのマーケットプレイスとプラグインをインストールするよう促します。

プロジェクトの `.claude/settings.json` に `extraKnownMarketplaces` を追加します。

```
{
  "extraKnownMarketplaces": {
    "my-team-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

`extraKnownMarketplaces` と `enabledPlugins` を含む完全な構成オプションについては、[プラグイン設定](#)を参照してください。

セキュリティ

プラグインとマーケットプレイスは、ユーザー権限でマシン上で任意のコードを実行できる、非常に信頼されたコンポーネントです。信頼できるソースからのみプラグインをインストールし、マーケットプレイスを追加してください。組織は、[管理マーケットプレイス制限](#)を使用してユーザーが追加できるマーケットプレイスを制限できます。

トラブルシューティング

/plugin コマンドが認識されない

「unknown command」が表示されるか、`/plugin` コマンドが表示されない場合：

1. **バージョンを確認する:** `claude --version` を実行します。プラグインにはバージョン 1.0.33 以降が必要です。
2. **Claude Code を更新する:**
 - **Homebrew:** `brew upgrade claude-code`
 - **npm:** `npm update -g @anthropic-ai/claude-code`
 - **ネイティブ インストーラー:** [セットアップ](#) からインストール コマンドを再実行します。
3. **Claude Code を再起動する:** 更新後、ターミナルを再起動して `claude` を再度実行します。

一般的な問題

- **マーケットプレイスが読み込まれない:** URL がアクセス可能であり、`.claude-plugin/marketplace.json` がパスに存在することを確認してください。
- **プラグイン インストール エラー:** プラグイン ソース URL がアクセス可能であり、リポジトリが公開されている（またはアクセス権がある）ことを確認してください。
- **インストール後にファイルが見つからない:** プラグインはキャッシュにコピーされるため、プラグインディレクトリ外のファイルを参照するパスは機能しません。
- **プラグイン スキルが表示されない:** `rm -rf ~/.claude/plugins/cache` でキャッシュをクリアし、Claude Code を再起動して、プラグインを再度インストールしてください。

詳細なトラブルシューティングとソリューションについては、マーケットプレイス ガイドの [トラブルシューティング](#) を参照してください。デバッグ ツールについては、[デバッグと開発ツール](#) を参照してください。

コードインテリジェンスの問題

- **言語サーバーが起動しない:** バイナリがインストールされており、`$PATH` で利用可能であることを確認してください。詳細については、`/plugin Errors` タブを確認してください。
- **メモリ使用量が多い:** `rust-analyzer` や `pyright` などの言語サーバーは、大規模なプロジェクトで大量のメモリを消費する可能性があります。メモリの問題が発生した場合は、`/plugin disable <plugin-name>` でプラグインを無効化し、代わりに Claude の組み込み検索ツールを使用してください。
- **モノレポでの誤検知診断:** ワークスペースが正しく構成されていない場合、言語サーバーは内部パッケージの未解決インポートエラーを報告する可能性があります。これはコードを編集する Claude の能力に影響しません。

次のステップ

- **独自のプラグインを構築する:** スキル、エージェント、フックを作成するには、[プラグイン](#)を参照してください。
- **マーケットプレイスを作成する:** チームまたはコミュニティにプラグインを配布するには、[プラグインマーケットプレイスを作成](#)を参照してください。
- **技術リファレンス:** 完全な仕様については、[プラグインリファレンス](#)を参照してください。

Part 6: Advanced & Automation

カスタムサブエージェントの作成

Claude Code でタスク固有のワークフローと改善されたコンテキスト管理のための特化した AI サブエージェントを作成して使用します。

サブエージェントは、特定の種類のタスクを処理する特化した AI アシスタントです。各サブエージェントは、カスタムシステムプロンプト、特定のツールアクセス、および独立した権限を備えた独自のコンテキストウィンドウで実行されます。Claude がサブエージェントの説明に一致するタスクに遭遇すると、そのサブエージェントに委譲され、サブエージェントは独立して動作し、結果を返します。

Note:

複数のエージェントが並行して動作し、互いに通信する必要がある場合は、代わりに [エージェントチーム](#) を参照してください。サブエージェントは単一のセッション内で動作します。エージェントチームは別々のセッション間で調整します。

サブエージェントは以下に役立ちます：

- ・ **コンテキストを保持する** ことで、探索と実装をメインの会話から分離します
- ・ **制約を強制する** ことで、サブエージェントが使用できるツールを制限します
- ・ **設定を再利用する** ことで、ユーザーレベルのサブエージェント全体でプロジェクト間で再利用します
- ・ **動作を特化させる** ことで、特定のドメイン向けの焦点を絞ったシステムプロンプトを使用します
- ・ **コストを制御する** ことで、Haiku のような高速で安価なモデルにタスクをルーティングします

Claude は各サブエージェントの説明を使用して、タスクを委譲するかどうかを決定します。サブエージェントを作成するときは、Claude がいつそれを使用するかを知るように明確な説明を書いてください。

Claude Code には、**Explore**、**Plan**、**general-purpose** などのいくつかの組み込みサブエージェントが含まれています。特定のタスクを処理するカスタムサブエージェントを作成することもできます。このページでは、[組み込みサブエージェント](#)、[独自のサブエージェントを作成する方法](#)、[完全な設定オプション](#)、[サブエージェントを使用するためのパターン](#)、および [サブエージェントの例](#) について説明します。

組み込みサブエージェント

Claude Code には、Claude が適切なときに自動的に使用する組み込みサブエージェントが含まれています。各サブエージェントは、親の会話の権限を継承し、追加のツール制限があります。

Explore

コードベースの検索と分析に最適化された高速な読み取り専用エージェント。

- **モデル:** Haiku（高速、低レイテンシ）
- **ツール:** 読み取り専用ツール（Write および Edit ツールへのアクセスは拒否）
- **目的:** ファイル検出、コード検索、コードベース探索

Claude は、変更を加えずにコードベースを検索または理解する必要があるときに Explore に委譲します。これにより、探索結果がメインの会話コンテキストから除外されます。

Explore を呼び出すときに、Claude は徹底度レベルを指定します：ターゲット検索の場合は **quick**、バランスの取れた探索の場合は **medium**、包括的な分析の場合は **very thorough**。

Plan

[プランモード](#) 中にプランを提示する前にコンテキストを収集するために使用される研究エージェント。

- **モデル:** メイン会話から継承
- **ツール:** 読み取り専用ツール（Write および Edit ツールへのアクセスは拒否）
- **目的:** 計画のためのコードベース研究

プランモード中に Claude がコードベースを理解する必要がある場合、研究を Plan サブエージェントに委譲します。これにより、無限ネストを防ぎながら（サブエージェントは他のサブエージェントを生成できません）、必要なコンテキストを収集します。

General-purpose

探索とアクションの両方を必要とする複雑なマルチステップタスク向けの有能なエージェント。

- ・ **モデル**: メイン会話から継承
- ・ **ツール**: すべてのツール
- ・ **目的**: 複雑な研究、マルチステップ操作、コード変更

Claude は、タスクが探索と変更の両方を必要とする場合、結果を解釈するための複雑な推論、または複数の依存ステップが必要な場合に `general-purpose` に委譲します。

Other

Claude Code には、特定のタスク向けの追加のヘルパーエージェントが含まれています。これらは通常自動的に呼び出されるため、直接使用する必要はありません。

エージェント	モデル	Claude がいつ使用するか
Bash	継承	別のコンテキストでターミナルコマンドを実行する場合
statusline-setup	Sonnet	<code>/statusline</code> を実行してステータスラインを設定する場合
Claude Code Guide	Haiku	Claude Code 機能について質問する場合

これらの組み込みサブエージェント以外に、カスタムプロンプト、ツール制限、権限モード、hooks、およびスキルを備えた独自のサブエージェントを作成できます。以下のセクションでは、開始方法とサブエージェントのカスタマイズ方法を示します。

クイックスタート：最初のサブエージェントを作成する

サブエージェントはYAML フロントマターを含む Markdown ファイルで定義されます。
[手動で作成](#) することも、`/agents` コマンドを使用することもできます。

このチュートリアルでは、`/agent` コマンドを使用してユーザーレベルのサブエージェントを作成する手順を説明します。サブエージェントはコードをレビューし、コードベースの改善を提案します。

Step 1: サブエージェントインターフェースを開く

Claude Code で、以下を実行します：

/agents

Step 2: 新しいユーザーレベルエージェントを作成する

Create new agent を選択し、**User-level** を選択します。これにより、サブエージェントが `~/.claude/agents/` に保存され、すべてのプロジェクトで利用可能になります。

Step 3: Claude で生成する

Generate with Claude を選択します。プロンプトが表示されたら、サブエージェントを説明します：

```
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 がシステムプロンプトと設定を生成します。カスタマイズしたい場合は、**e** を押してエディターで開きます。

Step 4: ツールを選択する

読み取り専用レビューアの場合は、**Read-only tools** 以外のすべてを選択解除します。すべてのツールを選択したままにすると、サブエージェントはメイン会話で利用可能なすべてのツールを継承します。

Step 5: モデルを選択する

サブエージェントが使用するモデルを選択します。この例のエージェントの場合は、**Sonnet** を選択します。これはコードパターンを分析するための機能と速度のバランスが取れています。

Step 6: 色を選択する

サブエージェントの背景色を選択します。これにより、UI でどのサブエージェントが実行されているかを識別するのに役立ちます。

Step 7: 保存して試す

サブエージェントを保存します。すぐに利用可能になります（再起動は不要）。試してみます：

```
Use the code-improver agent to suggest improvements in this project
```

Claude は新しいサブエージェントに委譲し、コードベースをスキャンして改善提案を返します。

これで、マシン上のプロジェクトでコードベースを分析し、改善を提案するために使用できるサブエージェントが完成しました。

Markdown ファイルとして手動でサブエージェントを作成したり、CLI フラグを使用して定義したり、プラグインを通じて配布したりすることもできます。以下のセクションでは、すべての設定オプションについて説明します。

サブエージェントを設定する

/agents コマンドを使用する

`/agents` コマンドは、サブエージェントを管理するためのインタラクティブインターフェースを提供します。 `/agents` を実行して以下を行います：

- すべての利用可能なサブエージェント（組み込み、ユーザー、プロジェクト、プラグイン）を表示する
- ガイド付きセットアップまたは Claude 生成を使用して新しいサブエージェントを作成する
- 既存のサブエージェント設定とツールアクセスを編集する
- カスタムサブエージェントを削除する
- 重複が存在する場合、どのサブエージェントがアクティブであるかを確認する

これはサブエージェントを作成および管理するための推奨される方法です。手動作成または自動化の場合は、サブエージェントファイルを直接追加することもできます。

インタラクティブセッションを開始せずにコマンドラインからすべての設定されたサブエージェントをリストするには、`claude agents` を実行します。これにより、エージェントがソース別にグループ化され、より高い優先度の定義によってオーバーライドされているものが示されます。

サブエージェントのスコープを選択する

サブエージェントは YAML フロントマターを含む Markdown ファイルです。スコープに応じて異なる場所に保存します。複数のサブエージェントが同じ名前を共有する場合、より高い優先度の場所が優先されます。

場所	スコープ	優先度	作成方法
<code>--agents</code> CLI フラグ	現在のセッション	1 (最高)	Claude Code を起動するときに JSON を渡す
<code>.claude/agents/</code>	現在のプロジェクト	2	インタラクティブまたは手動
<code>~/.claude/agents/</code>	すべてのプロジェクト	3	インタラクティブまたは手動
プラグインの <code>agents/</code> ディレクトリ	プラグインが有効な場所	4 (最低)	プラグイン でインストール

プロジェクトサブエージェント (`.claude/agents/`) は、コードベースに固有のサブエージェントに最適です。バージョン管理にチェックインして、チームが協力して使用および改善できるようにします。

ユーザーサブエージェント (`~/.claude/agents/`) は、すべてのプロジェクトで利用可能な個人用サブエージェントです。

CLI で定義されたサブエージェント は、Claude Code を起動するときに JSON として渡されます。そのセッションのみ存在し、ディスクに保存されないため、クイックテストまたは自動化スクリプトに役立ちます：

```
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"
  }
}'
```

`--agents` フラグは、ファイルベースのサブエージェントと同じ [フロントマター](#) フィールドを持つ JSON を受け入れます：`description`、`prompt`、`tools`、`disallowedTools`、`model`、`permissionMode`、`mcpServers`、`hooks`、`maxTurns`、

`skills`、および `memory`。システムプロンプトには `prompt` を使用します。これはファイルベースのサブエージェントの markdown 本体と同等です。完全な JSON 形式については、[CLI リファレンス](#) を参照してください。

プラグインサブエージェント は、インストールした [プラグイン](#) から提供されます。カスタムサブエージェントと一緒に `/agents` に表示されます。プラグインサブエージェントの作成の詳細については、[プラグインコンポーネントリファレンス](#) を参照してください。

サブエージェントファイルを書く

サブエージェントファイルは、設定用の YAML フロントマターの後に Markdown のシステムプロンプトを使用します：

Note:

サブエージェントはセッション開始時に読み込まれます。ファイルを手動で追加してサブエージェントを作成する場合は、セッションを再起動するか、`/agents` を使用してすぐに読み込みます。

```
---
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.
```

フロントマターはサブエージェントのメタデータと設定を定義します。本体はサブエージェントの動作をガイドするシステムプロンプトになります。サブエージェントは、このシステムプロンプト（および作業ディレクトリなどの基本的な環境詳細）のみを受け取り、完全な Claude Code システムプロンプトは受け取りません。

サポートされているフロントマターフィールド

以下のフィールドは YAML フロントマターで使用できます。`name` と `description` のみが必須です。

フィールド	必須	説明
<code>name</code>	はい	小文字とハイフンを使用した一意の識別子
<code>description</code>	はい	Claude がこのサブエージェントに委譲するべき場合
<code>tools</code>	いいえ	サブエージェントが使用できる ツール 。省略した場合はすべてのツールを継承
<code>disallowedTools</code>	いいえ	拒否するツール。継承または指定されたリストから削除
<code>model</code>	いいえ	使用する モデル ： <code>sonnet</code> 、 <code>opus</code> 、 <code>haiku</code> 、または <code>inherit</code> 。デフォルトは <code>inherit</code>
<code>permissionMode</code>	いいえ	権限モード ： <code>default</code> 、 <code>acceptEdits</code> 、 <code>dontAsk</code> 、 <code>bypassPermissions</code> 、または <code>plan</code>
<code>maxTurns</code>	いいえ	サブエージェントが停止する前の最大 agentic ターン数
<code>skills</code>	いいえ	スタートアップ時にサブエージェントのコンテキストに読み込む スキル 。呼び出しのために利用可能にするだけでなく、完全なスキルコンテンツが注入されます。サブエージェントは親の会話からスキルを継承しません

フィールド	必須	説明
<code>mcpServers</code>	いいえ	このサブエージェントで利用可能な MCP サーバー 。各エントリは、既に設定されたサーバーを参照するサーバー名（例：" <code>slack</code> "）か、サーバー名をキーとし、完全な MCP サーバー設定 を値とするインライン定義のいずれかです
<code>hooks</code>	いいえ	このサブエージェントにスコープされた ライフサイクルフック
<code>memory</code>	いいえ	永続メモリスコープ ： <code>user</code> 、 <code>project</code> 、または <code>local</code> 。クロスセッション学習を有効にします
<code>background</code>	いいえ	<code>true</code> に設定して、このサブエージェントを常に バックグラウンドタスク として実行します。デフォルト： <code>false</code>
<code>isolation</code>	いいえ	<code>worktree</code> に設定して、サブエージェントを一時的な git worktree で実行し、リポジトリの分離されたコピーを提供します。サブエージェントが変更を加えない場合、 <code>worktree</code> は自動的にクリーンアップされます

モデルを選択する

`model` フィールドは、サブエージェントが使用する [AI モデル](#) を制御します：

- **モデルエイリアス**: 利用可能なエイリアスの 1 つを使用します：`sonnet`、`opus`、または `haiku`
- **inherit**: メイン会話と同じモデルを使用します

- ・ **省略**: 指定されていない場合、デフォルトは `inherit`（メイン会話と同じモデルを使用）です

サブエージェント機能を制御する

ツールアクセス、権限モード、および条件付きルールを通じて、サブエージェントが実行できることを制御できます。

利用可能なツール

サブエージェントは、Claude Code の [内部ツール](#) のいずれかを使用できます。デフォルトでは、サブエージェントはメイン会話からすべてのツール（MCP ツールを含む）を継承します。

ツールを制限するには、`tools` フィールド（許可リスト）または `disallowedTools` フィールド（拒否リスト）を使用します：

```
---
name: safe-researcher
description: Research agent with restricted capabilities
tools: Read, Grep, Glob, Bash
disallowedTools: Write, Edit
---
```

生成できるサブエージェントを制限する

エージェントが `claude --agent` でメインスレッドとして実行される場合、Agent ツールを使用してサブエージェントを生成できます。生成できるサブエージェントの種類を制限するには、`tools` フィールドで `Agent(agent_type)` 構文を使用します。

Note:

バージョン 2.1.63 では、Task ツールが Agent に名前変更されました。設定とエージェント定義の既存の `Task(...)` 参照は引き続きエイリアスとして機能します。

```
---
name: coordinator
description: Coordinates work across specialized agents
tools: Agent(worker, researcher), Read, Bash
---
```

これは許可リストです： `worker` と `researcher` サブエージェントのみを生成できます。エージェントが他の種類を生成しようすると、リクエストは失敗し、エージェントはプロンプトで許可されたタイプのみを表示します。特定のエージェントをブロックしながら他のすべてを許可するには、代わりに `permissions.deny` を使用します。

制限なしでサブエージェントを生成できるようにするには、括弧なしで `Agent` を使用します：

```
tools: Agent, Read, Bash
```

`Agent` が `tools` リストから完全に省略されている場合、エージェントはサブエージェントを生成できません。この制限は、`claude --agent` でメインスレッドとして実行されるエージェントにのみ適用されます。サブエージェントは他のサブエージェントを生成できないため、`Agent(agent_type)` はサブエージェント定義に影響を与えません。

権限モード

`permissionMode` フィールドは、サブエージェントが権限プロンプトをどのように処理するかを制御します。サブエージェントはメイン会話から権限コンテキストを継承しますが、モードをオーバーライドできます。

モード	動作
<code>default</code>	プロンプト付きの標準権限チェック
<code>acceptEdits</code>	ファイル編集を自動受け入れ
<code>dontAsk</code>	権限プロンプトを自動拒否（明示的に許可されたツールは引き続き機能）
<code>bypassPermissions</code>	すべての権限チェックをスキップ
<code>plan</code>	プランモード（読み取り専用探索）

Warning:

`bypassPermissions` は注意して使用してください。すべての権限チェックをスキップし、サブエージェントが承認なしで任意の操作を実行できるようにします。

親が `bypassPermissions` を使用する場合、これが優先され、オーバーライドできません。

スキルをサブエージェントにプリロードする

skills フィールドを使用して、スキルコンテンツをスタートアップ時にサブエージェントのコンテキストに注入します。これにより、実行中にスキルを検出して読み込む必要なく、サブエージェントにドメイン知識を提供します。

```
---
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.
```

各スキルの完全なコンテンツがサブエージェントのコンテキストに注入され、呼び出しのために利用可能にするだけではありません。サブエージェントは親の会話からスキルを継承しません。明示的にリストする必要があります。

Note:

これは [サブエージェントでスキルを実行する](#) の逆です。サブエージェントの **skills** を使用すると、サブエージェントはシステムプロンプトを制御し、スキルコンテンツを読み込みます。スキルの **context: fork** を使用すると、スキルコンテンツが指定したエージェントに注入されます。どちらも同じ基盤システムを使用します。

永続メモリを有効にする

memory フィールドは、サブエージェントに会話全体で存続する永続ディレクトリを提供します。サブエージェントはこのディレクトリを使用して、コードベースパターン、デバッグインサイト、アーキテクチャ決定などの知識を時間をかけて構築します。

```
---
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.

メモリがどの程度広く適用されるべきかに基づいて、スコープを選択します：

スコープ	場所	使用する場合
user	<code>~/.claude/agent-memory/<name-of-agent>/</code>	サブエージェントがすべてのプロジェクト全体で学習を記憶すべき場合
project	<code>.claude/agent-memory/<name-of-agent>/</code>	サブエージェントの知識がプロジェクト固有で、バージョン管理を通じて共有可能な場合
local	<code>.claude/agent-memory-local/<name-of-agent>/</code>	サブエージェントの知識がプロジェクト固有だが、バージョン管理にチェックインすべきでない場合

メモリが有効な場合：

- ・サブエージェントのシステムプロンプトには、メモリディレクトリの読み取りと書き込みの指示が含まれます。
- ・サブエージェントのシステムプロンプトには、メモリディレクトリの `MEMORY.md` の最初の 200 行も含まれ、200 行を超える場合は `MEMORY.md` をキュレートする指示があります。
- ・Read、Write、および Edit ツールが自動的に有効になり、サブエージェントがメモリファイルを管理できるようになります。

永続メモリのヒント

- ・ `user` は推奨されるデフォルトスコープです。サブエージェントの知識が特定のコードベースにのみ関連する場合は、`project` または `local` を使用します。

- サブエージェントに作業を開始する前にメモリを確認するよう依頼します：「このブルリクエストをレビューし、以前に見たパターンについてメモリを確認してください。」
- タスク完了後、サブエージェントにメモリを更新するよう依頼します：「完了したので、学習したことをメモリに保存してください。」時間をかけて、これはサブエージェントをより効果的にする知識ベースを構築します。
- メモリ指示をサブエージェントの markdown ファイルに直接含めて、サブエージェントが積極的に独自の知識ベースを維持するようにします：

```
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.
```

hooks を使用した条件付きルール

ツール使用をより動的に制御するには、**PreToolUse** hooks を使用して、操作が実行される前に検証します。ツールの一部の操作を許可しながら他の操作をブロックする必要がある場合に役立ちます。

この例は、読み取り専用データベースクエリのみを許可するサブエージェントを作成します。**PreToolUse** hook は、各 Bash コマンドが実行される前に **command** で指定されたスクリプトを実行します：

```
---
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 は [hook 入力を JSON として](#) stdin を通じて hook コマンドに渡します。検証スクリプトはこの JSON を読み取り、Bash コマンドを抽出し、[終了コード 2](#) で書き込み操作をブロックします：

```
#!/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
```

完全な入力スキーマについては [Hook 入力](#) を参照し、終了コードが動作に与える影響については [終了コード](#) を参照してください。

特定のサブエージェントを無効にする

[設定](#) の `deny` 配列にサブエージェントを追加することで、Claude が特定のサブエージェントを使用するのを防ぐことができます。 `Agent(subagent-name)` 形式を使用します。ここで `subagent-name` はサブエージェントの `name` フィールドと一致します。

```
{
  "permissions": {
    "deny": ["Agent(Explore)", "Agent(my-custom-agent)"]
  }
}
```

これは組み込みとカスタムサブエージェントの両方で機能します。 `--disallowedTools` CLI フラグを使用することもできます：

```
claude --disallowedTools "Agent(Explore)"
```

詳細については、[権限ドキュメント](#) を参照してください。

サブエージェント用の hooks を定義する

サブエージェントは、サブエージェントのライフサイクル中に実行される `hooks` を定義できます。hooks を設定する方法は2つあります：

- 1. **サブエージェントのフロントマター内**: そのサブエージェントがアクティブな間のみ実行される hooks を定義します
- 2. **`settings.json` 内**: サブエージェントが開始または停止するときにメインセッションで実行される hooks を定義します

サブエージェントフロントマター内の hooks

hooks をサブエージェントの markdown ファイルに直接定義します。これらの hooks はその特定のサブエージェントがアクティブな間のみ実行され、完了時にクリーンアップされます。

すべての [hook イベント](#) がサポートされています。サブエージェント向けの最も一般的なイベントは：

イベント	マッチャー入力	いつ発火するか
<code>PreToolUse</code>	ツール名	サブエージェントがツールを使用する前
<code>PostToolUse</code>	ツール名	サブエージェントがツールを使用した後
<code>Stop</code>	(なし)	サブエージェントが完了するとき (実行時に <code>SubagentStop</code> に変換)

この例は `PreToolUse` hook で Bash コマンドを検証し、`PostToolUse` でファイル編集後にリンターを実行します：

```
---
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"
---
```

フロントマター内の **Stop** hooks は自動的に **SubagentStop** イベントに変換されます。

サブエージェントイベント向けのプロジェクトレベル hooks

メインセッションでサブエージェントのライフサイクルイベントに応答する hooks を **settings.json** で設定します。

イベント	マッチャー入力	いつ発火するか
SubagentStart	エージェント型名	サブエージェントが実行を開始するとき
SubagentStop	エージェント型名	サブエージェントが完了するとき

どちらのイベントも、名前で特定のエージェント型をターゲットにするマッチャーをサポートします。この例は、**db-agent** サブエージェントが開始するときのみセットアップスクリプトを実行し、サブエージェントが停止するときにクリーンアップスクリプトを実行します：

```

{
  "hooks": {
    "SubagentStart": [
      {
        "matcher": "db-agent",
        "hooks": [
          { "type": "command", "command": "./scripts/setup-db-connection.sh" }
        ]
      }
    ],
    "SubagentStop": [
      {
        "hooks": [
          { "type": "command", "command": "./scripts/cleanup-db-connection.sh" }
        ]
      }
    ]
  }
}

```

完全な hook 設定形式については、[Hooks](#) を参照してください。

サブエージェントを使用する

自動委譲を理解する

Claude は、リクエスト内のタスク説明、サブエージェント設定の `description` フィールド、および現在のコンテキストに基づいて、タスクを自動的に委譲します。積極的な委譲を促すには、サブエージェントの `description` フィールドに「use proactively」などのフレーズを含めます。

特定のサブエージェントを明示的にリクエストすることもできます：

```

Use the test-runner subagent to fix failing tests
Have the code-reviewer subagent look at my recent changes

```

サブエージェントをフォアグラウンドまたはバックグラウンドで実行する

サブエージェントはフォアグラウンド（ブロッキング）またはバックグラウンド（並行）で実行できます：

- ・ **フォアグラウンドサブエージェント** は、完了するまでメイン会話をブロックします。権限プロンプトと明確化の質問（[AskUserQuestion](#) など）はあなたに渡されます。
- ・ **バックグラウンドサブエージェント** は、作業を続ける間に並行して実行されます。起動前に、Claude Code はサブエージェントが必要とするツール権限をプロンプトし、必要な承認があることを確認します。実行開始後、サブエージェントはこれらの権限を継承し、事前に承認されていないものを自動拒否します。バックグラウンドサブエージェントが明確化の質問をする必要がある場合、そのツール呼び出しは失敗しますが、サブエージェントは続行します。

バックグラウンドサブエージェントが権限不足で失敗した場合、[再開](#) してフォアグラウンドで再試行できます。

Claude はタスクに基づいてサブエージェントをフォアグラウンドまたはバックグラウンドで実行するかどうかを決定します。以下のこともできます：

- ・ Claude に「run this in the background」と依頼する
- ・ **Ctrl+B** を押して実行中のタスクをバックグラウンドにする

すべてのバックグラウンドタスク機能を無効にするには、

`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` 環境変数を `1` に設定します。[環境変数](#) を参照してください。

一般的なパターン

大量操作を分離する

サブエージェントの最も効果的な用途の 1 つは、大量の出力を生成する操作を分離することです。テストの実行、ドキュメントの取得、またはログファイルの処理は、かなりのコンテキストを消費できます。これらをサブエージェントに委譲することで、詳細な出力はサブエージェントのコンテキストに留まり、関連する概要のみがメイン会話に返されます。

```
Use a subagent to run the test suite and report only the failing tests with their error messages
```

並行研究を実行する

独立した調査の場合、複数のサブエージェントを生成して同時に動作させます：

```
Research the authentication, database, and API modules in parallel using separate subagents
```

各サブエージェントは独立して領域を探索し、Claude が結果を統合します。これは研究パスが互いに依存しない場合に最適です。

Warning:

サブエージェントが完了すると、その結果がメイン会話に返されます。詳細な結果を返す多くのサブエージェントを実行すると、かなりのコンテキストを消費できます。

持続的な並列性が必要なタスクまたはコンテキストウィンドウを超えるタスクの場合、[エージェントチーム](#) は各ワーカーに独立したコンテキストを提供します。

サブエージェントをチェーンする

マルチステップワークフローの場合、Claude にサブエージェントを順序立てて使用するよう依頼します。各サブエージェントはタスクを完了して結果を Claude に返し、Claude は関連するコンテキストを次のサブエージェントに渡します。

```
Use the code-reviewer subagent to find performance issues, then use the optimizer subagent to fix them
```

サブエージェントとメイン会話の選択

メイン会話 を使用する場合：

- ・タスクが頻繁なやり取りまたは反復的な改善が必要な場合
- ・複数のフェーズが重要なコンテキストを共有する場合（計画 → 実装 → テスト）
- ・迅速でターゲット化された変更を行う場合
- ・レイテンシが重要な場合。サブエージェントは新規に開始し、コンテキストを収集するのに時間がかかる場合があります

サブエージェント を使用する場合：

- ・タスクがメインコンテキストで不要な詳細な出力を生成する場合
- ・特定のツール制限または権限を強制したい場合
- ・作業が自己完結型で、概要を返すことができる場合

代わりに [スキル](#) を検討してください。分離されたサブエージェントコンテキストではなく、メイン会話コンテキストで実行される再利用可能なプロンプトまたはワークフローが必要な場合。

会話に既にある何かについての簡単な質問の場合は、サブエージェントの代わりに [/btw](#) を使用します。完全なコンテキストを表示しますが、ツールアクセスはなく、回答は履歴に追加されるのではなく破棄されます。

Note:

サブエージェントは他のサブエージェントを生成できません。ワークフローがネストされた委譲を必要とする場合は、[スキル](#) または [メイン会話からサブエージェントをチェーン](#) します。

サブエージェントコンテキストを管理する

サブエージェントを再開する

各サブエージェント呼び出しは、新しいコンテキストで新しいインスタンスを作成します。最初からやり直すのではなく、既存のサブエージェントの作業を続けるには、Claude に再開するよう依頼します。

再開されたサブエージェントは、すべての前のツール呼び出し、結果、および推論を含む完全な会話履歴を保持します。サブエージェントは新規に開始するのではなく、停止した場所から正確に再開します。

サブエージェントが完了すると、Claude はそのエージェント ID を受け取ります。サブエージェントを再開するには、Claude に前の作業を続けるよう依頼します：

```
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]
```

明示的に参照したい場合は Claude にエージェント ID を依頼することもできます。または、`~/.claude/projects/{project}/{sessionId}/subagents/` のトランスクリプトファイルで ID を見つけることができます。各トランスクリプトは `agent-{agentId}.jsonl` として保存されます。

サブエージェントトランスクリプトはメイン会話とは独立して永続化されます：

- ・ **メイン会話圧縮**: メイン会話が圧縮されると、サブエージェントトランスクリプトは影響を受けません。別のファイルに保存されます。
- ・ **セッション永続化**: サブエージェントトランスクリプトはセッション内で永続化されます。同じセッションを再開することで、Claude Code を再起動した後に [サブエージェントを再開](#) できます。
- ・ **自動クリーンアップ**: トランスクリプトは `cleanupPeriodDays` 設定（デフォルト：30 日）に基づいてクリーンアップされます。

自動圧縮

サブエージェントは、メイン会話と同じロジックを使用した自動圧縮をサポートします。デフォルトでは、自動圧縮は約 95% 容量でトリガーされます。圧縮を早期にトリガーするには、`CLAUDE_AUTOCOMPACT_PCT_OVERRIDE` をより低いパーセンテージ（例：50）に設定します。詳細については、[環境変数](#) を参照してください。

圧縮イベントはサブエージェントトランスクリプトファイルにログされます：

```
{
  "type": "system",
  "subtype": "compact_boundary",
  "compactMetadata": {
    "trigger": "auto",
    "preTokens": 167189
  }
}
```

`preTokens` 値は、圧縮が発生する前に使用されたトークン数を示します。

サブエージェントの例

これらの例は、サブエージェントを構築するための効果的なパターンを示しています。出発点として使用するか、Claude を使用してカスタマイズされたバージョンを生成します。

Tip:

ベストプラクティス：

- ・ **焦点を絞ったサブエージェントを設計する**：各サブエージェントは 1 つの特定のタスクに優れるべき

- ・ **詳細な説明を書く**： Claude は説明を使用して委譲するかどうかを決定
- ・ **ツールアクセスを制限する**： 必要な権限のみを付与してセキュリティと焦点を確保
- ・ **バージョン管理にチェックインする**： プロジェクトサブエージェントをチームと共有

コードレビュー

コードを変更せずにレビューする読み取り専用サブエージェント。この例は、制限されたツールアクセス（Edit または Write なし）と、正確に何を探すべきか、および出力をどのようにフォーマットするかを指定する詳細なプロンプトを備えた焦点を絞ったサブエージェントを設計する方法を示しています。

```
---
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.

デバッガー

問題を分析して修正できるサブエージェント。コードレビューアとは異なり、バグ修正にはコード変更が必要なため、Edit が含まれています。プロンプトは診断から検証までの明確なワークフローを提供します。

```
---  
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.

データサイエンティスト

データ分析作業向けのドメイン固有のサブエージェント。この例は、典型的なコーディングタスク以外の特化したワークフロー向けのサブエージェントを作成する方法を示しています。より有能な分析のために `model: sonnet` を明示的に設定します。

```
---  
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.

データベースクエリバリデーター

Bash アクセスを許可しますが、読み取り専用 SQL クエリのみを許可するようにコマンドを検証するサブエージェント。この例は、`tools` フィールドが提供するよりも細かい制御が必要な場合に、条件付き検証に `PreToolUse` hooks を使用する方法を示しています。

```
---
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 は [hook 入力を JSON として](#) stdin を通じて hook コマンドに渡します。検証スクリプトはこの JSON を読み取り、実行されるコマンドを抽出し、SQL 書き込み操作のリストに対してチェックします。書き込み操作が検出された場合、スクリプトは [終了コード 2](#) で終了してエラーメッセージを stderr 経由で Claude に返します。

プロジェクト内の任意の場所に検証スクリプトを作成します。パスは hook 設定の `command` フィールドと一致する必要があります：

```
#!/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
```

スクリプトを実行可能にします：

```
chmod +x ./scripts/validate-readonly-query.sh
```

hook は stdin 経由で JSON を受け取り、Bash コマンドは `tool_input.command` にあります。終了コード 2 は操作をブロックし、エラーメッセージを Claude にフィードバックします。終了コードと [Hook 入力](#) の詳細については、[Hooks](#) を参照してください。

次のステップ

サブエージェントを理解したので、これらの関連機能を探索します：

- [プラグインでサブエージェントを配布する](#) ことで、チームまたはプロジェクト全体でサブエージェントを共有
- [Claude Code をプログラムで実行する](#) ことで、Agent SDK を使用して CI/CD と自動化を実現

- [MCP サーバーを使用する](#) ことで、サブエージェントに外部ツールとデータへのアクセスを提供

Claude Code セッションのチームを調整する

複数の Claude Code インスタンスがチームとして連携して動作するように調整し、共有タスク、エージェント間メッセージング、および一元管理を実現します。

Warning:

エージェントチームは実験的機能であり、デフォルトでは無効になっています。 [settings.json](#) または環境に `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` を追加して有効にしてください。エージェントチームには、セッション再開、タスク調整、シャットダウン動作に関する [既知の制限](#) があります。

エージェントチームを使用すると、複数の Claude Code インスタンスが連携して動作するように調整できます。1つのセッションがチームリーダーとして機能し、作業を調整し、タスクを割り当て、結果を統合します。チームメンバーは独立して動作し、それぞれ独自のコンテキストウィンドウで動作し、互いに直接通信します。

[subagents](#)（単一セッション内で実行され、メインエージェントにのみ報告できる）とは異なり、リーダーを経由せずに個別のチームメンバーと直接対話することもできます。

Note:

エージェントチームには Claude Code v2.1.32 以降が必要です。 `claude --version` でバージョンを確認してください。

このページでは、以下について説明します。

- [エージェントチームを使用する場合](#)（ユースケースと subagents との比較を含む）
- [チームを開始する](#)
- [チームメンバーを制御する](#)（表示モード、タスク割り当て、委任を含む）
- [並列作業のベストプラクティス](#)

エージェントチームを使用する場合

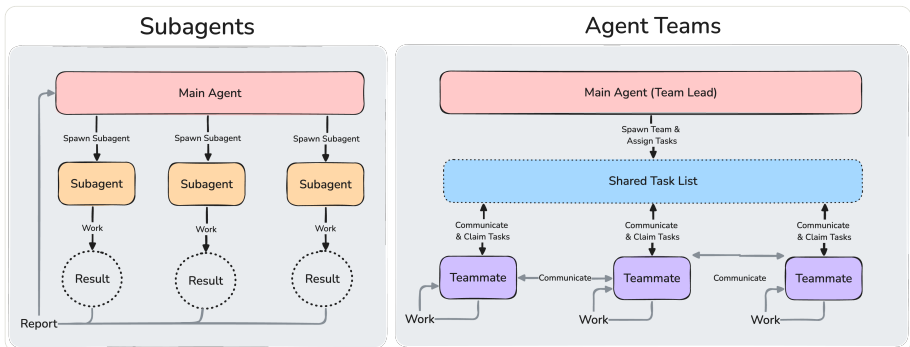
エージェントチームは、並列探索が実際の価値を追加するタスクに最も効果的です。完全なシナリオについては、[ユースケース例](#)を参照してください。最も強力なユースケースは以下の通りです。

- **調査とレビュー**：複数のチームメンバーが問題のさまざまな側面を同時に調査し、その後、各自の調査結果を共有して相互に検証できます
- **新しいモジュールまたは機能**：チームメンバーが互いに干渉することなく、それぞれ別々の部分を担当できます
- **競合する仮説でのデバッグ**：チームメンバーが異なる理論を並列でテストし、より迅速に答えに収束できます
- **クロスレイヤー調整**：フロントエンド、バックエンド、テストにまたがる変更で、それぞれ異なるチームメンバーが担当します

エージェントチームは調整オーバーヘッドを追加し、単一セッションよりも大幅に多くのトークンを使用します。チームメンバーが独立して動作できる場合に最も効果的です。順序付きタスク、同じファイルの編集、または多くの依存関係を持つ作業の場合は、単一セッションまたは [subagents](#) がより効果的です。

subagents との比較

エージェントチームと [subagents](#) の両方を使用すると、作業を並列化できますが、動作方法が異なります。ワーカーが互いに通信する必要があるかどうかに基づいて選択してください。



Subagent とエージェントチームのアーキテクチャを比較する図。Subagents はメインエージェントによって生成され、作業を実行し、結果を報告します。エージェントチームは共有タスクリストを通じて調整され、チームメンバーが互いに直接通信します。

	Subagents	エージェントチーム
コンテキスト	独自のコンテキストウィンドウ。結果は呼び出し元に返される	独自のコンテキストウィンドウ。完全に独立
通信	メインエージェントにのみ結果を報告	チームメンバーが互いに直接メッセージを送信
調整	メインエージェントがすべての作業を管理	自己調整を伴う共有タスクリスト
最適な用途	結果のみが重要な焦点を絞ったタスク	議論と協力が必要な複雑な作業
トークンコスト	低い：結果がメインコンテキストに要約されて返される	高い：各チームメンバーが個別の Claude インスタンス

結果を報告する必要がある迅速で焦点を絞ったワーカーが必要な場合は subagents を使用してください。チームメンバーが調査結果を共有し、互いに検証し、独立して調整する必要がある場合は、エージェントチームを使用してください。

エージェントチームを有効にする

エージェントチームはデフォルトでは無効になっています。

`CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` 環境変数を `1` に設定して有効にしてください。シェル環境または `settings.json` を通じて設定できます。

```
{
  "env": {
    "CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS": "1"
  }
}
```

最初のエージェントチームを開始する

エージェントチームを有効にした後、Claude に対してエージェントチームを作成するよう指示し、自然言語でタスクとチーム構成を説明してください。Claude がチームを作成し、チームメンバーを生成し、プロンプトに基づいて作業を調整します。

この例は、3つの役割が独立しており、互いに待つことなく問題を探索できるため、うまく機能します。

```
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.
```

その後、Claude は [共有タスクリスト](#) を持つチームを作成し、各視点のチームメンバーを生成し、問題を探索させ、調査結果を統合し、完了時に [チームをクリーンアップ](#) しようとしています。

リーダーのターミナルには、すべてのチームメンバーと彼らが取り組んでいる内容が表示されます。Shift+Down を使用してチームメンバーをサイクルして、直接メッセージを送信してください。最後のチームメンバーの後、Shift+Down はリーダーに戻ります。

各チームメンバーを独自の分割ペインに配置したい場合は、[表示モードを選択](#)を参照してください。

エージェントチームを制御する

リーダーに自然言語で実行したい内容を指示してください。チーム調整、タスク割り当て、および指示に基づいた委任を処理します。

表示モードを選択する

エージェントチームは2つの表示モードをサポートしています。

- **In-process** : すべてのチームメンバーがメインターミナル内で実行されます。Shift+Down を使用してチームメンバーをサイクルして、直接メッセージを入力してください。追加のセットアップなしで任意のターミナルで動作します。
- **分割ペイン** : 各チームメンバーが独自のペインを取得します。すべてのユーザーの出力を一度に表示でき、ペインをクリックして直接対話できます。tmux または iTerm2 が必要です。

Note:

tmux には特定のオペレーティングシステムでの既知の制限があり、従来は macOS で最も効果的に動作します。iTerm2 で **tmux -CC** を使用することが、**tmux** への推奨エントリーポイントです。

デフォルトは **"auto"** で、既に tmux セッション内で実行している場合は分割ペインを使用し、そうでない場合は in-process を使用します。**"tmux"** 設定は分割ペインモードを有効にし、ターミナルに基づいて tmux または iTerm2 を使用するかどうかを自動検出します。オーバーライドするには、[settings.json](#) で **teammateMode** を設定してください。

```
{  
  "teammateMode": "in-process"  
}
```

単一セッションに対して in-process モードを強制するには、フラグとして渡してください。

```
claude --teammate-mode in-process
```

分割ペインモードには、[tmux](#) または [it2 CLI](#) を備えた iTerm2 が必要です。手動でインストールするには、以下を実行してください。

- **tmux**：システムのパッケージマネージャーを通じてインストールしてください。プラットフォーム固有の手順については、[tmux wiki](#) を参照してください。
- **iTerm2**：[it2 CLI](#) をインストールし、**iTerm2 → Settings → General → Magic → Enable Python API** で Python API を有効にしてください。

チームメンバーとモデルを指定する

Claude はタスクに基づいて生成するチームメンバーの数を決定するか、正確に実行したい内容を指定できます。

```
Create a team with 4 teammates to refactor these modules in parallel.  
Use Sonnet for each teammate.
```

チームメンバーのプラン承認を要求する

複雑またはリスクの高いタスクの場合、チームメンバーが実装前にプランを立てることを要求できます。チームメンバーはリーダーがアプローチを承認するまで、読み取り専用プランモードで動作します。

```
Spawn an architect teammate to refactor the authentication module.  
Require plan approval before they make any changes.
```

チームメンバーがプランを完了すると、リーダーにプラン承認リクエストを送信します。リーダーはプランをレビューして、承認するか、フィードバック付きで却下するかのをいずれかを選択します。却下された場合、チームメンバーはプランモードのままで、フィードバックに基づいて修正し、再提出します。承認されると、チームメンバーはプランモードを終了し、実装を開始します。

リーダーは自律的に承認決定を下します。リーダーの判断に影響を与えるには、プロンプトで「テストカバレッジを含むプランのみを承認する」や「データベーススキーマを変更するプランを却下する」などの基準を指定してください。

チームメンバーと直接通信する

各チームメンバーは、完全に独立した Claude Code セッションです。任意のチームメンバーに直接メッセージを送信して、追加の指示を与えたり、フォローアップの質問をしたり、アプローチをリダイレクトしたりできます。

- **In-process モード**：Shift+Down を使用してチームメンバーをサイクルして、メッセージを入力してください。Enter キーを押してチームメンバーのセッションを表示し、Escape キーを押して現在のターンを中断してください。Ctrl+T を押してタスクリストを切り替えてください。
- **分割ペインモード**：チームメンバーのペインをクリックして、セッションと直接対話してください。各チームメンバーは独自のターミナルの完全なビューを持っています。

タスクを割り当てて要求する

共有タスクリストはチーム全体の作業を調整します。リーダーがタスクを作成し、チームメンバーがそれらを処理します。タスクには3つの状態があります。保留中、進行中、完了。タスクは他のタスクに依存することもできます。未解決の依存関係を持つ保留中のタスクは、それらの依存関係が完了するまで要求できません。

リーダーはタスクを明示的に割り当てるか、チームメンバーが自己要求できます。

- **リーダーが割り当て**：リーダーにどのタスクをどのチームメンバーに与えるかを指示してください
- **自己要求**：タスクを完了した後、チームメンバーは独立して次の未割り当て、ブロック解除されたタスクを選択します

タスク要求はファイルロックを使用して、複数のチームメンバーが同時に同じタスクを要求しようとするときの競合状態を防ぎます。

チームメンバーをシャットダウンする

チームメンバーのセッションを適切に終了するには、以下を実行してください。

```
Ask the researcher teammate to shut down
```

リーダーはシャットダウンリクエストを送信します。チームメンバーは承認して適切に終了するか、説明付きで却下できます。

チームをクリーンアップする

完了したら、リーダーにクリーンアップするよう指示してください。

```
Clean up the team
```

これにより、共有チームリソースが削除されます。リーダーがクリーンアップを実行すると、アクティブなチームメンバーをチェックし、まだ実行中の場合は失敗するため、最初にシャットダウンしてください。

Warning:

常にリーダーを使用してクリーンアップしてください。チームメンバーはクリーンアップを実行しないでください。チームメンバーのチームコンテキストが正しく解決されない可能性があります、リソースが不整合な状態のままになる可能性があります。

hooks で品質ゲートを実施する

[hooks](#) を使用して、チームメンバーが作業を完了したときまたはタスクが完了したときのルールを実施してください。

- `TeammateIdle` : チームメンバーがアイドル状態になろうとしているときに実行されます。終了コード 2 でフィードバックを送信し、チームメンバーを動作させ続けてください。
- `TaskCompleted` : タスクが完了としてマークされているときに実行されます。終了コード 2 で完了を防止し、フィードバックを送信してください。

エージェントチームの動作方法

このセクションでは、エージェントチームの背後にあるアーキテクチャとメカニクスについて説明します。使用を開始したい場合は、上記の [エージェントチームを制御する](#) を参照してください。

Claude がエージェントチームを開始する方法

エージェントチームが開始される方法は 2 つあります。

- **チームをリクエストする**：並列作業から利益を得るタスクを Claude に提供し、明示的にエージェントチームをリクエストしてください。Claude は指示に基づいてチームを作成します。
- **Claude がチームを提案する**：Claude がタスクが並列作業から利益を得ると判断した場合、チームの作成を提案する可能性があります。進行する前に確認してください。

どちらの場合でも、制御は維持されます。Claude はあなたの承認なしにチームを作成しません。

アーキテクチャ

エージェントチームは以下で構成されています。

コンポーネント	役割
チームリーダー	チームを作成し、チームメンバーを生成し、作業を調整するメイン Claude Code セッション
チームメンバー	割り当てられたタスクで動作する個別の Claude Code インスタンス
タスクリスト	チームメンバーが要求して完了する共有作業項目リスト
メールボックス	エージェント間の通信用メッセージングシステム

表示設定オプションについては、[表示モードを選択](#)を参照してください。チームメンバーのメッセージはリーダーに自動的に到着します。

システムはタスク依存関係を自動的に管理します。チームメンバーが他のタスクが依存するタスクを完了すると、ブロックされたタスクは手動介入なしにブロック解除されます。

チームとタスクはローカルに保存されます。

- **チーム設定**：`~/.claude/teams/{team-name}/config.json`
- **タスクリスト**：`~/.claude/tasks/{team-name}/`

チーム設定には、各チームメンバーの名前、エージェント ID、およびエージェントタイプを含む `members` 配列が含まれています。チームメンバーはこのファイルを読み取って、他のチームメンバーを発見できます。

権限

チームメンバーはリーダーの権限設定で開始します。リーダーが `--dangerously-skip-permissions` で実行する場合、すべてのチームメンバーも同様に実行します。生成後、個別のチームメンバーモードを変更できますが、生成時にチームメンバーごとのモードを設定することはできません。

コンテキストと通信

各チームメンバーは独自のコンテキストウィンドウを持っています。生成されると、チームメンバーは通常のセッションと同じプロジェクトコンテキストをロードします。CLAUDE.md、MCP servers、および skills。また、リーダーからの生成プロンプトを受け取ります。リーダーの会話履歴は引き継がれません。

チームメンバーが情報を共有する方法：

- **自動メッセージ配信**：チームメンバーがメッセージを送信すると、受信者に自動的に配信されます。リーダーは更新をポーリングする必要はありません。
- **アイドル通知**：チームメンバーが完了して停止すると、リーダーに自動的に通知します。
- **共有タスクリスト**：すべてのエージェントはタスクステータスを表示でき、利用可能な作業を要求できます。

チームメンバーメッセージング：

- **message**：特定のチームメンバーに1つのメッセージを送信します
- **broadcast**：すべてのチームメンバーに同時に送信します。チームサイズでコストがスケールするため、控えめに使用してください。

トークン使用量

エージェントチームは単一セッションよりも大幅に多くのトークンを使用します。各チームメンバーは独自のコンテキストウィンドウを持ち、トークン使用量はアクティブなチームメンバーの数でスケールします。調査、レビュー、および新機能作業の場合、追加のトークンは通常価値があります。ルーチンタスクの場合、単一セッションがより費用効果的です。使用ガイドンスについては、[エージェントチームトークンコスト](#)を参照してください。

ユースケース例

これらの例は、エージェントチームが並列探索が価値を追加するタスクをどのように処理するかを示しています。

並列コードレビューを実行する

単一のレビュアーは一度に 1 つのタイプの問題に傾く傾向があります。レビュー基準を独立したドメインに分割することで、セキュリティ、パフォーマンス、およびテストカバレッジがすべて同時に徹底的に注意を受けます。プロンプトは各チームメンバーに異なるレンズを割り当てるため、重複しません。

```
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.
```

各レビュアーは同じ PR から動作しますが、異なるフィルターを適用します。リーダーは完了後、3 つすべてにわたって調査結果を統合します。

競合する仮説で調査する

根本原因が不明な場合、単一のエージェントは 1 つのもっともらしい説明を見つけて停止する傾向があります。プロンプトはチームメンバーを明示的に敵対的にすることでこれと戦います。各チームメンバーの仕事は、独自の理論を調査するだけでなく、他の理論に異議を唱えることです。

```
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.
```

議論構造はここでの重要なメカニズムです。順序付き調査はアンカリングに悩まされます。1 つの理論が探索されると、その後の調査はそれに向かってバイアスされます。

複数の独立した調査官が互いに積極的に反証しようとしている場合、生き残る理論は実際の根本原因である可能性はるかに高くなります。

ベストプラクティス

チームメンバーに十分なコンテキストを提供する

チームメンバーはプロジェクトコンテキスト（CLAUDE.md、MCP servers、および skills を含む）を自動的にロードしますが、リーダーの会話履歴は継承しません。詳細については、[コンテキストと通信](#)を参照してください。生成プロンプトにタスク固有の詳細を含めてください。

```
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."
```

適切なチームサイズを選択する

チームメンバーの数に厳しい制限はありませんが、実際の制約が適用されます。

- ・ **トークンコストは線形にスケール**：各チームメンバーは独自のコンテキストウィンドウを持ち、独立してトークンを消費します。詳細については、[エージェントチームトークンコスト](#)を参照してください。
- ・ **調整オーバーヘッドが増加**：より多くのチームメンバーは、より多くの通信、タスク調整、および競合の可能性を意味します
- ・ **収穫逡減**：ある時点を超えると、追加のチームメンバーは作業を比例的に高速化しません

ほとんどのワークフローでは、3〜5 人のチームメンバーで開始してください。これは並列作業と管理可能な調整のバランスを取ります。このガイドの例では、3〜5 人のチームメンバーを使用しています。その範囲がさまざまなタスクタイプ全体でうまく機能するためです。

チームメンバーあたり 5〜6 個の [タスク](#) を持つことで、過度なコンテキストスイッチングなしに誰もが生産的に保たれます。15 個の独立したタスクがある場合、3 人のチームメンバーが良い出発点です。

作業が本当にチームメンバーが同時に動作することから利益を得る場合にのみスケールアップしてください。3 人の焦点を絞ったチームメンバーは、5 人の散らばったチームメンバーよりもしばしば優れています。

タスクを適切にサイズ設定する

- ・ **小さすぎる**：調整オーバーヘッドが利益を超える

- ・ **大きすぎる**：チームメンバーはチェックインなしで長時間動作し、無駄な努力のリスクが増加します
- ・ **ちょうど良い**：関数、テストファイル、またはレビューなど、明確な成果物を生成する自己完結型ユニット

Tip:

リーダーは作業をタスクに分割し、チームメンバーに自動的に割り当てます。十分なタスクを作成していない場合は、作業をより小さな部分に分割するよう指示してください。チームメンバーあたり 5〜6 個のタスクを持つことで、誰もが生産的に保たれ、誰かが立ち往生した場合、リーダーが作業を再割り当てできます。

チームメンバーが完了するまで待つ

時々、リーダーはチームメンバーを待つ代わりに、タスク自体を実装し始めます。これに気付いた場合は、以下を実行してください。

```
Wait for your teammates to complete their tasks before proceeding
```

調査とレビューから開始する

エージェントチームが初めての場合は、明確な境界があり、コードを書く必要がないタスクから開始してください。PR をレビューする、ライブラリを調査する、またはバグを調査します。これらのタスクは、並列実装に伴う調整の課題なしに、並列探索の価値を示しています。

ファイルの競合を回避する

2 人のチームメンバーが同じファイルを編集すると、上書きが発生します。作業を分割して、各チームメンバーが異なるファイルセットを所有するようにしてください。

監視と操舵

チームメンバーの進捗をチェックし、機能していないアプローチをリダイレクトし、調査結果が入ってくるにつれて統合してください。チームを長時間無人で実行させると、無駄な努力のリスクが増加します。

トラブルシューティング

チームメンバーが表示されない

Claude にチームを作成するよう指示した後、チームメンバーが表示されない場合は、以下を実行してください。

- In-process モードでは、チームメンバーは既に実行中ですが、表示されない可能性があります。Shift+Down を押してアクティブなチームメンバーをサイクルしてください。
- Claude に提供したタスクがチームを保証するのに十分複雑であることを確認してください。Claude はタスクに基づいてチームメンバーを生成するかどうかを決定します。
- 明示的に分割ペインをリクエストした場合は、tmux がインストールされ、PATH で利用可能であることを確認してください。

```
which tmux
```

- iTerm2 の場合、`it2` CLI がインストールされ、Python API が iTerm2 の設定で有効になっていることを確認してください。

権限プロンプトが多すぎる

チームメンバーの権限リクエストはリーダーにバブルアップし、摩擦を生じさせる可能性があります。チームメンバーを生成する前に、[権限設定](#)で一般的な操作を事前承認して、中断を減らしてください。

チームメンバーがエラーで停止する

チームメンバーはエラーが発生した後、回復する代わりに停止する可能性があります。In-process モードで Shift+Down を使用するか、分割モードでペインをクリックして出力を確認し、以下のいずれかを実行してください。

- 直接追加の指示を与える
- 作業を続行するために置き換えチームメンバーを生成する

リーダーが作業完了前にシャットダウンする

リーダーは、すべてのタスクが実際に完了する前に、チームが完了したと判断する可能性があります。これが発生した場合は、続行するよう指示してください。また、リーダーが委任する代わりに作業を開始する場合は、チームメンバーが完了するまで待つようリーダーに指示することもできます。

孤立した tmux セッション

チームが終了した後、tmux セッションが持続する場合、完全にクリーンアップされていない可能性があります。セッションをリストして、チームによって作成されたセッションを終了してください。

```
tmux ls
tmux kill-session -t <session-name>
```

制限事項

エージェントチームは実験的です。注意すべき現在の制限事項は以下の通りです。

- **In-process チームメンバーでのセッション再開なし**：`/resume` と `/rewind` は in-process チームメンバーを復元しません。セッションを再開した後、リーダーは存在しなくなったチームメンバーにメッセージを送信しようとする可能性があります。これが発生した場合は、リーダーに新しいチームメンバーを生成するよう指示してください。
- **タスクステータスが遅延する可能性**：チームメンバーはタスクを完了としてマークできず、依存タスクをブロックすることがあります。タスクが立ち往生しているように見える場合は、作業が実際に完了しているかどうかを確認し、タスクステータスを手動で更新するか、リーダーにチームメンバーをナッジするよう指示してください。
- **シャットダウンが遅い可能性**：チームメンバーは現在のリクエストまたはツール呼び出しを完了してからシャットダウンし、時間がかかる可能性があります。
- **セッションあたり 1 つのチーム**：リーダーは一度に 1 つのチームのみを管理できません。新しいチームを開始する前に、現在のチームをクリーンアップしてください。
- **ネストされたチームなし**：チームメンバーは独自のチームまたはチームメンバーを生成できません。リーダーのみがチームを管理できます。
- **リーダーは固定**：チームを作成するセッションは、その生涯のリーダーです。チームメンバーをリーダーに昇格させたり、リーダーシップを譲渡したりすることはできません。
- **権限は生成時に設定**：すべてのチームメンバーはリーダーの権限モードで開始します。生成後に個別のチームメンバーモードを変更できますが、生成時にチームメンバーごとのモードを設定することはできません。
- **分割ペインには tmux または iTerm2 が必要**：デフォルトの in-process モードは任意のターミナルで動作します。分割ペインモードは VS Code の統合ターミナル、Windows Terminal、または Ghostty ではサポートされていません。

Tip:

CLAUDE.md は正常に動作：チームメンバーは作業ディレクトリから **CLAUDE.md** ファイルを読み取ります。これを使用して、プロジェクト固有のガイダンスをすべてのチームメンバーに提供してください。

次のステップ

並列作業と委任の関連アプローチを探索してください。

- **軽量委任：**[subagents](#) はセッション内で調査または検証用のヘルパーエージェントを生成し、エージェント間調整が必要ないタスクに適しています
- **手動並列セッション：**[Git worktrees](#) を使用すると、自動チーム調整なしで複数の Claude Code セッションを自分で実行できます
- **アプローチを比較：**[subagent とエージェントチーム](#) の比較を参照して、並べて比較してください

Claude Code をプログラムで実行する

Agent SDK を使用して、CLI、Python、または TypeScript からプログラムで Claude Code を実行します。

[Agent SDK](#) は、Claude Code を支える同じツール、エージェントループ、およびコンテキスト管理を提供します。スクリプトと CI/CD 用の CLI として、または完全なプログラムによる制御のための [Python](#) および [TypeScript](#) パッケージとして利用できます。

Note:

CLI は以前「headless mode」と呼ばれていました。`-p` フラグとすべての CLI オプションは同じように機能します。

CLI からプログラムで Claude Code を実行するには、プロンプトと任意の [CLI オプション](#) を指定して `-p` を渡します。

```
claude -p "Find and fix the bug in auth.py" --allowedTools "Read,Edit,Bash"
```

このページでは、CLI（`claude -p`）経由で Agent SDK を使用することについて説明しています。構造化された出力、ツール承認コールバック、およびネイティブメッセージオブジェクトを備えた Python および TypeScript SDK パッケージについては、[完全な Agent SDK ドキュメント](#) を参照してください。

基本的な使用方法

任意の `claude` コマンドに `-p`（または `--print`）フラグを追加して、非対話的に実行します。すべての [CLI オプション](#) は `-p` で機能します。以下を含みます。

- `--continue` は [会話を続ける](#) 場合
- `--allowedTools` は [ツールを自動承認する](#) 場合
- `--output-format` は [構造化された出力を取得する](#) 場合

この例は、コードベースについて Claude に質問し、応答を出力します。

```
claude -p "What does the auth module do?"
```


例

これらの例は、一般的な CLI パターンを強調しています。

構造化された出力を取得する

`--output-format` を使用して、応答がどのように返されるかを制御します。

- `text` (デフォルト) : プレーンテキスト出力
- `json` : 結果、セッション ID、およびメタデータを含む構造化 JSON
- `stream-json` : リアルタイムストリーミング用の改行区切り JSON

この例は、セッションメタデータを含む JSON としてプロジェクト概要を返し、テキスト結果は `result` フィールドに含まれます。

```
claude -p "Summarize this project" --output-format json
```

特定のスキーマに準拠した出力を取得するには、`--output-format json` を `--json-schema` および [JSON Schema](#) 定義と共に使用します。応答には、リクエストに関するメタデータ（セッション ID、使用状況など）が含まれ、構造化された出力は `structured_output` フィールドに含まれます。

この例は、`auth.py` から関数名を抽出し、文字列の配列として返します。

```
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:

`jq` などのツールを使用して応答を解析し、特定のフィールドを抽出します。

```
## テキスト結果を抽出
claude -p "Summarize this project" --output-format json | jq -r '.result'

## 構造化された出力を抽出
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'
```

レスポンスをストリーミングする

`--output-format stream-json` を `--verbose` および `--include-partial-messages` と共に使用して、生成されるトークンをリアルタイムで受け取ります。各行は、イベントを表す JSON オブジェクトです。

```
claude -p "Explain recursion" --output-format stream-json --verbose --include-partial-messages
```

次の例は、`jq` を使用してテキストデルタをフィルタリングし、ストリーミングテキストのみを表示します。`-r` フラグは生の文字列を出力し（引用符なし）、`-j` は改行なしで結合するため、トークンは継続的にストリーミングされます。

```
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'
```

コールバックとメッセージオブジェクトを使用したプログラムによるストリーミングについては、Agent SDK ドキュメントの [リアルタイムでレスポンスをストリーミングする](#) を参照してください。

ツールを自動承認する

`--allowedTools` を使用して、Claude が確認を求めずに特定のツールを使用できるようにします。この例は、テストスイートを実行し、失敗を修正し、Claude が許可を求めずに Bash コマンドを実行し、ファイルを読み取り/編集できるようにします。

```
claude -p "Run the test suite and fix any failures" \  
--allowedTools "Bash,Read,Edit"
```

コミットを作成する

この例は、ステージされた変更を確認し、適切なメッセージを含むコミットを作成します。

```
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 *)"
```

`--allowedTools` フラグは [パーミッションルール構文](#) を使用します。末尾の `*` はプレフィックスマッチングを有効にするため、`Bash(git diff *)` は `git diff` で始まるすべてのコマンドを許可します。スペースは重要です。スペースがない場合、`Bash(git diff*)` は `git diff-index` にも一致します。

Note:

ユーザーが呼び出した [skills](#) (`/commit` など) および [組み込みコマンド](#) は、対話モードでのみ利用可能です。 `-p` モードでは、代わりに実行したいタスクを説明してください。

システムプロンプトをカスタマイズする

`--append-system-prompt` を使用して、Claude Code のデフォルト動作を保持しながら指示を追加します。この例は、PR diff を Claude にパイプし、セキュリティ脆弱性をレビューするよう指示します。

```
gh pr diff "$1" | claude -p \  
--append-system-prompt "You are a security engineer. Review for  
vulnerabilities." \  
--output-format json
```

デフォルトプロンプトを完全に置き換える `--system-prompt` を含む詳細なオプションについては、[システムプロンプトフラグ](#) を参照してください。

会話を続ける

`--continue` を使用して最新の会話を続けるか、`--resume` をセッション ID と共に使用して特定の会話を続けます。この例は、レビューを実行してから、フォローアッププロンプトを送信します。

```
## 最初のリクエスト
claude -p "Review this codebase for performance issues"

## 最新の会話を続ける
claude -p "Now focus on the database queries" --continue
claude -p "Generate a summary of all issues found" --continue
```

複数の会話を実行している場合は、セッション ID をキャプチャして特定の会話を再開します。

```
session_id=$(claude -p "Start a review" --output-format json | jq -r
'.session_id')
claude -p "Continue that review" --resume "$session_id"
```

次のステップ

- [Agent SDK クイックスタート](#)：Python または TypeScript で最初のエージェントを構築します
- [CLI リファレンス](#)：すべての CLI フラグとオプション
- [GitHub Actions](#)：GitHub ワークフローで Agent SDK を使用します
- [GitLab CI/CD](#)：GitLab パイプラインで Agent SDK を使用します

任意のデバイスからローカルセッションを続行する Remote Control

Remote Control を使用して、スマートフォン、タブレット、または任意のブラウザから Claude Code のローカルセッションを続行します。claude.ai/code と Claude モバイルアプリで動作します。

Note:

Remote Control はすべてのプランで利用可能です。Team および Enterprise 管理者は、まず [管理設定](#) で Claude Code を有効にする必要があります。

Remote Control は [claude.ai/code](#) または [iOS](#) および [Android](#) 用の Claude アプリをマシン上で実行されている Claude Code セッションに接続します。デスクで作業を開始し、ソファのスマートフォンまたは別のコンピュータのブラウザから続行できます。

マシン上で Remote Control セッションを開始すると、Claude はローカルで実行され続けるため、何もクラウドに移動しません。Remote Control を使用すると、以下のことができます。

- **ローカル環境全体をリモートで使用する:** ファイルシステム、[MCP servers](#)、ツール、プロジェクト設定がすべて利用可能なままです
- **複数のサーフェスから同時に作業する:** 会話はすべての接続されたデバイス間で同期されるため、ターミナル、ブラウザ、スマートフォンから相互に交換可能にメッセージを送信できます
- **中断に対応する:** ラップトップがスリープ状態になったり、ネットワークが切断されたりした場合、マシンがオンラインに戻ると、セッションは自動的に再接続されます

クラウドインフラストラクチャで実行される [Web 上の Claude Code](#) とは異なり、Remote Control セッションはマシン上で直接実行され、ローカルファイルシステムと相互作用します。Web およびモバイルインターフェースは、そのローカルセッションへのウィンドウにすぎません。

Note:

Remote Control には Claude Code v2.1.51 以降が必要です。 `claude --version` でバージョンを確認してください。

このページでは、セットアップ、セッションの開始と接続方法、および Remote Control と Web 上の Claude Code の比較について説明します。

要件

Remote Control を使用する前に、環境が以下の条件を満たしていることを確認してください。

- **サブスクリプション**: Pro、Max、Team、および Enterprise プランで利用可能です。Team および Enterprise 管理者は、まず [管理設定](#) で Claude Code を有効にする必要があります。API キーはサポートされていません。
- **認証**: `claude` を実行し、まだサインインしていない場合は `/login` を使用して claude.ai 経由でサインインします。
- **ワークスペース信頼**: プロジェクトディレクトリで少なくとも 1 回 `claude` を実行して、ワークスペース信頼ダイアログを受け入れます。

Remote Control セッションを開始する

新しいセッションを Remote Control で直接開始するか、既に実行中のセッションに接続できます。

新しいセッション

プロジェクトディレクトリに移動して、以下を実行します。

```
claude remote-control
```

プロセスはターミナルで実行され続け、リモート接続を待機します。[別のデバイスから接続](#)するために使用できるセッション URL が表示され、スペースバーを押してスマートフォンからの高速アクセス用の QR コードを表示できます。リモートセッションがアクティブな間、ターミナルは接続ステータスとツールアクティビティを表示します。

このコマンドは以下のフラグをサポートしています。

- **--name "My Project"**: claude.ai/code のセッションリストに表示されるカスタムセッションタイトルを設定します。名前を位置引数として渡すこともできます。
`claude remote-control "My Project"`
- **--verbose**: 詳細な接続とセッションログを表示します
- **--sandbox / --no-sandbox**: セッション中のファイルシステムとネットワーク分離のための [サンドボックス](#) を有効または無効にします。サンドボックスはデフォルトでオフです。

既存のセッションから

既に Claude Code セッションにいて、それをリモートで続行したい場合は、`/remote-control`（または `/rc`）コマンドを使用します。

```
/remote-control
```

カスタムセッションタイトルを設定するために、引数として名前を渡します。

```
/remote-control My Project
```

これにより、現在の会話履歴を引き継ぎ、[別のデバイスから接続](#)するために使用できるセッション URL と QR コードを表示する Remote Control セッションが開始されます。`--verbose`、`--sandbox`、および `--no-sandbox` フラグはこのコマンドでは利用できません。

別のデバイスから接続する

Remote Control セッションがアクティブになったら、別のデバイスから接続するいくつかの方法があります。

- **セッション URL を開く** 任意のブラウザで claude.ai/code のセッションに直接移動します。`claude remote-control` と `/remote-control` の両方がターミナルにこの URL を表示します。
- **QR コードをスキャン** セッション URL の横に表示される QR コードをスキャンして、Claude アプリで直接開きます。`claude remote-control` を使用する場合は、スペースバーを押して QR コード表示を切り替えます。
- **claude.ai/code または Claude アプリを開く** セッションリストで名前でセッションを見つけます。Remote Control セッションはオンラインの場合、コンピュータアイコンと緑色のステータスドットを表示します。

リモートセッションは、`--name` 引数（または `/remote-control` に渡された名前）、最後のメッセージ、`/rename` 値、または会話履歴がない場合は「Remote Control session」から名前を取得します。環境に既にアクティブなセッションがある場合は、それを続行するか新しいセッションを開始するかを尋ねられます。

Claude アプリをまだ持っていない場合は、Claude Code 内で `/mobile` コマンドを使用して、[iOS](#) または [Android](#) のダウンロード QR コードを表示します。

すべてのセッションで Remote Control を有効にする

デフォルトでは、Remote Control は `claude remote-control` または `/remote-control` を明示的に実行した場合にのみアクティブになります。すべてのセッションで自動的に有効にするには、Claude Code 内で `/config` を実行し、**Enable Remote Control for all sessions** を `true` に設定します。無効にするには `false` に設定します。

各 Claude Code インスタンスは、一度に 1 つのリモートセッションをサポートします。複数のインスタンスを実行する場合、各インスタンスは独自の環境とセッションを取得します。

接続とセキュリティ

ローカル Claude Code セッションは、アウトバウンド HTTPS リクエストのみを行い、マシン上のインバウンドポートを開くことはありません。Remote Control を開始すると、Anthropic API に登録され、作業をポーリングします。別のデバイスから接続すると、サーバーは Web またはモバイルクライアントとローカルセッション間のメッセージをストリーミング接続経由でルーティングします。

すべてのトラフィックは TLS 経由で Anthropic API を通過します。これは、任意の Claude Code セッションと同じトランスポートセキュリティです。接続は複数の短命の認証情報を使用し、各認証情報は単一の目的にスコープされ、独立して有効期限が切れます。

Remote Control と Web 上の Claude Code

Remote Control と [Web 上の Claude Code](#) の両方が `claude.ai/code` インターフェースを使用します。主な違いは、セッションが実行される場所です。Remote Control はマシン上で実行されるため、ローカル MCP サーバー、ツール、プロジェクト設定が利用可能なままです。Web 上の Claude Code は Anthropic 管理のクラウドインフラストラクチャで実行されます。

ローカル作業の途中で別のデバイスから続行したい場合は Remote Control を使用します。ローカルセットアップなしでタスクを開始したい場合、クローンしていないリポジトリで作業したい場合、または複数のタスクを並行して実行したい場合は Web 上の Claude Code を使用します。

制限事項

- ・一度に 1 つのリモートセッション: 各 Claude Code セッションは 1 つのリモート接続をサポートします。

- **ターミナルを開いたままにする必要があります:** Remote Control はローカルプロセスとして実行されます。ターミナルを閉じるか `claude` プロセスを停止すると、セッションは終了します。新しいセッションを開始するには、`claude remote-control` を再度実行します。
- **長時間のネットワーク障害:** マシンが起動しているがおよそ 10 分以上ネットワークに到達できない場合、セッションはタイムアウトしてプロセスは終了します。新しいセッションを開始するには、`claude remote-control` を再度実行します。

関連リソース

- [Web 上の Claude Code](#): マシン上ではなく Anthropic 管理のクラウド環境でセッションを実行します
- [認証](#): claude.ai の `/login` をセットアップし、認証情報を管理します
- [CLI リファレンス](#): `claude remote-control` を含むフラグとコマンドの完全なリスト
- [セキュリティ](#): Remote Control セッションが Claude Code セキュリティモデルにどのように適合するか
- [データ使用](#): ローカルおよびリモートセッション中に Anthropic API を通過するデータ

スケジュールに従ってプロンプトを実行する

/loop と cron スケジューリングツールを使用して、Claude Code セッション内でプロンプトを繰り返し実行したり、ステータスをポーリングしたり、1 回限りのリマインダーを設定したりします。

Note:

スケジュール済みタスクには Claude Code v2.1.72 以降が必要です。 `claude --version` でバージョンを確認してください。

スケジュール済みタスクを使用すると、Claude は一定の間隔でプロンプトを自動的に再実行できます。デプロイメントをポーリングしたり、PR を監視したり、長時間実行されるビルドをチェックバックしたり、後でセッション内で何かを実行するようにリマインダーを設定したりするために使用します。

タスクはセッションスコープです。現在の Claude Code プロセスに存在し、終了すると消えます。再起動後も存在し、アクティブなターミナルセッションなしで実行される永続的なスケジューリングについては、[Desktop スケジュール済みタスク](#) または [GitHub Actions](#) を参照してください。

/loop で定期的なプロンプトをスケジュールする

`/loop` [バンドルスキル](#) は、定期的なプロンプトをスケジュールする最速の方法です。オプションの間隔とプロンプトを渡すと、Claude はバックグラウンドで実行される cron ジョブをセットアップし、セッションは開いたままになります。

```
/loop 5m check if the deployment finished and tell me what happened
```

Claude は間隔を解析し、cron 式に変換し、ジョブをスケジュールし、頻度とジョブ ID を確認します。

間隔構文

間隔はオプションです。先頭に配置したり、末尾に配置したり、完全に省略したりできます。

形式	例	解析された間隔
先頭トークン	<code>/loop 30m check the build</code>	30 分ごと
末尾の <code>every</code> 句	<code>/loop check the build every 2 hours</code>	2 時間ごと
間隔なし	<code>/loop check the build</code>	デフォルトは 10 分ごと

サポートされている単位は、秒の場合は `s`、分の場合は `m`、時間の場合は `h`、日の場合は `d` です。cron は 1 分の粒度を持つため、秒は最も近い分に切り上げられます。`7m` や `90m` など、単位に均等に分割されない間隔は、最も近いクリーンな間隔に丸められ、Claude が選択したものを通知します。

別のコマンドをループする

スケジュール済みプロンプト自体がコマンドまたはスキル呼び出しである可能性があります。これは、既にパッケージ化したワークフローを再実行するのに便利です。

```
/loop 20m /review-pr 1234
```

ジョブが実行されるたびに、Claude は `/review-pr 1234` を入力したかのように実行します。

1 回限りのリマインダーを設定する

1 回限りのリマインダーの場合は、`/loop` を使用する代わりに、自然言語で実行したい内容を説明してください。Claude は実行後に自身を削除する単一実行タスクをスケジュールします。

```
remind me at 3pm to push the release branch
```

```
in 45 minutes, check whether the integration tests passed
```

Claude は cron 式を使用して火災時間を特定の分と時間に固定し、いつ実行されるかを確認します。

スケジュール済みタスクを管理する

自然言語で Claude にタスクをリストまたはキャンセルするよう依頼するか、基盤となるツールを直接参照してください。

```
what scheduled tasks do I have?
```

```
cancel the deploy check job
```

内部的には、Claude はこれらのツールを使用します。

ツール	目的
<code>CronCreate</code>	新しいタスクをスケジュールします。5 フィールド cron 式、実行するプロンプト、および繰り返すか 1 回実行するかを受け入れます。
<code>CronList</code>	ID、スケジュール、プロンプトを含むすべてのスケジュール済みタスクをリストします。
<code>CronDelete</code>	ID でタスクをキャンセルします。

各スケジュール済みタスクには、`CronDelete` に渡すことができる 8 文字の ID があります。セッションは一度に最大 50 個のスケジュール済みタスクを保持できます。

スケジュール済みタスクの実行方法

スケジューラは毎秒期限切れのタスクをチェックし、低優先度でキューに入れます。スケジュール済みプロンプトは、Claude が応答の途中ではなく、ターン間で実行されます。タスクが期限切れになったときに Claude がビジーの場合、プロンプトは現在のターンが終了するまで待機します。

すべての時間はローカルタイムゾーンで解釈されます。`09***` のような cron 式は、UTC ではなく、Claude Code を実行している場所の午前 9 時を意味します。

ジッター

すべてのセッションが同じ壁時計の瞬間に API にヒットするのを避けるために、スケジューラは火災時間に小さな決定論的オフセットを追加します。

- 定期的なタスクは、その期間の最大 10% 遅く実行され、15 分でキャップされます。時間ごとのジョブは `:00` から `:06` のどこかで実行される可能性があります。

- ・時間の上部または下部にスケジュールされた 1 回限りのタスクは、最大 90 秒早く実行されます。

オフセットはタスク ID から派生しているため、同じタスクは常に同じオフセットを取得します。正確なタイミングが重要な場合は、`0 9 * * *` の代わりに `3 9 * * *` など、`:00` または `:30` ではない分を選択してください。1 回限りのジッターは適用されません。

3 日間の有効期限

定期的なタスクは作成後 3 日で自動的に期限切れになります。タスクは最後に 1 回実行され、その後自身を削除します。これにより、忘れられたループが実行できる期間が制限されます。定期的なタスクをより長く続ける必要がある場合は、期限切れになる前にキャンセルして再作成するか、永続的なスケジューリングのために [Desktop スケジュール済みタスク](#) を使用してください。

Cron 式リファレンス

`CronCreate` は標準 5 フィールド cron 式を受け入れます。`minute hour day-of-month month day-of-week`。すべてのフィールドは、ワイルドカード（`*`）、単一値（`5`）、ステップ（`*/15`）、範囲（`1-5`）、およびカンマ区切りリスト（`1,15,30`）をサポートしています。

例	意味
<code>* / 5 * * * *</code>	5 分ごと
<code>0 * * * *</code>	毎時間、毎時間
<code>7 * * * *</code>	毎時間 7 分過ぎ
<code>0 9 * * *</code>	毎日午前 9 時（ローカル）
<code>0 9 * * 1-5</code>	平日午前 9 時（ローカル）
<code>30 14 15 3 *</code>	3 月 15 日午後 2 時 30 分（ローカル）

曜日は日曜日の場合は `0` または `7`、土曜日の場合は `6` を使用します。`L`、`W`、`?` などの拡張構文、および `MON` や `JAN` などの名前エイリアスはサポートされていません。

月の日と曜日の両方が制約されている場合、どちらかのフィールドが一致すれば日付が一致します。これは標準の vixie-cron セマンティクスに従います。

スケジュール済みタスクを無効にする

環境で `CLAUDE_CODE_DISABLE_CRON=1` を設定して、スケジューラ全体を無効にします。cron ツールと `/loop` は利用できなくなり、既にスケジュール済みのタスクは実行を停止します。無効化フラグの完全なリストについては、[環境変数](#) を参照してください。

制限事項

セッションスコープのスケジューリングには固有の制約があります。

- タスクは Claude Code が実行中でアイドル状態の場合にのみ実行されます。ターミナルを閉じるか、セッションを終了すると、すべてがキャンセルされます。
- 見落とされた火災のキャッチアップはありません。タスクのスケジュール済み時間が Claude が長時間実行されるリクエストでビジーの間に経過した場合、Claude がアイドル状態になったときに 1 回実行され、見落とされた間隔ごとに 1 回ではありません。
- 再起動間での永続性はありません。Claude Code を再起動すると、すべてのセッションスコープのタスクがクリアされます。

無人で実行する必要がある cron 駆動オートメーションの場合は、`schedule` トリガーを使用した [GitHub Actions ワークフロー](#)、またはグラフィカルセットアップフローが必要な場合は [Desktop スケジュール済みタスク](#) を使用してください。

コードレビュー

マルチエージェント分析を使用してフルコードベースを検査し、ロジックエラー、セキュリティ脆弱性、リグレッションを検出する自動化された PR レビューをセットアップします

Note:

Code Review はリサーチプレビュー段階であり、[Teams](#) および [Enterprise](#) サブスクリプションで利用可能です。[Zero Data Retention](#) が有効になっている組織では利用できません。

Code Review は GitHub プルリクエストを分析し、問題が見つかったコード行にインラインコメントとして結果を投稿します。特化したエージェントのフリートがフルコードベースのコンテキストでコード変更を検査し、ロジックエラー、セキュリティ脆弱性、壊れたエッジケース、微妙なリグレッションを探します。

結果は重大度でタグ付けされ、PR を承認またはブロックしないため、既存のレビューワークフローはそのまま機能します。リポジトリに `CLAUDE.md` または `REVIEW.md` ファイルを追加することで、Claude がフラグを立てる内容をチューニングできます。

独自の CI インフラストラクチャでこのマネージドサービスの代わりに Claude を実行する場合は、[GitHub Actions](#) または [GitLab CI/CD](#) を参照してください。

このページでは以下をカバーしています：

- [レビューの仕組み](#)
- [セットアップ](#)
- [`CLAUDE.md` と `REVIEW.md` を使用した[レビューのカスタマイズ](#)]
- [料金](#)

レビューの仕組み

管理者が組織の Code Review を [有効にする](#) と、プルリクエストが開かれたり更新されたりするたびにレビューが自動的に実行されます。複数のエージェントが Anthropic インフラストラクチャ上で並行してディフと周囲のコードを分析します。各エージェントは異なるクラスの問題を探し、その後検証ステップが候補を実際のコード動作に対してチェッ

くして、偽陽性を除外します。結果は重複排除され、重大度でランク付けされ、問題が見つかった特定の行にインラインコメントとして投稿されます。問題が見つからない場合、Claude は PR に短い確認コメントを投稿します。

レビューはコストが PR のサイズと複雑さに応じてスケーリングされ、平均 20 分で完了します。管理者は[分析ダッシュボード](#)を通じてレビューアクティビティと支出を監視できます。

重大度レベル

各結果は重大度レベルでタグ付けされます：

マーカー	重大度	意味
	Normal	マージ前に修正すべきバグ
	Nit	軽微な問題、修正する価値があるがブロックングではない
	Pre-existing	コードベースに存在するが、この PR で導入されなかったバグ

結果には展開可能な拡張推論セクションが含まれており、Claude が問題をフラグした理由と問題をどのように検証したかを理解するために展開できます。

Code Review がチェックする内容

デフォルトでは、Code Review は正確性に焦点を当てています：フォーマット設定の好みやテストカバレッジの欠落ではなく、本番環境を壊すバグです。リポジトリに[ガイドンスファイルを追加](#)することで、チェック対象を拡張できます。

Code Review をセットアップする

管理者は組織に対して Code Review を 1 回有効にし、含めるリポジトリを選択します。

Step 1: Claude Code 管理設定を開く

claude.ai/admin-settings/claude-code にアクセスして、Code Review セクションを見つけてください。Claude 組織への管理者アクセスと GitHub 組織に GitHub Apps をインストールする権限が必要です。

Step 2: セットアップを開始する

Setup をクリックします。これにより GitHub App インストールフローが開始されます。

Step 3: Claude GitHub App をインストールする

プロンプトに従って、Claude GitHub App を GitHub 組織にインストールします。アプリは以下のリポジトリ権限をリクエストします：

- **Contents:** 読み取りと書き込み
- **Issues:** 読み取りと書き込み
- **Pull requests:** 読み取りと書き込み

Code Review は contents への読み取りアクセスと pull requests への書き込みアクセスを使用します。より広い権限セットは、後で有効にする場合は [GitHub Actions](#) もサポートします。

Step 4: リポジトリを選択する

Code Review を有効にするリポジトリを選択します。リポジトリが表示されない場合は、インストール中に Claude GitHub App にアクセス権を付与したことを確認してください。後でリポジトリを追加できます。

Step 5: リポジトリごとにレビュートリガーを設定する

セットアップが完了すると、Code Review セクションはリポジトリをテーブルに表示します。各リポジトリについて、ドロップダウンを使用してレビューが実行される時期を選択します：

- **After PR creation only:** PR が開かれたか ready for review とマークされたときにレビューが 1 回実行されます
- **After every push to PR branch:** すべてのプッシュでレビューが実行され、PR が進化するにつれて新しい問題をキャッチし、フラグが立てられた問題を修正するとスレッドを自動的に解決します

すべてのプッシュでレビューするとより多くのレビューが実行され、コストが増加します。PR 作成のみで開始し、継続的なカバレッジと自動スレッドクリーンアップが必要なリポジトリではオンプッシュに切り替えます。

リポジトリテーブルは、最近のアクティビティに基づいて各リポジトリのレビューあたりの平均コストも表示します。行アクションメニューを使用して、リポジトリごとに Code Review をオンまたはオフにするか、リポジトリを完全に削除します。

セットアップを確認するには、テスト PR を開きます。**Claude Code Review** という名前のチェック実行が数分以内に表示されます。表示されない場合は、リポジトリが管理設定にリストされていることと、Claude GitHub App がそれにアクセスできることを確認してください。

レビューをカスタマイズする

Code Review はリポジトリから 2 つのファイルを読み取り、フラグを立てる内容をガイドします。どちらもデフォルトの正確性チェックの上に追加されます：

- **CLAUDE.md** : Claude Code がレビューだけでなくすべてのタスクに使用する共有プロジェクト指示。ガイダンスが対話的な Claude Code セッションにも適用される場合に使用します。
- **REVIEW.md** : レビューのみのガイダンス、コードレビュー中に排他的に読み取られます。レビュー中にフラグを立てるか、スキップするかについての厳密なルールに使用し、一般的な **CLAUDE.md** を乱雑にしないようにします。

CLAUDE.md

Code Review はリポジトリの **CLAUDE.md** ファイルを読み取り、新しく導入された違反を nit レベルの結果として扱います。これは双方向に機能します：PR が **CLAUDE.md** ステートメントを古くする方法でコードを変更する場合、Claude はドキュメントを更新する必要がありますことをフラグします。

Claude はディレクトリ階層のすべてのレベルで **CLAUDE.md** ファイルを読み取るため、サブディレクトリの **CLAUDE.md** のルールはそのパスの下ファイルにのみ適用されます。

CLAUDE.md の仕組みの詳細については、[メモリドキュメント](#)を参照してください。

一般的な Claude Code セッションに適用したくないレビュー固有のガイダンスについては、代わりに **REVIEW.md** を使用します。

REVIEW.md

レビュー固有のルールについては、リポジトリルートに **REVIEW.md** ファイルを追加します。以下をエンコードするために使用します：

- 会社またはチームのスタイルガイドライン：「ネストされた条件付きよりも早期リターンを優先する」
- リンターでカバーされていない言語またはフレームワーク固有の規約
- Claude が常にフラグを立てるべきもの：「新しい API ルートには対応する統合テストが必要」
- Claude がスキップすべきもの：「`/gen/` の下の生成されたコードのフォーマットについてコメントしない」

REVIEW.md の例：

```
## Code Review Guidelines

### Always check
- New API endpoints have corresponding integration tests
- Database migrations are backward-compatible
- Error messages don't leak internal details to users

### Style
- Prefer `match` statements over chained `isinstance` checks
- Use structured logging, not f-string interpolation in log calls

### Skip
- Generated files under `src/gen/`
- Formatting-only changes in `*.lock` files
```

Claude はリポジトリルートで **REVIEW.md** を自動検出します。設定は不要です。

使用状況を表示する

claude.ai/analytics/code-review にアクセスして、組織全体の Code Review アクティビティを確認します。ダッシュボードは以下を表示します：

セクション	表示内容
PRs reviewed	選択した時間範囲にわたってレビューされたプルリクエストの日次カウント
Cost weekly	Code Review の週次支出
Feedback	開発者が問題に対処したため自動的に解決されたレビューコメントのカウント
Repository breakdown	リポジトリごとのレビューされた PR とコメント解決のカウント

管理設定のリポジトリテーブルは、各リポジトリのレビューあたりの平均コストも表示します。

料金

Code Review はトークン使用量に基づいて請求されます。レビューは平均 \$15〜25 で、PR サイズ、コードベースの複雑さ、検証が必要な問題の数に応じてスケーリングされます。Code Review の使用は [extra usage](#) を通じて個別に請求され、プランの含まれた使用量にはカウントされません。

選択するレビュートリガーは総コストに影響します：

- **After PR creation only:** PR ごとに 1 回実行されます
- **After every push:** 各コミットで実行され、プッシュ数でコストが乗算されます

コストは、組織が他の Claude Code 機能に AWS Bedrock または Google Vertex AI を使用しているかどうかに関わらず、Anthropic 請求書に表示されます。Code Review の月次支出上限を設定するには、[claude.ai/admin-settings/usage](#) にアクセスして、Claude Code Review サービスの制限を設定します。

[分析](#)の週次コストチャートまたは管理設定のリポジトリごとの平均コスト列を通じて支出を監視します。

関連リソース

Code Review は Claude Code の残りの部分と連携するように設計されています。PR を開く前にローカルでレビューを実行したい場合、自己ホスト型セットアップが必要な場合、または **CLAUDE.md** がツール全体で Claude の動作をどのように形作るかについてさらに詳しく知りたい場合、これらのページは次の良い選択肢です：

- [Plugins](#): プラグインマーケットプレイスを参照します。プッシュ前にローカルでオンデマンドレビューを実行するための **code-review** プラグインを含みます
- [GitHub Actions](#): カスタムオートメーション用に独自の GitHub Actions ワークフローで Claude を実行します
- [GitLab CI/CD](#): GitLab パイプライン用の自己ホスト型 Claude 統合
- [Memory](#): Claude Code 全体で **CLAUDE.md** ファイルがどのように機能するか
- [Analytics](#): コードレビュー以外の Claude Code 使用状況を追跡します

Part 7: CI/CD & Integrations

Claude Code GitHub Actions

Claude Code を開発ワークフローに統合する Claude Code GitHub Actions について学びます

Claude Code GitHub Actions は、GitHub ワークフローに AI を活用した自動化をもたらします。任意の PR またはイシューで `@claude` とメンションするだけで、Claude はコードを分析し、プルリクエストを作成し、機能を実装し、バグを修正できます。すべてプロジェクトの標準に従いながら実行されます。すべての PR に自動的に投稿されるレビューについては、[GitHub Code Review](#) を参照してください。

Note:

Claude Code GitHub Actions は [Claude Agent SDK](#) の上に構築されており、Claude Code をアプリケーションにプログラマ的に統合できます。SDK を使用して、GitHub Actions を超えたカスタム自動化ワークフローを構築できます。

Info:

Claude Opus 4.6 が利用可能になりました。 Claude Code GitHub Actions はデフォルトで Sonnet を使用します。Opus 4.6 を使用するには、[model パラメータ](#) を `claude-opus-4-6` に設定してください。

Claude Code GitHub Actions を使用する理由

- **即座の PR 作成:** 必要なことを説明すると、Claude は必要なすべての変更を含む完全な PR を作成します
- **自動コード実装:** イシューを 1 つのコマンドで動作するコードに変換します
- **標準に従う:** Claude は `CLAUDE.md` ガイドラインと既存のコードパターンを尊重します
- **シンプルなセットアップ:** インストーラーと API キーで数分で開始できます
- **デフォルトで安全:** コードは Github のランナーに留まります

Claude は何ができますか？

Claude Code は、コードの操作方法を変える強力な GitHub Action を提供します。

Claude Code Action

この GitHub Action により、GitHub Actions ワークフロー内で Claude Code を実行できます。Claude Code の上に任意のカスタムワークフローを構築するために使用できます。

[リポジトリを表示 →](#)

セットアップ

クイックセットアップ

このアクションをセットアップする最も簡単な方法は、ターミナルで Claude Code を使用することです。claude を開いて `/install-github-app` を実行するだけです。

このコマンドは、GitHub アプリと必要なシークレットのセットアップをガイドします。

Note:

- GitHub アプリをインストールしてシークレットを追加するには、リポジトリ管理者である必要があります
- GitHub アプリは、Contents、Issues、Pull requests に対する読み取りと書き込みのアクセス許可をリクエストします
- このクイックスタート方法は、直接 Claude API ユーザーのみが利用できます。AWS Bedrock または Google Vertex AI を使用している場合は、[AWS Bedrock と Google Vertex AI での使用](#) セクションを参照してください。

手動セットアップ

`/install-github-app` コマンドが失敗した場合、または手動セットアップを希望する場合は、以下の手動セットアップ手順に従ってください。

1. **Claude GitHub アプリをリポジトリにインストール:** <https://github.com/apps/claude>

Claude GitHub アプリには、以下のリポジトリアクセス許可が必要です。

- **Contents:** 読み取りと書き込み（リポジトリファイルを変更するため）
- **Issues:** 読み取りと書き込み（イシューに応答するため）
- **Pull requests:** 読み取りと書き込み（PR を作成して変更をプッシュするため）

セキュリティとアクセス許可の詳細については、[セキュリティドキュメント](#)を参照してください。

2. `ANTHROPIC_API_KEY` をリポジトリシークレットに追加 ([GitHub Actions でシークレットを使用する方法を学ぶ](#))
3. ワークフローファイルをコピー [examples/claude.yml](#) からリポジトリの `.github/workflows/` にコピーします

Tip:

クイックスタートまたは手動セットアップのいずれかを完了した後、イシューまたは PR コメントで `@claude` をタグ付けしてアクションをテストします。

ベータ版からのアップグレード

Warning:

Claude Code GitHub Actions v1.0 は、ベータ版から v1.0 にアップグレードするためにワークフローファイルを更新する必要がある破壊的な変更を導入しています。

現在 Claude Code GitHub Actions のベータ版を使用している場合は、ワークフローを GA バージョンを使用するように更新することをお勧めします。新しいバージョンは、自動モード検出などの強力な新機能を追加しながら、設定を簡素化します。

重要な変更

すべてのベータユーザーは、アップグレードするためにワークフローファイルで以下の変更を行う必要があります。

1. **アクションバージョンを更新:** `@beta` を `@v1` に変更します
2. **モード設定を削除:** `mode: "tag"` または `mode: "agent"` を削除します（現在は自動検出）
3. **プロンプト入力を更新:** `direct_prompt` を `prompt` に置き換えます
4. **CLI オプションを移動:** `max_turns`、`model`、`custom_instructions`などを `claude_args` に変換します

破壊的な変更リファレンス

古いベータ入力	新しい v1.0 入力
<code>mode</code>	(削除 - 自動検出)

古いベータ入力	新しい v1.0 入力
<code>direct_prompt</code>	<code>prompt</code>
<code>override_prompt</code>	<code>prompt</code> with GitHub variables
<code>custom_instructions</code>	<code>claude_args: --append-system-prompt</code>
<code>max_turns</code>	<code>claude_args: --max-turns</code>
<code>model</code>	<code>claude_args: --model</code>
<code>allowed_tools</code>	<code>claude_args: --allowedTools</code>
<code>disallowed_tools</code>	<code>claude_args: --disallowedTools</code>
<code>claude_env</code>	<code>settings</code> JSON format

前後の例

ベータ版:

```
- 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 バージョン (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:

アクションは、設定に基づいて、インタラクティブモード（[@claude](#) メンションに応答）または自動化モード（プロンプト付きで即座に実行）で実行するかどうかを自動的に検出します。

使用例

Claude Code GitHub Actions は、さまざまなタスクに役立ちます。[examples ディレクトリ](#) には、さまざまなシナリオ用の使用可能なワークフローが含まれています。

基本的なワークフロー

```
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
```

skills を使用する

```
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"
```

プロンプトを使用したカスタム自動化

```
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"
```

一般的な使用例

イシューまたは PR コメント内：

```
@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 は自動的にコンテキストを分析し、適切に応答します。

ベストプラクティス

CLAUDE.md 設定

リポジトリルートに **CLAUDE.md** ファイルを作成して、コードスタイルガイドライン、レビュー基準、プロジェクト固有のルール、および推奨パターンを定義します。このファイルは、Claude のプロジェクト標準の理解をガイドします。

セキュリティに関する考慮事項

Warning:

API キーをリポジトリに直接コミットしないでください。

アクセス許可、認証、ベストプラクティスを含む包括的なセキュリティガイダンスについては、[Claude Code Action セキュリティドキュメント](#) を参照してください。

常に GitHub Secrets を API キーに使用します。

- API キーを **ANTHROPIC_API_KEY** という名前のリポジトリシークレットとして追加します
- ワークフローで参照します: **anthropic_api_key: \${ secrets.ANTHROPIC_API_KEY }**
- アクションのアクセス許可を必要なものだけに制限します
- マージする前に Claude の提案を確認します

常に GitHub Secrets（例えば、**\${ secrets.ANTHROPIC_API_KEY }**）を使用し、API キーをワークフローファイルに直接ハードコードしないでください。

パフォーマンスの最適化

イシューテンプレートを使用してコンテキストを提供し、**CLAUDE.md** を簡潔で焦点を絞ったものに保ち、ワークフローに適切なタイムアウトを設定します。

CI コスト

Claude Code GitHub Actions を使用する場合、関連するコストに注意してください。

GitHub Actions コスト:

- Claude Code は GitHub ホストランナーで実行され、GitHub Actions の分を消費します
- 詳細な価格設定と分の制限については、[GitHub の請求ドキュメント](#) を参照してください

API コスト:

- 各 Claude インタラクションは、プロンプトと応答の長さに基づいて API トークンを消費します
- トークン使用量は、タスクの複雑さとコードベースのサイズによって異なります
- 現在のトークンレートについては、[Claude の価格ページ](#) を参照してください

コスト最適化のヒント:

- 特定の `@claude` コマンドを使用して、不要な API 呼び出しを減らします
- `claude_args` で適切な `--max-turns` を設定して、過度な反復を防ぎます
- ワークフローレベルのタイムアウトを設定して、暴走ジョブを回避します
- GitHub の並行制御を使用して、並列実行を制限することを検討します

設定例

Claude Code Action v1 は、統一されたパラメータで設定を簡素化します。

```
- 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
```

主な機能：

- **統一されたプロンプトインターフェース** - すべての指示に `prompt` を使用します
- **Skills** - インストール済みの `skills` をプロンプトから直接呼び出します
- **CLI パススルー** - `claude_args` 経由の任意の Claude Code CLI 引数
- **柔軟なトリガー** - 任意の GitHub イベントで動作します

完全なワークフローファイルについては、[examples ディレクトリ](#) を参照してください。

Tip:

イシューまたは PR コメントに応答する場合、Claude は自動的に @claude メンションに応答します。その他のイベントについては、**prompt** パラメータを使用して指示を提供します。

AWS Bedrock と Google Vertex AI での使用

エンタープライズ環境では、Claude Code GitHub Actions を独自のクラウドインフラストラクチャで使用できます。このアプローチにより、データレジデンシーと請求を制御しながら、同じ機能を維持できます。

前提条件

クラウドプロバイダーで Claude Code GitHub Actions をセットアップする前に、以下が必要です。

Google Cloud Vertex AI の場合：

1. Vertex AI が有効な Google Cloud プロジェクト
2. GitHub Actions 用に設定された Workload Identity Federation
3. 必要なアクセス許可を持つサービスアカウント
4. GitHub App（推奨）または デフォルトの GITHUB_TOKEN を使用

AWS Bedrock の場合：

1. Amazon Bedrock が有効な AWS アカウント
2. AWS で設定された GitHub OIDC Identity Provider
3. Bedrock アクセス許可を持つ IAM ロール
4. GitHub App（推奨）または デフォルトの GITHUB_TOKEN を使用

Step 1: カスタム GitHub App を作成（3P プロバイダーに推奨）

Vertex AI や Bedrock などの 3P プロバイダーを使用する場合、最適な制御とセキュリティのために、独自の GitHub App を作成することをお勧めします。

1. <https://github.com/settings/apps/new> にアクセスします
 2. 基本情報を入力します。
- **GitHub App name:** 一意の名前を選択します（例：‘YourOrg Claude Assistant’）

- **Homepage URL:** 組織の Web サイトまたはリポジトリ URL

1. アプリ設定を設定します。

- **Webhooks:** 'Active' をオフにします（この統合には不要）

1. 必要なアクセス許可を設定します。

- **Repository permissions:**

- Contents: Read & Write
- Issues: Read & Write
- Pull requests: Read & Write

1. 'Create GitHub App' をクリックします

2. 作成後、'Generate a private key' をクリックしてダウンロードした `.pem` ファイルを保存します

3. アプリ設定ページからアプリ ID をメモします

4. アプリをリポジトリにインストールします。

- アプリの設定ページから、左側のサイドバーの 'Install App' をクリックします
- アカウントまたは組織を選択します
- 'Only select repositories' を選択して、特定のリポジトリを選択します
- 'Install' をクリックします

1. プライベートキーをリポジトリシークレットとして追加します。

- リポジトリの Settings → Secrets and variables → Actions に移動します
- `.pem` ファイルの内容を含む `APP_PRIVATE_KEY` という名前の新しいシークレットを作成します

1. アプリ ID をシークレットとして追加します。

- GitHub App の ID を含む `APP_ID` という名前の新しいシークレットを作成します

Note:

このアプリは [actions/create-github-app-token](#) アクションで使用され、ワークフロー内で認証トークンを生成します。

Claude API の場合、または独自の Github アプリをセットアップしたくない場合の代替案: 公式 Anthropic アプリを使用します。

1. <https://github.com/apps/claude> からインストールします

2. 認証に追加の設定は不要です

Step 2: クラウドプロバイダー認証を設定

クラウドプロバイダーを選択し、安全な認証をセットアップします。

GitHub Actions が認証情報を保存せずに安全に認証できるように AWS を設定します。

セキュリティに関する注意: リポジトリ固有の設定を使用し、最小限の必要なアクセス許可のみを付与します。

必要なセットアップ:

1. Amazon Bedrock を有効にします:

- Amazon Bedrock で Claude モデルへのアクセスをリクエストします
- クロスリージョンモデルの場合、すべての必要なリージョンでアクセスをリクエストします

1. GitHub OIDC Identity Provider をセットアップします:

- Provider URL: `https://token.actions.githubusercontent.com`
- Audience: `sts.amazonaws.com`

1. GitHub Actions 用の IAM ロールを作成します:

- Trusted entity type: Web identity
- Identity provider: `token.actions.githubusercontent.com`
- Permissions: `AmazonBedrockFullAccess` ポリシー
- 特定のリポジトリの信頼ポリシーを設定します

必要な値:

セットアップ後、以下が必要です。

- `AWS_ROLE_TO_ASSUME`: 作成した IAM ロールの ARN

Tip:

OIDC は、認証情報が一時的で自動的にローテーションされるため、静的な AWS アクセスキーを使用するよりも安全です。

詳細な OIDC セットアップ手順については、[AWS ドキュメント](#) を参照してください。

GitHub Actions が認証情報を保存せずに安全に認証できるように Google Cloud を設定します。

セキュリティに関する注意: リポジトリ固有の設定を使用し、最小限の必要なアクセス許可のみを付与します。

必要なセットアップ:

1. Google Cloud プロジェクトで API を有効にします:

- IAM Credentials API
- Security Token Service (STS) API
- Vertex AI API

1. Workload Identity Federation リソースを作成します:

- Workload Identity Pool を作成します
- GitHub OIDC プロバイダーを追加します。
- Issuer: <https://token.actions.githubusercontent.com>
- リポジトリと所有者の属性マッピング
- **セキュリティ推奨:** リポジトリ固有の属性条件を使用します

1. サービスアカウントを作成します:

- **Vertex AI User** ロールのみを付与します
- **セキュリティ推奨:** リポジトリごとに専用のサービスアカウントを作成します

1. IAM バインディングを設定します:

- Workload Identity Pool がサービスアカウントを偽装できるようにします
- **セキュリティ推奨:** リポジトリ固有のプリンシパルセットを使用します

必要な値:

セットアップ後、以下が必要です。

- **GCP_WORKLOAD_IDENTITY_PROVIDER:** 完全なプロバイダーリソース名
- **GCP_SERVICE_ACCOUNT:** サービスアカウントのメールアドレス

Tip:

Workload Identity Federation により、ダウンロード可能なサービスアカウントキーが不要になり、セキュリティが向上します。

詳細なセットアップ手順については、[Google Cloud Workload Identity Federation ドキュメント](#)を参照してください。

Step 3: 必要なシークレットを追加

リポジトリに以下のシークレットを追加します（Settings → Secrets and variables → Actions）：

Claude API（直接）の場合：

1. API 認証の場合：

- `ANTHROPIC_API_KEY` : console.anthropic.com からの Claude API キー

1. GitHub App を使用する場合（独自のアプリを使用している場合）：

- `APP_ID` : GitHub App の ID
- `APP_PRIVATE_KEY` : プライベートキー（.pem）の内容

Google Cloud Vertex AI の場合

1. GCP 認証の場合：

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

1. GitHub App を使用する場合（独自のアプリを使用している場合）：

- `APP_ID` : GitHub App の ID
- `APP_PRIVATE_KEY` : プライベートキー（.pem）の内容

AWS Bedrock の場合

1. AWS 認証の場合：

- `AWS_ROLE_TO_ASSUME`

1. GitHub App を使用する場合（独自のアプリを使用している場合）：

- `APP_ID` : GitHub App の ID
- `APP_PRIVATE_KEY` : プライベートキー（.pem）の内容

Step 4: ワークフローファイルを作成

クラウドプロバイダーと統合する GitHub Actions ワークフローファイルを作成します。以下の例は、AWS Bedrock と Google Vertex AI の両方の完全な設定を示しています。

前提条件：

- AWS Bedrock アクセスが有効で、Claude モデルのアクセス許可がある
- GitHub が AWS で OIDC ID プロバイダーとして設定されている
- Bedrock アクセス許可を持つ IAM ロールが GitHub Actions を信頼している

必要な GitHub シークレット:

Secret Name	Description
AWS_ROLE_TO_ASSUME	Bedrock アクセス用の IAM ロールの ARN
APP_ID	GitHub App ID (アプリ設定から)
APP_PRIVATE_KEY	GitHub App 用に生成したプライベートキー

```

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:

Bedrock のモデル ID 形式には、リージョンプレフィックスが含まれます（例：
`us.anthropic.claude-sonnet-4-6`）。

前提条件:

- GCP プロジェクトで Vertex AI API が有効
- GitHub 用に Workload Identity Federation が設定されている
- Vertex AI アクセス許可を持つサービスアカウント

必要な GitHub シークレット:

Secret Name	Description
<code>GCP_WORKLOAD_IDENTITY_PROVIDER</code>	Workload identity provider リソース名
<code>GCP_SERVICE_ACCOUNT</code>	Vertex AI アクセス権を持つサービスアカウントメール
<code>APP_ID</code>	GitHub App ID（アプリ設定から）
<code>APP_PRIVATE_KEY</code>	GitHub App 用に生成したプライベートキー

```

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:

プロジェクト ID は Google Cloud 認証ステップから自動的に取得されるため、ハードコードする必要はありません。

トラブルシューティング

Claude が @claude コマンドに応答しない

GitHub App が正しくインストールされていることを確認し、ワークフローが有効になっていることを確認し、API キーがリポジトリシークレットに設定されていることを確認し、コメントに @claude が含まれていることを確認します（/claude ではなく）。

CI が Claude のコミットで実行されない

GitHub App またはカスタムアプリを使用していることを確認します（Actions ユーザーではなく）、ワークフロートリガーに必要なイベントが含まれていることを確認し、アプリのアクセス許可に CI トリガーが含まれていることを確認します。

認証エラー

API キーが有効で十分なアクセス許可があることを確認します。Bedrock/Vertex の場合、認証情報の設定を確認し、シークレットがワークフロー内で正しく名前付けされていることを確認します。

高度な設定

アクションパラメータ

Claude Code Action v1 は、簡素化された設定を使用します。

Parameter	Description	Required
<code>prompt</code>	Claude の指示（プレーンテキストまたは <code>skill</code> 名）	No*
<code>claude_args</code>	Claude Code に渡される CLI 引数	No
<code>anthropic_api_key</code>	Claude API キー	Yes**
<code>github_token</code>	API アクセス用の GitHub トークン	No
<code>trigger_phrase</code>	カスタムトリガーフレーズ（デフォルト：「@claude」）	No
<code>use_bedrock</code>	Claude API の代わりに AWS Bedrock を使用	No
<code>use_vertex</code>	Claude API の代わりに Google Vertex AI を使用	No

*プロンプトはオプションです。イシュー/PR コメントで省略された場合、Claude はトリガーフレーズに応答します

**直接 Claude API に必要です。Bedrock/Vertex には不要です

CLI 引数を渡す

`claude_args` パラメータは、任意の Claude Code CLI 引数を受け入れます。

```
claude_args: "--max-turns 5 --model claude-sonnet-4-6 --mcp-config /path/to/config.json"
```

一般的な引数：

- `--max-turns` : 最大会話ターン数（デフォルト：10）
- `--model` : 使用するモデル（例： `claude-sonnet-4-6`）
- `--mcp-config` : MCP 設定へのパス
- `--allowed-tools` : 許可されたツールのカンマ区切りリスト
- `--debug` : デバッグ出力を有効にします

代替統合方法

`/install-github-app` コマンドは推奨されるアプローチですが、以下も実行できます。

- **カスタム GitHub App**: ブランド化されたユーザー名またはカスタム認証フローが必要な組織向け。必要なアクセス許可（contents、issues、pull requests）を持つ独自の GitHub App を作成し、actions/create-github-app-token アクションを使用してワークフロー内でトークンを生成します。
- **手動 GitHub Actions**: 最大の柔軟性のための直接ワークフロー設定
- **MCP 設定**: Model Context Protocol サーバーの動的読み込み

詳細なガイドについては、[Claude Code Action ドキュメント](#) を参照してください。認証、セキュリティ、高度な設定に関する詳細なガイドがあります。

Claude の動作をカスタマイズ

Claude の動作は 2 つの方法で設定できます。

1. **CLAUDE.md**: リポジトリのルートに `CLAUDE.md` ファイルを作成して、コーディング標準、レビュー基準、プロジェクト固有のルールを定義します。Claude は PR を作成し、リクエストに応答するときにこれらのガイドラインに従います。詳細については、[Memory ドキュメント](#) を確認してください。
2. **カスタムプロンプト**: ワークフローファイルの `prompt` パラメータを使用して、ワークフロー固有の指示を提供します。これにより、異なるワークフローまたはタスク用に Claude の動作をカスタマイズできます。

Claude は PR を作成し、リクエストに応答するときにこれらのガイドラインに従います。

Claude Code GitLab CI/CD

Claude Code を GitLab CI/CD で開発ワークフローに統合する方法を学びます

Info:

Claude Code for GitLab CI/CD は現在ベータ版です。機能と機能性は、エクスペリエンスを改善する際に進化する可能性があります。

この統合は GitLab によって保守されています。サポートについては、以下の [GitLab issue](#) を参照してください。

Note:

この統合は [Claude Code CLI and Agent SDK](#) の上に構築されており、CI/CD ジョブとカスタム自動化ワークフローで Claude をプログラマ的に使用できます。

GitLab で Claude Code を使用する理由

- **インスタント MR 作成:** 必要なことを説明すると、Claude は変更と説明を含む完全な MR を提案します
- **自動実装:** 単一のコマンドまたはメンションで issue を実行可能なコードに変換します
- **プロジェクト対応:** Claude は `CLAUDE.md` ガイドラインと既存のコードパターンに従います
- **シンプルなセットアップ:** `.gitlab-ci.yml` に 1 つのジョブとマスクされた CI/CD 変数を追加します
- **エンタープライズ対応:** Claude API、AWS Bedrock、または Google Vertex AI を選択して、データレジデンシーと調達のニーズを満たします
- **デフォルトでセキュア:** GitLab ランナーで実行され、ブランチ保護と承認が適用されます

仕組み

Claude Code は GitLab CI/CD を使用して AI タスクを分離されたジョブで実行し、MR 経由で結果をコミットバックします。

1. **イベント駆動型オーケストレーション:** GitLab は選択したトリガー（例えば、issue、MR、またはレビュースレッドで `@claude` をメンションするコメント）をリッスンします。ジョブはスレッドとリポジトリからコンテキストを収集し、その入力からプロンプトを構築し、Claude Code を実行します。
2. **プロバイダー抽象化:** 環境に適したプロバイダーを使用します。
 - Claude API (SaaS)
 - AWS Bedrock (IAM ベースのアクセス、クロスリージョンオプション)
 - Google Vertex AI (GCP ネイティブ、Workload Identity Federation)
3. **サンドボックス実行:** 各インタラクションは厳密なネットワークとファイルシステムルールを持つコンテナで実行されます。Claude Code はワークスペーススコープの権限を適用して書き込みを制限します。すべての変更は MR を通じてフローするため、レビューアは diff を確認でき、承認が引き続き適用されます。

地域エンドポイントを選択して、既存のクラウド契約を使用しながらレイテンシーを削減し、データソブリンティ要件を満たします。

Claude は何ができますか？

Claude Code は、コードの操作方法を変える強力な CI/CD ワークフローを実現します。

- issue の説明またはコメントから MR を作成および更新します
- パフォーマンス低下を分析し、最適化を提案します
- ブランチに直接機能を実装し、MR を開きます
- テストまたはコメントで特定されたバグと低下を修正します
- フォローアップコメントに応答して、リクエストされた変更を反復処理します

セットアップ

クイックセットアップ

最速で開始する方法は、`.gitlab-ci.yml` に最小限のジョブを追加し、API キーをマスクされた変数として設定することです。

1. **マスクされた CI/CD 変数を追加します**
 - **Settings** → **CI/CD** → **Variables** に移動します
 - `ANTHROPIC_API_KEY` を追加します（マスク、必要に応じて保護）

2. Claude ジョブを `.gitlab-ci.yml` に追加します

```

stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  # ジョブをトリガーする方法に合わせてルールを調整します。
  # - 手動実行
  # - マージリクエストイベント
  # - '@claude' を含むコメント時の web/API トリガー
  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:
    # オプション: セットアップが提供の場合は GitLab MCP サーバーを開始します
    - /bin/gitlab-mcp-server || true
    # web/API トリガーでコンテキストペイロードを使用して呼び出す場合は AI_FLOW_* 変数を使用し
    # ます
    - 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

```

ジョブと `ANTHROPIC_API_KEY` 変数を追加した後、**CI/CD → Pipelines** からジョブを手動で実行してテストするか、MR からトリガーして Claude が変更を提案し、必要に応じて MR を開くようにします。

Note:

Claude API の代わりに AWS Bedrock または Google Vertex AI で実行するには、以下の [Using with AWS Bedrock & Google Vertex AI](#) セクションを参照して、認証と環境セットアップを確認してください。

手動セットアップ（本番環境に推奨）

より制御されたセットアップが必要な場合、またはエンタープライズプロバイダーが必要な場合：

1. プロバイダーアクセスを構成します。

- **Claude API:** `ANTHROPIC_API_KEY` を作成してマスクされた CI/CD 変数として保存します
- **AWS Bedrock:** **Configure GitLab** → **AWS OIDC** を実行し、Bedrock 用の IAM ロールを作成します
- **Google Vertex AI:** **Configure Workload Identity Federation for GitLab** → **GCP** を実行します

2. GitLab API 操作用のプロジェクト認証情報を追加します。

- デフォルトで `CI_JOB_TOKEN` を使用するか、`api` スコープを持つ Project Access Token を作成します
- PAT を使用する場合は `GITLAB_ACCESS_TOKEN`（マスク）として保存します

3. Claude ジョブを `.gitlab-ci.yml` に追加します（以下の例を参照）**4. （オプション）メンション駆動型トリガーを有効にします。**

- プロジェクト webhook を「Comments (notes)」に追加して、イベントリスナーに追加します（使用する場合）
- コメントに `@claude` が含まれている場合、リスナーがパイプライントリガー API を `AI_FLOW_INPUT` や `AI_FLOW_CONTEXT` などの変数で呼び出すようにします

使用例**issue を MR に変換する**

issue コメント内：

```
@claude implement this feature based on the issue description
```

Claude は issue とコードベースを分析し、ブランチに変更を書き込み、レビュー用に MR を開きます。

実装ヘルプを取得する

MR ディスカッション内：

```
@claude suggest a concrete approach to cache the results of this API call
```

Claude は変更を提案し、適切なキャッシングを使用してコードを追加し、MR を更新します。

バグを素早く修正する

issue または MR コメント内：

```
@claude fix the TypeError in the user dashboard component
```

Claude はバグを特定し、修正を実装し、ブランチを更新するか新しい MR を開きます。

AWS Bedrock & Google Vertex AI での使用

エンタープライズ環境では、同じ開発者エクスペリエンスで Claude Code をクラウドインフラストラクチャ全体で実行できます。

AWS Bedrock

前提条件

AWS Bedrock で Claude Code をセットアップする前に、以下が必要です。

1. 目的の Claude モデルへのアクセス権を持つ Amazon Bedrock を備えた AWS アカウント
2. AWS IAM で OIDC ID プロバイダーとして構成された GitLab
3. Bedrock 権限と GitLab プロジェクト/refs に制限された信頼ポリシーを持つ IAM ロール
4. ロール仮定用の GitLab CI/CD 変数：
 - `AWS_ROLE_TO_ASSUME` (ロール ARN)
 - `AWS_REGION` (Bedrock リージョン)

セットアップ手順

GitLab CI ジョブが OIDC 経由で IAM ロールを仮定できるように AWS を構成します（静的キーなし）。

必須セットアップ：

1. Amazon Bedrock を有効にし、ターゲット Claude モデルへのアクセスをリクエストします
2. GitLab 用の IAM OIDC プロバイダーを作成します（まだ存在しない場合）
3. GitLab OIDC プロバイダーによって信頼され、プロジェクトと保護された refs に制限された IAM ロールを作成します
4. Bedrock invoke API に対する最小権限権限を付与します

CI/CD 変数に保存する必須値：

- `AWS_ROLE_TO_ASSUME`
- `AWS_REGION`

Settings → CI/CD → Variables で変数を追加します。

```
## AWS Bedrock の場合：
- AWS_ROLE_TO_ASSUME
- AWS_REGION
```

上記の AWS Bedrock ジョブの例を使用して、GitLab ジョブトークンを実行時に一時的な AWS 認証情報と交換します。

Google Vertex AI

前提条件

Google Vertex AI で Claude Code をセットアップする前に、以下が必要です。

1. 以下を備えた Google Cloud プロジェクト：
 - Vertex AI API が有効
 - GitLab OIDC を信頼するように構成された Workload Identity Federation
1. 必要な Vertex AI ロールのみを持つ専用サービスアカウント
2. WIF 用の GitLab CI/CD 変数：
 - `GCP_WORKLOAD_IDENTITY_PROVIDER`（完全なリソース名）
 - `GCP_SERVICE_ACCOUNT`（サービスアカウントメール）

セットアップ手順

GitLab CI ジョブが Workload Identity Federation 経由でサービスアカウントを偽装できるように Google Cloud を構成します。

必須セットアップ：

1. IAM Credentials API、STS API、および Vertex AI API を有効にします
2. GitLab OIDC 用の Workload Identity Pool とプロバイダーを作成します
3. Vertex AI ロールを持つ専用サービスアカウントを作成します
4. WIF プリンシパルにサービスアカウントを偽装する権限を付与します

CI/CD 変数に保存する必須値：

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

Settings → CI/CD → Variables で変数を追加します。

```
## Google Vertex AI の場合：  
- GCP_WORKLOAD_IDENTITY_PROVIDER  
- GCP_SERVICE_ACCOUNT  
- CLOUD_ML_REGION (例：us-east5)
```

上記の Google Vertex AI ジョブの例を使用して、キーを保存せずに認証します。

構成例

以下は、パイプラインに適応させることができる使用可能なスニペットです。

基本的な .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 は CI/CD 変数から ANTHROPIC_API_KEY を使用します

```

AWS Bedrock ジョブの例 (OIDC)

前提条件：

- Amazon Bedrock が有効で、選択した Claude モデルへのアクセス権がある
- GitLab OIDC が AWS で構成され、GitLab プロジェクトと refs を信頼するロールがある
- Bedrock 権限を持つ IAM ロール（最小権限を推奨）

必須 CI/CD 変数：

- `AWS_ROLE_TO_ASSUME` : Bedrock アクセス用の IAM ロールの ARN
- `AWS_REGION` : Bedrock リージョン（例： `us-west-2`）


```

claude-bedrock:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apk add --no-cache bash curl jq git python3 py3-pip
    - pip install --no-cache-dir awscli
    - curl -fsSL https://claude.ai/install.sh | bash
    # GitLab OIDC トークンを AWS 認証情報と交換します
    - export AWS_WEB_IDENTITY_TOKEN_FILE="${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:

Bedrock のモデル ID にはリージョン固有のプレフィックスが含まれます（例：`us.anthropic.claude-sonnet-4-6`）。ワークフローがサポートしている場合は、ジョブ構成またはプロンプト経由で目的のモデルを渡します。

Google Vertex AI ジョブの例（Workload Identity Federation）

前提条件：

- GCP プロジェクトで Vertex AI API が有効
- GitLab OIDC を信頼するように構成された Workload Identity Federation
- Vertex AI 権限を持つサービスアカウント

必須 CI/CD 変数：

- `GCP_WORKLOAD_IDENTITY_PROVIDER` : 完全なプロバイダーリソース名
- `GCP_SERVICE_ACCOUNT` : サービスアカウントメール
- `CLOUD_ML_REGION` : Vertex リージョン（例：`us-east5`）

```

claude-vertex:
  stage: ai
  image: gcr.io/google.com/cloudsdktool/google-cloud-cli:slim
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apt-get update && apt-get install -y git && apt-get clean
    - curl -fsSL https://claude.ai/install.sh | bash
    # WIF 経由で Google Cloud に認証します (ダウンロードされたキーなし)
    - >
      gcloud auth login --cred-file=<(cat <<EOF
      {
        "type": "external_account",
        "audience": "${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:

Workload Identity Federation では、サービスアカウントキーを保存する必要はありません。リポジトリ固有の信頼条件と最小権限サービスアカウントを使用します。

ベストプラクティス

CLAUDE.md 構成

リポジトリルートに **CLAUDE.md** ファイルを作成して、コーディング標準、レビュー基準、およびプロジェクト固有のルールを定義します。Claude は実行中にこのファイルを読み取り、変更を提案する際にあなたの規約に従います。

セキュリティに関する考慮事項

API キーやクラウド認証情報をリポジトリにコミットしないでください。常に GitLab CI/CD 変数を使用します。

- **ANTHROPIC_API_KEY** をマスクされた変数として追加します（必要に応じて保護）
- 可能な限りプロバイダー固有の OIDC を使用します（長期キーなし）
- ジョブ権限とネットワーク出力を制限します
- 他の貢献者と同じように Claude の MR をレビューします

パフォーマンスの最適化

- **CLAUDE.md** を焦点を絞った簡潔なものに保ちます
- issue/MR の説明を明確にして、反復を減らします
- 実行不可能な実行を避けるために、適切なジョブタイムアウトを構成します
- ランナーで npm とパッケージのインストールをキャッシュします（可能な場合）

CI コスト

GitLab CI/CD で Claude Code を使用する場合、関連するコストに注意してください。

- **GitLab Runner 時間：**
 - Claude は GitLab ランナーで実行され、コンピュート分を消費します
 - GitLab プランのランナー請求の詳細については、プランを参照してください
- **API コスト：**
 - 各 Claude インタラクションは、プロンプトと応答サイズに基づいてトークンを消費します
 - トークン使用量はタスクの複雑さとコードベースのサイズによって異なります

- 詳細については [Anthropic pricing](#) を参照してください
- **コスト最適化のヒント：**
 - 特定の `@claude` コマンドを使用して、不要なターンを減らします
 - 適切な `max_turns` とジョブタイムアウト値を設定します
 - 並列実行を制限して、並行実行を制御します

セキュリティとガバナンス

- 各ジョブは、ネットワークアクセスが制限された分離されたコンテナで実行されます
- Claude の変更は MR を通じてフローするため、レビュアーはすべての diff を確認できます
- ブランチ保護と承認ルールが AI 生成コードに適用されます
- Claude Code はワークスペーススコープの権限を使用して書き込みを制限します
- 独自のプロバイダー認証情報を持ち込むため、コストは制御下に置かれます

トラブルシューティング

Claude が `@claude` コマンドに応答しない

- パイプラインがトリガーされていることを確認します（手動、MR イベント、またはメソイベントリスナー/webhook 経由）
- CI/CD 変数（`ANTHROPIC_API_KEY` またはクラウドプロバイダー設定）が存在し、マスク解除されていることを確認します
- コメントに `@claude`（`/claude` ではなく）が含まれており、メンショントリガーが構成されていることを確認します

ジョブがコメントを書き込めない、または MR を開けない

- `CI_JOB_TOKEN` がプロジェクトに対して十分な権限を持っていることを確認するか、`api` スコープを持つ Project Access Token を使用します
- `mcp_gitlab` ツールが `--allowedTools` で有効になっていることを確認します
- ジョブが MR のコンテキストで実行されているか、`AI_FLOW_*` 変数経由で十分なコンテキストを持っていることを確認します

認証エラー

- **Claude API の場合:** `ANTHROPIC_API_KEY` が有効で期限切れでないことを確認します
- **Bedrock/Vertex の場合:** OIDC/WIF 構成、ロール偽装、シークレット名を確認します。リージョンとモデルの可用性を確認します

高度な構成

一般的なパラメータと変数

Claude Code は以下の一般的に使用される入力をサポートしています。

- `prompt / prompt_file`: インライン (`-p`) またはファイル経由で指示を提供します
- `max_turns`: バックアンドフォース反復の数を制限します
- `timeout_minutes`: 総実行時間を制限します
- `ANTHROPIC_API_KEY`: Claude API に必須 (Bedrock/Vertex では使用されません)
- プロバイダー固有の環境: `AWS_REGION`、Vertex のプロジェクト/リージョン変数

Note:

正確なフラグとパラメータは `@anthropic-ai/claude-code` のバージョンによって異なる場合があります。ジョブで `claude --help` を実行して、サポートされているオプションを確認してください。

Claude の動作をカスタマイズする

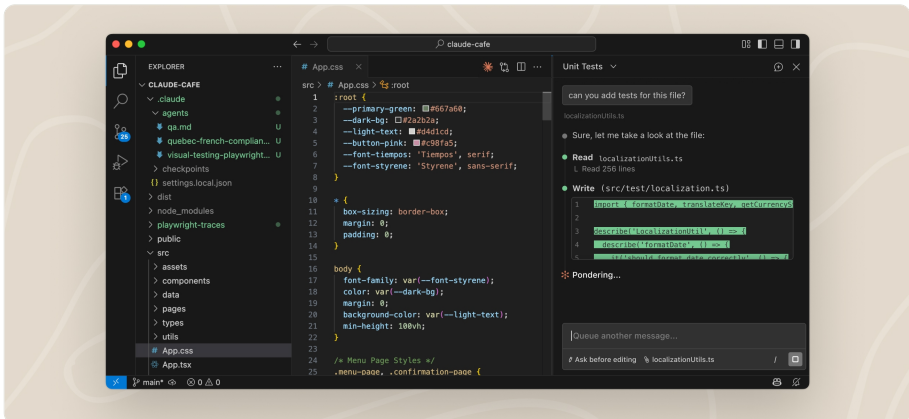
Claude をガイドするには、主に 2 つの方法があります。

1. **CLAUDE.md**: コーディング標準、セキュリティ要件、およびプロジェクト規約を定義します。Claude は実行中にこれを読み取り、ルールに従います。
2. **カスタムプロンプト**: ジョブで `prompt / prompt_file` 経由してタスク固有の指示を渡します。異なるジョブに異なるプロンプトを使用します (例: レビュー、実装、リファクタリング)。

Part 8: IDE & Platform Integration

VS Code で Claude Code を使用する

Claude Code 拡張機能を VS Code にインストールして設定します。インラインの差分表示、@-メンション、プラン確認、キーボードショートカットを使用した AI コーディング支援を取得します。



VS Code エディタの右側に Claude Code 拡張機能パネルが開いており、Claude との会話が表示されている

VS Code 拡張機能は、Claude Code 用のネイティブグラフィカルインターフェースを提供し、IDE に直接統合されています。これは VS Code で Claude Code を使用する推奨方法です。

この拡張機能を使用すると、Claude のプランを受け入れる前に確認および編集でき、編集が行われるにつれて自動的に受け入れることができ、選択範囲から特定の行範囲を持つファイルを @-メンションでき、会話履歴にアクセスでき、複数の会話を別々のタブまたはウィンドウで開くことができます。

前提条件

インストール前に、以下があることを確認してください。

- VS Code 1.98.0 以上

- Anthropic アカウント（拡張機能を初めて開くときにサインインします）。Amazon Bedrock や Google Vertex AI などのサードパーティプロバイダーを使用している場合は、代わりに[サードパーティプロバイダーを使用する](#)を参照してください。

Tip:

拡張機能には CLI（コマンドラインインターフェース）が含まれており、VS Code の統合ターミナルからアクセスして高度な機能を使用できます。詳細については、[VS Code 拡張機能と Claude Code CLI](#) を参照してください。

拡張機能をインストールする

IDE のリンクをクリックして直接インストールします。

- [VS Code 用にインストール](#)
- [Cursor 用にインストール](#)

または、VS Code で **Cmd+Shift+X**（Mac）または **Ctrl+Shift+X**（Windows/Linux）を押して拡張機能ビューを開き、「Claude Code」を検索して、**インストール**をクリックします。

Note:

インストール後に拡張機能が表示されない場合は、VS Code を再起動するか、コマンドパレットから「Developer: Reload Window」を実行してください。

はじめに

インストール後、VS Code インターフェースを通じて Claude Code の使用を開始できます。

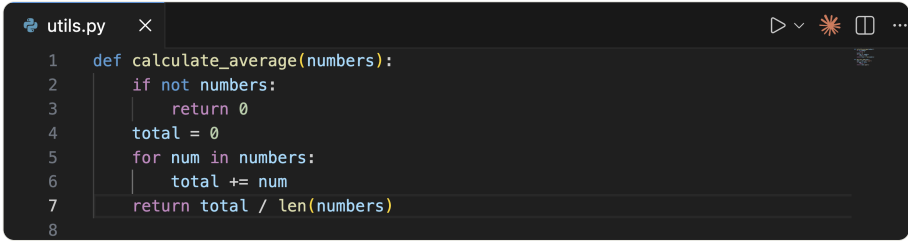
Step 1: Claude Code パネルを開く

VS Code 全体で、Spark アイコンは Claude Code を示します。



Spark icon

Claude を開く最速の方法は、**エディタツールバー**（エディタの右上隅）の Spark アイコンをクリックすることです。このアイコンは、ファイルが開いている場合にのみ表示されます。



VS Code エディタの右上隅のエディタツールバーに Spark アイコンが表示されている

Claude Code を開く他の方法：

- ・ **アクティビティバー**：左サイドバーの Spark アイコンをクリックしてセッションリストを開きます。任意のセッションをクリックしてフルエディタタブとして開くか、新しいセッションを開始します。このアイコンは常にアクティビティバーに表示されます。
- ・ **コマンドパレット**：`Cmd+Shift+P` (Mac) または `Ctrl+Shift+P` (Windows/Linux) を押し、「Claude Code」と入力して、「Open in New Tab」などのオプションを選択します。
- ・ **ステータスバー**：ウィンドウの右下隅の **Claude Code** をクリックします。ファイルが開いていない場合でも機能します。

パネルを初めて開くと、**Learn Claude Code** チェックリストが表示されます。**Show me** をクリックして各項目を実行するか、X で閉じます。後で再度開くには、VS Code 設定の Extensions → Claude Code で **Hide Onboarding** をオフにします。

Claude パネルをドラッグして VS Code 内の任意の場所に移動できます。詳細については、[ワークフローをカスタマイズする](#)を参照してください。

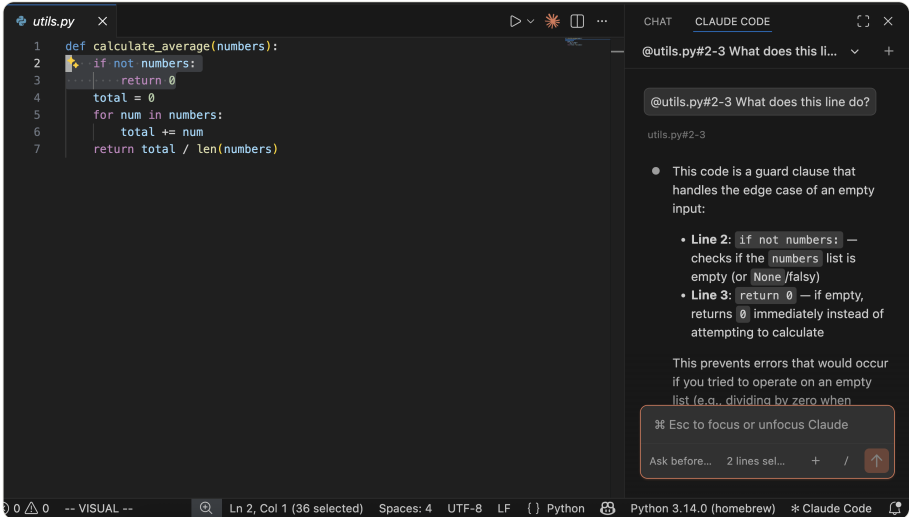
Step 2: プロンプトを送信する

Claude にコードやファイルの支援を依頼します。これには、何かがどのように機能するかを説明すること、問題をデバッグすること、または変更を加えることが含まれます。

Tip:

Claude は自動的に選択したテキストを表示します。`Option+K` (Mac) / `Alt+K` (Windows/Linux) を押して、@-メンション参照 (`@file.ts#5-10` など) をプロンプトに挿入することもできます。

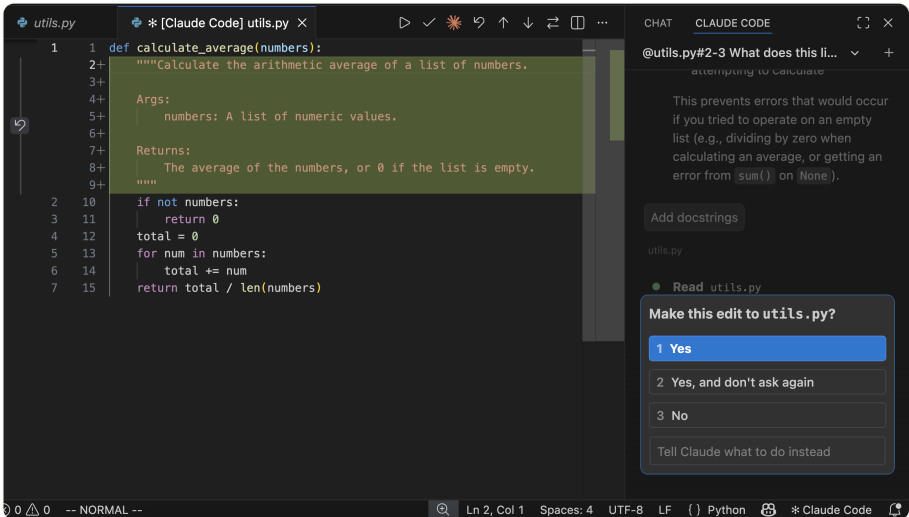
ファイル内の特定の行について質問する例を次に示します。



VS Code エディタで Python ファイルの 2〜3 行が選択されており、Claude Code パネルにそれらの行に関する質問と @-メンション参照が表示されている

Step 3: 変更を確認する

Claude がファイルを編集したい場合、元の内容と提案された変更の並列比較を表示し、許可を求めます。編集を受け入れるか、拒否するか、Claude に別の方法を指示できます。



VS Code が Claude の提案された変更の差分を表示し、編集を行うかどうかを尋ねる許可プロンプトが表示されている

Claude Code でできることについてのアイデアについては、[一般的なワークフロー](#)を参照してください。

Tip:

コマンドパレットから「Claude Code: Open Walkthrough」を実行して、基本的な機能のガイド付きツアーを取得します。

プロンプトボックスを使用する

プロンプトボックスは複数の機能をサポートしています。

- **許可モード**：プロンプトボックスの下部のモード指示器をクリックしてモードを切り替えます。通常モードでは、Claude は各アクション前に許可を求めます。Plan モードでは、Claude は実行内容を説明し、変更を加える前に承認を待ちます。VS Code は自動的にプランをフルマークダウンドキュメントとして開き、Claude が開始する前にフィードバックを提供するためのインラインコメントを追加できます。自動受け入れモードでは、Claude は許可を求めずに編集を行います。デフォルトは VS Code 設定の `claudeCode.initialPermissionMode` で設定します。
- **コマンドメニュー**：`/` をクリックするか `/` と入力してコマンドメニューを開きます。オプションには、ファイルの添付、モデルの切り替え、拡張思考の切り替え、プラン使用状況の表示 (`/usage`) が含まれます。カスタマイズセクションは、MCP サーバー、hooks、メモリ、権限、プラグインへのアクセスを提供します。ターミナルアイコン付きのアイテムは統合ターミナルで開きます。
- **コンテキスト指示器**：プロンプトボックスは Claude のコンテキストウィンドウをどの程度使用しているかを表示します。Claude は必要に応じて自動的にコンパクト化するか、`/compact` を手動で実行できます。
- **拡張思考**：Claude が複雑な問題を推論するためにより多くの時間を費やすことができます。コマンドメニュー (`/`) を使用してオンに切り替えます。詳細については、[拡張思考](#)を参照してください。
- **複数行入力**：`Shift+Enter` を押して送信せずに新しい行を追加します。これは質問ダイアログの「その他」フリーテキスト入力でも機能します。

ファイルとフォルダを参照する

@-メンションを使用して、特定のファイルまたはフォルダに関するコンテキストを Claude に提供します。@ の後にファイルまたはフォルダ名を入力すると、Claude はそのコンテンツを読み取り、それについて質問したり、変更を加えたりできます。Claude Code はあいまい一致をサポートしているため、部分的な名前を入力して必要なものを見つけることができます。

```
> Explain the logic in @auth (fuzzy matches auth.js, AuthService.ts, etc.)
> What's in @src/components/ (include a trailing slash for folders)
```

大きな PDF の場合、Claude にファイル全体ではなく特定のページを読むよう依頼できます。単一ページ、1~10 ページなどの範囲、または 3 ページ以降などのオープンエンド範囲です。

エディタでテキストを選択すると、Claude は強調表示されたコードを自動的に表示できます。プロンプトボックスのフッターは、選択されている行数を表示します。Option+K (Mac) / Alt+K (Windows/Linux) を押して、ファイルパスと行番号を含む @-メンション (例: @app.ts#5-10) を挿入します。選択指示器をクリックして、Claude が強調表示されたテキストを表示できるかどうかを切り替えます。目のスラッシュアイコンは、選択が Claude から非表示になっていることを意味します。

また、Shift を押しながらファイルをプロンプトボックスにドラッグして、添付ファイルとして追加することもできます。任意の添付ファイルの X をクリックしてコンテキストから削除します。

過去の会話を再開する

Claude Code パネルの上部のドロップダウンをクリックして、会話履歴にアクセスします。キーワードで検索するか、時間 (今日、昨日、過去 7 日間など) で参照できます。任意の会話をクリックして、完全なメッセージ履歴で再開します。セッションの上にマウスを置くと、名前変更と削除アクションが表示されます。説明的なタイトルを付けるために名前を変更するか、リストから削除するために削除します。セッションの再開の詳細については、[一般的なワークフロー](#)を参照してください。

Claude.ai からリモートセッションを再開する

[Web 上の Claude Code](#) を使用している場合、VS Code でそれらのリモートセッションを直接再開できます。これには、Anthropic Console ではなく **Claude.ai Subscription** でサインインする必要があります。

Step 1: 過去の会話を開く

Claude Code パネルの上部の **Past Conversations** ドロップダウンをクリックします。

Step 2: リモートタブを選択する

ダイアログには2つのタブが表示されます。ローカルとリモート。**リモート**をクリックして `claude.ai` からのセッションを表示します。

Step 3: 再開するセッションを選択する

リモートセッションを参照または検索します。任意のセッションをクリックしてダウンロードし、会話をローカルで続行します。

Note:

リモートタブに表示されるのは、GitHub リポジトリで開始された Web セッションのみです。再開するとローカルに会話履歴が読み込まれます。変更は `claude.ai` に同期されません。

ワークフローをカスタマイズする

起動して実行したら、Claude パネルを移動したり、複数のセッションを実行したり、ターミナルモードに切り替えたりできます。

Claude が存在する場所を選択する

Claude パネルをドラッグして VS Code 内の任意の場所に移動できます。パネルのタブまたはタイトルバーをつかんでドラッグします。

- **セカンダリサイドバー**：ウィンドウの右側。コード作成中に Claude を表示したままにします。
- **プライマリサイドバー**：Explorer、Search などのアイコンが付いた左サイドバー。
- **エディタ領域**：Claude をファイルの横のタブとして開きます。サイドタスクに便利です。

Tip:

メイン Claude セッションにはサイドバーを使用し、サイドタスク用に追加のタブを開きます。Claude は優先される場所を記憶します。アクティビティバーセッションリストアイコンは Claude パネルとは別です。セッションリストは常にアクティビティバーに表示されますが、Claude パネルアイコンはパネルが左サイドバーにドッキングされている場合にのみそこに表示されます。

複数の会話を実行する

コマンドパレットから **Open in New Tab** または **Open in New Window** を使用して、追加の会話を開始します。各会話は独自の履歴とコンテキストを保持し、異なるタスクで並列に作業できます。

タブを使用する場合、Spark アイコンの小さな色付きドットはステータスを示します。青は許可リクエストが保留中であることを意味し、オレンジはタブが非表示の間に Claude が完了したことを意味します。

ターミナルモードに切り替える

デフォルトでは、拡張機能はグラフィカルチャットパネルを開きます。CLI スタイルのインターフェースを使用する場合は、[ターミナル設定を使用](#)を開いてボックスをチェックします。

VS Code 設定（Mac で **Cmd+,** または Windows/Linux で **Ctrl+,**）を開き、Extensions → Claude Code に移動して、**Use Terminal** をチェックすることもできます。

プラグインを管理する

VS Code 拡張機能には、[プラグイン](#)をインストールおよび管理するためのグラフィカルインターフェースが含まれています。プロンプトボックスで **/plugins** と入力して、**プラグインを管理**インターフェースを開きます。

プラグインをインストールする

プラグインダイアログには2つのタブが表示されます。**プラグイン**と**マーケットプレイス**。

プラグインタブで：

- ・ **インストール済みプラグイン**は上部に表示され、トグルスイッチで有効または無効にできます。
- ・ **利用可能なプラグイン**は設定されたマーケットプレイスから下に表示されます。
- ・ 名前または説明でプラグインをフィルタリングするために検索します。
- ・ 利用可能なプラグインで **インストール**をクリックします。

プラグインをインストールするときは、インストールスコープを選択します。

- ・ **あなたのためにインストール**：すべてのプロジェクトで利用可能（ユーザースコープ）
- ・ **このプロジェクト用にインストール**：プロジェクト協力者と共有（プロジェクトスコープ）

- ・ **ローカルにインストール**：このリポジトリでのみ、あなただけのために（ローカルスコープ）

マーケットプレースを管理する

マーケットプレースタブに切り替えてプラグインソースを追加または削除します。

- ・ GitHub リポジトリ、URL、またはローカルパスを入力して新しいマーケットプレースを追加します。
- ・ 更新アイコンをクリックしてマーケットプレースのプラグインリストを更新します。
- ・ ゴミ箱アイコンをクリックしてマーケットプレースを削除します。

変更を加えた後、バナーが表示され、Claude Code を再起動して更新を適用するよう促します。

Note:

VS Code のプラグイン管理は、内部で同じ CLI コマンドを使用します。拡張機能で設定したプラグインとマーケットプレースは CLI でも利用可能であり、その逆も同様です。

プラグインシステムの詳細については、[プラグイン](#)と[プラグインマーケットプレース](#)を参照してください。

Chrome でブラウザタスクを自動化する

Claude を Chrome ブラウザに接続して、Web アプリをテストし、コンソールログでデバッグし、VS Code を離れることなくブラウザワークフローを自動化します。これには、[Chrome の Claude 拡張機能](#)バージョン 1.0.36 以上が必要です。

プロンプトボックスで `@browser` と入力し、その後に Claude に実行させたいことを入力します。

```
@browser go to localhost:3000 and check the console for errors
```

また、添付メニューを開いて、新しいタブを開いたりページコンテンツを読んだりするなど、特定のブラウザツールを選択することもできます。

Claude はブラウザタスク用に新しいタブを開き、ブラウザのログイン状態を共有するため、既にサインインしているサイトにアクセスできます。

セットアップ手順、機能の完全なリスト、トラブルシューティングについては、[Chrome で Claude Code を使用する](#)を参照してください。

VS Code コマンドとショートカット

コマンドパレット（Mac で **Cmd+Shift+P** または Windows/Linux で **Ctrl+Shift+P**）を開き、'Claude Code' と入力して、Claude Code 拡張機能で利用可能なすべての VS Code コマンドを表示します。

一部のショートカットは、どのパネルが'フォーカス'されているか（キーボード入力を受け取っているか）によって異なります。カーソルがコードファイルにある場合、エディタはフォーカスされています。カーソルが Claude のプロンプトボックスにある場合、Claude はフォーカスされています。 **Cmd+Esc** / **Ctrl+Esc** を使用してそれらを切り替えます。

Note:

これらは拡張機能を制御するための VS Code コマンドです。組み込みの Claude Code コマンドのすべてが拡張機能で利用可能なわけではありません。詳細については、[VS Code 拡張機能](#) と [Claude Code CLI](#) を参照してください。

コマンド	ショートカット	説明
Focus Input	Cmd+Esc (Mac) / Ctrl+Esc (Windows/Linux)	エディタと Claude 間のフォーカスを切り替える
Open in Side Bar	-	Claude を左サイドバーで開く
Open in Terminal	-	Claude をターミナルモードで開く
Open in New Tab	Cmd+Shift+Esc (Mac) / Ctrl+Shift+Esc (Windows/Linux)	新しい会話をエディタタブとして開く
Open in New Window	-	新しい会話を別のウィンドウで開く
New Conversation	Cmd+N (Mac) / Ctrl+N (Windows/Linux)	新しい会話を開始する (Claude がフォーカスされている必要があります)
Insert @-Mention Reference	Option+K (Mac) / Alt+K (Windows/Linux)	現在のファイルと選択への参照を挿入する (エディタがフォーカスされている必要があります)

コマンド	ショートカット	説明
Show Logs	-	拡張機能デバッグログを表示する
Logout	-	Anthropic アカウントからサインアウトする

設定を構成する

拡張機能には 2 種類の設定があります。

- **VS Code の拡張機能設定**：VS Code 内の拡張機能の動作を制御します。 `Cmd+,` (Mac) または `Ctrl+,` (Windows/Linux) で開き、Extensions → Claude Code に移動します。 `/` と入力して **General Config** を選択して設定を開くこともできます。
- **`~/.claude/settings.json` の Claude Code 設定**：拡張機能と CLI 間で共有されます。許可されたコマンド、環境変数、hooks、MCP サーバーに使用します。詳細については、[設定](#)を参照してください。

Tip:

`"$schema": "https://json.schemastore.org/claude-code-settings.json"` を `settings.json` に追加して、VS Code で利用可能なすべての設定のオートコンプリートとインライン検証を取得します。

拡張機能設定

設定	デフォルト	説明
<code>selectedModel</code>	<code>default</code>	新しい会話のモデル。 <code>/ model</code> でセッションごとに変更します。
<code>useTerminal</code>	<code>false</code>	グラフィカルパネルの代わりにターミナルモードで Claude を起動します。
<code>initialPermissionMode</code>	<code>default</code>	承認プロンプトを制御します。 <code>default</code> (毎回尋ねる)、 <code>plan</code> 、 <code>acceptEdits</code> 、または <code>bypassPermissions</code>

設定	デフォルト	説明
<code>preferredLocation</code>	<code>panel</code>	Claude が開く場所： <code>sidebar</code> （右）または <code>panel</code> （新しいタブ）
<code>autosave</code>	<code>true</code>	Claude が読み取りまたは 書き込みする前にファイルを 自動保存します。
<code>useCtrlEnterToSend</code>	<code>false</code>	Enter の代わりに Ctrl/ Cmd+Enter を使用してプ ロンプトを送信します。
<code>enableNewConversation Shortcut</code>	<code>true</code>	Cmd/Ctrl+N を有効にして 新しい会話を開始します。
<code>hideOnboarding</code>	<code>false</code>	オンボーディングチェック リスト（卒業キャップアイ コン）を非表示にします。
<code>respectGitIgnore</code>	<code>true</code>	ファイル検索か ら .gitignore パターンを除 外します。
<code>environmentVariables</code>	<code>[]</code>	Claude プロセスの環境変 数を設定します。共有設定 には Claude Code 設定を 使用します。
<code>disableLoginPrompt</code>	<code>false</code>	認証プロンプトをスキップ します（サードパーティプ ロバイダーセットアップ 用）。
<code>allowDangerouslySkipP ermissions</code>	<code>false</code>	すべての許可プロンプトを バイパスします。 極度の注 意を持って使用してくださ い。
<code>claudeProcessWrapper</code>	-	Claude プロセスを起動す るために使用される実行可 能ファイルパス

VS Code 拡張機能と Claude Code CLI

Claude Code は VS Code 拡張機能（グラフィカルパネル）と CLI（ターミナルのコマンドラインインターフェース）の両方として利用可能です。一部の機能は CLI でのみ利用可能です。CLI のみの機能が必要な場合は、VS Code の統合ターミナルで `claude` を実行します。

機能	CLI	VS Code 拡張機能
コマンドと skills	すべて	サブセット（ <code>/</code> と入力して利用可能なものを表示）
MCP サーバー設定	はい	部分的（CLI 経由でサーバーを追加。チャットパネルで <code>/mcp</code> を使用して既存のサーバーを管理）
チェックポイント	はい	はい
 bash ショートカット	はい	いいえ
タブ補完	はい	いいえ

チェックポイントで巻き戻す

VS Code 拡張機能はチェックポイントをサポートしており、Claude のファイル編集を追跡し、以前の状態に巻き戻すことができます。任意のメッセージの上にマウスを置いて巻き戻しボタンを表示し、3 つのオプションから選択します。

- ・ **ここから会話をフォーク**：このメッセージから新しい会話ブランチを開始し、すべてのコード変更をそのまま保持します。
- ・ **ここにコードを巻き戻す**：会話の完全な履歴を保持しながら、ファイル変更をこのポイントに戻します。
- ・ **会話をフォークしてコードを巻き戻す**：新しい会話ブランチを開始し、ファイル変更をこのポイントに戻します。

チェックポイントの動作方法と制限事項の詳細については、[チェックポイント](#)を参照してください。

VS Code で CLI を実行する

VS Code にいながら CLI を使用するには、統合ターミナル（Windows/Linux で `Ctrl+`` または Mac で `Cmd+``）を開き、`claude` を実行します。CLI は自動的に IDE と統合され、差分表示や診断共有などの機能を提供します。

外部ターミナルを使用している場合は、Claude Code 内で `/ide` を実行して VS Code に接続します。

拡張機能と CLI を切り替える

拡張機能と CLI は同じ会話履歴を共有します。拡張機能の会話を CLI で続行するには、ターミナルで `claude --resume` を実行します。これにより、会話を検索して選択できるインタラクティブピッカーが開きます。

プロンプトにターミナル出力を含める

プロンプトで `@terminal:name` を使用してターミナル出力を参照します。ここで `name` はターミナルのタイトルです。これにより、Claude はコマンド出力、エラーメッセージ、またはログをコピーペーストせずに表示できます。

バックグラウンドプロセスを監視する

Claude が長時間実行されるコマンドを実行すると、拡張機能はステータスバーに進行状況を表示します。ただし、バックグラウンドタスクの可視性は CLI と比較して制限されています。より良い可視性のために、Claude にコマンドを出力させて、VS Code の統合ターミナルで実行できるようにします。

MCP で外部ツールに接続する

MCP (Model Context Protocol) サーバーは Claude に外部ツール、データベース、API へのアクセスを提供します。

MCP サーバーを追加するには、統合ターミナル (`Ctrl+`` または `Cmd+``) を開き、以下を実行します。

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

設定後、Claude にツールを使用するよう依頼します（例：「Review PR #456」）。

VS Code を離れることなく MCP サーバーを管理するには、チャットパネルで `/mcp` と入力します。MCP 管理ダイアログでは、サーバーを有効または無効にし、サーバーに再接続し、OAuth 認証を管理できます。利用可能なサーバーについては、[MCP ドキュメント](#) を参照してください。

git で作業する

Claude Code は git と統合され、VS Code でバージョン管理ワークフローを直接支援します。Claude にコミット変更、プルリクエスト作成、またはブランチ間での作業を依頼します。

コミットとプルリクエストを作成する

Claude は変更をステージング、コミットメッセージを作成、作業に基づいてプルリクエストを作成できます。

```
> commit my changes with a descriptive message
> create a pr for this feature
> summarize the changes I've made to the auth module
```

プルリクエストを作成するときに、Claude は実際のコード変更に基づいて説明を生成し、テストまたは実装の決定に関するコンテキストを追加できます。

並列タスク用に git worktrees を使用する

`--worktree` (`-w`) フラグを使用して、独自のファイルとブランチを持つ分離された `worktree` で Claude を開始します。

```
claude --worktree feature-auth
```

各 `worktree` は git 履歴を共有しながら独立したファイル状態を保持します。これにより、異なるタスクで作業するときに Claude インスタンスが互いに干渉するのを防ぎます。詳細については、[Git worktrees で並列セッションを実行する](#)を参照してください。

サードパーティプロバイダーを使用する

デフォルトでは、Claude Code は Anthropic の API に直接接続します。組織が Amazon Bedrock、Google Vertex AI、または Microsoft Foundry を使用して Claude にアクセスする場合は、代わりにプロバイダーを使用するように拡張機能を設定します。

Step 1: ログインプロンプトを無効にする

[ログインプロンプトを無効にする設定](#)を開いてボックスをチェックします。

VS Code 設定 (Mac で `Cmd+`, または Windows/Linux で `Ctrl+`) を開き、'Claude Code login' を検索して、**Disable Login Prompt** をチェックすることもできます。

Step 2: プロバイダーを設定する

プロバイダーのセットアップガイドに従います。

- [Amazon Bedrock の Claude Code](#)
- [Google Vertex AI の Claude Code](#)
- [Microsoft Foundry の Claude Code](#)

これらのガイドは `~/.claude/settings.json` でプロバイダーを設定することをカバーしており、VS Code 拡張機能と CLI 間で設定が共有されることを保証します。

セキュリティとプライバシー

コードはプライベートのままです。Claude Code はコード支援を提供するためにコードを処理しますが、モデルのトレーニングには使用しません。データ処理とログアウトの方法の詳細については、[データとプライバシー](#)を参照してください。

自動編集権限が有効な場合、Claude Code は VS Code が自動的に実行する可能性がある VS Code 設定ファイル（`settings.json` や `tasks.json` など）を変更できます。信頼できないコードで作業するときのリスクを軽減するには、以下を実行します。

- 信頼できないワークスペースに対して [VS Code 制限モード](#) を有効にします。
- 編集の自動受け入れの代わりに手動承認モードを使用します。
- 変更を受け入れる前に慎重に確認します。

一般的な問題を修正する

拡張機能がインストールされない

- VS Code の互換バージョン（1.98.0 以上）があることを確認します。
- VS Code に拡張機能をインストールする権限があることを確認します。
- [VS Code Marketplace](#) から直接インストールしてみてください。

Spark アイコンが表示されない

Spark アイコンは、ファイルが開いている場合、**エディタツールバー**（エディタの右上）に表示されます。表示されない場合：

1. **ファイルを開く**：アイコンはファイルが開いている必要があります。フォルダだけを開いているのでは不十分です。
2. **VS Code バージョンを確認**：1.98.0 以上が必要です（Help → About）
3. **VS Code を再起動**：コマンドパレットから「Developer: Reload Window」を実行します。
4. **競合する拡張機能を無効にする**：他の AI 拡張機能（Cline、Continue など）を一時的に無効にします。
5. **ワークスペーストラストを確認**：拡張機能は制限モードでは機能しません。

または、**ステータスバー**（右下隅）の「**Claude Code**」をクリックします。これはファイルが開いていなくても機能します。**コマンドパレット**（`Cmd+Shift+P` / `Ctrl+Shift+P`）を使用して「Claude Code」と入力することもできます。

Claude Code が応答しない

Claude Code がプロンプトに応答しない場合：

1. **インターネット接続を確認**：安定したインターネット接続があることを確認します。
2. **新しい会話を開始**：新しい会話を開始して、問題が続くかどうかを確認します。
3. **CLIを試す**：ターミナルから `claude` を実行して、より詳細なエラーメッセージが表示されるかどうかを確認します。

問題が続く場合は、エラーの詳細を含めて [GitHub で問題を報告](#) してください。

拡張機能をアンインストールする

Claude Code 拡張機能をアンインストールするには：

1. 拡張機能ビューを開きます（Mac で `Cmd+Shift+X` または Windows/Linux で `Ctrl+Shift+X`）
2. 「Claude Code」を検索します。
3. **アンインストール**をクリックします。

拡張機能データを削除してすべての設定をリセットするには：

```
rm -rf ~/.vscode/globalStorage/anthropic.claude-code
```

追加のヘルプについては、[トラブルシューティングガイド](#)を参照してください。

次のステップ

VS Code で Claude Code をセットアップしたので：

- [一般的なワークフローを探索](#)して Claude Code を最大限に活用します。
- [MCP サーバーをセットアップ](#)して、外部ツールで Claude の機能を拡張します。CLI を使用してサーバーを追加し、チャットパネルで `/mcp` を使用して管理します。
- [Claude Code 設定を構成](#)して、許可されたコマンド、hooksなどをカスタマイズします。これらの設定は拡張機能と CLI 間で共有されます。

JetBrains IDEs

Claude Code を IntelliJ、PyCharm、WebStorm など JetBrains IDEs で使用する

Claude Code は専用プラグインを通じて JetBrains IDEs と統合され、インタラクティブな diff ビューイング、選択コンテキスト共有など、様々な機能を提供します。

サポートされている IDE

Claude Code プラグインは、以下を含むほとんどの JetBrains IDEs で動作します。

- IntelliJ IDEA
- PyCharm
- Android Studio
- WebStorm
- PhpStorm
- GoLand

機能

- **クイック起動:** `Cmd+Esc` (Mac) または `Ctrl+Esc` (Windows/Linux) を使用してエディタから Claude Code を直接開くか、UI の Claude Code ボタンをクリックします
- **Diff ビューイング:** コードの変更をターミナルではなく IDE の diff ビューアに直接表示できます
- **選択コンテキスト:** IDE の現在の選択/タブが Claude Code と自動的に共有されます
- **ファイル参照ショートカット:** `Cmd+Option+K` (Mac) または `Alt+Ctrl+K` (Linux/Windows) を使用してファイル参照を挿入します (例: `@File#L1-99`)
- **診断共有:** IDE からの診断エラー (lint、構文など) が作業中に Claude と自動的に共有されます

インストール

マーケットプレイスからのインストール

JetBrains マーケットプレイスから [Claude Code プラグイン](#) を見つけてインストールし、IDE を再起動します。

Claude Code をまだインストールしていない場合は、[クイックスタートガイド](#) でインストール手順を確認してください。

Note:

プラグインをインストール後、IDE を完全に再起動する必要がある場合があります。

使用方法

IDE から

IDE の統合ターミナルから `claude` を実行すると、すべての統合機能がアクティブになります。

外部ターミナルから

任意の外部ターミナルで `/ide` コマンドを使用して Claude Code を JetBrains IDE に接続し、すべての機能をアクティブにします。

```
claude
```

```
/ide
```

Claude が IDE と同じファイルにアクセスできるようにしたい場合は、IDE プロジェクトルートと同じディレクトリから Claude Code を起動してください。

設定

Claude Code 設定

Claude Code の設定を通じて IDE 統合を設定します。

1. `claude` を実行します
2. `/config` コマンドを入力します
3. diff ツールを `auto` に設定して自動 IDE 検出を有効にします

プラグイン設定

Settings → Tools → Claude Code [Beta] に移動して Claude Code プラグインを設定します。

一般設定

- **Claude command:** Claude を実行するカスタムコマンドを指定します（例：
`claude`、`/usr/local/bin/claude`、または `npm @anthropic/claude`）
- **Suppress notification for Claude command not found:** Claude コマンドが見つからないことに関する通知をスキップします
- **Enable using Option+Enter for multi-line prompts**（macOS のみ）：有効にすると、Option+Enter は Claude Code プロンプトに新しい行を挿入します。Option キーが予期せずキャプチャされる問題が発生している場合は無効にしてください（ターミナルの再起動が必要）
- **Enable automatic updates:** プラグインの更新を自動的にチェックしてインストールします（再起動時に適用）

Tip:

WSL ユーザー向け: Claude コマンドとして `wsl -d Ubuntu -- bash -lic "claude"` を設定します（`Ubuntu` を WSL ディストリビューション名に置き換えてください）

ESC キー設定

ESC キーが JetBrains ターミナルで Claude Code 操作を中断しない場合：

1. **Settings → Tools → Terminal** に移動します
2. 以下のいずれかを実行します。
 - 「Move focus to the editor with Escape」をオフにするか、
 - 「Configure terminal keybindings」をクリックして「Switch focus to Editor」ショートカットを削除します
3. 変更を適用します

これにより、ESC キーが Claude Code 操作を適切に中断できるようになります。

特別な設定

リモート開発

Warning:

JetBrains リモート開発を使用する場合、**Settings → Plugin (Host)** を通じてリモートホストにプラグインをインストールする必要があります。

プラグインはローカルクライアントマシンではなく、リモートホストにインストールする必要があります。

WSL 設定

Warning:

WSL ユーザーは IDE 検出が正常に機能するために追加の設定が必要な場合があります。詳細なセットアップ手順については、[WSL トラブルシューティングガイド](#)を参照してください。

WSL 設定には以下が必要な場合があります。

- 適切なターミナル設定
- ネットワークモード調整
- ファイアウォール設定の更新

トラブルシューティング

プラグインが動作しない

- Claude Code をプロジェクトルートディレクトリから実行していることを確認してください
- JetBrains プラグインが IDE 設定で有効になっていることを確認してください
- IDE を完全に再起動してください（複数回実行する必要がある場合があります）
- リモート開発の場合、プラグインがリモートホストにインストールされていることを確認してください

IDE が検出されない

- プラグインがインストールされて有効になっていることを確認してください
- IDE を完全に再起動してください
- 統合ターミナルから Claude Code を実行していることを確認してください
- WSL ユーザーの場合、[WSL トラブルシューティングガイド](#)を参照してください

コマンドが見つからない

Claude アイコンをクリックして「command not found」が表示される場合：

1. Claude Code がインストールされていることを確認します：`npm list -g @anthropic-ai/claude-code`
2. プラグイン設定で Claude コマンドパスを設定します

3. WSL ユーザーの場合、設定セクションで説明されている WSL コマンド形式を使用します

セキュリティに関する考慮事項

Claude Code が自動編集権限が有効な JetBrains IDE で実行される場合、IDE によって自動的に実行される可能性のある IDE 設定ファイルを変更できる場合があります。これにより、自動編集モードで Claude Code を実行するリスクが増加し、bash 実行に対する Claude Code の権限プロンプトをバイパスできる可能性があります。

JetBrains IDEs で実行する場合は、以下を検討してください。

- 編集に対して手動承認モードを使用する
- Claude が信頼できるプロンプトでのみ使用されることを確認するために特に注意する
- Claude Code がアクセスして変更できるファイルを認識する

追加のヘルプについては、[トラブルシューティングガイド](#)を参照してください。

Claude Code Desktop を使用する

Claude Code Desktop をさらに活用する：Git 分離による並列セッション、ビジュアル diff レビュー、アプリプレビュー、PR 監視、権限モード、コネクタ、エンタープライズ設定。

Claude Desktop アプリ内の Code タブを使用すると、ターミナルではなくグラフィカルインターフェースを通じて Claude Code を使用できます。

Desktop は標準的な Claude Code エクスペリエンスに以下の機能を追加します：

- [ビジュアル diff レビュー](#)（インラインコメント付き）
- [ライブアプリプレビュー](#)（dev サーバー付き）
- [GitHub PR 監視](#)（自動修正と自動マージ付き）
- [並列セッション](#)（自動 Git worktree 分離付き）
- [スケジュール済みタスク](#)（定期的に Claude を実行）
- [コネクタ](#)（GitHub、Slack、Linear など）
- ローカル、[SSH](#)、および[クラウド](#)環境

Tip:

Desktop は初めてですか？[はじめに](#)から始めて、アプリをインストールして最初の編集を行ってください。

このページでは、[コードの操作](#)、[セッションの管理](#)、[Claude Code の拡張](#)、[スケジュール済みタスク](#)、および[設定](#)について説明します。また、[CLI との比較](#)と[トラブルシューティング](#)も含まれています。

セッションを開始する

最初のメッセージを送信する前に、プロンプト領域で 4 つのことを設定してください：

- **環境**：Claude が実行される場所を選択します。ローカルマシンの場合は**Local**、Anthropic ホスト型クラウドセッションの場合は**Remote**、管理するリモートマシンの場合は**SSH 接続**を選択します。[環境設定](#)を参照してください。
- **プロジェクトフォルダ**：Claude が作業するフォルダまたはリポジトリを選択します。リモートセッションの場合、[複数のリポジトリ](#)を追加できます。

- **モデル**：送信ボタンの横のドロップダウンから [モデル](#) を選択します。セッションが開始されるとモデルはロックされます。
- **権限モード**： [モードセレクト](#) から Claude が持つ自律性の程度を選択します。セッション中にこれを変更できます。

タスクを入力して **Enter** キーを押してセッションを開始します。各セッションは独自のコンテキストと変更を追跡します。

コードの操作

Claude に適切なコンテキストを提供し、それが独立して実行する量を制御し、変更内容を確認します。

プロンプトボックスを使用する

Claude に実行してほしいことを入力して **Enter** キーを押して送信します。Claude はプロジェクトファイルを読み取り、[権限モード](#) に基づいて変更を加えてコマンドを実行します。いつでも Claude を中断できます：停止ボタンをクリックするか、修正を入力して **Enter** キーを押します。Claude は実行を停止し、入力に基づいて調整します。

プロンプトボックスの横の ******* ボタンをクリックすると、ファイル添付、[スキル](#)、[コネクタ](#)、および [プラグイン](#) にアクセスできます。

ファイルとコンテキストをプロンプトに追加する

プロンプトボックスは外部コンテキストを取り込む 2 つの方法をサポートしています：

- **@mention ファイル**： **@** の後にファイル名を入力して、ファイルを会話コンテキストに追加します。Claude はそのファイルを読み取り、参照できます。
- **ファイルを添付**：添付ボタンを使用して画像、PDF、その他のファイルをプロンプトに添付するか、ファイルをプロンプトに直接ドラッグアンドドロップします。これはバグのスクリーンショット、デザインモックアップ、または参照ドキュメントを共有するのに便利です。

権限モードを選択する

権限モードは、セッション中に Claude が持つ自律性の程度を制御します：ファイルの編集、コマンドの実行、またはその両方の前に確認するかどうかです。送信ボタンの横のモードセレクトを使用して、いつでもモードを切り替えることができます。「Ask permissions」から始めて Claude が実行する内容を正確に確認してから、慣れてきたら「Auto accept edits」または「Plan mode」に移動します。

モード	設定キー	動作
Ask permissions	<code>default</code>	Claude はファイルの編集またはコマンドの実行の前に確認を求めます。diff を確認して、各変更を受け入れるか拒否するかを選択できます。新規ユーザーに推奨されます。
Auto accept edits	<code>acceptEdits</code>	Claude はファイル編集を自動的に受け入れますが、ターミナルコマンドの実行前には確認を求めます。ファイル変更を信頼し、より高速な回復を望む場合に使用します。
Plan mode	<code>plan</code>	Claude はコードを分析し、ファイルを変更したりコマンドを実行したりせずにプランを作成します。アプローチを最初に確認したい複雑なタスクに適しています。
Bypass permissions	<code>bypassPermissions</code>	Claude は権限プロンプトなしで実行され、CLI の <code>-dangerously-skip-permissions</code> と同等です。Settings → Claude Code の「Allow bypass permissions mode」で有効にします。サンドボックス化されたコンテナまたは VM でのみ使用してください。エンタープライズ管理者はこのオプションを無効にできます。

`dontAsk` 権限モードは [CLI](#) でのみ利用可能です。

Tip:

複雑なタスクを Plan mode で開始して、Claude が変更を加える前にアプローチをマップアウトするようにします。プランを承認したら、'Auto accept edits' または 'Ask permissions' に切り替えて実行します。このワークフローの詳細については、[最初に探索してからプランしてからコードを書く](#)を参照してください。

リモートセッションは 'Auto accept edits' と 'Plan mode' をサポートしています。'Ask permissions' はリモートセッションがデフォルトでファイル編集を自動受け入れするため利用できず、'Bypass permissions' はリモート環境が既にサンドボックス化されているため利用できません。

エンタープライズ管理者は利用可能な権限モードを制限できます。詳細については、[エンタープライズ設定](#)を参照してください。

アプリをプレビューする

Claude は dev サーバーを起動し、埋め込みブラウザを開いて変更内容を確認できます。これはフロントエンド Web アプリとバックエンドサーバーの両方で機能します：Claude は API エンドポイントをテストし、サーバーログを表示し、見つけた問題を反復処理できます。ほとんどの場合、Claude はプロジェクトファイルを編集した後、サーバーを自動的に起動します。いつでも Claude にプレビューを要求することもできます。デフォルトでは、Claude は編集後に [変更を自動検証](#)します。

プレビューパネルから、以下を実行できます：

- 埋め込みブラウザで実行中のアプリと直接対話する
- Claude が自動的に独自の変更を検証するのを監視する：スクリーンショットを撮影し、DOM を検査し、要素をクリックし、フォームに入力し、見つけた問題を修正する
- セッションツールバーの **Preview** ドロップダウンからサーバーを開始または停止する
- ドロップダウンで **Persist sessions** を選択して、サーバーの再起動時にクッキーとローカルストレージを保持し、開発中に再度ログインする必要があるようにする
- サーバー設定を編集するか、すべてのサーバーを一度に停止する

Claude はプロジェクトに基づいて初期サーバー設定を作成します。アプリがカスタム dev コマンドを使用する場合、[.claude/launch.json](#) を編集してセットアップに合わせます。完全なリファレンスについては、[プレビューサーバーを設定する](#)を参照してください。

保存されたセッションデータをクリアするには、Settings → Claude Code で **Persist preview sessions** をオフに切り替えます。プレビューを完全に無効にするには、Settings → Claude Code で **Preview** をオフに切り替えます。

diff ビューで変更を確認する

Claude がコードに変更を加えた後、diff ビューを使用して、プルリクエストを作成する前にファイルごとに変更内容を確認できます。

Claude がファイルを変更すると、**+12 -1** などの追加および削除された行数を示す diff 統計インジケータが表示されます。このインジケータをクリックして diff ビューを開きます。左側にファイルリストが表示され、右側に各ファイルの変更が表示されます。

特定の行にコメントするには、diff 内の任意の行をクリックしてコメントボックスを開きます。フィードバックを入力して **Enter** キーを押してコメントを追加します。複数の行にコメントを追加した後、すべてのコメントを一度に送信します：

- **macOS** : **Cmd+Enter** を押す
- **Windows** : **Ctrl+Enter** を押す

Claude はコメントを読み取り、要求された変更を加えます。これは確認できる新しい diff として表示されます。

コードを確認する

diff ビューで、右上のツールバーの **Review code** をクリックして、Claude にコミット前に変更内容を評価するよう依頼します。Claude は現在の diff を検査し、diff ビューに直接コメントを残します。任意のコメントに応答するか、Claude に修正を依頼できます。

レビューは高シグナルの問題に焦点を当てます：コンパイルエラー、明確なロジックエラー、セキュリティ脆弱性、および明らかなバグです。スタイル、フォーマット、既存の問題、またはリンターが検出するものにはフラグを立てません。

プルリクエストステータスを監視する

プルリクエストを開いた後、CI ステータスバーがセッションに表示されます。Claude Code は GitHub CLI を使用してチェック結果をポーリングし、失敗を表示します。

- **Auto-fix** : 有効にすると、Claude は失敗出力を読み取り、反復することで、失敗した CI チェックを自動的に修正しようとします。
- **Auto-merge** : 有効にすると、Claude はすべてのチェックが成功したら PR をマージします。マージ方法はスカッシュです。Auto-merge がこれを機能させるために [GitHub リポジトリ設定で有効にされている](#) 必要があります。

CI ステータスバーの **Auto-fix** および **Auto-merge** トグルを使用して、いずれかのオプションを有効にします。Claude Code は CI が完了したときにデスクトップ通知も送信します。

Note:

PR 監視には、[GitHub CLI \(gh \)](#) がマシンにインストールされ、認証されている必要があります。gh がインストールされていない場合、Desktop は PR を作成しようとする最初の時点でインストールを促します。

セッションを管理する

各セッションは独立した会話で、独自のコンテキストと変更を持ちます。複数のセッションを並列で実行するか、作業をクラウドに送信できます。

セッションで並列に作業する

サイドバーの ****+ New session**** をクリックして、複数のタスクを並列で作業します。Git リポジトリの場合、各セッションは [Git worktrees](#) を使用してプロジェクトの独立したコピーを取得するため、1 つのセッションの変更は、コミットするまで他のセッションに影響しません。

Worktrees はデフォルトで `<project-root>/ .claude/worktrees/` に保存されます。Settings → Claude Code の 'Worktree location' でカスタムディレクトリに変更できます。また、すべての worktree ブランチ名の前に付加されるブランチプレフィックスを設定することもできます。これは Claude が作成したブランチを整理するのに便利です。完了したら、サイドバーのセッションにマウスを置いてアーカイブアイコンをクリックして worktree を削除します。

Note:

セッション分離には Git が必要です。ほとんどの Mac には Git がデフォルトで含まれています。Terminal で `git --version` を実行して確認してください。Windows では、Code タブが機能するために Git が必要です：[Git for Windows](#) をダウンロードし、インストールしてアプリを再起動してください。Git エラーが発生した場合は、Cowork セッションを試してセットアップのトラブルシューティングを行ってください。

サイドバーの上部のフィルタアイコンを使用して、ステータス (Active、Archived) と環境 (Local、Cloud) でセッションをフィルタリングします。セッションタイトルを確認するか、コンテキスト使用量を確認するには、アクティブセッションの上部のツールバーのセッションタイトルをクリックします。コンテキストがいっぱいになると、Claude は会話を自動的に要約して作業を続けます。`/compact` を入力して要約をより早くトリガーし、コンテキストスペースを解放することもできます。[コンテキストウィンドウ](#)を参照して、圧縮がどのように機能するかについての詳細を確認してください。

長時間実行されるタスクをリモートで実行する

大規模なリファクタリング、テストスイート、マイグレーション、またはその他の長時間実行されるタスクの場合、セッションを開始するときに**Local**ではなく**Remote**を選択します。リモートセッションはAnthropicのクラウドインフラストラクチャで実行され、アプリを閉じたりコンピュータをシャットダウンしたりしても続行します。いつでも戻ってきて進捗を確認するか、Claudeを別の方向に導くことができます。claude.ai/codeまたはClaude iOS アプリからリモートセッションを監視することもできます。

リモートセッションは複数のリポジトリもサポートしています。クラウド環境を選択した後、リポジトリピルの横の***ボタンをクリックして、セッションに追加のリポジトリを追加します。各リポジトリは独自のブランチセレクトアを取得します。これは共有ライブラリとそのコンシューマーの更新など、複数のコードベースにまたがるタスクに便利です。

リモートセッションがどのように機能するかについての詳細については、[Web上のClaude Code](#)を参照してください。

別のサーフェスで続行する

セッションツールバーの右下のVS Codeアイコンからアクセスできる**Continue in**メニューを使用すると、セッションを別のサーフェスに移動できます：

- **Claude Code on the Web**：ローカルセッションをリモートで実行し続けるために送信します。Desktopはブランチをプッシュし、会話の要約を生成し、完全なコンテキストを持つ新しいリモートセッションを作成します。その後、ローカルセッションをアーカイブするか保持するかを選択できます。これはクリーンなワーキングツリーが必要であり、SSHセッションでは利用できません。
- **Your IDE**：現在の作業ディレクトリでサポートされているIDEでプロジェクトを開きます。

Claude Code を拡張する

外部サービスを接続し、再利用可能なワークフローを追加し、Claudeの動作をカスタマイズし、プレビューサーバーを設定します。

外部ツールを接続する

ローカルおよびSSHセッションの場合、プロンプトボックスの横の***ボタンをクリックしてConnectors**を選択し、Google Calendar、Slack、GitHub、Linear、Notionなどの統合を追加します。セッション前またはセッション中にコネクタを追加できます。コネクタはリモートセッションでは利用できません。

コネクタを管理または切断するには、デスクトップアプリの Settings → Connectors に移動するか、プロンプトボックスの Connectors メニューから **Manage connectors** を選択します。

接続すると、Claude はカレンダーを読み取り、メッセージを送信し、問題を作成し、ツールと直接対話できます。セッションで設定されているコネクタについて Claude に尋ねることができます。

コネクタは [MCP サーバー](#) で、グラフィカルセットアップフローを備えています。サポートされているサービスとの迅速な統合に使用します。Connectors にリストされていない統合の場合、[設定ファイル](#) を使用して MCP サーバーを手動で追加します。[カスタムコネクタを作成](#) することもできます。

スキルを使用する

[スキル](#) は Claude ができることを拡張します。Claude は関連する場合に自動的にロードするか、直接呼び出すことができます：プロンプトボックスで **/** を入力するか、**+** ボタンをクリックして **Slash commands** を選択して、利用可能なものを参照します。これには [組み込みコマンド](#)、[カスタムスキル](#)、コードベースからのプロジェクトスキル、および [インストール済みプラグイン](#) からのスキルが含まれます。1 つを選択すると、入力フィールドで強調表示されます。その後タスクを入力して、通常どおり送信します。

プラグインをインストールする

[プラグイン](#) は、スキル、エージェント、フック、MCP サーバー、および LSP 設定を Claude Code に追加する再利用可能なパッケージです。ターミナルを使用せずにデスクトップアプリからプラグインをインストールできます。

ローカルおよび [SSH](#) セッションの場合、プロンプトボックスの横の ****+ ボタン** をクリックして **Plugins** を選択して、**インストール済みプラグイン** とそのコマンドを確認します。**プラグインを追加するには、サブメニューから Add plugin**** を選択してプラグインブラウザを開きます。これは、公式 Anthropic マーケットプレイスを含む、設定された [マーケットプレイス](#) から利用可能なプラグインを表示します。**Manage plugins** を選択して、プラグインを有効化、無効化、またはアンインストールします。

プラグインはユーザーアカウント、特定のプロジェクト、またはローカルのみにスコープできます。プラグインはリモートセッションでは利用できません。独自のプラグインを作成することを含む、完全なプラグインリファレンスについては、[プラグイン](#) を参照してください。

プレビューサーバーを設定する

Claude は dev サーバーセットアップを自動的に検出し、セッション開始時に選択したフォルダのルートの `.claude/launch.json` に設定を保存します。Preview はこのフォルダを作業ディレクトリとして使用するため、親フォルダを選択した場合、独自の dev サーバーを持つサブフォルダは自動的に検出されません。サブフォルダのサーバーで作業するには、そのフォルダで直接セッションを開始するか、設定を手動で追加します。

サーバーの起動方法をカスタマイズするには、たとえば `npm run dev` の代わりに `yarn dev` を使用するか、ポートを変更するには、ファイルを手動で編集するか、Preview ドロップダウンの **Edit configuration** をクリックしてコードエディタで開きます。ファイルはコメント付き JSON をサポートしています。

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000
    }
  ]
}
```

同じプロジェクトから異なるサーバーを実行するために複数の設定を定義できます。たとえば、フロントエンドと API です。以下の[例](#)を参照してください。

変更を自動検証する

`autoVerify` が有効な場合、Claude はファイル編集後に自動的にコード変更を検証します。スクリーンショットを撮影し、エラーをチェックし、変更が機能することを確認してから応答を完了します。

Auto-verify はデフォルトで有効です。`.claude/launch.json` に `"autoVerify": false` を追加してプロジェクトごとに無効にするか、**Preview** ドロップダウンメニューから切り替えます。

```
{
  "version": "0.0.1",
  "autoVerify": false,
  "configurations": [...]
}
```

無効にした場合、プレビューツールは引き続き利用可能で、いつでも Claude に検証を依頼できます。Auto-verify は編集後に自動的に行います。

設定フィールド

`configurations` 配列の各エントリは、以下のフィールドを受け入れます：

フィールド	型	説明
<code>name</code>	string	このサーバーの一意的識別子
<code>runtimeExecutable</code>	string	実行するコマンド (<code>npm</code> 、 <code>yarn</code> 、 <code>node</code> など)
<code>runtimeArgs</code>	string[]	<code>runtimeExecutable</code> に渡される引数 (<code>["run", "dev"]</code> など)
<code>port</code>	number	サーバーがリッスンするポート。デフォルトは 3000
<code>cwd</code>	string	プロジェクトルートに相対的な作業ディレクトリ。デフォルトはプロジェクトルート。プロジェクトルートを明示的に参照するには <code>\${workspaceFolder}</code> を使用

フィールド	型	説明
<code>env</code>	object	<code>{ "NODE_ENV": "development" }</code> などのキーと値のペアとしての追加環境変数。このファイルはリポジトリにコミットされるため、ここにシークレットを入れないください。シェルスクリプトで設定されたシークレットは自動的に継承されます。
<code>autoPort</code>	boolean	ポート競合の処理方法。以下を参照
<code>program</code>	string	<code>node</code> で実行するスクリプト。 <code>program vs runtimeExecutable</code> を 使用する場合 を参照
<code>args</code>	string[]	<code>program</code> に渡される引数。 <code>program</code> が設定されている場合のみ使用

`program` vs `runtimeExecutable` を使用する場合

`runtimeExecutable` を `runtimeArgs` と共に使用して、パッケージマネージャーを通じて dev サーバーを起動します。たとえば、`"runtimeExecutable": "npm"` と `"runtimeArgs": ["run", "dev"]` は `npm run dev` を実行します。

`node` で直接実行したいスタンドアロンスクリプトがある場合は `program` を使用します。たとえば、`"program": "server.js"` は `node server.js` を実行します。 `args` で追加フラグを渡します。

ポート競合

`autoPort` フィールドは、優先ポートが既に使用中の場合に何が起るかを制御します：

- `true` : Claude は自動的に空きポートを見つけて使用します。ほとんどの dev サーバーに適しています。
- `false` : Claude はエラーで失敗します。OAuth コールバックまたは CORS 許可リストなど、サーバーが特定のポートを使用する必要がある場合に使用します。

- ・ **設定されていない（デフォルト）**：Claude はサーバーがそのポートを必要とするかどうかを尋ねてから、答えを保存します。

Claude が別のポートを選択すると、割り当てられたポートを **PORT** 環境変数を通じてサーバーに渡します。

例

これらの設定は、異なるプロジェクトタイプの一般的なセットアップを示しています：

Next.js

この設定は、Yarn を使用してポート 3000 で Next.js アプリを実行します：

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "web",
      "runtimeExecutable": "yarn",
      "runtimeArgs": ["dev"],
      "port": 3000
    }
  ]
}
```

複数のサーバー

フロントエンドと API サーバーを持つモノレポの場合、複数の設定を定義します。フロントエンドは **autoPort: true** を使用して、3000 が使用中の場合は空きポートを選択し、API サーバーはポート 8080 を正確に必要とします：


```
{
  "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 スクリプト

パッケージマネージャーコマンドを使用する代わりに Node.js スクリプトを直接実行するには、`program` フィールドを使用します：

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "server",
      "program": "server.js",
      "args": ["--verbose"],
      "port": 4000
    }
  ]
}
```

定期的なタスクをスケジュールする

スケジュール済みタスクは、選択した時間と頻度で自動的に新しいローカルセッションを開始します。毎日のコードレビュー、依存関係の更新チェック、またはカレンダーとインボックスから取得する朝のブリーフィングなどの定期的な作業に使用します。

タスクはマシンで実行されるため、デスクトップアプリが開いていてコンピュータが起動していないと実行されません。詳細については、[スケジュール済みタスクがどのように実行されるか](#)を参照してください。

Note:

デフォルトでは、スケジュール済みタスクはコミットされていない変更を含む、作業ディレクトリの現在の状態に対して実行されます。プロンプト入力の `worktree` トグルを有効にして、各実行に独立した Git worktree を与えます。これは[並列セッション](#)と同じ方法です。

スケジュール済みタスクを作成するには、サイドバーの **Schedule** をクリックしてから ****+ New task**** をクリックします。これらのフィールドを設定します：

フィールド	説明
Name	タスクの識別子。小文字のケバブケースに変換され、ディスク上のフォルダ名として使用されます。タスク全体で一意である必要があります。
Description	タスクリストに表示される短い要約。

フィールド	説明
Prompt	タスクが実行されるときに Claude に送信される指示。プロンプトボックスで任意のメッセージを書くのと同じ方法で書きます。プロンプト入力には、モデル、権限モード、作業フォルダ、および worktree のコントロールも含まれます。
Frequency	タスクが実行される頻度。以下の 頻度オプション を参照してください。

任意のセッションで実行したいことを説明することでタスクを作成することもできます。たとえば、「毎朝 9 時に実行される毎日のコードレビューを設定する」。

頻度オプション

- **Manual** : スケジュールなし、**Run now**をクリックしたときのみ実行されます。オンデマンドでトリガーするプロンプトを保存するのに便利です
- **Hourly** : 毎時間実行されます。各タスクは API トラフィックを分散させるために、時間の上部から最大 10 分の固定オフセットを取得します
- **Daily** : 時間ピッカーを表示し、デフォルトは現地時間の午前 9 時です
- **Weekdays** : Daily と同じですが、土曜日と日曜日をスキップします
- **Weekly** : 時間ピッカーと日付ピッカーを表示します

ピッカーが提供しないインターバル（15 分ごと、毎月 1 日など）の場合、任意の Desktop セッションで Claude に尋ねてスケジュールを設定します。プレーンテキストを使用します。たとえば、「6 時間ごとにすべてのテストを実行するタスクをスケジュールする」。

スケジュール済みタスクがどのように実行されるか

スケジュール済みタスクはマシンでローカルに実行されます。Desktop はアプリが開いている間、毎分スケジュールをチェックし、開いている手動セッションとは独立して、タスクが期限になったときに新しいセッションを開始します。各タスクは、API トラフィックを分散させるために、スケジュール時刻の後に最大 10 分の固定遅延を取得します。遅延は決定的です：同じタスクは常に同じオフセットで開始されます。

タスクが実行されると、デスクトップ通知が表示され、新しいセッションがサイドバーの **Scheduled** セクションの下に表示されます。それを開いて Claude が実行した内容を確認し、変更を確認するか、権限プロンプトに応答します。セッションは他のセッションと同じように機能します：Claude はファイルを編集し、コマンドを実行し、コミットを作成し、プルリクエストを開くことができます。

タスクはデスクトップアプリが実行されていてコンピュータが起動している間のみ実行されます。コンピュータがスケジュール時刻を通じてスリープ状態になった場合、実行はスキップされます。アイドルスリープを防ぐには、Settings の **Desktop app** → **General** で **Keep computer awake** を有効にします。ラップトップの蓋を閉じるとまだスリープ状態になります。

見落とされた実行

アプリが起動するか、コンピュータが起動すると、Desktop は過去 7 日間に各タスクが見落とされた実行があるかどうかをチェックします。そうした場合、Desktop は最近見落とされた時刻に対して正確に 1 つのキャッチアップ実行を開始し、それより古いものは破棄します。6 日間見落とされた毎日のタスクは起動時に 1 回実行されます。Desktop はキャッチアップ実行が開始されたときに通知を表示します。

プロンプトを書くときはこれを念頭に置いてください。午前 9 時にスケジュールされたタスクは、コンピュータが一日中スリープ状態だった場合、午後 11 時に実行される可能性があります。タイミングが重要な場合は、プロンプト自体にガードレールを追加します。たとえば、「今日のコミットのみを確認してください。午後 5 時以降の場合は、レビューをスキップして、見落とされた内容の要約を投稿してください。」

スケジュール済みタスクの権限

各タスクは独自の権限モードを持ち、タスクを作成または編集するときに設定します。

`~/.claude/settings.json` からの許可ルールもスケジュール済みタスクセッションに適用されます。タスクが Ask モードで実行され、権限がないツールを実行する必要がある場合、実行は承認するまで停止します。セッションはサイドバーで開いたままなので、後で応答できます。

停止を避けるには、タスクを作成した後に **Run now** をクリックし、権限プロンプトを監視し、各プロンプトに対して「常に許可」を選択します。そのタスクの将来の実行は、プロンプトなしで同じツールを自動承認します。タスクの詳細ページからこれらの承認を確認および取り消すことができます。

スケジュール済みタスクを管理する

Schedule リストのタスクをクリックして、その詳細ページを開きます。ここから以下を実行できます：

- **Run now**：次のスケジュール時刻を待たずにタスクを直ちに開始する
- **Toggle repeats**：タスクを削除せずにスケジュール実行を一時停止または再開する
- **Edit**：プロンプト、頻度、フォルダ、またはその他の設定を変更する
- **Review history**：コンピュータがスリープ状態だったためにスキップされたものを含む、すべての過去の実行を確認する

- **Review allowed permissions : Always allowed** パネルからこのタスクの保存されたツール承認を確認および取り消す
- **Delete** : タスクを削除し、それが作成したすべてのセッションをアーカイブする

任意の Desktop セッションで Claude に尋ねることでタスクを管理することもできます。たとえば、「dependency-audit タスクを一時停止する」、「standup-prep タスクを削除する」、または「スケジュール済みタスクを表示する」。

ディスク上のタスクのプロンプトを編集するには、`~/.claude/scheduled-tasks/<task-name>/SKILL.md` を開きます（`CLAUDE_CONFIG_DIR` が設定されている場合はその下）。ファイルは `name` と `description` に YAML frontmatter を使用し、プロンプトを本文として使用します。変更は次の実行時に有効になります。スケジュール、フォルダ、モデル、および有効状態はこのファイルにはありません：Edit フォームを通じて変更するか、Claude に尋ねます。

環境設定

[セッションを開始する](#) ときに選択する環境は、Claude が実行される場所と接続方法を決定します：

- **Local** : マシンで実行され、ファイルに直接アクセスできます
- **Remote** : Anthropic のクラウドインフラストラクチャで実行されます。アプリを閉じて、セッションは続行されます。
- **SSH** : SSH 経由で接続するリモートマシンで実行されます。たとえば、独自のサーバー、クラウド VM、または dev コンテナなど

ローカルセッション

ローカルセッションはシェルから環境変数を継承します。追加の変数が必要な場合は、`~/.zshrc` または `~/.bashrc` などのシェルスクリプトファイルで設定し、デスクトップアプリを再起動します。サポートされている変数の完全なリストについては、[環境変数](#) を参照してください。

[拡張思考](#) はデフォルトで有効になっており、複雑な推論タスクのパフォーマンスを向上させますが、追加のトークンを使用します。思考を完全に無効にするには、シェルスクリプトファイルで `MAX_THINKING_TOKENS=0` を設定します。Opus では、適応的推論が思考の深さを制御するため、`MAX_THINKING_TOKENS` は `0` を除いて無視されます。

リモートセッション

リモートセッションはアプリを閉じて、バックグラウンドで続行されます。使用量は[サブスクリプションプランの制限](#)にカウントされ、別の計算料金はありありません。

異なるネットワークアクセスレベルと環境変数を持つカスタムクラウド環境を作成できます。リモートセッションを開始するときに環境ドロップダウンを選択し、**Add environment**を選択します。ネットワークアクセスと環境変数の設定の詳細については、[クラウド環境](#)を参照してください。

SSH セッション

SSH セッションを使用すると、デスクトップアプリをインターフェースとして使用しながら、リモートマシンで Claude Code を実行できます。これは、クラウド VM、dev コンテナ、または特定のハードウェアまたは依存関係を持つサーバーに存在するコードベースで作業するのに便利です。

SSH 接続を追加するには、セッションを開始する前に環境ドロップダウンをクリックして、**+ Add SSH connection**を選択します。ダイアログは以下を要求します：

- **Name**：この接続のフレンドリーラベル
- **SSH Host**：`user@hostname` または `~/.ssh/config` で定義されたホスト
- **SSH Port**：空のままの場合はデフォルトの 22、または SSH config からのポート
- **Identity File**：`~/.ssh/id_rsa` などの秘密鍵へのパス。デフォルトキーまたは SSH config を使用するには空のままにします。

追加されると、接続は環境ドロップダウンに表示されます。それを選択してそのマシンでセッションを開始します。Claude はリモートマシンで実行され、そのファイルとツールにアクセスできます。

Claude Code はリモートマシンにインストールされている必要があります。接続されると、SSH セッションは権限モード、コネクタ、プラグイン、および MCP サーバーをサポートします。

エンタープライズ設定

Teams または Enterprise プランの組織は、管理コンソールコントロール、管理設定ファイル、およびデバイス管理ポリシーを通じてデスクトップアプリの動作を管理できます。

管理コンソールコントロール

これらの設定は[管理設定コンソール](#)を通じて設定されます：

- **Code タブを有効または無効にする**：組織内のユーザーがデスクトップアプリで Claude Code にアクセスできるかどうかを制御する
- **Bypass permissions モードを無効にする**：組織内のユーザーが bypass permissions モードを有効にするのを防ぐ

- **Claude Code on the Web を無効にする**：組織のリモートセッションを有効または無効にする

管理設定

管理設定はプロジェクトおよびユーザー設定をオーバーライドし、Desktop が CLI セッションを生成するときに適用されます。これらのキーを組織の[管理設定](#)ファイルで設定するか、管理コンソールを通じてリモートでプッシュできます。

キー	説明
<code>disableBypassPermissionsMode</code>	Bypass permissions モードを有効にするのを防ぐために "disable" に設定します。 管理設定 を参照してください。

`allowManagedPermissionRulesOnly` および `allowManagedHooksOnly` を含む管理専用設定の完全なリストについては、[管理専用設定](#)を参照してください。

管理コンソールを通じてアップロードされたリモート管理設定は、現在 CLI および IDE セッションにのみ適用されます。Desktop 固有の制限については、上記の管理コンソールコントロールを使用します。

デバイス管理ポリシー

IT チームは、macOS の MDM または Windows のグループポリシーを通じてデスクトップアプリを管理できます。利用可能なポリシーには、Claude Code 機能の有効化または無効化、自動更新の制御、およびカスタムデプロイメント URL の設定が含まれます。

- **macOS**：Jamf または Kandji などのツールを使用して `com.anthropic.Claude` 優先ドメインを通じて設定
- **Windows**：`SOFTWARE\Policies\Claude` のレジストリを通じて設定

認証と SSO

エンタープライズ組織はすべてのユーザーに SSO を要求できます。プランレベルの詳細については[認証](#)を参照し、SAML および OIDC 設定については[SSO の設定](#)を参照してください。

データ処理

Claude Code はローカルセッションではコードをローカルで処理するか、リモートセッションでは Anthropic のクラウドインフラストラクチャで処理します。会話とコードコンテキストは処理のために Anthropic の API に送信されます。データ保持、プライバシー、およびコンプライアンスの詳細については、[データ処理](#)を参照してください。

デプロイメント

Desktop はエンタープライズデプロイメントツールを通じて配布できます：

- **macOS**：Jamf または Kandji などの MDM を使用して `.dmg` インストーラーを通じて配布
- **Windows**：MSIX パッケージまたは `.exe` インストーラーを通じてデプロイします。サイレントインストールを含むエンタープライズデプロイメントオプションについては、[Deploy Claude Desktop for Windows](#)を参照してください。

プロキシ設定、ファイアウォール許可リスト、LLM ゲートウェイなどのネットワーク設定については、[ネットワーク設定](#)を参照してください。

完全なエンタープライズ設定リファレンスについては、[エンタープライズ設定ガイド](#)を参照してください。

CLI から来ましたか？

既に Claude Code CLI を使用している場合、Desktop は同じ基盤となるエンジンをグラフィカルインターフェースで実行します。同じマシン上で、同じプロジェクト上でも、両方を同時に実行できます。各々は個別のセッション履歴を保持しますが、`CLAUDE.md` ファイルを通じて設定とプロジェクトメモリを共有します。

CLI セッションを Desktop に移動するには、ターミナルで `/desktop` を実行します。Claude はセッションを保存し、デスクトップアプリで開いてから CLI を終了します。このコマンドは macOS と Windows でのみ利用可能です。

Tip:

Desktop と CLI をいつ使用するか：ビジュアル diff レビュー、ファイル添付、またはサイドバーのセッション管理が必要な場合は Desktop を使用します。スクリプト、自動化、サードパーティプロバイダー、またはターミナルワークフローを優先する場合は CLI を使用します。

CLI フラグの同等物

このテーブルは、一般的な CLI フラグの Desktop 同等物を示しています。リストされていないフラグは、スクリプトまたは自動化用に設計されているため、Desktop 同等物がありません。

CLI	Desktop 同等物
<code>--model sonnet</code>	セッション開始前に送信ボタンの横のモデルドロップダウン

CLI	Desktop 同等物
<code>--resume</code> 、 <code>--continue</code>	サイドバーのセッションをクリック
<code>--permission-mode</code>	送信ボタンの横のモードセレクト
<code>--dangerously-skip-permissions</code>	Bypass permissions モード。Settings → Claude Code → 「Allow bypass permissions mode」で有効にします。エンタープライズ管理者はこの設定を無効にできます。
<code>--add-dir</code>	リモートセッションで***ボタンで複数のリポジトリを追加
<code>--allowedTools</code> 、 <code>--disallowedTools</code>	Desktop では利用できません
<code>--verbose</code>	利用できません。システムログを確認してください：macOS の Console.app、Windows の Event Viewer → Windows Logs → Application
<code>--print</code> 、 <code>--output-format</code>	利用できません。Desktop はインタラクティブのみです。
<code>ANTHROPIC_MODEL</code> env var	送信ボタンの横のモデルドロップダウン
<code>MAX_THINKING_TOKENS</code> env var	シェルスクリプトファイルで設定；ローカルセッションに適用されます。 環境設定 を参照してください。

共有設定

Desktop と CLI は同じ設定ファイルを読み取るため、セットアップが引き継がれます：

- プロジェクト内の**[CLAUDE.md](#)
- **ファイルは両方で使用されます
- `~/.claude.json` または `.mcp.json` で設定された**[MCP サーバー](#)
- **は両方で機能します
- 設定で定義された**[フックおよびスキル](#)
- **は両方に適用されます
- `~/.claude.json` および `~/.claude/settings.json` の**[設定](#)
- **は共有されます。`settings.json` の権限ルール、許可されたツール、およびその他の設定は Desktop セッションに適用されます。
- モデル**：Sonnet、Opus、Haiku は両方で利用可能です。Desktop では、セッションを開始する前に送信ボタンの横のドロップダウンからモデルを選択します。アクティブなセッション中にモデルを変更することはできません。

Note:

MCP サーバー：Desktop チャットアプリ vs Claude Code： Claude Desktop チャットアプリの `claude_desktop_config.json` で設定された MCP サーバーは Claude Code とは別であり、Code タブに表示されません。Claude Code で MCP サーバーを使用するには、`~/claude.json` またはプロジェクトの `.mcp.json` ファイルで設定します。詳細については、[MCP 設定](#) を参照してください。

機能比較

このテーブルは、CLI と Desktop 間のコア機能を比較しています。CLI フラグの完全なリストについては、[CLI リファレンス](#) を参照してください。

機能	CLI	Desktop
権限モード	<code>dontAsk</code> を含むすべてのモード	Ask permissions、Auto accept edits、Plan mode、および Settings 経由の Bypass permissions
<code>--dangerously-skip-permissions</code>	CLI フラグ	Bypass permissions モード。Settings → Claude Code → 「Allow bypass permissions mode」で有効にします
サードパーティプロバイダー	Bedrock、Vertex、Foundry	利用できません。Desktop は Anthropic の API に直接接続します。
MCP サーバー	設定ファイルで設定	ローカルおよび SSH セッションの Connectors UI、または設定ファイル
プラグイン	<code>/plugin</code> コマンド	プラグインマネージャー UI
@mention ファイル	テキストベース	オートコンプリート付き
ファイル添付	利用できません	画像、PDF
セッション分離	<code>--worktree</code> フラグ	自動 worktrees
複数セッション	別のターミナル	サイドバータブ

機能	CLI	Desktop
定期的なタスク	cron ジョブ、CI パイプライン	スケジュール済みタスク
スクリプトと自動化	<code>--print</code> 、 Agent SDK	利用できません

Desktop では利用できないもの

以下の機能は CLI または VS Code 拡張機能でのみ利用可能です：

- ・ **サードパーティプロバイダー**：Desktop は Anthropic の API に直接接続します。代わりに Bedrock、Vertex、または Foundry で [CLI](#) を使用してください。
- ・ **Linux**：デスクトップアプリは macOS と Windows でのみ利用可能です。
- ・ **インラインコード提案**：Desktop はオートコンプリートスタイルの提案を提供しません。会話型プロンプトと明示的なコード変更を通じて機能します。
- ・ **エージェントチーム**：マルチエージェントオーケストレーションは [CLI](#) および [Agent SDK](#) を通じて利用可能で、Desktop では利用できません。

トラブルシューティング

バージョンを確認する

実行しているデスクトップアプリのバージョンを確認するには：

- ・ **macOS**：メニューバーの **Claude** をクリックしてから、**About Claude** をクリック
- ・ **Windows**：**Help** をクリックしてから、**About** をクリック

バージョン番号をクリックしてクリップボードにコピーします。

Code タブの 403 またはエラー認証

Code タブを使用するときに **Error 403: Forbidden** またはその他の認証エラーが表示される場合：

1. アプリメニューからサインアウトしてから再度サインインします。これが最も一般的な修正です。
2. アクティブな有料サブスクリプション（Pro、Max、Teams、または Enterprise）があることを確認します。
3. CLI は機能するが Desktop は機能しない場合、デスクトップアプリを完全に終了し（ウィンドウを閉じるだけでなく）、再度開いてサインインします。
4. インターネット接続とプロキシ設定を確認します。

起動時に空白または停止画面

アプリが開いても空白または応答しない画面が表示される場合：

1. アプリを再起動します。
2. 保留中の更新を確認します。アプリは起動時に自動更新されます。
3. Windows では、**Windows Logs → Application**の Event Viewer でクラッシュログを確認します。

「Failed to load session」

Failed to load session が表示される場合、選択したフォルダが存在しなくなった可能性があります。Git リポジトリがインストールされていない Git LFS を必要とする可能性があります。またはファイル権限がアクセスを防ぐ可能性があります。別のフォルダを選択するか、アプリを再起動してみてください。

セッションがインストール済みツールを見つけれられない

Claude が **npm**、**node**、またはその他の CLI コマンドなどのツールを見つけれられない場合、ツールが通常のターミナルで機能することを確認し、シェルスクリプトファイルが PATH を正しく設定していることを確認し、デスクトップアプリを再起動して環境変数を再ロードします。

Git および Git LFS エラー

Windows では、Code タブがローカルセッションを開始するために Git が必要です。

「Git is required」が表示される場合、[Git for Windows](#)をインストールしてアプリを再起動します。

「Git LFS is required by this repository but is not installed」が表示される場合、[git-lfs.com](#)から Git LFS をインストールし、**git lfs install** を実行してアプリを再起動します。

Windows で MCP サーバーが機能しない

MCP サーバートグルが応答しない場合、または Windows でサーバーが接続に失敗する場合、サーバーが設定で正しく設定されていることを確認し、アプリを再起動し、Task Manager でサーバープロセスが実行されていることを確認し、接続エラーについてサーバーログを確認します。

アプリが終了しない

- **macOS** : Cmd+Q を押します。アプリが応答しない場合、Cmd+Option+Esc で Force Quit を使用し、Claude を選択して Force Quit をクリックします。

- **Windows** : Ctrl+Shift+Esc で Task Manager を使用して Claude プロセスを終了します。

Windows 固有の問題

- **インストール後に PATH が更新されない** : 新しいターミナルウィンドウを開きます。PATH の更新は新しいターミナルセッションにのみ適用されます。
- **同時インストールエラー** : 別のインストールが進行中であるというエラーが表示されるが、実際には進行中でない場合、インストーラーを管理者として実行してみてください。
- **ARM64** : Windows ARM64 デバイスは完全にサポートされています。

Intel Mac で Cowork タブが利用できない

Cowork タブは macOS で Apple Silicon (M1 以降) が必要です。Windows では、Cowork はサポートされているすべてのハードウェアで利用可能です。Chat タブと Code タブは Intel Mac で正常に機能します。

CLI で開くときに「Branch doesn't exist yet」

リモートセッションはローカルマシンに存在しないブランチを作成できます。セッションツールバーのブランチ名をクリックしてコピーしてから、ローカルでフェッチします：

```
git fetch origin <branch-name>
git checkout <branch-name>
```

まだ困っていますか？

- [GitHub Issues](#) で検索するか、バグを報告する
- [Claude サポートセンター](#) にアクセス

バグを報告するときは、デスクトップアプリのバージョン、オペレーティングシステム、正確なエラーメッセージ、および関連ログを含めます。macOS では Console.app を確認してください。Windows では Event Viewer → Windows Logs → Application を確認してください。

Chrome で Claude Code を使用する（ベータ版）

Claude Code を Chrome ブラウザに接続して、Web アプリをテストし、コンソールログでデバッグし、フォーム入力を自動化し、Web ページからデータを抽出します。

Claude Code は Claude in Chrome ブラウザ拡張機能と統合され、CLI または [VS Code 拡張機能](#) からブラウザ自動化機能を提供します。コードをビルドしてから、コンテキストを切り替えることなくブラウザでテストおよびデバッグできます。

Claude はブラウザタスク用に新しいタブを開き、ブラウザのログイン状態を共有するため、既にサインインしているサイトにアクセスできます。ブラウザアクションはリアルタイムで表示される Chrome ウィンドウで実行されます。Claude がログインページまたは CAPTCHA に遭遇した場合、一時停止して手動で処理するよう求めます。

Note:

Chrome 統合はベータ版であり、現在 Google Chrome のみで動作します。Brave、Arc、またはその他の Chromium ベースのブラウザではまだサポートされていません。WSL（Windows Subsystem for Linux）もサポートされていません。

機能

Chrome が接続されている場合、単一のワークフロー内でブラウザアクションとコーディングタスクをチェーンできます。

- **ライブデバッグ**：コンソールエラーと DOM 状態を直接読み取り、それらを引き起こしたコードを修正します
- **デザイン検証**：Figma モックから UI をビルドしてから、ブラウザで開いて一致することを確認します
- **Web アプリテスト**：フォーム検証をテストし、ビジュアルリグレッションをチェックするか、ユーザーフローを検証します
- **認証済み Web アプリ**：API コネクタなしで、ログインしている Google Docs、Gmail、Notion、またはその他のアプリと対話します
- **データ抽出**：Web ページから構造化情報を取得してローカルに保存します
- **タスク自動化**：データ入力、フォーム入力、またはマルチサイトワークフローなどの反復的なブラウザタスクを自動化します

- ・ **セッション記録**：ブラウザインタラク션을 GIF として記録して、何が起こったかを文書化または共有します

前提条件

Claude Code を Chrome で使用する前に、以下が必要です。

- ・ [Google Chrome](#) ブラウザ
- ・ [Claude in Chrome 拡張機能](#) バージョン 1.0.36 以上
- ・ [Claude Code](#) バージョン 2.0.73 以上
- ・ 直接 Anthropic プラン（Pro、Max、Team、または Enterprise）

Note:

Chrome 統合は Amazon Bedrock、Google Cloud Vertex AI、Microsoft Foundry などのサードパーティプロバイダーを通じては利用できません。Claude にサードパーティプロバイダーを通じてのみアクセスする場合、この機能を使用するには別の claude.ai アカウントが必要です。

CLI で開始する

Step 1: Chrome で Claude Code を起動する

`--chrome` フラグで Claude Code を起動します。

```
claude --chrome
```

既存のセッション内から `/chrome` を実行して Chrome を有効にすることもできます。

Step 2: Claude にブラウザを使用するよう依頼する

この例は、ページに移動し、それと対話し、ターミナルまたはエディターからすべてを報告します。

```
Go to code.claude.com/docs, click on the search box,  
type "hooks", and tell me what results appear
```

いつでも `/chrome` を実行して接続ステータスを確認し、権限を管理するか、拡張機能を再接続できます。

VS Code については、[VS Code でのブラウザ自動化](#) を参照してください。

Chrome をデフォルトで有効にする

各セッションで `--chrome` を渡すことを避けるには、`/chrome` を実行して「Enabled by default」を選択します。

[VS Code 拡張機能](#) では、Chrome 拡張機能がインストールされている場合、Chrome はいつでも利用可能です。追加のフラグは必要ありません。

Note:

CLI で Chrome をデフォルトで有効にすると、ブラウザツールが常にロードされるため、コンテキスト使用量が増加します。コンテキスト消費の増加に気付いた場合、この設定を無効にして、必要な場合にのみ `--chrome` を使用してください。

サイト権限を管理する

サイトレベルの権限は Chrome 拡張機能から継承されます。Chrome 拡張機能の設定で権限を管理して、Claude がブラウズ、クリック、入力できるサイトを制御します。

ワークフロー例

これらの例は、ブラウザアクションとコーディングタスクを組み合わせる一般的な方法を示しています。`/mcp` を実行して `claude-in-chrome` を選択すると、利用可能なブラウザツールの完全なリストが表示されます。

ローカル Web アプリケーションをテストする

Web アプリを開発する場合、変更が正しく機能することを確認するよう Claude に依頼します。

```
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 はローカルサーバーに移動し、フォームと対話し、観察したことを報告します。

コンソールログでデバッグする

Claude はコンソール出力を読み取って問題の診断を支援できます。ログが詳細になる可能性があるため、すべてのコンソール出力を要求するのではなく、探すパターンを Claude に伝えます。


```
Open the dashboard page and check the console for any errors when
the page loads.
```

Claude はコンソールメッセージを読み取り、特定のパターンまたはエラータイプでフィルタリングできます。

フォーム入力を自動化する

反復的なデータ入力タスクを高速化します。

```
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 はローカルファイルを読み取り、Web インターフェースをナビゲートし、各レコードのデータを入力します。

Google Docs でコンテンツをドラフトする

API セットアップなしで Claude を使用してドキュメントに直接書き込みます。

```
Draft a project update based on the recent commits and add it to my
Google Doc at docs.google.com/document/d/abc123
```

Claude はドキュメントを開き、エディターをクリックしてコンテンツを入力します。これは、ログインしているあらゆる Web アプリで機能します。Gmail、Notion、Sheets など。

Web ページからデータを抽出する

Web サイトから構造化情報を取得します。

```
Go to the product listings page and extract the name, price, and
availability for each item. Save the results as a CSV file.
```

Claude はページに移動し、コンテンツを読み取り、データを構造化形式にコンパイルします。

マルチサイトワークフローを実行する

複数の Web サイト間でタスクを調整します。

```
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 はタブ間で動作して情報を収集し、ワークフローを完了します。

デモ GIF を記録する

ブラウザインタラクションの共有可能な記録を作成します。

```
Record a GIF showing how to complete the checkout flow, from adding
an item to the cart through to the confirmation page.
```

Claude はインタラクションシーケンスを記録し、GIF ファイルとして保存します。

トラブルシューティング

拡張機能が検出されない

Claude Code が「Chrome extension not detected」を表示する場合。

1. Chrome 拡張機能が `chrome://extensions` にインストールされ、有効になっていることを確認します
2. `claude --version` を実行して Claude Code が最新であることを確認します
3. Chrome が実行されていることを確認します
4. `/chrome` を実行して「Reconnect extension」を選択し、接続を再確立します
5. 問題が解決しない場合、Claude Code と Chrome の両方を再起動します

Chrome 統合を初めて有効にすると、Claude Code はネイティブメッセージングホスト設定ファイルをインストールします。Chrome はスタートアップ時にこのファイルを読み取るため、最初の試行で拡張機能が検出されない場合、Chrome を再起動して新しい設定を取得します。

接続がまだ失敗する場合、ホスト設定ファイルが以下の場所に存在することを確認します。

- **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** : Windows レジストリで `HKCU\Software\Google\Chrome\NativeMessagingHosts\` を確認します

ブラウザが応答しない

Claude のブラウザコマンドが機能しなくなった場合。

1. モーダルダイアログ（alert、confirm、prompt）がページをブロックしているかどうかを確認します。JavaScript ダイアログはブラウザイベントをブロックし、Claude がコマンドを受け取るのを防ぎます。ダイアログを手動で閉じてから、Claude に続行するよう伝えます。
2. Claude に新しいタブを作成して再度試すよう依頼します
3. `chrome://extensions` で拡張機能を無効にしてから再度有効にして Chrome 拡張機能を再起動します

長いセッション中に接続が切れる

Chrome 拡張機能のサービスワーカーは長時間のセッション中にアイドル状態になる可能性があります、接続が切れます。非アクティブ期間後にブラウザツールが機能しなくなった場合、`/chrome` を実行して「Reconnect extension」を選択します。

Windows 固有の問題

Windows では、以下の問題が発生する可能性があります。

- **名前付きパイプの競合（EADDRINUSE）** : 別のプロセスが同じ名前付きパイプを使用している場合、Claude Code を再起動します。Chrome を使用している他の Claude Code セッションを閉じます。
- **ネイティブメッセージングホストエラー** : ネイティブメッセージングホストがスタートアップ時にクラッシュする場合、Claude Code を再インストールしてホスト設定を再生成してみてください。

一般的なエラーメッセージ

これらは最も頻繁に遭遇するエラーと、それらを解決する方法です。

エラー	原因	修正
“Browser extension is not connected”	ネイティブメッセージングホストが拡張機能に到達できない	Chrome と Claude Code を再起動してから、 <code>/chrome</code> を実行して再接続します

エラー	原因	修正
“Extension not detected”	Chrome 拡張機能がインストールされていないか、無効になっている	<code>chrome://extensions</code> で拡張機能をインストールまたは有効にします
“No tab available”	Claude がタブの準備ができる前に動作しようとした	Claude に新しいタブを作成して再度試すよう依頼します
“Receiving end does not exist”	拡張機能サービスワーカーがアイドル状態になった	<code>/chrome</code> を実行して「Reconnect extension」を選択します

関連項目

- [VS Code で Claude Code を使用する](#)：VS Code 拡張機能でのブラウザ自動化
- [CLI リファレンス](#)：`--chrome` を含むコマンドラインフラグ
- [一般的なワークフロー](#)：Claude Code を使用するその他の方法
- [データとプライバシー](#)：Claude Code がデータを処理する方法
- [Chrome で Claude を使い始める](#)：ショートカット、スケジューリング、権限を含む Chrome 拡張機能の完全なドキュメント

Slack での Claude Code

Slack ワークスペースから直接コーディングタスクを委任する

Slack での Claude Code は、Claude Code の機能を Slack ワークスペースに直接もたらしめます。`@Claude` にコーディングタスクをメンションすると、Claude は自動的に意図を検出し、ウェブ上で Claude Code セッションを作成します。これにより、チームの会話を離れることなく開発作業を委任できます。

この統合は既存の Claude for Slack アプリに基づいていますが、コーディング関連のリクエストに対して Claude Code ウェブへのインテリジェントなルーティングを追加しています。

ユースケース

- ・ **バグ調査と修正:** Slack チャンネルで報告されたバグを Claude に調査・修正させます。
- ・ **迅速なコードレビューと修正:** Claude にチームのフィードバックに基づいて小さな機能を実装したりコードをリファクタリングさせます。
- ・ **協調的なデバッグ:** チームの議論が重要なコンテキスト（エラーの再現やユーザーレポートなど）を提供する場合、Claude はその情報を使用してデバッグアプローチを知らせることができます。
- ・ **並列タスク実行:** Slack でコーディングタスクを開始しながら他の作業を続け、完了時に通知を受け取ります。

前提条件

Claude Code in Slack を使用する前に、以下を確認してください：

要件	詳細
Claude ブラン	Pro、Max、Team、または Claude Code アクセス付き Enterprise（プレミアムシート）
ウェブ上の Claude Code	ウェブ上の Claude Code へのアクセスが有効になっている必要があります
GitHub アカウント	ウェブ上の Claude Code に接続され、少なくとも 1 つのリポジトリが認証されている

要件	詳細
Slack 認証	Slack アカウントが Claude アプリを通じて Claude アカウントにリンクされている

Slack での Claude Code のセットアップ

Step 1: Slack に Claude アプリをインストールする

ワークスペース管理者は Slack App Marketplace から Claude アプリをインストールする必要があります。[Slack App Marketplace](#) にアクセスして「Add to Slack」をクリックしてインストールプロセスを開始します。

Step 2: Claude アカウントを接続する

アプリがインストールされた後、個別の Claude アカウントを認証します：

1. Apps セクションで「Claude」をクリックして Slack で Claude アプリを開きます
2. App Home タブに移動します
3. 「Connect」をクリックして Slack アカウントを Claude アカウントにリンクします
4. ブラウザで認証フローを完了します

Step 3: ウェブ上の Claude Code を設定する

ウェブ上の Claude Code が適切に設定されていることを確認します：

- claude.ai/code にアクセスして、Slack に接続したのと同じアカウントでサインインします
- GitHub アカウントがまだ接続されていない場合は接続します
- Claude が作業するリポジトリを少なくとも 1 つ認証します

Step 4: ルーティングモードを選択する

アカウントを接続した後、Claude が Slack のメッセージをどのように処理するかを設定します。Slack の Claude App Home に移動して、**ルーティングモード**設定を見つけます。

モード	動作
Code のみ	Claude はすべての @mentions を Claude Code セッションにルーティングします。 Claude を Slack で開発タスク専用使用するチームに最適です。

モード	動作
Code + Chat	Claude は各メッセージを分析し、Claude Code（コーディングタスク用）と Claude Chat（執筆、分析、一般的な質問用）の間でインテリジェントにルーティングします。すべてのタイプの作業に対して単一の @Claude エントリーポイントが必要なチームに最適です。

Note:

Code + Chat モードでは、Claude がメッセージを Chat にルーティングしたがコーディングセッションが必要な場合は、「Retry as Code」をクリックして Claude Code セッションを作成できます。同様に、Code にルーティングされたが Chat セッションが必要な場合は、そのスレッドでそのオプションを選択できます。

仕組み

自動検出

Slack チャンネルまたはスレッドで @Claude をメンションすると、Claude は自動的にメッセージを分析してコーディングタスクかどうかを判断します。Claude がコーディング意図を検出した場合、通常のチャットアシスタントとして応答する代わりに、リクエストをウェブ上の Claude Code にルーティングします。

また、Claude が自動的に検出しない場合でも、リクエストをコーディングタスクとして処理するよう Claude に明示的に指示することもできます。

Note:

Slack での Claude Code はチャンネル（公開または非公開）でのみ機能します。ダイレクトメッセージ（DM）では機能しません。

コンテキスト収集

スレッドから: スレッドで @Claude をメンションすると、そのスレッド内のすべてのメッセージからコンテキストを収集して、完全な会話を理解します。

チャンネルから: チャンネルで直接メンションされた場合、Claude は関連するコンテキストについて最近のチャンネルメッセージを確認します。

このコンテキストは Claude が問題を理解し、適切なリポジトリを選択し、タスクへのアプローチを知らせるのに役立ちます。

Warning:

@Claude が Slack で呼び出されると、Claude はリクエストをより良く理解するために会話コンテキストへのアクセスが与えられます。Claude は他のメッセージからの指示に従う可能性があるため、ユーザーは信頼できる Slack 会話でのみ Claude を使用するようになっています。

セッションフロー

1. **開始:** @Claude にコーディングリクエストをメンションします
2. **検出:** Claude がメッセージを分析してコーディング意図を検出します
3. **セッション作成:** claude.ai/code で新しい Claude Code セッションが作成されます
4. **進捗更新:** Claude は作業が進むにつれて Slack スレッドにステータス更新を投稿します
5. **完了:** 完了時に、Claude は概要とアクションボタンを含めてあなたをメンションします
6. **レビュー:** 「View Session」をクリックして完全なトランスクリプトを表示するか、「Create PR」をクリックしてプルリクエストを開きます

ユーザーインターフェース要素

App Home

App Home タブは接続ステータスを表示し、Claude アカウントを Slack から接続または切断できます。

メッセージアクション

- **View Session:** ブラウザで完全な Claude Code セッションを開き、実行されたすべての作業、セッションの継続、または追加のリクエストを確認できます。
- **Create PR:** セッションの変更から直接プルリクエストを作成します。
- **Retry as Code:** Claude が最初はチャットアシスタントとして応答したがコーディングセッションが必要な場合は、このボタンをクリックしてリクエストを Claude Code タスクとして再試行します。
- **Change Repo:** Claude が誤って選択した場合、別のリポジトリを選択できます。

リポジトリ選択

Claude は Slack 会話のコンテキストに基づいてリポジトリを自動的に選択します。複数のリポジトリが適用される可能性がある場合、Claude は正しいものを選択できるドロップダウンを表示する場合があります。

アクセスと権限

ユーザーレベルのアクセス

アクセスタイプ	要件
Claude Code セッション	各ユーザーは自分の Claude アカウントでセッションを実行します
使用状況とレート制限	セッションは個別ユーザーのプラン制限に対してカウントされます
リポジトリアクセス	ユーザーは個人的に接続したリポジトリにのみアクセスできます
セッション履歴	セッションは <code>claude.ai/code</code> の Claude Code 履歴に表示されます

ワークスペース管理者の権限

Slack ワークスペース管理者は、Claude アプリをワークスペースにインストールできるかどうかを制御します。その後、個別ユーザーが自分の Claude アカウントで認証して統合を使用します。

どこでアクセスできるか

Slack で: ステータス更新、完了概要、アクションボタンが表示されます。完全なトランスクリプトは保存され、常にアクセス可能です。

ウェブで: 完全な Claude Code セッション、完全な会話履歴、すべてのコード変更、ファイル操作、セッションの継続またはプルリクエストの作成機能があります。

ベストプラクティス

効果的なリクエストの作成

- **具体的に:** ファイル名、関数名、またはエラーメッセージが関連する場合は含めます。
- **コンテキストを提供:** 会話から明確でない場合はリポジトリまたはプロジェクトをメンションします。
- **成功を定義:** 「完了」とはどういう意味か説明します。Claude はテストを書くべきですか？ドキュメントを更新しますか？PR を作成しますか？
- **スレッドを使用:** バグや機能について議論する場合はスレッドで返信して、Claude が完全なコンテキストを収集できるようにします。

Slack とウェブの使い分け

Slack を使用する場合: コンテキストが既に Slack の議論に存在する場合、タスクを非同期で開始したい場合、またはチームメイトが可視性を必要とする場合に協力しています。

ウェブを直接使用する場合: ファイルをアップロードする必要がある場合、開発中のリアルタイムインタラクションが必要な場合、またはより長く複雑なタスクに取り組んでいる場合。

トラブルシューティング

セッションが開始しない

1. Claude アカウントが Claude App Home で接続されていることを確認します
2. ウェブ上の Claude Code アクセスが有効になっていることを確認します
3. Claude Code に接続された GitHub リポジトリが少なくとも 1 つあることを確認します

リポジトリが表示されない

1. claude.ai/code で Claude Code on the web でリポジトリを接続します
2. そのリポジトリの GitHub 権限を確認します
3. GitHub アカウントを切断して再接続してみます

誤ったリポジトリが選択された

1. 「Change Repo」 ボタンをクリックして別のリポジトリを選択します
2. より正確な選択のためにリクエストにリポジトリ名を含めます

認証エラー

1. App Home で Claude アカウントを切断して再接続します
2. ブラウザで正しい Claude アカウントにサインインしていることを確認します
3. Claude プランに Claude Code アクセスが含まれていることを確認します

セッション有効期限

1. セッションはウェブ上の Claude Code 履歴でアクセス可能なままです
2. claude.ai/code から過去のセッションを継続または参照できます

現在の制限事項

- **GitHub のみ:** 現在、GitHub 上のリポジトリのみをサポートしています。
- **一度に 1 つの PR:** 各セッションは 1 つのプルリクエストを作成できます。

- **レート制限が適用:** セッションは個別の Claude プランのレート制限を使用します。
- **ウェブアクセスが必要:** ユーザーは Claude Code on the web アクセスを持つ必要があります。持たないユーザーは標準的な Claude チャット応答のみを取得します。

関連リソース

- [ウェブ上の Claude Code](#) ウェブ上の Claude Code について詳しく学ぶ
- [Claude for Slack](#) Claude for Slack の一般的なドキュメント
- [Slack App Marketplace](#) Slack Marketplace から Claude アプリをインストール
- [Claude ヘルプセンター](#) 追加サポートを取得

ウェブ上の Claude Code

セキュアなクラウドインフラストラクチャで Claude Code タスクを非同期に実行します

Note:

ウェブ上の Claude Code は現在リサーチプレビュー段階です。

ウェブ上の Claude Code とは？

ウェブ上の Claude Code を使用すると、開発者は Claude アプリから Claude Code を開始できます。これは以下の場合に最適です：

- **質問への回答**：コードアーキテクチャと機能の実装方法について質問する
- **バグ修正とルーチンタスク**：頻繁な操舵を必要としない明確に定義されたタスク
- **並列作業**：複数のバグ修正を並列で処理する
- **ローカルマシンにないリポジトリ**：ローカルにチェックアウトしていないコードで作業する
- **バックエンド変更**：Claude Code がテストを作成してからそのテストに合格するコードを作成できる場所

Claude Code は [iOS](#) および [Android](#) の Claude アプリでも利用可能で、外出先でタスクを開始し、進行中の作業を監視できます。

[ターミナルから `--remote` でウェブ上で新しいタスクを開始](#)するか、[ウェブセッションをターミナルにテレポートして](#)ローカルで続行できます。クラウドインフラストラクチャの代わりに自分のマシンで Claude Code を実行しながらウェブインターフェースを使用するには、[リモートコントロール](#)を参照してください。

ウェブ上の Claude Code は誰が使用できますか？

ウェブ上の Claude Code はリサーチプレビューで以下のユーザーが利用できます：

- **Pro ユーザー**
- **Max ユーザー**
- **Team ユーザー**
- **Enterprise ユーザー**（プレミアムシートまたは Chat + Claude Code シート付き）

はじめに

1. claude.ai/code にアクセスします
2. GitHub アカウントを接続します
3. リポジトリに Claude GitHub アプリをインストールします
4. デフォルト環境を選択します
5. コーディングタスクを送信します
6. diff ビューで変更を確認し、コメントで反復処理してから、プルリクエストを作成します

仕組み

ウェブ上の Claude Code でタスクを開始すると：

1. **リポジトリのクローン**：リポジトリが Anthropic 管理の仮想マシンにクローンされます
2. **環境セットアップ**：Claude はコードを含むセキュアなクラウド環境を準備し、設定されている場合は[セットアップスクリプト](#)を実行します
3. **ネットワーク設定**：インターネットアクセスは設定に基づいて構成されます
4. **タスク実行**：Claude はコードを分析し、変更を加え、テストを実行し、その作業を確認します
5. **完了**：完了時に通知され、変更を含むプルリクエストを作成できます
6. **結果**：変更がブランチにプッシュされ、プルリクエスト作成の準備ができます

diff ビューで変更を確認する

Diff ビューを使用すると、プルリクエストを作成する前に Claude が何を変更したかを正確に確認できます。GitHub で「Create PR」をクリックして変更を確認する代わりに、アプリで diff を直接表示し、変更の準備ができるまで Claude と反復処理します。

Claude がファイルに変更を加えると、追加および削除された行数を示す diff 統計インジケータ（例：**+12 -1**）が表示されます。このインジケータを選択して diff ビューを開くと、左側にファイルリストが表示され、右側に各ファイルの変更が表示されます。

diff ビューから、以下のことができます：

- ・ ファイルごとに変更を確認する
- ・ 特定の変更にコメントして修正をリクエストする
- ・ 表示内容に基づいて Claude との反復処理を続行する

これにより、ドラフト PR を作成したり GitHub に切り替えたりすることなく、複数ラウンドのフィードバックを通じて変更を改善できます。

ウェブとターミナル間でタスクを移動する

ターミナルからウェブ上で新しいタスクを開始するか、ウェブセッションをターミナルにプルしてローカルで続行できます。ウェブセッションはラップトップを閉じてでも保持され、Claude モバイルアプリを含む任意の場所から監視できます。

Note:

セッションハンドオフは一方です：ウェブセッションをターミナルにプルできますが、既存のターミナルセッションをウェブにプッシュすることはできません。 `--remote` フラグは現在のリポジトリの新しいウェブセッションを作成します。

ターミナルからウェブへ

`--remote` フラグを使用してコマンドラインからウェブセッションを開始します：

```
claude --remote "Fix the authentication bug in src/auth/login.ts"
```

これにより claude.ai 上に新しいウェブセッションが作成されます。タスクはクラウドで実行され、ローカルで作業を続行できます。 `/tasks` を使用して進捗を確認するか、claude.ai または Claude モバイルアプリでセッションを開いて直接対話します。そこから Claude を操舵し、フィードバックを提供するか、他のすべての会話と同じように質問に答えることができます。

リモートタスクのヒント

ローカルで計画し、リモートで実行する：複雑なタスクの場合、Claude をプランモードで開始してアプローチについて協力し、その後ウェブに作業を送信します：

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

プランモードでは、Claude はファイルの読み取りとコードベースの探索のみができます。計画に満足したら、自律実行のためにリモートセッションを開始します：

```
claude --remote "Execute the migration plan in docs/migration-plan.md"
```

このパターンにより、戦略を制御しながら Claude がクラウドで自律的に実行できます。

タスクを並列で実行する：各 `--remote` コマンドは独立して実行される独自のウェブセッションを作成します。複数のタスクを開始でき、すべて別々のセッションで同時に実行されます：

```
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"
```

`/tasks` ですべてのセッションを監視します。セッションが完了したら、ウェブインターフェースから PR を作成するか、[セッションをテレポート](#)してターミナルで作業を続行できます。

ウェブからターミナルへ

ウェブセッションをターミナルにプルするにはいくつかの方法があります：

- **`/teleport` を使用する：**Claude Code 内から `/teleport`（または `/tp`）を実行して、ウェブセッションのインタラクティブピッカーを表示します。コミットされていない変更がある場合は、最初にそれらをスタッシュするよう求められます。
- **`--teleport` を使用する：**コマンドラインから `claude --teleport` を実行してインタラクティブセッションピッカーを表示するか、`claude --teleport <session-id>` を実行して特定のセッションを直接再開します。
- **`/tasks` から：**`/tasks` を実行してバックグラウンドセッションを表示し、`t` を押してセッションにテレポートします
- **ウェブインターフェースから：**「Open in CLI」をクリックしてターミナルに貼り付けられるコマンドをコピーします

セッションをテレポートすると、Claude は正しいリポジトリにいることを確認し、リモートセッションからブランチをフェッチしてチェックアウトし、完全な会話履歴をターミナルに読み込みます。

テレポートの要件

テレポートはセッションを再開する前にこれらの要件をチェックします。要件が満たされていない場合は、エラーが表示されるか、問題を解決するよう求められます。

要件	詳細
クリーンな git 状態	作業ディレクトリにコミットされていない変更がないことが必要です。テレポートは必要に応じて変更をスタッシュするよう求めます。

要件	詳細
正しいリポジトリ	フォークではなく、同じリポジトリのチェックアウトから <code>--teleport</code> を実行する必要があります。
ブランチが利用可能	ウェブセッションからのブランチがリモートにプッシュされている必要があります。テレポートは自動的にフェッチしてチェックアウトします。
同じアカウント	ウェブセッションで使用された同じ Claude.ai アカウントに認証される必要があります。

セッションの共有

セッションを共有するには、以下のアカウントタイプに従ってその可視性を切り替えます。その後、セッションリンクをそのまま共有します。共有セッションを開いた受信者は、読み込み時にセッションの最新状態を表示しますが、受信者のページはリアルタイムで更新されません。

Enterprise または Teams アカウントから共有する

Enterprise および Teams アカウントの場合、2つの可視性オプションは**プライベート**と**チーム**です。チーム可視性により、セッションは Claude.ai 組織の他のメンバーに表示されます。リポジトリアクセス検証はデフォルトで有効になっており、受信者のアカウントに接続された GitHub アカウントに基づいています。アカウントの表示名はアクセス権を持つすべての受信者に表示されます。[Claude in Slack](#) セッションは自動的にチーム可視性で共有されます。

Max または Pro アカウントから共有する

Max および Pro アカウントの場合、2つの可視性オプションは**プライベート**と**パブリック**です。パブリック可視性により、セッションは `claude.ai` にログインしているすべてのユーザーに表示されます。

共有する前にセッションで機密コンテンツを確認してください。セッションにはプライベート GitHub リポジトリのコードと認証情報が含まれる可能性があります。リポジトリアクセス検証はデフォルトで有効になっていません。

Settings > Claude Code > Sharing settings に移動して、リポジトリアクセス検証を有効にするか、共有セッションから名前を保留します。

セッションの管理

セッションのアーカイブ

セッションをアーカイブしてセッションリストを整理できます。アーカイブされたセッションはデフォルトのセッションリストから非表示になりますが、アーカイブされたセッションをフィルタリングして表示できます。

セッションをアーカイブするには、サイドバーのセッションにマウスを合わせてアーカイブアイコンをクリックします。

セッションの削除

セッションを削除すると、セッションとそのデータが永続的に削除されます。このアクションは取り消せません。セッションは2つの方法で削除できます：

- ・ **サイドバーから**：アーカイブされたセッションをフィルタリングし、削除するセッションにマウスを合わせて削除アイコンをクリックします
- ・ **セッションメニューから**：セッションを開き、セッションタイトルの横のドロップダウンをクリックして、**削除**を選択します

セッションが削除される前に確認するよう求められます。

クラウド環境

デフォルトイメージ

一般的なツールチェーンと言語エコシステムがプリインストールされた汎用イメージを構築および保守します。このイメージには以下が含まれます：

- ・ 一般的なプログラミング言語とランタイム
- ・ 一般的なビルドツールとパッケージマネージャー
- ・ テストフレームワークとリンター

利用可能なツールの確認

環境にプリインストールされているものを確認するには、Claude Code に以下を実行するよう依頼します：

```
check-tools
```

このコマンドは以下を表示します：

- ・ プログラミング言語とそのバージョン

- 利用可能なパッケージマネージャー
- インストールされた開発ツール

言語固有のセットアップ

汎用イメージには以下の事前設定環境が含まれます：

- **Python**：pip、poetry、および一般的な科学ライブラリを備えた Python 3.x
- **Node.js**：npm、yarn、pnpm、および bun を備えた最新 LTS バージョン
- **Ruby**：バージョン 3.1.6、3.2.6、3.3.6（デフォルト：3.3.6）、gem、bundler、およびバージョン管理用の rbenv
- **PHP**：バージョン 8.4.14
- **Java**：Maven および Gradle を備えた OpenJDK
- **Go**：モジュールサポート付きの最新安定版
- **Rust**：cargo を備えた Rust ツールチェーン
- **C++**：GCC および Clang コンパイラ

データベース

汎用イメージには以下のデータベースが含まれます：

- **PostgreSQL**：バージョン 16
- **Redis**：バージョン 7.0

環境設定

ウェブ上の Claude Code でセッションを開始すると、内部で以下が発生します：

1. **環境準備**：リポジトリをクローンし、設定されている[セットアップスクリプト](#)を実行します。リポジトリは GitHub リポジトリのデフォルトブランチでクローンされます。特定のブランチをチェックアウトする場合は、プロンプトで指定できます。
2. **ネットワーク設定**：エージェントのインターネットアクセスを設定します。インターネットアクセスはデフォルトで制限されていますが、ニーズに基づいて環境をインターネットなしまたは完全なインターネットアクセスを持つように設定できます。
3. **Claude Code 実行**：Claude Code を実行してタスクを完了し、コードを作成し、テストを実行し、その作業を確認します。ウェブインターフェースを通じてセッション全体で Claude をガイドおよび操舵できます。Claude は `CLAUDE.md` で定義したコンテキストを尊重します。
4. **結果**：Claude が作業を完了すると、ブランチをリモートにプッシュします。ブランチの PR を作成できるようになります。

Note:

Claude は環境で利用可能なターミナルと CLI ツールを完全に通じて動作します。汎用イメージのプリインストールツールと、フック依存管理を通じてインストールする追加ツールを使用します。

新しい環境を追加するには：現在の環境を選択して環境セクターを開き、「Add environment」を選択します。これにより、環境名、ネットワークアクセスレベル、環境変数、および[セットアップスクリプト](#)を指定できるダイアログが開きます。

既存の環境を更新するには：現在の環境を選択し、環境名の右側にある設定ボタンを選択します。これにより、環境名、ネットワークアクセス、環境変数、およびセットアップスクリプトを更新できるダイアログが開きます。

ターミナルからデフォルト環境を選択するには：複数の環境が設定されている場合、`/remote-env` を実行して、`--remote` でターミナルからウェブセッションを開始するとき使用するものを選択します。単一の環境では、このコマンドは現在の設定を表示します。

Note:

環境変数は [.env 形式](#) でキーと値のペアとして指定する必要があります。例：

```
API_KEY=your_api_key
DEBUG=true
```

セットアップスクリプト

セットアップスクリプトは、新しいクラウドセッションが開始されるときに実行される Bash スクリプトで、Claude Code が起動する前に実行されます。セットアップスクリプトを使用して、依存関係をインストールし、ツールを設定するか、クラウド環境が必要とするものを準備します。これは[デフォルトイメージ](#)にはありません。

スクリプトは Ubuntu 24.04 でルートとして実行されるため、`apt install` とほとんどの言語パッケージマネージャーが機能します。

Tip:

スクリプトに追加する前に何がすでにインストールされているかを確認するには、Claude にクラウドセッションで `check-tools` を実行するよう依頼します。

セットアップスクリプトを追加するには、環境設定ダイアログを開き、**Setup script** フィールドにスクリプトを入力します。

この例は、デフォルトイメージにない **gh** CLI をインストールします：

```
#!/bin/bash
apt update && apt install -y gh
```

セットアップスクリプトは新しいセッションを作成するときのみ実行されます。既存のセッションを再開するときはスキップされます。

スクリプトがゼロ以外で終了する場合、セッションは開始に失敗します。不安定なインストールでセッションをブロックするのを避けるために、重要でないコマンドに `|| true` を追加します。

Note:

パッケージをインストールするセットアップスクリプトはレジストリに到達するためにネットワークアクセスが必要です。デフォルトのネットワークアクセスは `npm`、`PyPI`、`RubyGems`、`crates.io` を含む [一般的なパッケージレジストリ](#) への接続を許可します。環境がネットワークアクセスを無効にしている場合、スクリプトはパッケージのインストールに失敗します。

セットアップスクリプト対 SessionStart フック

クラウドが必要とするがラップトップがすでに持っているもの（言語ランタイムや CLI ツールなど）をインストールするにはセットアップスクリプトを使用します。クラウドとローカルの両方で実行する必要があるプロジェクトセットアップ（`npm install` など）には [SessionStart フック](#) を使用します。

どちらもセッションの開始時に実行されますが、異なる場所に属しています：

	セットアップスクリプト	SessionStart フック
添付先	クラウド環境	リポジトリ
設定場所	クラウド環境 UI	リポジトリの <code>.claude/settings.json</code>
実行	Claude Code が起動する前、新しいセッションのみ	Claude Code が起動した後、再開を含むすべてのセッション

	セットアップスクリプト	SessionStart フック
スコープ	クラウド環境のみ	ローカルとクラウド両方

SessionStart フックはローカルのユーザーレベル `~/.claude/settings.json` でも定義できますが、ユーザーレベルの設定はクラウドセッションに引き継がれません。クラウドでは、リポジトリにコミットされたフックのみが実行されます。

依存関係管理

カスタム環境イメージとスナップショットはまだサポートされていません。セッション開始時にパッケージをインストールするには[セットアップスクリプト](#)を使用するか、ローカル環境でも実行する必要がある依存関係インストールには[SessionStart フック](#)を使用します。SessionStart フックには[既知の制限](#)があります。

セットアップスクリプトで自動依存関係インストールを設定するには、環境設定を開いてスクリプトを追加します：

```
#!/bin/bash
npm install
pip install -r requirements.txt
```

または、リポジトリの `.claude/settings.json` ファイルで SessionStart フックを使用して、ローカル環境でも実行する必要がある依存関係インストールを実行できます：

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/scripts/install_pkgs.sh"
          }
        ]
      }
    ]
  }
}
```

`scripts/install_pkgs.sh` に対応するスクリプトを作成します：

```
#!/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
```

実行可能にします：`chmod +x scripts/install_pkgs.sh`

環境変数を永続化する

SessionStart フックは `CLAUDE_ENV_FILE` 環境変数で指定されたファイルに書き込むことで、後続の Bash コマンドの環境変数を永続化できます。詳細については、フック参照の [SessionStart フック](#) を参照してください。

依存関係管理の制限

- **すべてのセッションに対してフックが発火する**：SessionStart フックはローカルとリモート環境の両方で実行されます。リモートセッションのみにフックをスコープするフック設定はありません。ローカル実行をスキップするには、上記のように `CLAUDE_CODE_REMOTE` 環境変数をスクリプトで確認します。
- **ネットワークアクセスが必要**：インストールコマンドはパッケージレジストリに到達するためにネットワークアクセスが必要です。環境が「No internet」アクセスで設定されている場合、これらのフックは失敗します。「Limited」（デフォルト）または「Full」ネットワークアクセスを使用します。[デフォルト許可リスト](#)には npm、PyPI、RubyGems、crates.io などの一般的なレジストリが含まれます。
- **プロキシ互換性**：リモート環境のすべてのアウトバウンドトラフィックは[セキュリティプロキシ](#)を通じて渡されます。一部のパッケージマネージャーはこのプロキシで正しく機能しません。Bun は既知の例です。
- **すべてのセッション開始時に実行**：フックはセッションが開始または再開されるたびに実行され、スタートアップレイテンシーが追加されます。依存関係が既に存在するかどうかを確認してから再インストールすることで、インストールスクリプトを高速に保ちます。

ネットワークアクセスとセキュリティ

ネットワークポリシー

GitHub プロキシ

セキュリティのため、すべての GitHub 操作は、すべての git インタラクションを透過的に処理する専用プロキシサービスを通じて行われます。サンドボックス内では、git クライアントはカスタムビルトのスコープ付き認証情報を使用して認証します。このプロキシは：

- GitHub 認証をセキュアに管理します。git クライアントはサンドボックス内のスコープ付き認証情報を使用し、プロキシはそれを検証して実際の GitHub 認証トークンに変換します
- 安全性のため git push 操作を現在のワーキングブランチに制限します
- セキュリティ境界を維持しながらシームレスなクローン、フェッチ、PR 操作を有効にします

セキュリティプロキシ

環境はセキュリティと不正使用防止のため HTTP/HTTPS ネットワークプロキシの背後で実行されます。すべてのアウトバウンドインターネットトラフィックはこのプロキシを通じて渡され、以下を提供します：

- 悪意のあるリクエストに対する保護
- レート制限と不正使用防止
- 強化されたセキュリティのためのコンテンツフィルタリング

アクセスレベル

デフォルトでは、ネットワークアクセスは[許可リストドメイン](#)に制限されています。

カスタムネットワークアクセスを設定でき、ネットワークアクセスを無効にすることも含まれます。

デフォルト許可ドメイン

「Limited」ネットワークアクセスを使用する場合、以下のドメインはデフォルトで許可されます：

Anthropic サービス

- api.anthropic.com
- statsig.anthropic.com
- platform.claude.com
- code.claude.com
- claude.ai

バージョン管理

- 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

コンテナレジストリ

- 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

クラウドプラットフォーム

- 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

パッケージマネージャー - JavaScript/Node

- registry.npmjs.org
- www.npmjs.com
- www.npmjs.org
- npmjs.com
- npmjs.org
- yarnpkg.com
- registry.yarnpkg.com

パッケージマネージャー - Python

- pypi.org
- www.pypi.org
- files.pythonhosted.org
- pythonhosted.org
- test.pypi.org
- pypi.python.org

- [pypa.io](https://pypi.org/)
- [www.pypi.io](https://www.pypi.org/)

パッケージマネージャー - 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

パッケージマネージャー - Rust

- crates.io
- www.crates.io
- index.crates.io
- static.crates.io
- rustup.rs
- static.rust-lang.org
- www.rust-lang.org

パッケージマネージャー - Go

- proxy.golang.org
- sum.golang.org
- index.golang.org
- golang.org
- www.golang.org
- goproxy.io
- pkg.go.dev

パッケージマネージャー - 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

パッケージマネージャー - その他の言語

- 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 ディストリビューション

- archive.ubuntu.com
- security.ubuntu.com
- ubuntu.com
- www.ubuntu.com
- *.ubuntu.com
- ppa.launchpad.net
- launchpad.net
- www.launchpad.net

開発ツール&プラットフォーム

- 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

クラウドサービス & 監視

- statsig.com
- www.statsig.com
- api.statsig.com
- sentry.io
- *.sentry.io
- http-intake.logs.datadoghq.com
- *.datadoghq.com
- *.datadoghq.eu

コンテンツ配信 & ミラー

- sourceforge.net
- *.sourceforge.net
- packagecloud.io
- *.packagecloud.io

スキーマ & 設定

- json-schema.org
- www.json-schema.org
- json.schemastore.org
- www.schemastore.org

Model Context Protocol

- *.modelcontextprotocol.io

Note:

- * でマークされたドメインはワイルドカードサブドメインマッチングを示します。例えば、`*.gcr.io` は `gcr.io` のすべてのサブドメインへのアクセスを許可します。

カスタマイズされたネットワークアクセスのセキュリティベストプラクティス

1. **最小権限の原則**：必要な最小限のネットワークアクセスのみを有効にします
2. **定期的に監査する**：許可されたドメインを定期的に確認します
3. **HTTPS を使用する**：常に HTTP エンドポイントより HTTPS エンドポイントを優先します

セキュリティと分離

ウェブ上の Claude Code は強力なセキュリティ保証を提供します：

- **分離された仮想マシン**：各セッションは分離された Anthropic 管理の VM で実行されます
- **ネットワークアクセス制御**：ネットワークアクセスはデフォルトで制限され、無効にできます

Note:

ネットワークアクセスを無効にして実行する場合、Claude Code は Anthropic API と通信することが許可されており、これにより分離された Claude Code VM からデータが出ることを許可する可能性があります。

- **認証情報保護**：機密認証情報（git 認証情報や署名キーなど）はサンドボックス内の Claude Code と一緒にありません。認証はスコープ付き認証情報を使用するセキュアプロキシを通じて処理されます
- **セキュアな分析**：コードは PR を作成する前に分離された VM 内で分析および変更されます

価格とレート制限

ウェブ上の Claude Code はアカウント内のすべての他の Claude および Claude Code 使用とレート制限を共有します。複数のタスクを並列で実行すると、レート制限をより多く消費します。

制限事項

- **リポジトリ認証**：ウェブからローカルにセッションを移動できるのは、同じアカウントに認証されている場合のみです
- **プラットフォーム制限**：ウェブ上の Claude Code は GitHub でホストされているコードでのみ機能します。GitLab およびその他の非 GitHub リポジトリはクラウドセッションで使用できません

ベストプラクティス

1. **環境セットアップを自動化する**：[セットアップスクリプト](#)を使用して Claude Code が起動する前に依存関係をインストールし、ツールを設定します。より高度なシナリオの場合は、[SessionStart フック](#)を設定します。
2. **要件を文書化する**：`CLAUDE.md` ファイルで依存関係とコマンドを明確に指定します。`AGENTS.md` ファイルがある場合は、`@AGENTS.md` を使用して `CLAUDE.md` でソースして、単一の情報源を維持できます。

関連リソース

- [フック設定](#)
- [設定リファレンス](#)
- [セキュリティ](#)
- [データ使用](#)

Part 9: Security & Privacy

セキュリティ

Claude Code のセキュリティ対策とセキュアな使用方法のベストプラクティスについて学びます。

セキュリティへのアプローチ方法

セキュリティの基盤

コードのセキュリティは最優先事項です。Claude Code はセキュリティを中核に据えて構築されており、Anthropic の包括的なセキュリティプログラムに従って開発されています。詳細情報とリソース（SOC 2 Type 2 レポート、ISO 27001 証明書など）については、[Anthropic Trust Center](#) をご覧ください。

パーミッションベースのアーキテクチャ

Claude Code はデフォルトで厳密な読み取り専用パーミッションを使用します。追加のアクション（ファイルの編集、テストの実行、コマンドの実行）が必要な場合、Claude Code は明示的なパーミッションをリクエストします。ユーザーは、アクションを 1 回だけ承認するか、自動的に許可するかを制御できます。

Claude Code は透明性とセキュリティを備えるように設計されています。例えば、bash コマンドを実行する前に承認が必要であり、直接制御できます。このアプローチにより、ユーザーと組織はパーミッションを直接設定できます。

詳細なパーミッション設定については、[Permissions](#) を参照してください。

組み込み保護機能

agentic システムのリスクを軽減するために：

- **サンドボックス化された bash ツール:** [Sandbox](#) bash コマンドをファイルシステムとネットワークの分離で実行し、パーミッションプロンプトを減らしながらセキュリティを維持します。 `/sandbox` で有効にして、Claude Code が自律的に動作できる境界を定義します
- **書き込みアクセス制限:** Claude Code は開始されたフォルダとそのサブフォルダにのみ書き込みでき、明示的なパーミッションなしに親ディレクトリのファイルを変更することはできません。Claude Code は作業ディレクトリ外のファイルを読み取ること

ができます（システムライブラリと依存関係へのアクセスに便利です）が、書き込み操作はプロジェクトスコープに厳密に限定され、明確なセキュリティ境界を作成します

- **プロンプト疲労の軽減:** ユーザーごと、コードベースごと、または組織ごとに頻繁に使用される安全なコマンドのホワイトリスト化をサポート
- **Accept Edits モード:** 複数の編集をバッチで受け入れながら、副作用のあるコマンドのパーミッションプロンプトを維持

ユーザーの責任

Claude Code は、ユーザーが付与したパーミッションのみを持ちます。承認前に、提案されたコードとコマンドのセキュリティを確認する責任があります。

プロンプトインジェクションから保護する

プロンプトインジェクションは、攻撃者が悪意のあるテキストを挿入することで AI アシスタントの指示をオーバーライドまたは操作しようとする手法です。Claude Code にはこれらの攻撃に対する複数のセーフガードが含まれています：

コア保護機能

- **パーミッションシステム:** 機密操作には明示的な承認が必要です
- **コンテキスト認識分析:** 完全なリクエストを分析して潜在的に有害な指示を検出します
- **入力サニタイゼーション:** ユーザー入力を処理することでコマンドインジェクションを防止します
- **コマンドブロックリスト:** `curl` や `wget` のようにウェブから任意のコンテンツを取得するリスクのあるコマンドをデフォルトでブロックします。明示的に許可する場合は、[パーミッションパターンの制限](#) に注意してください

プライバシーセーフガード

データを保護するために、複数のセーフガードを実装しています：

- 機密情報の保持期間の制限（詳細については [Privacy Center](#) を参照してください）
- ユーザーセッションデータへのアクセス制限
- データトレーニング設定に対するユーザーコントロール。コンシューマーユーザーは [プライバシー設定](#) をいつでも変更できます。

詳細については、[Commercial Terms of Service](#)（Team、Enterprise、API ユーザー向け）または [Consumer Terms](#)（Free、Pro、Max ユーザー向け）および [Privacy Policy](#) をご確認ください。

追加のセーフガード

- **ネットワークリクエスト承認:** ネットワークリクエストを行うツールはデフォルトでユーザー承認が必要です
- **分離されたコンテキストウィンドウ:** Web fetch は潜在的に悪意のあるプロンプトの注入を避けるために別のコンテキストウィンドウを使用します
- **信頼検証:** 初回のコードベース実行と新しい MCP サーバーには信頼検証が必要です
 - 注: 信頼検証は `-p` フラグで非対話的に実行する場合は無効になります
- **コマンドインジェクション検出:** 疑わしい bash コマンドは、以前にホワイトリストに登録されていても手動承認が必要です
- **ファイルクローズドマッチング:** マッチしないコマンドはデフォルトで手動承認が必要です
- **自然言語説明:** 複雑な bash コマンドにはユーザーの理解のための説明が含まれます
- **セキュアな認証情報ストレージ:** API キーとトークンは暗号化されます。 [Credential Management](#) を参照してください

Warning:

Windows WebDAV セキュリティリスク: Windows で Claude Code を実行する場合、WebDAV を有効にしたり、Claude Code に `\\*` などの WebDAV サブディレクトリを含む可能性のあるパスへのアクセスを許可することはお勧めしません。 [WebDAV は Microsoft によって非推奨になっています](#) セキュリティリスクのため。WebDAV を有効にすると、Claude Code がリモートホストへのネットワークリクエストをトリガーし、パーミッションシステムをバイパスする可能性があります。

信頼できないコンテンツを使用する場合のベストプラクティス：

1. 承認前に提案されたコマンドを確認します
2. 信頼できないコンテンツを Claude に直接パイプすることを避けます
3. 重要なファイルへの提案された変更を確認します
4. 仮想マシン（VM）を使用してスクリプトを実行し、ツール呼び出しを行います。特に外部 Web サービスと対話する場合
5. `/bug` で疑わしい動作を報告します

Warning:

これらの保護機能はリスクを大幅に軽減しますが、どのシステムもすべての攻撃に完全に免疫があるわけではありません。AI ツールを使用する場合は常に良好なセキュリティプラクティスを維持してください。

MCP セキュリティ

Claude Code ユーザーは Model Context Protocol (MCP) サーバーを設定できます。許可された MCP サーバーのリストは、エンジニアがソース管理にチェックインする Claude Code 設定の一部として、ソースコードで設定されます。

独自の MCP サーバーを作成するか、信頼できるプロバイダーからの MCP サーバーを使用することをお勧めします。Claude Code パーミッションを MCP サーバー用に設定できます。Anthropic は MCP サーバーを管理または監査しません。

IDE セキュリティ

IDE で Claude Code を実行する場合の詳細については、[VS Code security and privacy](#) を参照してください。

クラウド実行セキュリティ

[Claude Code on the web](#) を使用する場合、追加のセキュリティ制御が実施されます：

- **分離された仮想マシン:** 各クラウドセッションは分離された Anthropic 管理 VM で実行されます
- **ネットワークアクセス制御:** ネットワークアクセスはデフォルトで制限され、無効にするか特定のドメインのみを許可するように設定できます
- **認証情報保護:** 認証はサンドボックス内でスコープされた認証情報を使用するセキュアプロキシを通じて処理され、その後実際の GitHub 認証トークンに変換されます
- **ブランチ制限:** Git push 操作は現在のワーキングブランチに制限されます
- **監査ログ:** クラウド環境内のすべての操作はコンプライアンスと監査目的でログされます
- **自動クリーンアップ:** クラウド環境はセッション完了後に自動的に終了されます

クラウド実行の詳細については、[Claude Code on the web](#) を参照してください。

[Remote Control](#) セッションは異なる方法で動作します：Web インターフェースはローカルマシンで実行されている Claude Code プロセスに接続します。すべてのコード実行とファイルアクセスはローカルに留まり、ローカル Claude Code セッション中に流れるのと同じデータが TLS 経由で Anthropic API を通じて流れます。クラウド VM またはサンドボックスは関与しません。接続は複数の短命で狭くスコープされた認証情報を使用し、各認証情報は特定の目的に限定され、独立して有効期限が切れ、単一の侵害された認証情報のブラストレイアウトを制限します。

セキュリティベストプラクティス

機密コードの使用

- 承認前にすべての提案された変更を確認します
- 機密リポジトリにはプロジェクト固有のパーミッション設定を使用します
- 追加の分離のために [devcontainers](#) の使用を検討します
- `/permissions` で定期的にパーミッション設定を監査します

チームセキュリティ

- [managed settings](#) を使用して組織標準を実施します
- 承認されたパーミッション設定をバージョン管理を通じて共有します
- チームメンバーにセキュリティベストプラクティスについてトレーニングを行います
- [OpenTelemetry metrics](#) を通じて Claude Code の使用を監視します
- [ConfigChange hooks](#) でセッション中の設定変更を監査またはブロックします

セキュリティ問題の報告

Claude Code でセキュリティ脆弱性を発見した場合：

1. 公開で開示しないでください
2. [HackerOne program](#) を通じて報告してください
3. 詳細な再現手順を含めてください
4. 公開開示前に問題に対処する時間を与えてください

関連リソース

- [Sandboxing](#) - bash コマンドのファイルシステムとネットワーク分離
- [Permissions](#) - パーミッションとアクセス制御を設定します
- [Monitoring usage](#) - Claude Code アクティビティを追跡および監査します
- [Development containers](#) - セキュアで分離された環境
- [Anthropic Trust Center](#) - セキュリティ認証とコンプライアンス

サンドボックス

Claude Code のサンドボックス化された bash ツールがファイルシステムとネットワークの分離を提供し、より安全で自律的なエージェント実行を実現する方法について学びます。

概要

Claude Code はネイティブサンドボックス機能を備えており、エージェント実行のためのより安全な環境を提供しながら、継続的な許可プロンプトの必要性を軽減します。各 bash コマンドの実行許可を求める代わりに、サンドボックス化により事前に定義された境界が作成され、Claude Code はリスクを軽減しながらより自由に動作できます。

サンドボックス化された bash ツールは OS レベルのプリミティブを使用して、ファイルシステムとネットワークの両方の分離を実施します。

サンドボックス化が重要な理由

従来の許可ベースのセキュリティでは、bash コマンドごとに継続的なユーザー承認が必要です。これは制御を提供しますが、以下の問題につながる可能性があります。

- **承認疲れ**：「承認」を繰り返しクリックすると、ユーザーが承認内容に注意を払わなくなる可能性があります
- **生産性の低下**：継続的な中断により開発ワークフローが遅くなります
- **自律性の制限**：Claude Code は承認を待つ際に効率的に動作できません

サンドボックス化はこれらの課題に以下の方法で対処します。

1. **明確な境界を定義**：Claude Code がアクセスできるディレクトリとネットワークホストを正確に指定します
2. **許可プロンプトを削減**：サンドボックス内の安全なコマンドは承認を必要としません
3. **セキュリティを維持**：サンドボックス外のリソースへのアクセス試行は即座に通知をトリガーします
4. **自律性を有効化**：Claude Code は定義された制限内でより独立して実行できます

Warning:

効果的なサンドボックス化には、ファイルシステムとネットワークの両方の分離が必要です。ネットワーク分離がない場合、侵害されたエージェントは SSH キーなどの機密ファイルを流出させる可能性があります。ファイルシステム分離がない場合、侵害されたエージェントはシステムリソースにバックドアを仕掛けてネットワークアクセスを取得する可能性があります。サンドボックス化を設定する際は、設定がこれらのシステムのバイパスを作成しないことを確認することが重要です。

仕組み

ファイルシステム分離

サンドボックス化された bash ツールはファイルシステムアクセスを特定のディレクトリに制限します。

- **デフォルトの書き込み動作**：現在の作業ディレクトリとそのサブディレクトリへの読み取りおよび書き込みアクセス
- **デフォルトの読み取り動作**：特定の拒否ディレクトリを除く、コンピュータ全体への読み取りアクセス
- **ブロックされたアクセス**：明示的な許可なしに現在の作業ディレクトリ外のファイルを変更できません
- **設定可能**：設定を通じてカスタム許可パスと拒否パスを定義します

`sandbox.filesystem.allowWrite` を設定で使用して、追加のパスへの書き込みアクセスを付与できます。これらの制限は OS レベル（macOS の Seatbelt、Linux の bubblewrap）で実施されるため、Claude のファイルツールだけでなく、`kubectl`、`terraform`、`npm` などのツールを含むすべてのサブプロセスコマンドに適用されます。

ネットワーク分離

ネットワークアクセスはサンドボックス外で実行されるプロキシサーバーを通じて制御されます。

- **ドメイン制限**：承認されたドメインのみにアクセスできます
- **ユーザー確認**：新しいドメインリクエストは許可プロンプトをトリガーします（`allowManagedDomainsOnly` が有効な場合を除き、許可されていないドメインを自動的にブロックします）
- **カスタムプロキシサポート**：高度なユーザーは発信トラフィックにカスタムルールを実装できます
- **包括的なカバレッジ**：制限はすべてのスクリプト、プログラム、およびコマンドによって生成されるサブプロセスに適用されます

OS レベルの実施

サンドボックス化された bash ツールは OS セキュリティプリミティブを活用します。

- **macOS** : Seatbelt をサンドボックス実施に使用します
- **Linux** : 分離に [bubblewrap](#) を使用します
- **WSL2** : Linux と同じく bubblewrap を使用します

WSL1 は bubblewrap が WSL2 でのみ利用可能なカーネル機能が必要とするため、サポートされていません。

これらの OS レベルの制限により、Claude Code のコマンドによって生成されたすべての子プロセスが同じセキュリティ境界を継承することが保証されます。

開始方法

前提条件

macOS では、サンドボックス化は組み込みの Seatbelt フレームワークを使用してすぐに動作します。

Linux と WSL2 では、まず必要なパッケージをインストールします。

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

サンドボックス化を有効化

`/sandbox` コマンドを実行してサンドボックス化を有効化できます。

```
/sandbox
```

これはサンドボックスモードを選択できるメニューを開きます。必要な依存関係 (Linux の `bubblewrap` や `socat` など) が不足している場合、メニューはプラットフォームのインストール手順を表示します。

サンドボックスモード

Claude Code は 2 つのサンドボックスモードを提供します。

自動許可モード：Bash コマンドはサンドボックス内で実行を試みられ、許可なしに自動的に許可されます。サンドボックス化できないコマンド（許可されていないホストへのネットワークアクセスが必要なコマンドなど）は通常の許可フローにフォールバックします。設定した明示的な許可/拒否ルールは常に尊重されます。

通常の許可モード：すべての bash コマンドは、サンドボックス化されている場合でも標準的な許可フローを通じます。これはより多くの制御を提供しますが、より多くの承認が必要です。

両方のモードで、サンドボックスは同じファイルシステムとネットワーク制限を実施します。違いは、サンドボックス化されたコマンドが自動承認されるか明示的な許可が必要かどうかだけです。

Info:

自動許可モードは許可モード設定とは独立して動作します。「編集を受け入れる」モードでない場合でも、自動許可が有効な場合、サンドボックス化された bash コマンドは自動的に実行されます。これは、ファイル編集ツールが通常は承認を必要とする場合でも、サンドボックス境界内のファイルを変更する bash コマンドはプロンプトなしに実行されることを意味します。

サンドボックス化を設定

`settings.json` ファイルを通じてサンドボックス動作をカスタマイズします。完全な設定リファレンスについては [Settings](#) を参照してください。

特定のパスへのサブプロセス書き込みアクセスの付与

デフォルトでは、サンドボックス化されたコマンドは現在の作業ディレクトリにのみ書き込みできます。`kubectl`、`terraform`、`npm` などのサブプロセスコマンドがプロジェクトディレクトリ外に書き込む必要がある場合、`sandbox.filesystem.allowWrite` を使用して特定のパスへのアクセスを付与します。

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "allowWrite": ["~/.kube", "//tmp/build"]
    }
  }
}
```

これらのパスは OS レベルで実施されるため、サンドボックス内で実行されるすべてのコマンド（その子プロセスを含む）がそれらを尊重します。これは、`excludedCommands` でツールをサンドボックスから除外するのではなく、ツールが特定の場所への書き込みアクセスを必要とする場合の推奨アプローチです。

`allowWrite`（または `denyWrite` / `denyRead`）が複数の **設定スコープ** で定義されている場合、配列は**マージ**されます。つまり、すべてのスコープからのパスが結合され、置き換えられません。たとえば、管理設定が `//opt/company-tools` への書き込みを許可し、ユーザーが個人設定で `~/.kube` を追加する場合、両方のパスが最終的なサンドボックス設定に含まれます。これは、ユーザーとプロジェクトが、より高い優先度のスコープで設定されたパスを複製または上書きすることなく、リストを拡張できることを意味します。

パスプレフィックスはパスの解決方法を制御します。

プレフィックス	意味	例
<code>//</code>	ファイルシステムルートからの絶対パス	<code>//tmp/build</code> は <code>/tmp/build</code> になります
<code>~/</code>	ホームディレクトリからの相対パス	<code>~/.kube</code> は <code>\$HOME/.kube</code> になります
<code>/</code>	設定ファイルのディレクトリからの相対パス	<code>/build</code> は <code>\$SETTINGS_DIR/build</code> になります
<code>./</code> またはプレフィックスなし	相対パス（サンドボックスランタイムで解決）	<code>./output</code>

`sandbox.filesystem.denyWrite` と `sandbox.filesystem.denyRead` を使用して書き込みまたは読み取りアクセスを拒否することもできます。これらは `Edit(...)` と `Read(...)` 許可ルールからのパスとマージされます。

Tip:

すべてのコマンドがサンドボックス化と互換性があるわけではありません。サンドボックスを最大限に活用するのに役立つ可能性のあるいくつかのメモ：

- 多くの CLI ツールは特定のホストへのアクセスを必要とします。これらのツールを使用すると、特定のホストへのアクセス許可をリクエストします。許可を付与すると、これらのホストに今後アクセスでき、サンドボックス内で安全に実行できるようになります。
- `watchman` はサンドボックス内での実行と互換性がありません。`jest` を実行している場合は、`jest --no-watchman` の使用を検討してください
- `docker` はサンドボックス内での実行と互換性がありません。`excludedCommands` で `docker` を指定して、サンドボックス外で実行するように強制することを検討してください。

Note:

Claude Code には、必要に応じてコマンドをサンドボックス外で実行できるようにする意図的なエスケープハッチメカニズムが含まれています。コマンドがサンドボックス制限（ネットワーク接続の問題や互換性のないツールなど）により失敗した場合、Claude は失敗を分析するよう促され、`dangerouslyDisableSandbox` パラメータでコマンドを再試行する可能性があります。このパラメータを使用するコマンドは、実行するユーザー許可を必要とする通常の Claude Code 許可フローを通じます。これにより、Claude Code は特定のツールまたはネットワーク操作がサンドボックス制約内で機能できないエッジケースを処理できます。

このエスケープハッチは、[サンドボックス設定](#) で `"allowUnsandboxedCommands": false` を設定することで無効化できます。無効化されると、`dangerouslyDisableSandbox` パラメータは完全に無視され、すべてのコマンドはサンドボックス化されるか、`excludedCommands` に明示的にリストされている必要があります。

セキュリティ上の利点

プロンプトインジェクションからの保護

攻撃者がプロンプトインジェクションを通じて Claude Code の動作を正常に操作した場合でも、サンドボックスはシステムのセキュリティを確保します。

ファイルシステム保護：

- `~/.bashrc` などの重要な設定ファイルを変更できません
- `/bin/` のシステムレベルファイルを変更できません
- [Claude 許可設定](#) で拒否されたファイルを読み取ることができません

ネットワーク保護：

- 攻撃者が制御するサーバーへのデータ流出はできません

- ・許可されていないドメインから悪意のあるスクリプトをダウンロードできません
- ・承認されていないサービスへの予期しない API 呼び出しを行うことができません
- ・明示的に許可されていないドメインに連絡することはできません

監視と制御：

- ・サンドボックス外のすべてのアクセス試行は OS レベルでブロックされます
- ・境界がテストされるときに即座に通知を受け取ります
- ・リクエストを拒否、1 回だけ許可、または設定を永続的に更新することを選択できます

攻撃面の削減

サンドボックス化は以下からの潜在的な損害を制限します。

- ・ **悪意のある依存関係**：有害なコードを含む NPM パッケージまたは他の依存関係
- ・ **侵害されたスクリプト**：セキュリティ脆弱性を持つビルドスクリプトまたはツール
- ・ **ソーシャルエンジニアリング**：ユーザーに危険なコマンドを実行させるための攻撃
- ・ **プロンプトインジェクション**：Claude に危険なコマンドを実行させるための攻撃

透過的な操作

Claude Code がサンドボックス外のネットワークリソースにアクセスしようとする場合：

1. 操作は OS レベルでブロックされます
2. 即座に通知を受け取ります
3. 以下を選択できます。
 - リクエストを拒否する
 - 1 回だけ許可する
 - サンドボックス設定を永続的に更新して許可する

セキュリティ上の制限

- ・ ネットワークサンドボックス化の制限：ネットワークフィルタリングシステムは、プロセスが接続を許可されるドメインを制限することで動作します。プロキシを通じて渡されるトラフィックを検査することではなく、ユーザーはポリシーで許可されたドメインが信頼できるドメインのみであることを確認する責任があります。

Warning:

ユーザーは、データ流出を許可する可能性のある `github.com` などの広いドメインを許可することに伴う潜在的なリスクに注意する必要があります。また、場合によっては [ドメインフロンティング](#) を通じてネットワークフィルタリングをバイパスすることが可能な場合があります。

- Unix ソケットを通じた権限昇格： `allowUnixSockets` 設定は、サンドボックスバイパスにつながる可能性のある強力なシステムサービスへのアクセスを不注意に付与する可能性があります。たとえば、 `/var/run/docker.sock` へのアクセスを許可するために使用される場合、docker ソケットを悪用してホストシステムへのアクセスを効果的に付与します。ユーザーはサンドボックスを通じて許可する Unix ソケットを慎重に検討することをお勧めします。
- ファイルシステム許可昇格：過度に広いファイルシステム書き込み許可は権限昇格攻撃を有効にする可能性があります。 `$PATH` の実行可能ファイルを含むディレクトリ、システム設定ディレクトリ、またはユーザーシェル設定ファイル（ `.bashrc` 、 `.zshrc` ）への書き込みを許可すると、他のユーザーまたはシステムプロセスがこれらのファイルにアクセスするときに異なるセキュリティコンテキストでコード実行につながる可能性があります。
- Linux サンドボックス強度：Linux 実装は強力なファイルシステムとネットワーク分離を提供しますが、特権のない名前空間なしで Docker 環境内で動作できるようにする `enableWeakerNestedSandbox` モードが含まれています。このオプションはセキュリティを大幅に弱め、追加の分離が別の方法で実施される場合にのみ使用する必要があります。

サンドボックス化が許可とどのように関連するか

サンドボックス化と [許可](#) は、一緒に動作する補完的なセキュリティレイヤーです。

- **許可** は Claude Code が使用できるツールを制御し、任意のツールが実行される前に評価されます。これらは Bash、Read、Edit、WebFetch、MCP などすべてのツールに適用されます。
- **サンドボックス化** は、Bash コマンドがファイルシステムとネットワークレベルでアクセスできるものを制限する OS レベルの実施を提供します。これは Bash コマンドとその子プロセスにのみ適用されます。

ファイルシステムとネットワーク制限は、サンドボックス設定と許可ルールの両方を通じて設定されます。

- `sandbox.filesystem.allowWrite` を使用して、作業ディレクトリ外のパスへのサブプロセス書き込みアクセスを付与します

- `sandbox.filesystem.denyWrite` と `sandbox.filesystem.denyRead` を使用して、特定のパスへのサブプロセスアクセスをブロックします
- `Read` と `Edit` 拒否ルールを使用して、特定のファイルまたはディレクトリへのアクセスをブロックします
- `WebFetch` 許可/拒否ルールを使用してドメインアクセスを制御します
- サンドボックス `allowedDomains` を使用して、Bash コマンドが到達できるドメインを制御します

`sandbox.filesystem` 設定と許可ルールからのパスは、最終的なサンドボックス設定にマージされます。

この [リポジトリ](#) には、一般的なデプロイメントシナリオ（サンドボックス固有の例を含む）のスターター設定が含まれています。これらを出発点として使用し、ニーズに合わせて調整してください。

高度な使用方法

カスタムプロキシ設定

高度なネットワークセキュリティを必要とする組織の場合、カスタムプロキシを実装して以下を行うことができます。

- HTTPS トラフィックを復号化して検査する
- カスタムフィルタリングルールを適用する
- すべてのネットワークリクエストをログに記録する
- 既存のセキュリティインフラストラクチャと統合する

```
{
  "sandbox": {
    "network": {
      "httpProxyPort": 8080,
      "socksProxyPort": 8081
    }
  }
}
```

既存のセキュリティツールとの統合

サンドボックス化された bash ツールは以下と連携します。

- 許可ルール: [許可設定](#) と組み合わせて多層防御を実現します

- ・ **開発コンテナ**：[devcontainers](#) と共に使用して追加の分離を実現します
- ・ **エンタープライズポリシー**：[管理設定](#) を通じてサンドボックス設定を実施します

ベストプラクティス

1. **制限的に開始**：最小限の許可で開始し、必要に応じて拡張します
2. **ログを監視**：サンドボックス違反の試みを確認して、Claude Code のニーズを理解します
3. **環境固有の設定を使用**：開発環境と本番環境で異なるサンドボックスルールを使用します
4. **許可と組み合わせる**：包括的なセキュリティのためにサンドボックス化を IAM ポリシーと共に使用します
5. **設定をテスト**：サンドボックス設定が正当なワークフローをブロックしないことを確認します

オープンソース

サンドボックスランタイムは、独自のエージェントプロジェクトで使用するためのオープンソース npm パッケージとして利用可能です。これにより、より広い AI エージェントコミュニティがより安全で安全な自律システムを構築できます。これは、サンドボックス化したい他のプログラムをサンドボックス化するためにも使用できます。たとえば、MCP サーバーをサンドボックス化するには、以下を実行できます。

```
npx @anthropic-ai/sandbox-runtime <command-to-sandbox>
```

実装の詳細とソースコードについては、[GitHub リポジトリ](#) を参照してください。

制限事項

- ・ **パフォーマンスオーバーヘッド**：最小限ですが、一部のファイルシステム操作はわずかに遅くなる可能性があります
- ・ **互換性**：特定のシステムアクセスパターンを必要とするツールの中には、設定調整が必要な場合や、サンドボックス外で実行する必要がある場合があります
- ・ **プラットフォームサポート**：macOS、Linux、WSL2 をサポートします。WSL1 はサポートされていません。ネイティブ Windows サポートは計画中です。

関連項目

- ・ [Security](#) - 包括的なセキュリティ機能とベストプラクティス

- [Permissions](#) - 許可設定とアクセス制御
- [Settings](#) - 完全な設定リファレンス
- [CLI reference](#) - コマンドラインオプション

Checkpointing

Claude の編集を自動的に追跡し、不要な変更から素早く復旧するために以前の状態に巻き戻します。

Claude Code は、作業中に Claude のファイル編集を自動的に追跡し、何か問題が発生した場合に変更を素早く取り消し、以前の状態に巻き戻すことができます。

Checkpointing の仕組み

Claude と作業する際、checkpointing は各編集の前にコードの状態を自動的にキャプチャします。このセーフティネットにより、野心的で大規模なタスクを実行する際に、いつでも以前のコード状態に戻ることができるという安心感を持って作業できます。

自動追跡

Claude Code は、ファイル編集ツールによって行われたすべての変更を追跡します：

- ユーザープロンプトごとに新しい checkpoint が作成されます
- Checkpoint はセッション間で保持されるため、再開された会話でアクセスできます
- セッションとともに 30 日後に自動的にクリーンアップされます（設定可能）

変更の巻き戻し

Esc キーを 2 回（**Esc** + **Esc**）押すか、**/rewind** コマンドを使用して rewind メニューを開きます。以下のいずれかを復元することを選択できます：

- **会話のみ**: ユーザーメッセージに巻き戻しながらコード変更を保持します
- **コードのみ**: 会話を保持しながらファイル変更を元に戻します
- **コードと会話の両方**: セッション内の以前の時点に両方を復元します

一般的なユースケース

Checkpoint は以下の場合に特に有用です：

- **代替案の検討**: 開始点を失わずに異なる実装アプローチを試します
- **ミスからの復旧**: バグを導入したり機能を破損させた変更を素早く取り消します
- **機能の反復**: 動作する状態に戻ることができることを知りながら、バリエーションを試験します

制限事項

Bash コマンドの変更は追跡されません

Checkpointing は bash コマンドによって変更されたファイルを追跡しません。例えば、Claude Code が以下を実行する場合：

```
rm file.txt
mv old.txt new.txt
cp source.txt dest.txt
```

これらのファイル変更は rewind を通じて取り消すことはできません。Claude のファイル編集ツールを通じて行われた直接的なファイル編集のみが追跡されます。

外部の変更は追跡されません

Checkpointing は、現在のセッション内で編集されたファイルのみを追跡します。Claude Code の外部で手動で行ったファイルの変更と、他の同時セッションからの編集は、通常はキャプチャされません。ただし、現在のセッションと同じファイルを変更する場合は除きます。

バージョン管理の代替ではありません

Checkpoint はセッションレベルの迅速な復旧のために設計されています。永続的なバージョン履歴とコラボレーションのために：

- ・バージョン管理（例：Git）を引き続き使用してコミット、ブランチ、長期履歴を管理します
- ・Checkpoint は適切なバージョン管理を補完しますが、置き換えるものではありません
- ・Checkpoint を「ローカルアンドゥ」、Git を「永続履歴」と考えてください

関連項目

- ・ [Interactive mode](#) - キーボードショートカットとセッション制御
- ・ [Built-in commands](#) - `/rewind` を使用した checkpoint へのアクセス
- ・ [CLI reference](#) - コマンドラインオプション

データ使用

Anthropic の Claude のデータ使用ポリシーについて学習します

データポリシー

データトレーニングポリシー

コンシューマーユーザー（Free、Pro、Max プラン）：将来の Claude モデルの改善のためにデータを使用することを許可するかどうかを選択できます。この設定がオンの場合、Free、Pro、Max アカウントからのデータを使用して新しいモデルをトレーニングします（これらのアカウントから Claude Code を使用する場合も含みます）。

商用ユーザー：（Team および Enterprise プラン、API、サードパーティプラットフォーム、Claude Gov）既存のポリシーを維持します。Anthropic は、商用条件の下で Claude Code に送信されたコードまたはプロンプトを使用して生成モデルをトレーニングしません。ただし、カスタマーがモデル改善のためにデータを提供することを選択した場合は除きます（例えば、[Developer Partner Program](#)）。

Development Partner Program

[Development Partner Program](#) などを通じて、トレーニング用の資料を提供する方法に明示的にオプトインした場合、提供された資料を使用してモデルをトレーニングする可能性があります。組織管理者は、組織の Development Partner Program に明示的にオプトインできます。このプログラムは Anthropic ファーストパーティ API でのみ利用可能であり、Bedrock または Vertex ユーザーは利用できないことに注意してください。

/bug コマンドを使用したフィードバック

/bug コマンドを使用して Claude Code に関するフィードバックを送信することを選択した場合、製品とサービスを改善するためにフィードバックを使用する可能性があります。**/bug** を通じて共有されたトランスクリプトは 5 年間保持されます。

セッション品質調査

Claude Code で「How is Claude doing this session?」プロンプトが表示されたときに、この調査に応答する場合（「Dismiss」を選択する場合を含む）、数値評価（1、2、3、または dismiss）のみが記録されます。この調査の一部として、会話トランスクリプト、入力、出力、またはその他のセッションデータは収集または保存されません。サムズ

アップ/ダウンフィードバックまたは [/bug](#) レポートとは異なり、このセッション品質調査は単純な製品満足度メトリックです。この調査への応答は、データトレーニング設定に影響を与えず、AI モデルをトレーニングするために使用することはできません。

これらの調査を無効にするには、`CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1` を設定します。サードパーティプロバイダー（Bedrock、Vertex、Foundry）を使用する場合、またはテレメトリが無効になっている場合、調査は自動的に無効になります。

データ保持

Anthropic は、アカウントタイプと設定に基づいて Claude Code データを保持します。

コンシューマーユーザー（Free、Pro、Max プラン）：

- モデル改善のためのデータ使用を許可するユーザー：モデル開発とセキュリティ改善をサポートするための 5 年間の保持期間
- モデル改善のためのデータ使用を許可しないユーザー：30 日間の保持期間
- プライバシー設定は、claude.ai/settings/data-privacy-controls でいつでも変更できます。

商用ユーザー（Team、Enterprise、API）：

- 標準：30 日間の保持期間
- [Zero data retention](#)：Claude for Enterprise の Claude Code で利用可能。ZDR は組織ごとに有効になります。新しい各組織は、アカウントチームによって個別に ZDR を有効にする必要があります
- ローカルキャッシング：Claude Code クライアントは、セッション再開を有効にするために、セッションをローカルに最大 30 日間保存できます（設定可能）

Web 上の個別の Claude Code セッションはいつでも削除できます。セッションを削除すると、セッションのイベントデータが永久に削除されます。セッションの削除方法については、[Managing sessions](#) を参照してください。

データ保持慣行の詳細については、[Privacy Center](#) を参照してください。

詳細については、[Commercial Terms of Service](#)（Team、Enterprise、API ユーザー向け）または [Consumer Terms](#)（Free、Pro、Max ユーザー向け）および [Privacy Policy](#) を確認してください。

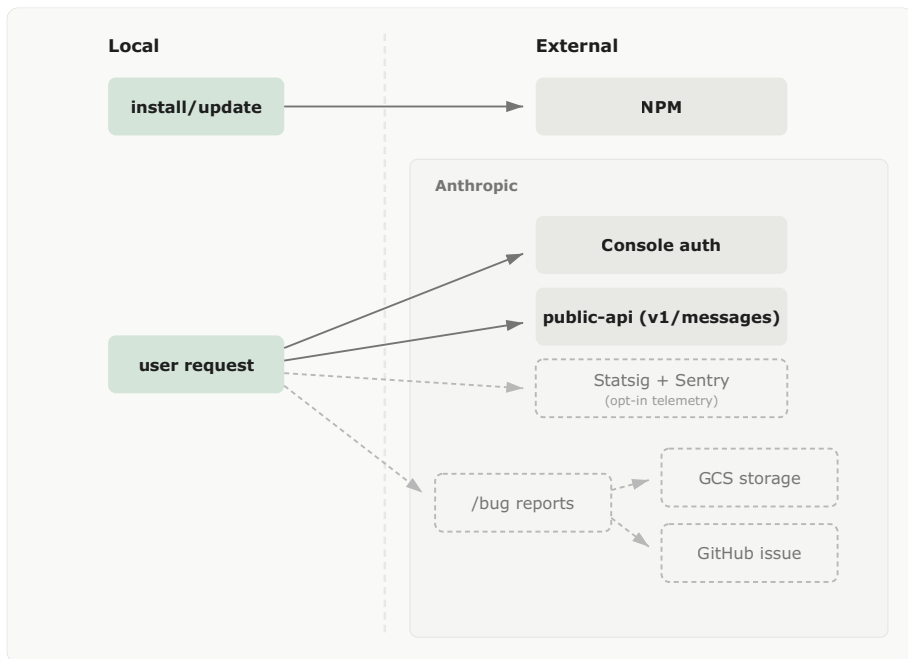
データアクセス

すべてのファーストパーティユーザーの場合、[ローカル Claude Code](#) および [リモート Claude Code](#) に対してログされるデータについて詳しく知ることができます。[Remote Control](#) セッションは、すべての実行がマシン上で行われるため、ローカルデータフロー

に従います。リモート Claude Code の場合、Claude は Claude Code セッションを開始するリポジトリにアクセスします。Claude は接続したが、セッションを開始していないリポジトリにはアクセスしません。

ローカル Claude Code：データフローと依存関係

以下の図は、インストール中および通常の操作中に Claude Code が外部サービスにどのように接続するかを示しています。実線は必須の接続を示し、破線はオプションまたはユーザーが開始したデータフローを表します。



Claude Code の外部接続を示す図：インストール/更新は NPM に接続し、ユーザーリクエストは Console 認証、public-api、およびオプションで Statsig、Sentry、バグレポートを含む Anthropic サービスに接続します

Claude Code は [NPM](#) からインストールされます。Claude Code はローカルで実行されます。LLM と対話するために、Claude Code はネットワーク経由でデータを送信します。このデータには、すべてのユーザープロンプトとモデル出力が含まれます。データは TLS 経由で転送中に暗号化され、保存時には暗号化されません。Claude Code はほとんどの一般的な VPN および LLM プロキシと互換性があります。

Claude Code は Anthropic の API 上に構築されています。API のセキュリティ制御（API ロギング手順を含む）の詳細については、[Anthropic Trust Center](#) で提供されるコンプライアンスアーティファクトを参照してください。

クラウド実行：データフローと依存関係

[Claude Code on the web](#) を使用する場合、セッションはローカルではなく Anthropic が管理する仮想マシンで実行されます。クラウド環境では：

- **コードとデータストレージ**：リポジトリは分離された VM にクローンされます。コードとセッションデータは、アカウントタイプのデータ保持および使用ポリシーの対象となります（上記のデータ保持セクションを参照）
- **認証情報**：GitHub 認証はセキュアプロキシを通じて処理されます。GitHub 認証情報がサンドボックスに入ることはありません
- **ネットワークトラフィック**：すべてのアウトバウンドトラフィックは、監査ログと不正使用防止のためのセキュリティプロキシを通じて行われます
- **セッションデータ**：プロンプト、コード変更、出力は、ローカル Claude Code 使用と同じデータポリシーに従います

クラウド実行のセキュリティの詳細については、[Security](#) を参照してください。

テレメトリサービス

Claude Code は、ユーザーのマシンから Statsig サービスに接続して、レイテンシ、信頼性、使用パターンなどの運用メトリックをログします。このログには、コードまたはファイルパスは含まれません。データは TLS を使用して転送中に暗号化され、256 ビット AES 暗号化を使用して保存時に暗号化されます。詳細については、[Statsig security documentation](#) を参照してください。Statsig テレメトリをオプトアウトするには、`DISABLE_TELEMETRY` 環境変数を設定します。

Claude Code は、ユーザーのマシンから Sentry に接続して、運用エラーログを記録します。データは TLS を使用して転送中に暗号化され、256 ビット AES 暗号化を使用して保存時に暗号化されます。詳細については、[Sentry security documentation](#) を参照してください。エラーログをオプトアウトするには、`DISABLE_ERROR_REPORTING` 環境変数を設定します。

ユーザーが `/bug` コマンドを実行すると、コードを含む完全な会話履歴のコピーが Anthropic に送信されます。データは転送中および保存時に暗号化されます。オプションで、公開リポジトリに Github イssueが作成されます。バグレポートをオプトアウトするには、`DISABLE_BUG_COMMAND` 環境変数を設定します。

API プロバイダーのデフォルト動作

デフォルトでは、Bedrock、Vertex、または Foundry を使用する場合、すべての非必須トラフィック（エラーレポート、テレメトリ、バグレポート機能、セッション品質調査を含む）を無効にします。 `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` 環境変数を設定することで、これらすべてを一度にオプトアウトすることもできます。以下は完全なデフォルト動作です：

サービス	Claude API	Vertex API	Bedrock API
Statsig (メトリクス)	デフォルトオン。 <code>DISABLE_TELEMETRY=1</code> で無効にします。	デフォルトオフ。 <code>CLAUDE_CODE_USE_VERTEX</code> は 1 である必要があります。	デフォルトオフ。 <code>CLAUDE_CODE_U</code> は 1 である必要が
Sentry (エラー)	デフォルトオン。 <code>DISABLE_ERROR_REPORTING=1</code> で無効にします。	デフォルトオフ。 <code>CLAUDE_CODE_USE_VERTEX</code> は 1 である必要があります。	デフォルトオフ。 <code>CLAUDE_CODE_U</code> は 1 である必要が
Claude API (/ bug レポート)	デフォルトオン。 <code>DISABLE_BUG_COMMAND=1</code> で無効にします。	デフォルトオフ。 <code>CLAUDE_CODE_USE_VERTEX</code> は 1 である必要があります。	デフォルトオフ。 <code>CLAUDE_CODE_U</code> は 1 である必要が

サービス	Claude API	Vertex API	Bedrock API
セッション品質調査	デフォルトオン。 CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1 で無効にします。	デフォルトオフ。 CLAUDE_CODE_USE_VERTEX は 1 である必要があります。	デフォルトオフ。 CLAUDE_CODE_U は 1 である必要が

すべての環境変数は settings.json にチェックインできます（[詳細を読む](#)）。

法的小よびコンプライアンス

Claude Codeの法的小契約、コンプライアンス認証、およびセキュリティ情報。

法的小契約

ライセンス

Claude Codeの使用は、以下の対象となります：

- ・ [商用利用規約](#) - Team、Enterprise、およびClaude APIユーザー向け
- ・ [コンシューマー利用規約](#) - Free、Pro、およびMaxユーザー向け

商用契約

Claude APIを直接（1P）使用している場合でも、AWS BedrockまたはGoogle Vertex経由でアクセスしている場合（3P）でも、別途合意がない限り、既存の商用契約がClaude Codeの使用に適用されます。

コンプライアンス

ヘルスケアコンプライアンス（BAA）

顧客がビジネスアソシエイト契約（BAA）を当社と締結しており、Claude Codeを使用したい場合、顧客がBAAを実行済みでゼロデータリテンション（ZDR）が有効化されていれば、BAAは自動的にClaude Codeをカバーするように拡張されます。BAAはClaude Codeを通じて流れるその顧客のAPIトラフィックに適用されます。

セキュリティと信頼

信頼とセーフティ

詳細情報は、[Anthropic Trust Center](#)および[Transparency Hub](#)で確認できます。

セキュリティ脆弱性報告

Anthropicはセキュリティプログラムを HackerOne を通じて管理しています。[このフォームを使用して脆弱性を報告してください](#)。

© Anthropic PBC. All rights reserved. Use is subject to applicable Anthropic Terms of Service.

ゼロデータ保持

Claude for Enterprise での Claude Code のゼロデータ保持（ZDR）について、スコープ、無効化される機能、有効化のリクエスト方法を学びます。

ゼロデータ保持（ZDR）は、Claude for Enterprise を通じて使用される Claude Code で利用可能です。ZDR が有効になると、Claude Code セッション中に生成されたプロンプトとモデル応答はリアルタイムで処理され、法令遵守またはミスユース対策が必要な場合を除き、応答が返された後は Anthropic によって保存されません。

Claude for Enterprise 上の ZDR により、エンタープライズカスタマーは Claude Code をゼロデータ保持で使用し、管理機能にアクセスできます：

- ユーザーごとのコスト管理
- [Analytics](#) ダッシュボード
- [Server-managed settings](#)
- 監査ログ

Claude for Enterprise 上の Claude Code の ZDR は、Anthropic の直接プラットフォームにのみ適用されます。AWS Bedrock、Google Vertex AI、または Microsoft Foundry 上の Claude デプロイメントについては、これらのプラットフォームのデータ保持ポリシーを参照してください。

ZDR スコープ

ZDR は Claude for Enterprise 上の Claude Code 推論をカバーします。

Warning:

ZDR は組織ごとに有効化されます。新しい各組織では、Anthropic アカウントチームによって ZDR を個別に有効化する必要があります。ZDR は同じアカウントの下に作成された新しい組織に自動的に適用されません。新しい組織に対して ZDR を有効化するには、アカウントチームに連絡してください。

ZDR がカバーする内容

ZDR は Claude for Enterprise 上の Claude Code を通じて行われたモデル推論呼び出しをカバーします。ターミナルで Claude Code を使用する場合、送信するプロンプトと Claude が生成する応答は Anthropic によって保持されません。これは、どの Claude モデルが使用されているかに関係なく適用されます。

ZDR がカバーしない内容

ZDR は、ZDR が有効化されている組織であっても、以下には適用されません。これらの機能は[標準データ保持ポリシー](#)に従います：

機能	詳細
claude.ai 上のチャット	Claude for Enterprise ウェブインターフェース経由のチャット会話は ZDR でカバーされません。
Cowork	Cowork セッションは ZDR でカバーされません。
Claude Code Analytics	プロンプトまたはモデル応答を保存しませんが、アカウントメールや使用統計などの生産性メタデータを収集します。貢献メトリクスは ZDR 組織では利用できません。 analytics ダッシュボード は使用メトリクスのみを表示します。
ユーザーとシート管理	アカウントメールやシート割り当てなどの管理データは標準ポリシーの下で保持されます。
サードパーティ統合	サードパーティツール、MCP servers、またはその他の外部統合によって処理されたデータは ZDR でカバーされません。これらのサービスのデータ処理慣行を独立して確認してください。

ZDR の下で無効化される機能

Claude for Enterprise 上の Claude Code 組織に対して ZDR が有効化されると、プロンプトまたは完了を保存する必要がある特定の機能はバックエンドレベルで自動的に無効化されます：

機能	理由
Web 上の Claude Code	会話履歴のサーバー側ストレージが必要です。
Desktop アプリからの リモートセッション	プロンプトと完了を含む永続的なセッションデータが必要です。

機能	理由
フィードバック送信（ /feedback ）	フィードバックを送信すると、会話データが Anthropic に送信されます。

これらの機能はクライアント側の表示に関係なく、バックエンドでブロックされます。Claude Code ターミナルの起動中に無効化された機能が表示される場合、それを使用しようとするエラーが返され、組織のポリシーがそのアクションを許可していないことが示されます。

プロンプトまたは完了を保存する必要がある場合、将来の機能も無効化される可能性があります。

ポリシー違反のためのデータ保持

ZDR が有効化されている場合でも、法律で必要な場合または Usage Policy 違反に対処するために、Anthropic はデータを保持する場合があります。セッションがポリシー違反でフラグが立てられた場合、Anthropic は関連する入力と出力を最大 2 年間保持する場合があります、これは Anthropic の標準 ZDR ポリシーと一致しています。

ZDR をリクエストする

Claude for Enterprise 上の Claude Code に対して ZDR をリクエストするには、Anthropic アカウントチームに連絡してください。アカウントチームが内部でリクエストを送信し、Anthropic は適格性を確認した後、組織に対して ZDR を確認して有効化します。すべての有効化アクションは監査ログに記録されます。

現在、従量課金制 API キーを介して Claude Code に対して ZDR を使用している場合、Claude for Enterprise に移行して、Claude Code の ZDR を維持しながら管理機能にアクセスできます。移行を調整するには、アカウントチームに連絡してください。

Part 10: Enterprise & Monitoring

チームの使用状況を分析で追跡する

Claude Code の使用メトリクスを表示し、採用状況を追跡し、分析ダッシュボードでエンジニアリング速度を測定します。

Claude Code は、組織が開発者の使用パターンを理解し、貢献メトリクスを追跡し、Claude Code がエンジニアリング速度にどのような影響を与えるかを測定するのに役立つ分析ダッシュボードを提供します。お客様のプランに応じたダッシュボードにアクセスしてください。

プラン	ダッシュボード URL	含まれる内容	詳細
Claude for Teams / Enterprise	claude.ai/analytics/claude-code	使用メトリクス、GitHub 統合による貢献メトリクス、リーダーボード、データエクスポート	詳細
API (Claude Console)	platform.claude.com/claude-code	使用メトリクス、支出追跡、チームインサイト	詳細

Teams と Enterprise の分析にアクセスする

claude.ai/analytics/claude-code に移動してください。管理者とオーナーがダッシュボードを表示できます。

Teams と Enterprise ダッシュボードには以下が含まれます。

- ・ **使用メトリクス**：受け入れられたコード行数、提案受け入れ率、日次アクティブユーザー数とセッション数
- ・ **貢献メトリクス**：[GitHub 統合](#)を使用した Claude Code 支援による PR とシブされたコード行数
- ・ **リーダーボード**：Claude Code 使用量でランク付けされたトップコントリビューター
- ・ **データエクスポート**：カスタムレポート用に貢献データを CSV としてダウンロード

貢献メトリクスを有効にする

Note:

貢献メトリクスはパブリックベータ版であり、Claude for Teams と Claude for Enterprise プランで利用可能です。これらのメトリクスは、[claude.ai](#) 組織内のユーザーのみをカバーしています。Claude Console API または サードパーティ統合を通じた使用は含まれていません。

使用状況と採用データは、すべての Claude for Teams と Claude for Enterprise アカウントで利用可能です。貢献メトリクスには、GitHub 組織を接続するための追加セットアップが必要です。

分析設定を構成するには、オーナーロールが必要です。GitHub 管理者が GitHub アプリをインストールする必要があります。

Warning:

[Zero Data Retention](#) が有効になっている組織では、貢献メトリクスは利用できません。分析ダッシュボードは使用メトリクスのみを表示します。

Step 1: GitHub アプリをインストールする

GitHub 管理者が、github.com/apps/claude で組織の GitHub アカウントに Claude GitHub アプリをインストールします。

Step 2: Claude Code 分析を有効にする

Claude オーナーが claude.ai/admin-settings/claude-code に移動し、Claude Code 分析機能を有効にします。

Step 3: GitHub 分析を有効にする

同じページで、'GitHub analytics' トグルを有効にします。

Step 4: GitHub で認証する

GitHub 認証フローを完了し、分析に含める GitHub 組織を選択します。

データは通常、有効化後 24 時間以内に表示され、毎日更新されます。データが表示されない場合は、以下のいずれかのメッセージが表示される可能性があります。

- 「**GitHub app required**」：貢献メトリクスを表示するには GitHub アプリをインストールしてください
- 「**Data processing in progress**」：数日後に確認し、データが表示されない場合は GitHub アプリがインストールされていることを確認してください

貢献メトリクスは GitHub Cloud と GitHub Enterprise Server をサポートしています。

サマリーメトリクスを確認する

Note:

これらのメトリクスは意図的に保守的であり、Claude Code の実際の影響の過小評価を表しています。Claude Code の関与に高い信頼度がある行と PR のみがカウントされます。

ダッシュボードの上部に以下のサマリーメトリクスが表示されます。

- **PRs with CC** : Claude Code で記述された少なくとも 1 行のコードを含むマージされたプルリクエストの総数
- **Lines of code with CC** : Claude Code 支援で記述されたすべてのマージされた PR 全体のコード行数。「有効な行」のみがカウントされます。正規化後に 3 文字以上の行で、空行と括弧またはトリビアルな句読点のみの行を除きます。
- **PRs with Claude Code (%)** : Claude Code 支援コードを含むすべてのマージされた PR のパーセンテージ
- **Suggestion accept rate** : ユーザーが Claude Code のコード編集提案を受け入れる回数のパーセンテージ。Edit、Write、NotebookEdit ツール使用を含みます。
- **Lines of code accepted** : ユーザーがセッション内で受け入れた Claude Code で記述されたコード行の総数。これは拒否された提案を除外し、その後の削除を追跡しません。

チャートを探索する

ダッシュボードには、時系列でトレンドを視覚化するためのいくつかのチャートが含まれています。

採用を追跡する

採用チャートは日次使用トレンドを表示します。

- **users** : 日次アクティブユーザー
- **sessions** : 1 日あたりのアクティブな Claude Code セッション数

ユーザーあたりの PR を測定する

このチャートは、時系列での個々の開発者アクティビティを表示します。

- **PRs per user** : 1 日にマージされた PR の総数を日次アクティブユーザーで割った値
- **users** : 日次アクティブユーザー

これを使用して、Claude Code の採用が増加するにつれて個々の生産性がどのように変化するかを理解します。

プルリクエストの内訳を表示する

プルリクエストチャートは、マージされた PR の日次内訳を表示します。

- **PRs with CC** : Claude Code 支援コードを含むプルリクエスト
- **PRs without CC** : Claude Code 支援コードを含まないプルリクエスト

Lines of code ビューに切り替えて、PR 数ではなくコード行数による同じ内訳を表示します。

トップコントリビューターを見つける

リーダーボードは、貢献量でランク付けされたトップ 10 ユーザーを表示します。以下を切り替えます。

- **Pull requests** : 各ユーザーの Claude Code を使用した PR とすべての PR を表示
- **Lines of code** : 各ユーザーの Claude Code を使用した行とすべての行を表示

Export all users をクリックして、すべてのユーザーの完全な貢献データを CSV ファイルとしてダウンロードします。エクスポートには、表示されているトップ 10 だけでなく、すべてのユーザーが含まれます。

PR 属性

貢献メトリクスが有効になっている場合、Claude Code はマージされたプルリクエストを分析して、Claude Code 支援で記述されたコードを判定します。これは、Claude Code セッションアクティビティを各 PR のコードと照合することで行われます。

タグ付け基準

PR は、Claude Code セッション中に記述された少なくとも 1 行のコードを含む場合、「with Claude Code」としてタグ付けされます。システムは保守的なマッチングを使用します。Claude Code の関与に高い信頼度がある場合のみ、支援されたコードとしてカウントされます。

属性プロセス

プルリクエストがマージされるとき。

1. 追加された行が PR diff から抽出されます
2. 時間ウィンドウ内で一致するファイルを編集した Claude Code セッションが識別されます
3. PR 行が複数の戦略を使用して Claude Code 出力と照合されます

4. AI 支援行と総行数のメトリクスが計算されます

比較前に、行は正規化されます。空白がトリミングされ、複数のスペースが折りたたまれ、引用符が標準化され、テキストが小文字に変換されます。

Claude Code 支援行を含むマージされたプルリクエストは、GitHub で `claude-code-assisted` としてラベル付けされます。

時間ウィンドウ

PR マージ日の 21 日前から 2 日後のセッションが属性マッチングの対象と見なされます。

除外されたファイル

特定のファイルは自動生成されるため、分析から自動的に除外されます。

- ロックファイル：package-lock.json、yarn.lock、Cargo.lock など
- 生成されたコード：Protobuf 出力、ビルドアーティファクト、縮小ファイル
- ビルドディレクトリ：dist/、build/、node_modules/、target/
- テストフィクスチャ：スナップショット、カセット、モックデータ
- 1,000 文字を超える行（縮小または生成されている可能性が高い）

属性に関する注記

属性データを解釈する際は、以下の追加の詳細に注意してください。

- 開発者によって大幅に書き直されたコード（20% 以上の差がある場合）は Claude Code に属性されません
- 21 日ウィンドウ外のセッションは考慮されません
- アルゴリズムは属性を実行するときに PR ソースまたは宛先ブランチを考慮しません

分析から最大限の価値を得る

貢献メトリクスを使用して ROI を実証し、採用パターンを特定し、他のユーザーが開始するのを支援できるチームメンバーを見つけます。

採用を監視する

採用チャートとユーザー数を追跡して、以下を特定します。

- ベストプラクティスを共有できるアクティブユーザー
- 組織全体の全体的な採用トレンド
- 摩擦または問題を示す可能性のある使用の低下

ROI を測定する

貢献メトリクスは、独自のコードベースからのデータを使用して「このツールは投資する価値があるか？」という質問に答えるのに役立ちます。

- 採用が増加するにつれて、時系列でユーザーあたりの PR の変化を追跡します
- Claude Code を使用した場合と使用しない場合の PR とシippされたコード行を比較します
- [DORA メトリクス](#)、スプリント速度、または他のエンジニアリング KPI と一緒に使用して、Claude Code の採用による変化を理解します

パワーユーザーを特定する

リーダーボードは、高い Claude Code 採用を持つチームメンバーを見つけるのに役立ちます。彼らは以下を行うことができます。

- プロンプティング技術とワークフローをチームと共有する
- 何がうまく機能しているかについてのフィードバックを提供する
- 新しいユーザーのオンボーディングを支援する

プログラムでデータにアクセスする

GitHub を通じてこのデータをクエリするには、`claude-code-assisted` でラベル付けされた PR を検索します。

API カスタマー向けの分析にアクセスする

Claude Console を使用している API カスタマーは、platform.claude.com/claude-code で分析にアクセスできます。ダッシュボードにアクセスするには UsageView 権限が必要です。これは Developer、Billing、Admin、Owner、Primary Owner ロールに付与されます。

Note:

GitHub 統合による貢献メトリクスは、現在 API カスタマーには利用できません。Console ダッシュボードは使用メトリクスと支出メトリクスのみを表示します。

Console ダッシュボードは以下を表示します。

- **Lines of code accepted** : ユーザーがセッション内で受け入れた Claude Code で記述されたコード行の総数。これは拒否された提案を除外し、その後の削除を追跡しません。

- **Suggestion accept rate**：ユーザーがコード編集ツール使用を受け入れる回数のパーセンテージ。Edit、Write、NotebookEdit ツールを含みます。
- **Activity**：チャートに表示される日次アクティブユーザーとセッション。
- **Spend**：ユーザー数と並んでドルでの日次 API コスト。

チームインサイトを表示する

チームインサイトテーブルはユーザーごとのメトリクスを表示します。

- **Members**：Claude Code に認証されたすべてのユーザー。API キーユーザーはキー識別子で表示され、OAuth ユーザーはメールアドレスで表示されます。
- **Spend this month**：ユーザーごとの現在の月の API コストの合計。
- **Lines this month**：ユーザーごとの現在の月の受け入れられたコード行の合計。

Note:

Console ダッシュボードの支出数値は分析目的の推定値です。実際のコストについては、請求ページを参照してください。

関連リソース

- [OpenTelemetry での監視](#)：リアルタイムメトリクスとイベントを可観測性スタックにエクスポート
- [コストを効果的に管理する](#)：支出制限を設定し、トークン使用を最適化
- [権限](#)：ロールと権限を構成

監視

Claude Code の OpenTelemetry を有効にして設定する方法を学びます。

Claude Code は、監視と可観測性のための OpenTelemetry (OTel) メトリクスとイベントをサポートしています。

すべてのメトリクスは OpenTelemetry の標準メトリクスプロトコルを介してエクスポートされる時系列データであり、イベントは OpenTelemetry のログ/イベントプロトコルを介してエクスポートされます。メトリクスとログのバックエンドが適切に設定されており、集約の粒度が監視要件を満たしていることを確認するのはユーザーの責任です。

クイックスタート

環境変数を使用して OpenTelemetry を設定します:

```
## 1. テレメトリを有効にする
export CLAUDE_CODE_ENABLE_TELEMETRY=1

## 2. エクスポートを選択する（両方はオプション - 必要なものを設定してください）
export OTEL_METRICS_EXPORTER=otlp          # オプション: otlp, prometheus, console
export OTEL_LOGS_EXPORTER=otlp            # オプション: otlp, console

## 3. OTLP エンドポイントを設定する（OTLP エクスポート用）
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## 4. 認証を設定する（必要な場合）
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer your-token"

## 5. デバッグ用: エクスポート間隔を短縮する
export OTEL_METRIC_EXPORT_INTERVAL=10000  # 10 秒（デフォルト: 60000ms）
export OTEL_LOGS_EXPORT_INTERVAL=5000    # 5 秒（デフォルト: 5000ms）

## 6. Claude Code を実行する
claude
```

Note:

デフォルトのエクスポート間隔は、メトリクスが 60 秒、ログが 5 秒です。セットアップ中は、デバッグ目的で短い間隔を使用することをお勧めします。本番環境での使用に向けてこれらをリセットすることを忘れないでください。

完全な設定オプションについては、[OpenTelemetry 仕様](#)を参照してください。

管理者設定

管理者は、[管理設定ファイル](#)を通じてすべてのユーザーの OpenTelemetry 設定を設定できます。これにより、組織全体のテレメトリ設定を一元管理できます。設定がどのように適用されるかについては、[設定の優先順位](#)を参照してください。

管理設定の設定例:

```
{
  "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:

管理設定は MDM (Mobile Device Management) または他のデバイス管理ソリューションを通じて配布できます。管理設定ファイルで定義された環境変数は優先度が高く、ユーザーによってオーバーライドすることはできません。

設定の詳細

一般的な設定変数

環境変数	説明	例の値
<code>CLAUDE_CODE_ENABLE_TELEMETRY</code>	テレメトリ収集を有効にする (必須)	<code>1</code>
<code>OTEL_METRICS_EXPORTER</code>	メトリクスエクスポートのタイプ (カンマ区切り)	<code>console</code> , <code>otlp</code> , <code>prometheus</code>
<code>OTEL_LOGS_EXPORTER</code>	ログ/イベントエクスポートのタイプ (カンマ区切り)	<code>console</code> , <code>otlp</code>
<code>OTEL_EXPORTER_OTLP_PROTOCOL</code>	OTLP エクスポートのプロトコル (すべてのシグナル)	<code>grpc</code> , <code>http/json</code> , <code>http/protobuf</code>
<code>OTEL_EXPORTER_OTLP_ENDPOINT</code>	OTLP コレクターエンドポイント (すべてのシグナル)	<code>http://localhost:4317</code>
<code>OTEL_EXPORTER_OTLP_METRICS_PROTOCOL</code>	メトリクスのプロトコル (一般的な設定をオーバーライド)	<code>grpc</code> , <code>http/json</code> , <code>http/protobuf</code>
<code>OTEL_EXPORTER_OTLP_METRICS_ENDPOINT</code>	OTLP メトリクスエンドポイント (一般的な設定をオーバーライド)	<code>http://localhost:4318/v1/metrics</code>
<code>OTEL_EXPORTER_OTLP_LOGS_PROTOCOL</code>	ログのプロトコル (一般的な設定をオーバーライド)	<code>grpc</code> , <code>http/json</code> , <code>http/protobuf</code>
<code>OTEL_EXPORTER_OTLP_LOGS_ENDPOINT</code>	OTLP ログエンドポイント (一般的な設定をオーバーライド)	<code>http://localhost:4318/v1/logs</code>
<code>OTEL_EXPORTER_OTLP_HEADERS</code>	OTLP の認証ヘッダー	<code>Authorization=Bearer token</code>
<code>OTEL_EXPORTER_OTLP_METRICS_CLIENT_KEY</code>	mTLS 認証用のクライアントキー	クライアントキーファイルへのパス
<code>OTEL_EXPORTER_OTLP_METRICS_CLIENT_CERTIFICATE</code>	mTLS 認証用のクライアント証明書	クライアント証明書ファイルへのパス

環境変数	説明	例の値
<code>OTEL_METRIC_EXPORT_INTERVAL</code>	エクスポート間隔 (ミリ秒単位、デフォルト: 60000)	<code>5000</code> , <code>60000</code>
<code>OTEL_LOGS_EXPORT_INTERVAL</code>	ログエクスポート間隔 (ミリ秒単位、デフォルト: 5000)	<code>1000</code> , <code>10000</code>
<code>OTEL_LOG_USER_PROMPTS</code>	ユーザープロンプトコンテンツのログを有効にする (デフォルト: 無効)	<code>1</code> で有効化
<code>CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS</code>	動的ヘッダーを更新するための間隔 (デフォルト: 1740000ms / 29 分)	<code>900000</code>

メトリクスカーディナリティ制御

以下の環境変数は、カーディナリティを管理するためにメトリクスに含まれる属性を制御します:

環境変数	説明	デフォルト値	無効化する例
<code>OTEL_METRICS_INCLUDE_SESSION_ID</code>	メトリクスに <code>session.id</code> 属性を含める	<code>true</code>	<code>false</code>
<code>OTEL_METRICS_INCLUDE_VERSION</code>	メトリクスに <code>app.version</code> 属性を含める	<code>false</code>	<code>true</code>
<code>OTEL_METRICS_INCLUDE_ACCOUNT_UUID</code>	メトリクスに <code>user.account_uuid</code> 属性を含める	<code>true</code>	<code>false</code>

これらの変数は、メトリクスのカーディナリティを制御するのに役立ちます。これはメトリクスバックエンドのストレージ要件とクエリパフォーマンスに影響します。カーディナリティが低いほど、一般的にパフォーマンスが向上し、ストレージコストが低くなりますが、分析用のより詳細なデータは少なくなります。

動的ヘッダー

動的認証が必要なエンタープライズ環境では、ヘッダーを動的に生成するスクリプトを設定できます:

設定ファイルの設定

`.claude/settings.json` に追加します:

```
{
  "otelHeadersHelper": "/bin/generate_opentelemetry_headers.sh"
}
```

スクリプト要件

スクリプトは HTTP ヘッダーを表す文字列キーと値のペアを持つ有効な JSON を出力する必要があります:

```
#!/bin/bash
## 例: 複数のヘッダー
echo '{"Authorization": \"Bearer $(get-token.sh)\", \"X-API-Key\": \"$(get-api-key.sh)\"}'
```

リフレッシュ動作

ヘッダーヘルパースクリプトはスタートアップ時に実行され、その後定期的に実行されてトークンリフレッシュをサポートします。デフォルトでは、スクリプトは 29 分ごとに実行されます。 `CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS` 環境変数で間隔をカスタマイズします。

マルチチーム組織サポート

複数のチームまたは部門を持つ組織は、 `OTEL_RESOURCE_ATTRIBUTES` 環境変数を使用してカスタム属性を追加し、異なるグループを区別できます:

```
## チーム識別用のカスタム属性を追加する
export OTEL_RESOURCE_ATTRIBUTES="department=engineering,team.id=platform,cost_center=eng-123"
```

これらのカスタム属性はすべてのメトリクスとイベントに含まれ、以下のことが可能になります:

- ・ チームまたは部門別にメトリクスをフィルタリングする
- ・ コストセンターごとのコストを追跡する
- ・ チーム固有のダッシュボードを作成する

- 特定のチームのアラートを設定する

Warning:**OTEL_RESOURCE_ATTRIBUTES の重要なフォーマット要件:**

`OTEL_RESOURCE_ATTRIBUTES` 環境変数は [W3C Baggage 仕様](#) に従い、厳密なフォーマット要件があります:

- **スペースは許可されません:** 値にスペースを含めることはできません。例えば、`user.organizationName=My Company` は無効です
- **フォーマット:** カンマ区切りのキー=値ペアである必要があります:
`key1=value1,key2=value2`
- **許可される文字:** 制御文字、空白、ダブルクォート、カンマ、セミコロン、バックスラッシュを除く US-ASCII 文字のみ
- **特殊文字:** 許可された範囲外の文字はパーセントエンコードする必要があります

例:

```
## ❌ 無効 - スペースを含む
export OTEL_RESOURCE_ATTRIBUTES="org.name=John's Organization"

## ✅ 有効 - アンダースコアまたはキャメルケースを代わりに使用する
export OTEL_RESOURCE_ATTRIBUTES="org.name=Johns_Organization"
export OTEL_RESOURCE_ATTRIBUTES="org.name=JohnsOrganization"

## ✅ 有効 - 必要に応じて特殊文字をパーセントエンコードする
export OTEL_RESOURCE_ATTRIBUTES="org.name=John%27s%20Organization"
```

注: 値をクォートで囲むことはスペースをエスケープしません。例えば、`org.name="My Company"` は `My Company` ではなく、リテラル値 `"My Company"` (クォート付き) になります。

監視

設定例

コンソールデバッグ (1 秒間隔)

```
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
```

複数のエクスポーター

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console,otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=http/json
```

メトリクスとログの異なるエンドポイント/バックエンド

```
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
```

メトリクスのみ (イベント/ログなし)

```
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
```

イベント/ログのみ (メトリクスなし)

```
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
```

利用可能なメトリクスとイベント

標準属性

すべてのメトリクスとイベントは、これらの標準属性を共有します:

属性	説明	制御者
<code>session.id</code>	一意のセッション識別子	<code>OTEL_METRICS_INCLUDE_SESSION_ID</code> (デフォルト: true)
<code>app.version</code>	現在の Claude Code バージョン	<code>OTEL_METRICS_INCLUDE_VERSION</code> (デフォルト: false)
<code>organization.id</code>	組織 UUID (認証時)	利用可能な場合は常に含まれます
<code>user.account.uuid</code>	アカウント UUID (認証時)	<code>OTEL_METRICS_INCLUDE_ACCOUNT_UUID</code> (デフォルト: true)
<code>terminal.type</code>	ターミナルタイプ (例: <code>iTerm.app</code> , <code>vscode</code> , <code>cursor</code> , <code>tmux</code>)	検出された場合は常に含まれます

メトリクス

Claude Code は以下のメトリクスをエクスポートします:

メトリクス名	説明	単位
<code>claude_code.session.count</code>	開始された CLI セッションの数	count
<code>claude_code.lines_of_code.count</code>	変更されたコード行の数	count
<code>claude_code.pull_request.count</code>	作成されたプルリクエストの数	count

メトリクス名	説明	単位
<code>claude_code.commit.count</code>	作成された git コミットの数	count
<code>claude_code.cost.usd</code>	Claude Code セッションのコスト	USD
<code>claude_code.token.usage</code>	使用されたトークンの数	tokens
<code>claude_code.code_edit_tool.decision</code>	コード編集ツールの権限決定の数	count
<code>claude_code.active_time.total</code>	合計アクティブ時間 (秒単位)	s

メトリクスの詳細

セッションカウンター

各セッションの開始時にインクリメントされます。

属性:

- すべての[標準属性](#)

コード行カウンター

コードが追加または削除されたときにインクリメントされます。

属性:

- すべての[標準属性](#)
- `type: ("added", "removed")`

プルリクエストカウンター

Claude Code を介してプルリクエストを作成するときにインクリメントされます。

属性:

- すべての[標準属性](#)

コミットカウンター

Claude Code を介して git コミットを作成するときにインクリメントされます。

属性:

- すべての[標準属性](#)

コストカウンター

各 API リクエスト後にインクリメントされます。

属性:

- すべての[標準属性](#)
- `model`: モデル識別子 (例: "claude-sonnet-4-5-20250929")

トークンカウンター

各 API リクエスト後にインクリメントされます。

属性:

- すべての[標準属性](#)
- `type`: ("input", "output", "cacheRead", "cacheCreation")
- `model`: モデル識別子 (例: "claude-sonnet-4-5-20250929")

コード編集ツール決定カウンター

ユーザーが Edit、Write、または NotebookEdit ツールの使用を受け入れるか拒否するときにインクリメントされます。

属性:

- すべての[標準属性](#)
- `tool`: ツール名 ("Edit", "Write", "NotebookEdit")
- `decision`: ユーザーの決定 ("accept", "reject")
- `language`: 編集されたファイルのプログラミング言語 (例: "TypeScript", "Python", "JavaScript", "Markdown")。認識されないファイル拡張子の場合は "unknown" を返します。

アクティブ時間カウンター

Claude Code を積極的に使用している実際の時間を追跡します (アイドル時間ではありません)。このメトリクスは、プロンプトの入力や応答の受信などのユーザーインタラクション中にインクリメントされます。

属性:

- すべての[標準属性](#)

イベント

Claude Code は、OpenTelemetry ログ/イベント経由で以下のイベントをエクスポートします (`OTEL_LOGS_EXPORTER` が設定されている場合):

ユーザープロンプトイベント

ユーザーがプロンプトを送信するときにログされます。

イベント名: `claude_code.user_prompt`

属性:

- すべての [標準属性](#)
- `event.name`: `"user_prompt"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `prompt_length`: プロンプトの長さ
- `prompt`: プロンプトコンテンツ (デフォルトではマスク、`OTEL_LOG_USER_PROMPTS=1` で有効化)

ツール結果イベント

ツールが実行を完了するときにログされます。

イベント名: `claude_code.tool_result`

属性:

- すべての [標準属性](#)
- `event.name`: `"tool_result"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `tool_name`: ツールの名前
- `success`: `"true"` または `"false"`
- `duration_ms`: 実行時間 (ミリ秒単位)
- `error`: エラーメッセージ (失敗した場合)
- `decision`: `"accept"` または `"reject"`
- `source`: 決定ソース - `"config"`, `"user_permanent"`, `"user_temporary"`, `"user_abort"`, または `"user_reject"`
- `tool_parameters`: ツール固有のパラメーターを含む JSON 文字列 (利用可能な場合)
 - Bash ツールの場合: `bash_command`, `full_command`, `timeout`, `description`, `sandbox` を含む

API リクエストイベント

Claude への各 API リクエストについてログされます。

イベント名: `claude_code.api_request`

属性:

- すべての[標準属性](#)
- `event.name`: `"api_request"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `model`: 使用されたモデル (例: “claude-sonnet-4-5-20250929”)
- `cost_usd`: 推定コスト (USD)
- `duration_ms`: リクエスト期間 (ミリ秒単位)
- `input_tokens`: 入力トークンの数
- `output_tokens`: 出力トークンの数
- `cache_read_tokens`: キャッシュから読み取られたトークンの数
- `cache_creation_tokens`: キャッシュ作成に使用されたトークンの数

API エラーイベント

Claude への API リクエストが失敗するときにログされます。

イベント名: `claude_code.api_error`

属性:

- すべての[標準属性](#)
- `event.name`: `"api_error"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `model`: 使用されたモデル (例: “claude-sonnet-4-5-20250929”)
- `error`: エラーメッセージ
- `status_code`: HTTP ステータスコード (該当する場合)
- `duration_ms`: リクエスト期間 (ミリ秒単位)
- `attempt`: 試行番号 (再試行されたリクエストの場合)

ツール決定イベント

ツール権限決定 (受け入れ/拒否) が行われるときにログされます。

イベント名: `claude_code.tool_decision`

属性:

- すべての標準属性
- `event.name`: "tool_decision"
- `event.timestamp`: ISO 8601 タイムスタンプ
- `tool_name`: ツールの名前 (例: "Read", "Edit", "Write", "NotebookEdit")
- `decision`: "accept" または "reject"
- `source`: 決定ソース - "config", "user_permanent", "user_temporary", "user_abort", または "user_reject"

メトリクスとイベントデータの解釈

Claude Code によってエクスポートされるメトリクスは、使用パターンと生産性に関する貴重な洞察を提供します。作成できる一般的な可視化と分析は次のとおりです:

使用状況監視

メトリクス	分析の機会
<code>claude_code.token.usage</code>	<code>type</code> (入力/出力)、ユーザー、チーム、またはモデル別に分類
<code>claude_code.session.count</code>	時間経過に伴う採用と関与を追跡
<code>claude_code.lines_of_code.count</code>	コード追加/削除を追跡して生産性を測定
<code>claude_code.commit.count</code> & <code>claude_code.pull_request.count</code>	開発ワークフローへの影響を理解

コスト監視

`claude_code.cost.usage` メトリクスは以下に役立ちます:

- チームまたは個人全体の使用トレンドを追跡する
- 最適化のための高使用セッションを特定する

Note:

コストメトリクスは概算です。公式な請求データについては、API プロバイダー (Claude Console、AWS Bedrock、または Google Cloud Vertex) を参照してください。

アラートとセグメンテーション

検討すべき一般的なアラート:

- ・ コストスパイク
- ・ 異常なトークン消費
- ・ 特定のユーザーからの高いセッションボリューム

すべてのメトリクスは、`user.account.uuid`、`organization.id`、`session.id`、`model`、および `app.version` でセグメント化できます。

イベント分析

イベントデータは Claude Code インタラクションに関する詳細な洞察を提供します:

ツール使用パターン: ツール結果イベントを分析して以下を特定します:

- ・ 最も頻繁に使用されるツール
- ・ ツール成功率
- ・ 平均ツール実行時間
- ・ ツールタイプ別のエラーパターン

パフォーマンス監視: API リクエスト期間とツール実行時間を追跡して、パフォーマンスボトルネックを特定します。

バックエンドに関する考慮事項

メトリクスとログバックエンドの選択により、実行できる分析のタイプが決まります:

メトリクスの場合

- ・ **時系列データベース (例: Prometheus):** レート計算、集約メトリクス
- ・ **カラムナーストア (例: ClickHouse):** 複雑なクエリ、一意のユーザー分析
- ・ **フル機能の可観測性プラットフォーム (例: Honeycomb、Datadog):** 高度なクエリ、可視化、アラート

イベント/ログの場合

- ・ **ログ集約システム (例: Elasticsearch、Loki):** 全文検索、ログ分析
- ・ **カラムナーストア (例: ClickHouse):** 構造化イベント分析
- ・ **フル機能の可観測性プラットフォーム (例: Honeycomb、Datadog):** メトリクスとイベント間の相関

日次/週次/月次アクティブユーザー (DAU/WAU/MAU) メトリクスが必要な組織の場合は、効率的な一意値クエリをサポートするバックエンドを検討してください。

サービス情報

すべてのメトリクスとイベントは、以下のリソース属性でエクスポートされます:

- `service.name`: `claude-code`
- `service.version`: 現在の Claude Code バージョン
- `os.type`: オペレーティングシステムタイプ (例: `linux`, `darwin`, `windows`)
- `os.version`: オペレーティングシステムバージョン文字列
- `host.arch`: ホストアーキテクチャ (例: `amd64`, `arm64`)
- `wsl.version`: WSL バージョン番号 (Windows Subsystem for Linux で実行している場合のみ存在)
- メーター名: `com.anthropic.claude_code`

ROI 測定リソース

テレメトリセットアップ、コスト分析、生産性メトリクス、自動レポート生成を含む Claude Code の投資収益率 (ROI) 測定に関する包括的なガイドについては、[Claude Code ROI 測定ガイド](#)を参照してください。このリポジトリは、すぐに使用できる Docker Compose 設定、Prometheus と OpenTelemetry セットアップ、Linear などのツールと統合された生産性レポート生成テンプレートを提供します。

セキュリティ/プライバシーに関する考慮事項

- テレメトリはオプトインであり、明示的な設定が必要です
- API キーやファイルコンテンツなどの機密情報は、メトリクスやイベントに含まれることはありません
- ユーザープロンプトコンテンツはデフォルトではマスクされます - プロンプト長のみが記録されます。ユーザープロンプトログを有効にするには、`OTEL_LOG_USER_PROMPTS=1` を設定します

Amazon Bedrock での Claude Code の監視

Amazon Bedrock での Claude Code 使用状況監視ガイドランスの詳細については、[Claude Code 監視実装 \(Bedrock\)](#)を参照してください。

エンタープライズデプロイメント概要

Claude Codeがさまざまなサードパーティサービスとインフラストラクチャと統合して、エンタープライズデプロイメント要件を満たす方法について学びます。

このページでは、利用可能なデプロイメントオプションの概要を提供し、組織に適した構成を選択するのに役立ちます。

プロバイダー比較

機能 Anthropic Amazon Bedrock Google Vertex AI Microsoft Foundry

リージョン サポート対象国 複数のAWS [リージョン](#) 複数のGCP [リージョン](#) 複数のAzure [リージョン](#)

プロンプトキャッシング デフォルトで有効 デフォルトで有効 デフォルトで有効 デフォルトで有効

認証 APIキー APIキーまたはAWS認証情報 GCP認証情報 APIキーまたはMicrosoft Entra ID

コスト追跡 ダッシュボード AWS Cost Explorer GCP Billing Azure Cost Management

エンタープライズ機能 チーム、使用状況監視 IAMポリシー、CloudTrail IAMロール、Cloud Audit Logs RBACポリシー、Azure Monitor

クラウドプロバイダー

- [Amazon Bedrock](#) AWSインフラストラクチャを通じてClaudeモデルを使用し、APIキーまたはIAMベースの認証とAWSネイティブ監視を備えています
- [Google Vertex AI](#) Google Cloud Platformを介してClaudeモデルにアクセスし、エンタープライズグレードのセキュリティとコンプライアンスを備えています
- [Microsoft Foundry](#) Azureを通じてClaudeにアクセスし、APIキーまたはMicrosoft Entra ID認証とAzure課金を備えています

企業インフラストラクチャ

- [Enterprise Network](#) Claude Codeを組織のプロキシサーバーとSSL/TLS要件と連携するように構成します
- [LLM Gateway](#) 使用状況追跡、予算設定、監査ログを備えた集中型モデルアクセスをデプロイします

構成概要

Claude Codeは、異なるプロバイダーとインフラストラクチャを組み合わせることができる柔軟な構成オプションをサポートしています。

Note:

以下の違いを理解してください：

- ・ **企業プロキシ**: トラフィックをルーティングするためのHTTP/HTTPSプロキシ (`HTTPS_PROXY` または `HTTP_PROXY` 経由で設定)
- ・ **LLM Gateway**: 認証を処理し、プロバイダー互換のエンドポイントを提供するサービス (`ANTHROPIC_BASE_URL` 、 `ANTHROPIC_BEDROCK_BASE_URL` 、または `ANTHROPIC_VERTEX_BASE_URL` 経由で設定)

両方の構成を同時に使用できます。

企業プロキシでBedrockを使用する

Bedrockトラフィックを企業HTTP/HTTPSプロキシを通じてルーティングします：

```
## Bedrockを有効にする
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1

## 企業プロキシを構成する
export HTTPS_PROXY='https://proxy.example.com:8080'
```

LLM GatewayでBedrockを使用する

Bedrock互換のエンドポイントを提供するゲートウェイサービスを使用します：

```
## Bedrockを有効にする
export CLAUDE_CODE_USE_BEDROCK=1

## LLM Gatewayを構成する
export ANTHROPIC_BEDROCK_BASE_URL='https://your-llm-gateway.com/bedrock'
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1 # ゲートウェイがAWS認証を処理する場合
```

企業プロキシでFoundryを使用する

Azureトラフィックを企業HTTP/HTTPSプロキシを通じてルーティングします：

```
## Microsoft Foundryを有効にする
export CLAUDE_CODE_USE_FOUNDRY=1
export ANTHROPIC_FOUNDRY_RESOURCE=your-resource
export ANTHROPIC_FOUNDRY_API_KEY=your-api-key # またはEntra ID認証の場合は省略

## 企業プロキシを構成する
export HTTPS_PROXY='https://proxy.example.com:8080'
```

LLM GatewayでFoundryを使用する

Azure互換のエンドポイントを提供するゲートウェイサービスを使用します：

```
## Microsoft Foundryを有効にする
export CLAUDE_CODE_USE_FOUNDRY=1

## LLM Gatewayを構成する
export ANTHROPIC_FOUNDRY_BASE_URL='https://your-llm-gateway.com'
export CLAUDE_CODE_SKIP_FOUNDRY_AUTH=1 # ゲートウェイがAzure認証を処理する場合
```

企業プロキシでVertex AIを使用する

Vertex AIトラフィックを企業HTTP/HTTPSプロキシを通じてルーティングします：

```
## Vertexを有効にする
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
export ANTHROPIC_VERTEX_PROJECT_ID=your-project-id

## 企業プロキシを構成する
export HTTPS_PROXY='https://proxy.example.com:8080'
```

LLM GatewayでVertex AIを使用する

Google Vertex AIモデルをLLM Gatewayと組み合わせて、集中管理を実現します：

```
## Vertexを有効にする
export CLAUDE_CODE_USE_VERTEX=1

## LLM Gatewayを構成する
export ANTHROPIC_VERTEX_BASE_URL='https://your-llm-gateway.com/vertex'
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1 # ゲートウェイがGCP認証を処理する場合
```

認証構成

Claude Codeは、必要に応じて `Authorization` ヘッダーに `ANTHROPIC_AUTH_TOKEN` を使用します。 `SKIP_AUTH` フラグ（ `CLAUDE_CODE_SKIP_BEDROCK_AUTH` 、 `CLAUDE_CODE_SKIP_VERTEX_AUTH` ）は、ゲートウェイがプロバイダー認証を処理するLLM Gatewayシナリオで使用されます。

適切なデプロイメント構成を選択する

デプロイメントアプローチを選択する際は、以下の要因を考慮してください：

直接プロバイダーアクセス

以下の組織に最適です：

- ・ 最もシンプルなセットアップを望む
- ・ 既存のAWSまたはGCPインフラストラクチャを持っている
- ・ プロバイダーネイティブの監視とコンプライアンスが必要

企業プロキシ

以下の組織に最適です：

- ・ 既存の企業プロキシ要件を持っている
- ・ トラフィック監視とコンプライアンスが必要
- ・ すべてのトラフィックを特定のネットワークパスを通じてルーティングする必要がある

LLM Gateway

以下の組織に最適です：

- ・ チーム全体での使用状況追跡が必要
- ・ モデル間を動的に切り替えたい
- ・ カスタムレート制限または予算が必要

- ・集中認証管理が必要

デバッグ

デプロイメントをデバッグする場合：

- ・ `claude /status` [スラッシュコマンド](#)を使用してください。このコマンドは、適用されている認証、プロキシ、およびURL設定の可視性を提供します。
- ・ 環境変数 `export ANTHROPIC_LOG=debug` を設定してリクエストをログします。

組織のベストプラクティス

1. ドキュメントとメモリに投資する

Claude Codeがコードベースを理解するようにドキュメントに投資することを強くお勧めします。組織は複数のレベルでCLAUDE.mdファイルをデプロイできます：

- ・ **組織全体:** `/Library/Application Support/ClaudeCode/CLAUDE.md` (macOS)などのシステムディレクトリにデプロイして、会社全体の標準を設定します
- ・ **リポジトリレベル:** リポジトリルートに `CLAUDE.md` ファイルを作成し、プロジェクトアーキテクチャ、ビルドコマンド、貢献ガイドラインを含めます。これらをソース管理にチェックインして、すべてのユーザーが利益を得られるようにします
[詳細を学ぶ](#)。

2. デプロイメントを簡素化する

カスタム開発環境がある場合、Claude Codeをインストールする「ワンクリック」の方法を作成することが、組織全体での採用を促進するための鍵となります。

3. ガイド付き使用から始める

新しいユーザーにClaude CodeをコードベースのQ&Aや小さなバグ修正または機能リクエストで試すことを勧めます。Claude Codeに計画を立てるよう依頼してください。Claudeの提案を確認し、外れている場合はフィードバックを提供してください。時間が経つにつれて、ユーザーがこの新しいパラダイムをより理解するようになると、Claude Codeをより自律的に実行させるのに効果的になります。

4. セキュリティポリシーを構成する

セキュリティチームは、Claude Codeが実行できることと実行できないことに関する管理権限を構成できます。これはローカル構成によって上書きされません。[詳細を学ぶ](#)。

5. 統合にMCPを活用する

MCPは、チケット管理システムやエラーログへの接続など、Claude Codeにより多くの情報を提供する優れた方法です。1つの中央チームがMCPサーバーを構成し、`.mcp.json` 構成をコードベースにチェックインして、すべてのユーザーが利益を得られるようにすることをお勧めします。[詳細を学ぶ](#)。

Anthropicでは、Claude CodeがすべてのAnthropicコードベース全体の開発を支援することを信賴しています。Claude Codeを使用することを楽しんでいただければ幸いです。

次のステップ

- [Amazon Bedrockをセットアップする](#) - AWS固有のデプロイメント用
- [Google Vertex AIを構成する](#) - GCPデプロイメント用
- [Microsoft Foundryをセットアップする](#) - Azureデプロイメント用
- [Enterprise Networkを構成する](#) - ネットワーク要件用
- [LLM Gatewayをデプロイする](#) - エンタープライズ管理用
- [設定](#) - 構成オプションと環境変数用

サーバー管理設定を構成する（パブリックベータ版）

デバイス管理インフラストラクチャを必要とせずに、サーバー配信設定を通じて組織全体で Claude Code を一元管理します。

サーバー管理設定により、管理者は Claude.ai 上のウェブベースインターフェースを通じて Claude Code を一元管理できます。Claude Code クライアントは、ユーザーが組織の認証情報で認証すると、これらの設定を自動的に受け取ります。

このアプローチは、デバイス管理インフラストラクチャが導入されていない組織、または管理されていないデバイス上のユーザーの設定を管理する必要がある組織向けに設計されています。

Note:

サーバー管理設定はパブリックベータ版であり、[Claude for Teams](#) および [Claude for Enterprise](#) カスタマーが利用できます。一般提供前に機能が変更される可能性があります。

要件

サーバー管理設定を使用するには、以下が必要です。

- Claude for Teams または Claude for Enterprise プラン
- Claude for Teams の場合はバージョン 2.1.38 以降、Claude for Enterprise の場合はバージョン 2.1.30 以降の Claude Code
- api.anthropic.com へのネットワークアクセス

サーバー管理設定とエンドポイント管理設定の選択

Claude Code は一元管理設定のための 2 つのアプローチをサポートしています。サーバー管理設定は Anthropic のサーバーから設定を配信します。[エンドポイント管理設定](#)は、ネイティブ OS ポリシー（macOS 管理設定、Windows レジストリ）または管理設定ファイルを通じてデバイスに直接配置されます。

アプローチ	最適な用途	セキュリティモデル
サーバー管理設定	MDM がない組織、または管理されていないデバイス上のユーザー	認証時に Anthropic のサーバーから配信される設定

アプローチ	最適な用途	セキュリティモデル
エンドポイント管理設定	MDM またはエンドポイント管理がある組織	MDM 構成プロファイル、レジストリポリシー、または管理設定ファイルを通じてデバイスに配置される設定

デバイスが MDM またはエンドポイント管理ソリューションに登録されている場合、エンドポイント管理設定はより強力なセキュリティ保証を提供します。これは、設定ファイルを OS レベルでユーザーの変更から保護できるためです。

サーバー管理設定を構成する

Step 1: 管理コンソールを開く

[Claude.ai](#) で、**Admin Settings > Claude Code > Managed settings** に移動します。

Step 2: 設定を定義する

設定を JSON として追加します。[settings.json](#) で利用可能なすべての設定がサポートされており、`disableBypassPermissionsMode` などの[管理専用設定](#)も含まれます。

この例は、権限拒否リストを適用し、ユーザーが権限をバイパスするのを防ぎます。

```
{
  "permissions": {
    "deny": [
      "Bash(curl *)",
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)"
    ]
  },
  "disableBypassPermissionsMode": "disable"
}
```

Step 3: 保存してデプロイする

変更を保存します。Claude Code クライアントは、次の起動時または 1 時間ごとのポーリングサイクルで更新された設定を受け取ります。

設定配信の確認

設定が適用されていることを確認するには、ユーザーに Claude Code を再起動するように依頼します。設定に[セキュリティ承認ダイアログ](#)をトリガーする設定が含まれている場合、ユーザーは起動時に管理設定を説明するプロンプトを表示します。また、ユーザーに `/permissions` を実行して有効な権限ルールを表示させることで、管理権限ルールがアクティブであることを確認することもできます。

アクセス制御

以下のロールがサーバー管理設定を管理できます。

- **Primary Owner**
- **Owner**

信頼できる担当者へのアクセスを制限してください。設定の変更は組織内のすべてのユーザーに適用されます。

現在の制限事項

サーバー管理設定には、ベータ期間中に以下の制限があります。

- 設定は組織内のすべてのユーザーに均一に適用されます。グループごとの設定はまだサポートされていません。
- [MCP サーバー構成](#)はサーバー管理設定を通じて配布することはできません。

設定配信

設定の優先順位

サーバー管理設定と[エンドポイント管理設定](#)は、Claude Code [設定階層](#)の最上位を占めます。コマンドライン引数を含む他の設定レベルはこれらをオーバーライドできません。両方が存在する場合、サーバー管理設定が優先され、エンドポイント管理設定は使用されません。

フェッチとキャッシング動作

Claude Code は起動時に Anthropic のサーバーから設定をフェッチし、アクティブなセッション中は1時間ごとに更新をポーリングします。

キャッシュされた設定なしの初回起動：

- Claude Code は非同期で設定をフェッチします
- フェッチが失敗した場合、Claude Code は管理設定なしで続行します
- 設定が読み込まれるまでの短い期間があり、その間は制限が適用されません

キャッシュされた設定での後続の起動：

- キャッシュされた設定は起動時に直ちに適用されます
- Claude Code はバックグラウンドで新しい設定をフェッチします
- キャッシュされた設定はネットワーク障害を通じて保持されます

Claude Code は再起動なしで設定更新を自動的に適用します。ただし、OpenTelemetry 構成などの高度な設定は、有効にするために完全な再起動が必要です。

セキュリティ承認ダイアログ

セキュリティリスクをもたらす可能性のある特定の設定には、適用される前に明示的なユーザー承認が必要です。

- **シェルコマンド設定**：シェルコマンドを実行する設定
- **カスタム環境変数**：既知の安全なアローリストにない変数
- **フック構成**：任意のフック定義

これらの設定が存在する場合、ユーザーは構成されている内容を説明するセキュリティダイアログを表示します。ユーザーは続行するために承認する必要があります。ユーザーが設定を拒否した場合、Claude Code は終了します。

Note:

-p フラグを使用した非対話モードでは、Claude Code はセキュリティダイアログをスキップし、ユーザー承認なしで設定を適用します。

プラットフォームの可用性

サーバー管理設定は api.anthropic.com への直接接続が必要であり、サードパーティのモデルプロバイダーを使用する場合は利用できません。

- Amazon Bedrock
- Google Vertex AI
- Microsoft Foundry
- **ANTHROPIC_BASE_URL** または **LLM ゲートウェイ** を通じたカスタム API エンドポイント

監査ログ

設定変更の監査ログイベントは、コンプライアンス API または監査ログエクスポートを通じて利用できます。アクセスについては、Anthropic アカウントチームにお問い合わせください。

監査イベントには、実行されたアクションのタイプ、アクションを実行したアカウントとデバイス、および前の値と新しい値への参照が含まれます。

セキュリティに関する考慮事項

サーバー管理設定は一元化されたポリシー適用を提供しますが、クライアント側の制御として機能します。管理されていないデバイスでは、管理者またはスーパーユーザーアクセス権を持つユーザーは Claude Code バイナリ、ファイルシステム、またはネットワーク構成を変更できます。

シナリオ	動作
ユーザーがキャッシュされた設定ファイルを編集する	改ざんされたファイルは起動時に適用されますが、次のサーバーフェッチで正しい設定が復元されます
ユーザーがキャッシュされた設定ファイルを削除する	初回起動動作が発生します。設定は非同期でフェッチされ、短い未適用ウィンドウがあります
API が利用できない	キャッシュされた設定が利用可能な場合は適用されます。そうでない場合、管理設定は次の成功したフェッチまで適用されません
ユーザーが別の組織で認証する	管理対象組織外のアカウントには設定が配信されません
ユーザーがデフォルト以外の <code>ANTHROPIC_BASE_URL</code> を設定する	サードパーティの API プロバイダーを使用する場合、サーバー管理設定はバイパスされます

ランタイム構成の変更を検出するには、`ConfigChange` フックを使用して変更をログに記録するか、変更が有効になる前に不正な変更をブロックします。

より強力な適用保証については、MDM ソリューションに登録されているデバイスで [エンドポイント管理設定](#) を使用します。

関連項目

Claude Code 構成を管理するための関連ページ。

- [Settings](#)：すべての利用可能な設定を含む完全な構成リファレンス
- [Endpoint-managed settings](#)：IT によってデバイスに配置される管理設定
- [Authentication](#)：Claude Code へのユーザーアクセスのセットアップ
- [Security](#)：セキュリティ保護とベストプラクティス

Part 11: Cloud Providers

Amazon Bedrock 上の Claude Code

Amazon Bedrock を通じた Claude Code の設定方法（セットアップ、IAM 設定、トラブルシューティングを含む）について学習します。

前提条件

Claude Code を Bedrock で設定する前に、以下を確認してください。

- Bedrock アクセスが有効になっている AWS アカウント
- Bedrock で目的の Claude モデル（例：Claude Sonnet 4.6）へのアクセス
- AWS CLI がインストールされ、設定されていること（オプション - 認証情報を取得する別のメカニズムがない場合のみ必要）
- 適切な IAM 権限

Note:

Claude Code を複数のユーザーにデプロイする場合は、[モデルバージョンをピン留めして](#)、Anthropic が新しいモデルをリリースしたときの破損を防いでください。

セットアップ

1. ユースケースの詳細を送信

Anthropic モデルの初回ユーザーは、モデルを呼び出す前にユースケースの詳細を送信する必要があります。これはアカウントごとに 1 回行われます。

1. 適切な IAM 権限があることを確認してください（以下を参照）
2. [Amazon Bedrock コンソール](#) に移動します
3. **Chat/Text playground** を選択します
4. Anthropic モデルを選択すると、ユースケースフォームの入力を求められます

2. AWS 認証情報を設定

Claude Code は、デフォルトの AWS SDK 認証情報チェーンを使用します。以下のいずれかの方法を使用して認証情報を設定してください。

オプション A : AWS CLI 設定

```
aws configure
```

オプション B : 環境変数 (アクセスキー)

```
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
```

オプション C : 環境変数 (SSO プロファイル)

```
aws sso login --profile=<your-profile-name>  
  
export AWS_PROFILE=your-profile-name
```

オプション D : AWS Management Console 認証情報

```
aws login
```

`aws login` について[詳しく学習](#)してください。

オプション E : Bedrock API キー

```
export AWS_BEARER_TOKEN_BEDROCK=your-bedrock-api-key
```

Bedrock API キーは、完全な AWS 認証情報を必要としない、より簡単な認証方法を提供します。[Bedrock API キーについて詳しく学習](#)してください。

高度な認証情報設定

Claude Code は、AWS SSO および企業 ID プロバイダーの自動認証情報更新をサポートしています。これらの設定を Claude Code 設定ファイルに追加してください（ファイルの場所については[Settings](#)を参照）。

Claude Code が AWS 認証情報の有効期限が切れていることを検出した場合（ローカルのタイムスタンプに基づくか、Bedrock が認証情報エラーを返した場合）、設定された `awsAuthRefresh` および/または `awsCredentialExport` コマンドを自動的に実行して、リクエストを再試行する前に新しい認証情報を取得します。

設定例

```
{
  "awsAuthRefresh": "aws sso login --profile myprofile",
  "env": {
    "AWS_PROFILE": "myprofile"
  }
}
```

設定の説明

awsAuthRefresh : `.aws` ディレクトリを変更するコマンド（認証情報、SSO キャッシュ、または設定ファイルの更新など）に使用します。コマンドの出力はユーザーに表示されますが、対話的な入力はサポートされていません。これは、CLI が URL またはコードを表示し、ブラウザで認証を完了するブラウザベースの SSO フローに適しています。

awsCredentialExport : `.aws` を変更せず、認証情報を直接返す必要がある場合にのみ使用します。出力はサイレントにキャプチャされ、ユーザーに表示されません。コマンドは次の形式で JSON を出力する必要があります。

```
{
  "Credentials": {
    "AccessKeyId": "value",
    "SecretAccessKey": "value",
    "SessionToken": "value"
  }
}
```

3. Claude Code を設定

Bedrock を有効にするために、以下の環境変数を設定します。

```
## Bedrock 統合を有効にする
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1 # または希望するリージョン

## オプション：小型/高速モデル (Haiku) のリージョンをオーバーライド
export ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION=us-west-2
```

Claude Code で Bedrock を有効にする場合は、以下に注意してください。

- **AWS_REGION** は必須の環境変数です。Claude Code はこの設定について **.aws** 設定ファイルから読み込みません。
- Bedrock を使用する場合、**/login** および **/logout** コマンドは無効になります。認証は AWS 認証情報を通じて処理されるためです。
- 他のプロセスに漏らしたくない **AWS_PROFILE** などの環境変数に設定ファイルを使用できます。詳細については [Settings](#) を参照してください。

4. モデルバージョンをピン留め

Warning:

すべてのデプロイメントに対して特定のモデルバージョンをピン留めしてください。モデルエイリアス（**sonnet**、**opus**、**haiku**）をピン留めなしで使用する場合、Claude Code は Bedrock アカウントで利用できない新しいモデルバージョンを使用しようとする可能性があります、Anthropic がアップデートをリリースしたときに既存のユーザーが破損します。

これらの環境変数を特定の Bedrock モデル ID に設定します。

```
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'
```

これらの変数は、クロスリージョン推論プロファイル ID（**us.** プレフィックス付き）を使用します。別のリージョンプレフィックスまたはアプリケーション推論プロファイルを使用する場合は、それに応じて調整してください。現在および従来のモデル ID については、[Models overview](#) を参照してください。環境変数の完全なリストについては、[Model configuration](#) を参照してください。

Claude Code は、ピン留め変数が設定されていない場合、これらのデフォルトモデルを使用します。

モデルタイプ	デフォルト値
プライマリモデル	<code>global.anthropic.claude-sonnet-4-6</code>
小型/高速モデル	<code>us.anthropic.claude-haiku-4-5-20251001-v1:0</code>

モデルをさらにカスタマイズするには、以下のいずれかの方法を使用します。

```
## 推論プロファイル ID を使用
export ANTHROPIC_MODEL='global.anthropic.claude-sonnet-4-6'
export ANTHROPIC_SMALL_FAST_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'

## アプリケーション推論プロファイル ARN を使用
export ANTHROPIC_MODEL='arn:aws:bedrock:us-east-2:your-account-id:application-
inference-profile/your-model-id'

## オプション：必要に応じてプロンプトキャッシングを無効にする
export DISABLE_PROMPT_CACHING=1
```

Note:

[プロンプトキャッシング](#)は、すべてのリージョンで利用できない場合があります。

各モデルバージョンを推論プロファイルにマップ

`ANTHROPIC_DEFAULT_*_MODEL` 環境変数は、モデルファミリーごとに1つの推論プロファイルを設定します。組織が同じファミリーの複数のバージョンを `/model` ピッカーで公開し、それぞれを独自のアプリケーション推論プロファイル ARN にルーティングする必要がある場合は、代わりに `settings file` の `modelOverrides` 設定を使用してください。

この例は、3つの Opus バージョンを異なる ARN にマップするため、ユーザーは組織の推論プロファイルをバイパスすることなく、それらを切り替えることができます。

```
{
  "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"
  }
}
```

ユーザーが `/model` でこれらのバージョンのいずれかを選択すると、Claude Code はマップされた ARN で Bedrock を呼び出します。オーバーライドのないバージョンは、組み込みの Bedrock モデル ID またはスタートアップで検出された一致する推論プロファイルにフォールバックします。オーバーライドが `availableModels` および他のモデル設定とどのように相互作用するかについては、[Override model IDs per version](#) を参照してください。

IAM 設定

Claude Code に必要な権限を持つ IAM ポリシーを作成します。

```

{
  "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"
        }
      }
    }
  ]
}

```

より制限的な権限の場合は、リソースを特定の推論プロファイル ARN に制限できます。

詳細については、[Bedrock IAM documentation](#) を参照してください。

Note:

コスト追跡とアクセス制御を簡素化するために、Claude Code 用の専用 AWS アカウントを作成してください。

AWS Guardrails

[Amazon Bedrock Guardrails](#) を使用すると、Claude Code のコンテンツフィルタリングを実装できます。[Amazon Bedrock コンソール](#)で Guardrail を作成し、バージョンを公開してから、Guardrail ヘッダーを [settings file](#) に追加します。クロスリージョン推論プロファイルを使用している場合は、Guardrail でクロスリージョン推論を有効にしてください。

設定例：

```
{
  "env": {
    "ANTHROPIC_CUSTOM_HEADERS": "X-Amzn-Bedrock-GuardrailIdentifier: your-guardrail-id\nX-Amzn-Bedrock-GuardrailVersion: 1"
  }
}
```

トラブルシューティング

リージョンの問題が発生した場合：

- モデルの可用性を確認：`aws bedrock list-inference-profiles --region your-region`
- サポートされているリージョンに切り替え：`export AWS_REGION=us-east-1`
- クロスリージョンアクセスに推論プロファイルの使用を検討

「on-demand throughput isn't supported」エラーが表示される場合：

- モデルを [inference profile](#) ID として指定します

Claude Code は Bedrock [Invoke API](#) を使用し、Converse API はサポートしていません。

追加リソース

- [Bedrock documentation](#)
- [Bedrock pricing](#)

- [Bedrock inference profiles](#)
- [Claude Code on Amazon Bedrock: Quick Setup Guide](#)
- [Claude Code Monitoring Implementation \(Bedrock\)](#)

Google Vertex AI 上の Claude Code

Google Vertex AI を通じた Claude Code の設定方法について学びます。セットアップ、IAM 設定、トラブルシューティングを含みます。

前提条件

Claude Code を Vertex AI で設定する前に、以下を確認してください。

- 請求が有効になっている Google Cloud Platform (GCP) アカウント
- Vertex AI API が有効になっている GCP プロジェクト
- 目的の Claude モデルへのアクセス（例：Claude Sonnet 4.5）
- Google Cloud SDK（`gcloud`）がインストールおよび設定されていること
- 目的の GCP リージョンに割り当てられたクォータ

リージョン設定

Claude Code は Vertex AI の [グローバル](#) エンドポイントとリージョナルエンドポイントの両方で使用できます。

Note:

Vertex AI は、すべてのリージョンで Claude Code のデフォルトモデルをサポートしていない場合があります。[サポートされているリージョンまたはモデル](#)に切り替える必要がある場合があります。

Note:

Vertex AI は、グローバルエンドポイントで Claude Code のデフォルトモデルをサポートしていない場合があります。リージョナルエンドポイントまたは[サポートされているモデル](#)に切り替える必要がある場合があります。

セットアップ

1. Vertex AI API を有効にする

GCP プロジェクトで Vertex AI API を有効にします。

```
## プロジェクト ID を設定
gcloud config set project YOUR-PROJECT-ID

## Vertex AI API を有効にする
gcloud services enable aiplatform.googleapis.com
```

2. モデルアクセスをリクエストする

Vertex AI で Claude モデルへのアクセスをリクエストします。

1. [Vertex AI Model Garden](#) に移動します
2. 「Claude」モデルを検索します
3. 目的の Claude モデルへのアクセスをリクエストします（例：Claude Sonnet 4.5）
4. 承認を待ちます（24 ～ 48 時間かかる場合があります）

3. GCP 認証情報を設定する

Claude Code は標準的な Google Cloud 認証を使用します。

詳細については、[Google Cloud 認証ドキュメント](#)を参照してください。

Note:

認証時に、Claude Code は `ANTHROPIC_VERTEX_PROJECT_ID` 環境変数からプロジェクト ID を自動的に使用します。これをオーバーライドするには、`GCLOUD_PROJECT`、`GOOGLE_CLOUD_PROJECT`、または `GOOGLE_APPLICATION_CREDENTIALS` のいずれかの環境変数を設定します。

4. Claude Code を設定する

以下の環境変数を設定します。

```
## Vertex AI 統合を有効にする
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=global
export ANTHROPIC_VERTEX_PROJECT_ID=YOUR-PROJECT-ID

## オプション：必要に応じてプロンプトキャッシングを無効にする
export DISABLE_PROMPT_CACHING=1

## CLOUD_ML_REGION=global の場合、サポートされていないモデルのリージョンをオーバーライドする
export VERTEX_REGION_CLAUDE_3_5_HAIKU=us-east5

## オプション：他の特定のモデルのリージョンをオーバーライドする
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:

[プロンプトキャッシング](#)は、`cache_control` エフェメラルフラグを指定すると自動的にサポートされます。これを無効にするには、`DISABLE_PROMPT_CACHING=1` を設定します。レート制限を高くするには、Google Cloud サポートにお問い合わせください。

Note:

Vertex AI を使用する場合、Google Cloud 認証情報を通じて認証が処理されるため、`/login` および `/logout` コマンドは無効になります。

5. モデル設定

Claude Code は Vertex AI に対して以下のデフォルトモデルを使用します。

モデルタイプ	デフォルト値
プライマリモデル	<code>claude-sonnet-4-5@20250929</code>
小型/高速モデル	<code>claude-haiku-4-5@20251001</code>

Note:

Vertex AI ユーザーの場合、Claude Code は Haiku 3.5 から Haiku 4.5 に自動的にアップグレードされません。新しい Haiku モデルに手動で切り替えるには、

`ANTHROPIC_DEFAULT_HAIKU_MODEL` 環境変数をフルモデル名に設定します（例： `claude-haiku-4-5@20251001` ）。

モデルをカスタマイズするには、以下を実行します。

```
export ANTHROPIC_MODEL='claude-opus-4-1@20250805'  
export ANTHROPIC_SMALL_FAST_MODEL='claude-haiku-4-5@20251001'
```

IAM 設定

必要な IAM パーミッションを割り当てます。

`roles/aipplatform.user` ロールには、必要なパーミッションが含まれています。

- `aipplatform.endpoints.predict` - モデル呼び出しとトークンカウントに必要

より制限的なパーミッションの場合は、上記のパーミッションのみを持つカスタムロールを作成します。

詳細については、[Vertex IAM ドキュメント](#)を参照してください。

Note:

コスト追跡とアクセス制御を簡素化するために、Claude Code 用に専用の GCP プロジェクトを作成することをお勧めします。

100 万トークンコンテキストウィンドウ

Claude Sonnet 4 および Sonnet 4.5 は Vertex AI で[100 万トークンコンテキストウィンドウ](#)をサポートしています。

Note:

100 万トークンコンテキストウィンドウは現在ベータ版です。拡張コンテキストウィンドウを使用するには、Vertex AI リクエストに `context-1m-2025-08-07` ベータヘッダーを含めます。

トラブルシューティング

クォータの問題が発生した場合：

- [Cloud Console](#) を通じて現在のクォータを確認するか、クォータ増加をリクエストします

「モデルが見つかりません」 404 エラーが発生した場合：

- [Model Garden](#) でモデルが有効になっていることを確認します
- 指定されたリージョンへのアクセス権があることを確認します
- `CLOUD_ML_REGION=global` を使用している場合、[Model Garden](#) の「Supported features」でモデルがグローバルエンドポイントをサポートしていることを確認します。グローバルエンドポイントをサポートしていないモデルの場合は、以下のいずれかを実行します。
 - `ANTHROPIC_MODEL` または `ANTHROPIC_SMALL_FAST_MODEL` を通じてサポートされているモデルを指定するか、
 - `VERTEX_REGION_<MODEL_NAME>` 環境変数を使用してリージョナルエンドポイントを設定します

429 エラーが発生した場合：

- リージョナルエンドポイントの場合、プライマリモデルと小型/高速モデルが選択したリージョンでサポートされていることを確認します
- より良い可用性のために `CLOUD_ML_REGION=global` への切り替えを検討します

追加リソース

- [Vertex AI ドキュメント](#)
- [Vertex AI 価格](#)
- [Vertex AI クォータと制限](#)

Claude Code on Microsoft Foundry

Microsoft Foundryを通じてClaude Codeを構成する方法について学びます。セットアップ、構成、トラブルシューティングを含みます。

前提条件

Microsoft FoundryでClaude Codeを構成する前に、以下を確認してください：

- Microsoft Foundryへのアクセス権を持つAzureサブスクリプション
- Microsoft Foundryリソースとデプロイメントを作成するためのRBAC権限
- Azure CLIがインストールされ、構成されている（オプション - 認証情報を取得する別のメカニズムがない場合のみ必要）

セットアップ

1. Microsoft Foundryリソースをプロビジョニングする

まず、AzureでClaudeリソースを作成します：

1. [Microsoft Foundryポータル](#)に移動します
2. 新しいリソースを作成し、リソース名をメモします
3. Claudeモデルのデプロイメントを作成します：
 - Claude Opus
 - Claude Sonnet
 - Claude Haiku

2. Azure認証情報を構成する

Claude CodeはMicrosoft Foundryの2つの認証方法をサポートしています。セキュリティ要件に最適な方法を選択してください。

オプションA：APIキー認証

1. Microsoft Foundryポータルでリソースに移動します
2. エンドポイントとキーセクションに移動します
3. APIキーをコピーします
4. 環境変数を設定します：

```
export ANTHROPIC_FOUNDRY_API_KEY=your-azure-api-key
```

オプションB：Microsoft Entra ID認証

`ANTHROPIC_FOUNDRY_API_KEY` が設定されていない場合、Claude CodeはAzure SDKデフォルト認証情報チェーンを自動的に使用します。これは、ローカルおよびリモートワークロードを認証するためのさまざまな方法をサポートしています。

ローカル環境では、一般的にAzure CLIを使用できます：

```
az login
```

Note:

Microsoft Foundryを使用する場合、認証がAzure認証情報を通じて処理されるため、`/login` および `/logout` コマンドは無効になります。

3. Claude Codeを構成する

Microsoft Foundryを有効にするには、以下の環境変数を設定します。デプロイメント名はClaude Codeのモデル識別子として設定されます（推奨されるデプロイメント名を使用している場合はオプションの場合があります）。

```
## Microsoft Foundry統合を有効にする
export CLAUDE_CODE_USE_FOUNDRY=1

## Azureリソース名（{resource}をリソース名に置き換えます）
export ANTHROPIC_FOUNDRY_RESOURCE={resource}

## または完全なベースURLを提供します：
## export ANTHROPIC_FOUNDRY_BASE_URL=https://{resource}.services.ai.azure.com

## モデルをリソースのデプロイメント名に設定します
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'
```

モデル構成オプションの詳細については、[モデル構成](#)を参照してください。

Azure RBAC構成

`Azure AI User` および `Cognitive Services User` デフォルトロールには、Claudeモデルを呼び出すために必要なすべての権限が含まれています。

より制限的な権限の場合は、以下を含むカスタムロールを作成します：

```
{
  "permissions": [
    {
      "dataActions": [
        "Microsoft.CognitiveServices/accounts/providers/*"
      ]
    }
  ]
}
```

詳細については、[Microsoft Foundry RBACドキュメント](#)を参照してください。

トラブルシューティング

「Failed to get token from azureADTokenProvider: ChainedTokenCredential authentication failed」というエラーが表示される場合：

- 環境でEntra IDを構成するか、`ANTHROPIC_FOUNDRY_API_KEY`を設定してください。

その他のリソース

- [Microsoft Foundryドキュメント](#)
- [Microsoft Foundryモデル](#)
- [Microsoft Foundry価格](#)

LLM gateway設定

Claude CodeをLLM gatewayソリューションと連携するための設定方法を学びます。gateway要件、認証設定、モデル選択、プロバイダー固有のエンドポイント設定をカバーしています。

LLM gatewayは、Claude Codeとモデルプロバイダー間の集中型プロキシレイヤーを提供し、以下のような機能をしばしば提供します：

- ・ **集中型認証** - API キー管理の単一ポイント
- ・ **使用状況追跡** - チームとプロジェクト全体での使用状況の監視
- ・ **コスト管理** - 予算とレート制限の実装
- ・ **監査ログ** - コンプライアンスのためのすべてのモデル相互作用の追跡
- ・ **モデルルーティング** - コード変更なしでプロバイダー間の切り替え

Gateway要件

LLM gatewayがClaude Codeと連携するには、以下の要件を満たす必要があります：

API形式

gatewayは、クライアントに対して以下のAPI形式の少なくとも1つを公開する必要があります：

1. **Anthropic Messages:** `/v1/messages` , `/v1/messages/count_tokens`
 - ・ リクエストヘッダーを転送する必要があります: `anthropic-beta` , `anthropic-version`
2. **Bedrock InvokeModel:** `/invoke` , `/invoke-with-response-stream`
 - ・ リクエストボディフィールドを保持する必要があります: `anthropic_beta` , `anthropic_version`
3. **Vertex rawPredict:** `:rawPredict` , `:streamRawPredict` , `/count-tokens:rawPredict`
 - ・ リクエストヘッダーを転送する必要があります: `anthropic-beta` , `anthropic-version`

ヘッダーの転送またはボディフィールドの保持に失敗すると、機能が低下したり、Claude Code機能を使用できなくなる可能性があります。

Note:

Claude Codeは、API形式に基づいて有効にする機能を決定します。Anthropic Messages形式をBedrocまたはVertexで使用する場合、環境変数

`CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS=1` を設定する必要があります。

設定

モデル選択

デフォルトでは、Claude Codeは選択したAPI形式の標準モデル名を使用します。

gatewayでカスタムモデル名を設定している場合は、[モデル設定](#)に記載されている環境変数を使用して、カスタム名と一致させてください。

LiteLLM設定

Note:

LiteLLMはサードパーティのプロキシサービスです。Anthropicは、LiteLLMのセキュリティまたは機能を推奨、保守、または監査していません。このガイドは情報提供目的で提供されており、古くなる可能性があります。自己判断で使用してください。

前提条件

- Claude Codeが最新バージョンに更新されている
- LiteLLM Proxy Serverがデプロイされてアクセス可能
- 選択したプロバイダーを通じてClaudeモデルへのアクセス

基本的なLiteLLMセットアップ

Claude Codeを設定する:

認証方法

静的APIキー

固定APIキーを使用した最も簡単な方法：

```
## 環境で設定
export ANTHROPIC_AUTH_TOKEN=sk-litellm-static-key

## またはClaude Code設定で
{
  "env": {
    "ANTHROPIC_AUTH_TOKEN": "sk-litellm-static-key"
  }
}
```

この値は **Authorization** ヘッダーとして送信されます。

ヘルパーを使用した動的APIキー

キーのローテーションまたはユーザーごとの認証の場合：

1. APIキーヘルパースクリプトを作成します：

```
#!/bin/bash
## ~/bin/get-litellm-key.sh

## 例：vaultからキーを取得
vault kv get -field=api_key secret/litellm/claude-code

## 例：JWTトークンを生成
jwt encode \
  --secret="${JWT_SECRET}" \
  --exp="+1h" \
  '{"User":"'${USER}'","team":"engineering"}'
```

1. ヘルパーを使用するようにClaude Code設定を構成します：

```
{
  "apiKeyHelper": "~/bin/get-litellm-key.sh"
}
```

1. トークンリフレッシュ間隔を設定します：

```
## 1時間ごとにリフレッシュ (3600000 ms)
export CLAUDE_CODE_API_KEY_HELPER_TTL_MS=3600000
```

この値は `Authorization` および `X-API-Key` ヘッダーとして送信されます。

`apiKeyHelper` は `ANTHROPIC_AUTH_TOKEN` または `ANTHROPIC_API_KEY` より優先度が低くなります。

統合エンドポイント（推奨）

LiteLLMの[Anthropic形式エンドポイント](#)を使用：

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000
```

統合エンドポイントのパススルーエンドポイント上での利点：

- ・ロードバランシング
- ・フェイルオーバー
- ・コスト追跡とエンドユーザー追跡の一貫したサポート

プロバイダー固有のパススルーエンドポイント（代替）

LiteLLMを通じたClaude API

[パススルーエンドポイント](#)を使用：

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000/anthropic
```

LiteLLMを通じたAmazon Bedrock

[パススルーエンドポイント](#)を使用：

```
export ANTHROPIC_BEDROCK_BASE_URL=https://litellm-server:4000/bedrock
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1
export CLAUDE_CODE_USE_BEDROCK=1
```

LiteLLMを通じたGoogle Vertex AI

[パススルーエンドポイント](#)を使用：

```
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
```

詳細については、[LiteLLMドキュメント](#)を参照してください。

追加リソース

- [LiteLLMドキュメント](#)
- [Claude Code設定](#)
- [エンタープライズネットワーク設定](#)
- [サードパーティ統合の概要](#)

エンタープライズネットワーク設定

プロキシサーバー、カスタム認証局（CA）、相互 Transport Layer Security（mTLS）認証を使用して、エンタープライズ環境向けに Claude Code を設定します。

Claude Code は、環境変数を通じてさまざまなエンタープライズネットワークおよびセキュリティ設定をサポートしています。これには、企業プロキシサーバーを経由したトラフィックのルーティング、カスタム認証局（CA）の信頼、およびセキュリティ強化のための相互 Transport Layer Security（mTLS）証明書による認証が含まれます。

Note:

このページに表示されているすべての環境変数は、`settings.json` でも設定できます。

プロキシ設定

環境変数

Claude Code は標準的なプロキシ環境変数に対応しています。

```
## HTTPS プロキシ (推奨)
export HTTPS_PROXY=https://proxy.example.com:8080

## HTTP プロキシ (HTTPS が利用できない場合)
export HTTP_PROXY=http://proxy.example.com:8080

## 特定のリクエストのプロキシをバイパス - スペース区切り形式
export NO_PROXY="localhost 192.168.1.1 example.com .example.com"
## 特定のリクエストのプロキシをバイパス - カンマ区切り形式
export NO_PROXY="localhost,192.168.1.1,example.com,.example.com"
## すべてのリクエストのプロキシをバイパス
export NO_PROXY="*"
```

Note:

Claude Code は SOCKS プロキシをサポートしていません。

基本認証

プロキシが基本認証を必要とする場合は、プロキシ URL に認証情報を含めます。

```
export HTTPS_PROXY=http://username:password@proxy.example.com:8080
```

Warning:

スクリプトにパスワードをハードコーディングすることは避けてください。代わりに環境変数またはセキュアな認証情報ストレージを使用してください。

Tip:

高度な認証（NTLM、Kerberos など）が必要なプロキシの場合は、認証方法をサポートする LLM Gateway サービスの使用を検討してください。

カスタム CA 証明書

エンタープライズ環境で HTTPS 接続用のカスタム CA を使用している場合（プロキシ経由でも直接 API アクセスでも）、Claude Code をそれらを信頼するように設定します。

```
export NODE_EXTRA_CA_CERTS=/path/to/ca-cert.pem
```

mTLS 認証

クライアント証明書認証が必要なエンタープライズ環境の場合：

認証用のクライアント証明書

```
export CLAUDE_CODE_CLIENT_CERT=/path/to/client-cert.pem
```

クライアント秘密鍵

```
export CLAUDE_CODE_CLIENT_KEY=/path/to/client-key.pem
```

オプション：暗号化された秘密鍵のパスフレーズ

```
export CLAUDE_CODE_CLIENT_KEY_PASSPHRASE="your-passphrase"
```

ネットワークアクセス要件

Claude Code は以下の URL へのアクセスが必要です。

- `api.anthropic.com` : Claude API エンドポイント
- `claude.ai` : claude.ai アカウント用の認証
- `platform.claude.com` : Anthropic Console アカウント用の認証

これらの URL がプロキシ設定とファイアウォールルールでホワイトリストに登録されていることを確認してください。これは、Claude Code をコンテナ化された環境または制限されたネットワーク環境で使用する場合に特に重要です。

その他のリソース

- [Claude Code 設定](#)
- [環境変数リファレンス](#)
- [トラブルシューティングガイド](#)

Part 12: Environment Setup

開発コンテナ

一貫性のある安全な環境が必要なチーム向けのClaude Code開発コンテナについて学びます。

リファレンス[devcontainerセットアップ](#)と関連する[Dockerfile](#)は、そのまま使用することも、ニーズに合わせてカスタマイズすることもできる事前設定済みの開発コンテナを提供します。このdevcontainerはVisual Studio Code [Dev Containers拡張機能](#)および同様のツールと連携します。

コンテナの強化されたセキュリティ対策（分離とファイアウォールルール）により、`claude --dangerously-skip-permissions` を実行して権限プロンプトをバイパスし、無人操作を行うことができます。

Warning:

devcontainerは実質的な保護を提供していますが、すべての攻撃に完全に耐性のあるシステムはありません。 `--dangerously-skip-permissions` で実行する場合、devcontainerはClaude Codeの認証情報を含むdevcontainer内でアクセス可能なものを悪意のあるプロジェクトが流出させることを防ぎません。信頼できるリポジトリで開発する場合にのみdevcontainerを使用することをお勧めします。常に良好なセキュリティプラクティスを維持し、Claudeのアクティビティを監視してください。

主な機能

- **本番環境対応のNode.js:** Node.js 20上に構築され、必須の開発依存関係を含む
- **設計によるセキュリティ:** 必要なサービスのみへのネットワークアクセスを制限するカスタムファイアウォール
- **開発者向けツール:** git、生産性向上機能付きZSH、fzfなどを含む
- **シームレスなVS Code統合:** 事前設定済みの拡張機能と最適化された設定
- **セッション永続性:** コンテナ再起動間でコマンド履歴と設定を保持
- **どこでも動作:** macOS、Windows、Linuxの開発環境と互換性あり

4ステップで始める

1. VS Codeとリモート - コンテナ拡張機能をインストール
2. [Claude Codeのリファレンス実装](#)リポジトリをクローン
3. VS Codeでリポジトリを開く
4. プロンプトが表示されたら、「コンテナで再度開く」をクリック（またはコマンドパレットを使用: Cmd+Shift+P → 「Remote-Containers: Reopen in Container」）

設定の詳細

devcontainerセットアップは3つの主要なコンポーネントで構成されています:

- [devcontainer.json](#): コンテナ設定、拡張機能、ボリュームマウントを制御
- [Dockerfile](#): コンテナイメージとインストール済みツールを定義
- [init-firewall.sh](#): ネットワークセキュリティルールを確立

セキュリティ機能

コンテナはファイアウォール設定を使用した多層防御セキュリティアプローチを実装しています:

- **正確なアクセス制御**: ホワイトリストに登録されたドメインのみへのアウトバウンド接続を制限（npmレジストリ、GitHub、Claude APIなど）
- **許可されたアウトバウンド接続**: ファイアウォールはアウトバウンドDNSおよびSSH接続を許可
- **デフォルト拒否ポリシー**: その他すべての外部ネットワークアクセスをブロック
- **スタートアップ検証**: コンテナ初期化時にファイアウォールルールを検証
- **分離**: メインシステムから分離された安全な開発環境を作成

カスタマイズオプション

devcontainer設定はニーズに適応するように設計されています:

- ワークフローに基づいてVS Code拡張機能を追加または削除
- 異なるハードウェア環境のリソース割り当てを変更
- ネットワークアクセス権限を調整
- シェル設定と開発者ツールをカスタマイズ

使用例

セキュアなクライアント作業

devcontainerを使用して異なるクライアントプロジェクトを分離し、コードと認証情報が環境間で混在しないようにします。

チームオンボーディング

新しいチームメンバーは数分で完全に設定された開発環境を取得でき、すべての必要なツールと設定が事前にインストールされています。

一貫したCI/CD環境

devcontainer設定をCI/CDパイプラインにミラーリングして、開発環境と本番環境が一致することを確認します。

関連リソース

- [VS Code devcontainersドキュメント](#)
- [Claude Codeセキュリティベストプラクティス](#)
- [エンタープライズネットワーク設定](#)

ターミナルセットアップを最適化する

Claude Codeはターミナルが適切に設定されているときに最適に機能します。これらのガイドラインに従って、エクスペリエンスを最適化してください。

テーマと外観

Claudeはターミナルのテーマを制御することはできません。これはターミナルアプリケーションによって処理されます。 `/config` コマンドを使用して、いつでもClaude Codeのテーマをターミナルに合わせることができます。

Claude Codeインターフェース自体のさらなるカスタマイズについては、[カスタムステータスライン](#)を設定して、ターミナルの下部に現在のモデル、作業ディレクトリ、またはgitブランチなどのコンテキスト情報を表示することができます。

改行

Claude Codeに改行を入力するためのいくつかのオプションがあります：

- **クイックエスケープ**：`\` の後にEnterキーを押して改行を作成します
- **Shift+Enter**：iTerm2、WezTerm、Ghostty、およびKittyではそのまま機能します
- **キーボードショートカット**：他のターミナルで改行を挿入するためのキーバインディングを設定します

他のターミナルでShift+Enterを設定する

Claude Code内で `/terminal-setup` を実行して、VS Code、Alacritty、Zed、およびWarpのShift+Enterを自動的に設定します。

Note:

`/terminal-setup` コマンドは、手動設定が必要なターミナルにのみ表示されます。iTerm2、WezTerm、Ghostty、またはKittyを使用している場合、Shift+Enterはすでにネイティブに機能するため、このコマンドは表示されません。

Option+Enter (VS Code、iTerm2、またはmacOS Terminal.app) を設定する

Mac Terminal.appの場合：

1. 設定 → プロファイル → キーボードを開く
2. 「OptionキーをMetaキーとして使用」をチェックする

iTerm2およびVS Codeターミナルの場合：

1. 設定 → プロファイル → キーを開く
2. 「一般」で、左右のOptionキーを「Esc+」に設定する

通知設定

適切な通知設定でClaudeがタスクを完了したときを見逃さないようにしてください：

iTerm 2システム通知

iTerm 2でタスク完了時にアラートを表示する場合：

1. iTerm 2の環境設定を開く
2. プロファイル → ターミナルに移動する
3. 「ベルを消音」を有効にし、フィルターアラート → 「エスケープシーケンスで生成されたアラートを送信」を選択する
4. 希望する通知遅延を設定する

これらの通知はiTerm 2に固有であり、デフォルトのmacOS Terminalでは利用できないことに注意してください。

カスタム通知フック

高度な通知処理については、[通知フック](#)を作成して独自のロジックを実行できます。

大量入力の処理

広範なコードまたは長い指示を扱う場合：

- **直接貼り付けを避ける**：Claude Codeは非常に長い貼り付けコンテンツで問題が発生する可能性があります
- **ファイルベースのワークフローを使用する**：コンテンツをファイルに書き込み、Claudeに読み込むよう依頼します
- **VS Codeの制限に注意する**：VS Codeターミナルは特に長い貼り付けを切り詰める傾向があります

Vimモード

Claude Codeは `/vim` で有効にするか、`/config` 経由で設定できるVimキーバインディングのサブセットをサポートしています。

サポートされているサブセットには以下が含まれます：

- モード切り替え：`Esc` (NORMAL) 、`i/I` 、`a/A` 、`o/O` (INSERT)

- ナビゲーション： `h/j/k/l`、`w/e/b`、`0/$/^`、`gg/G`、`f/F/t/T`（`;`、`,` リピート付き）
- 編集： `x`、`dw/de/db/dd/D`、`cw/ce/cb/cc/C`、`.`（リピート）
- ヤンク/ペースト： `yy/Y`、`yw/ye/yb`、`p/P`
- テキストオブジェクト： `iw/aw`、`iW/aW`、`i"/a"`、`i'/a'`、`i(/a(`、`i[/a[`、`i{/a{`
- インデント： `>>` / `<<`
- 行操作： `J`（行を結合）

完全なリファレンスについては、[インタラクティブモード](#)を参照してください。

ステータスラインをカスタマイズする

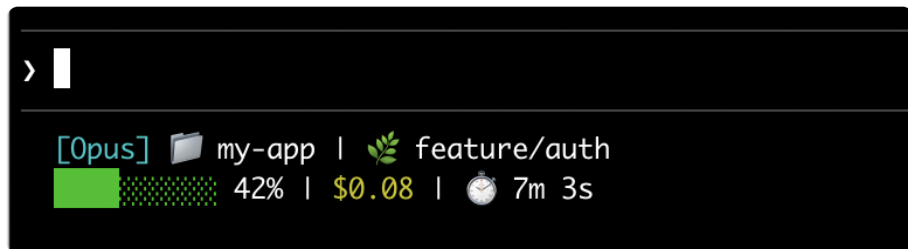
Claude Code でコンテキストウィンドウの使用状況、コスト、git ステータスを監視するカスタムステータスバーを設定します

ステータスラインは Claude Code の下部にあるカスタマイズ可能なバーで、設定したシェルスクリプトを実行します。stdin 経由で JSON セッションデータを受け取り、スクリプトが出力したものを表示し、コンテキスト使用状況、コスト、git ステータス、またはその他の追跡したい情報を一目で確認できる永続的なビューを提供します。

ステータスラインは以下の場合に便利です：

- ・作業中にコンテキストウィンドウの使用状況を監視したい
- ・セッションコストを追跡する必要がある
- ・複数のセッション間で作業し、それらを区別する必要がある
- ・git ブランチとステータスを常に表示したい

以下は、最初の行に git 情報を表示し、2 番目の行にカラーコード化されたコンテキストバーを表示する [複数行ステータスライン](#) の例です。



モデル名、ディレクトリ、git ブランチを最初の行に表示し、コンテキスト使用状況プログレスバー、コスト、期間を 2 番目の行に表示する複数行ステータスライン

このページでは、[基本的なステータスラインの設定](#) について説明し、Claude Code からスクリプトへの [データフロー](#) について説明し、[表示できるすべてのフィールド](#) をリストアップし、git ステータス、コスト追跡、プログレスバーなどの一般的なパターンの [すぐ使える例](#) を提供します。

ステータスラインを設定する

`/statusline` [コマンド](#) を使用して Claude Code にスクリプトを生成させるか、[手動でスクリプトを作成](#) して設定に追加します。

/statusline コマンドを使用する

`/statusline` コマンドは、表示したい内容を説明する自然言語の指示を受け入れます。Claude Code は `~/.claude/` にスクリプトファイルを生成し、設定を自動的に更新します：

```
/statusline show model name and context percentage with a progress bar
```

ステータスラインを手動で設定する

ユーザー設定 (`~/.claude/settings.json`、`~` はホームディレクトリ) または [プロジェクト設定](#) に `statusLine` フィールドを追加します。 `type` を `"command"` に設定し、`command` をスクリプトパスまたはインラインシェルコマンドに指定します。スクリプト作成の完全なチュートリアルについては、[ステータスラインをステップバイステップで構築する](#) を参照してください。

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh",
    "padding": 2
  }
}
```

`command` フィールドはシェルで実行されるため、スクリプトファイルの代わりにインラインコマンドを使用することもできます。この例では `jq` を使用して JSON 入力を解析し、モデル名とコンテキスト割合を表示します：

```
{
  "statusLine": {
    "type": "command",
    "command": "jq -r \"[\\\"(.model.display_name)\\\"] \\\"(.context_window.used_percent age // 0)% context\\\"\""
  }
}
```


オプションの `padding` フィールドは、ステータスラインコンテンツに追加の水平スペース（文字単位）を追加します。デフォルトは `0` です。このパディングはインターフェイスの組み込みスペースに加えて追加されるため、ターミナルエッジからの絶対距離ではなく相対的なインデントを制御します。

ステータスラインを無効にする

`/statusline` を実行し、ステータスラインを削除またはクリアするよう指示します（例： `/statusline delete` 、 `/statusline clear` 、 `/statusline remove it` ）。

settings.json から `statusLine` フィールドを手動で削除することもできます。

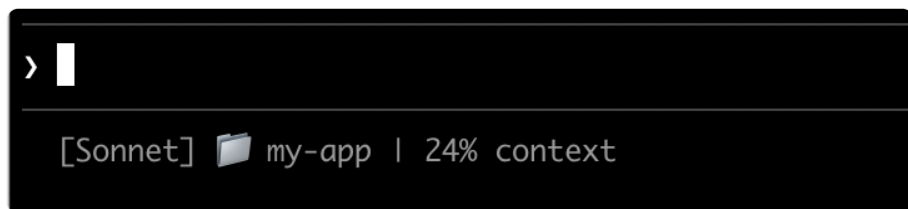
ステータスラインをステップバイステップで構築する

このチュートリアルでは、現在のモデル、作業ディレクトリ、コンテキストウィンドウ使用率を表示するステータスラインを手動で作成することで、内部で何が起こっているかを示します。

Note:

`/statusline` を実行して、表示したい内容を説明すると、これらすべてが自動的に設定されます。

これらの例では Bash スクリプトを使用しており、macOS と Linux で動作します。Windows では、[Windows 設定](#) を参照して PowerShell と Git Bash の例を確認してください。



モデル名、ディレクトリ、コンテキスト割合を表示するステータスライン

Step 1: JSON を読み取り、出力を出力するスクリプトを作成する

Claude Code は stdin 経由で JSON データをスクリプトに送信します。このスクリプトは `jq`（コマンドラインの JSON パーサーで、インストールが必要な場合があります）を使用して、モデル名、ディレクトリ、コンテキスト割合を抽出し、フォーマットされた行を出力します。

これを `~/claude/statusline.sh` に保存します（`~` はホームディレクトリ、macOS では `/Users/username`、Linux では `/home/username` など）：

```
#!/bin/bash
## Claude Code がstdin に送信する JSON データを読み取る
input=$(cat)

## jq を使用してフィールドを抽出する
MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
## "// 0" はフィールドが null の場合のフォールバックを提供します
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## ステータスラインを出力します - ${DIR##*/} はフォルダ名のみを抽出します
echo "[${MODEL}] 📁 ${DIR##*/} | ${PCT}% context"
```

Step 2: 実行可能にする

スクリプトを実行可能にマークして、シェルが実行できるようにします：

```
chmod +x ~/claude/statusline.sh
```

Step 3: 設定に追加する

Claude Code にスクリプトをステータスラインとして実行するよう指示します。この設定を `~/claude/settings.json` に追加します。これは `type` を `"command"`（「このシェルコマンドを実行する」という意味）に設定し、`command` をスクリプトに指定します：

```
{
  "statusLine": {
    "type": "command",
    "command": "~/claude/statusline.sh"
  }
}
```

ステータスラインはインターフェイスの下部に表示されます。設定は自動的に再読み込みされますが、Claude Code との次の相互作用まで変更は表示されません。

ステータスラインの仕組み

Claude Code はスクリプトを実行し、stdin 経由で [JSON セッションデータ](#) をパイプします。スクリプトは JSON を読み取り、必要なものを抽出し、stdout にテキストを出力します。Claude Code はスクリプトが出力したものを表示します。

更新のタイミング

スクリプトは新しいアシスタントメッセージの後、パーミッションモードが変更されたとき、または vim モードが切り替わったときに実行されます。更新は 300ms でデバウンスされます。つまり、急速な変更がバッチ処理され、スクリプトは物事が落ち着いたら 1 回実行されます。スクリプトがまだ実行中に新しい更新がトリガーされた場合、実行中の実行はキャンセルされます。スクリプトを編集した場合、Claude Code との次の相互作用がトリガーされるまで変更は表示されません。

スクリプトが出力できるもの

- ・ **複数行**：各 `echo` または `print` ステートメントは別の行として表示されます。[複数行の例](#) を参照してください。
- ・ **色**：`\033[32m` のような [ANSI エスケープコード](#) を使用して緑色を表示します（ターミナルがサポートしている必要があります）。[git ステータスの例](#) を参照してください。
- ・ **リンク**：[OSC 8 エスケープシーケンス](#) を使用してテキストをクリック可能にします（macOS では Cmd+クリック、Windows/Linux では Ctrl+クリック）。iTerm2、Kitty、WezTerm などのハイパーリンクをサポートするターミナルが必要です。[クリック可能なリンクの例](#) を参照してください。

Note:

ステータスラインはローカルで実行され、API トークンを消費しません。オートコンプリート提案、ヘルプメニュー、パーミッションプロンプトなど、特定の UI 相互作用中は一時的に非表示になります。

利用可能なデータ

Claude Code は以下の JSON フィールドを stdin 経由でスクリプトに送信します：

フィールド	説明
<code>model.id</code> 、 <code>model.display_name</code>	現在のモデル識別子と表示名

フィールド	説明
<code>cwd</code> 、 <code>workspace.current_dir</code>	現在の作業ディレクトリ。両方のフィールドに同じ値が含まれます。 <code>workspace.current_dir</code> は <code>workspace.project_dir</code> との一貫性のために推奨されます。
<code>workspace.project_dir</code>	Claude Code が起動されたディレクトリ。セッション中に作業ディレクトリが変更された場合、 <code>cwd</code> と異なる場合があります
<code>cost.total_cost_usd</code>	USD でのセッションの総コスト
<code>cost.total_duration_ms</code>	セッション開始からの総経過時間（ミリ秒）
<code>cost.total_api_duration_ms</code>	API レスポンスを待つのに費やされた総時間（ミリ秒）
<code>cost.total_lines_added</code> 、 <code>cost.total_lines_removed</code>	変更されたコード行
<code>context_window.total_input_tokens</code> 、 <code>context_window.total_output_tokens</code>	セッション全体の累積トークン数
<code>context_window.context_window_size</code>	トークン単位の最大コンテキストウィンドウサイズ。デフォルトは 200000、拡張コンテキストを持つモデルの場合は 1000000
<code>context_window.used_percentage</code>	事前計算されたコンテキストウィンドウ使用率
<code>context_window.remaining_percentage</code>	事前計算されたコンテキストウィンドウ残り率
<code>context_window.current_usage</code>	最後の API 呼び出しからのトークン数。 コンテキストウィンドウフィールド で説明されています
<code>exceeds_200k_tokens</code>	最新の API レスポンスからの総トークン数（入力、キャッシュ、出力トークンの組み合わせ）が 200k を超えるかどうか。これは実際のコンテキストウィンドウサイズに関係なく固定閾値です。
<code>session_id</code>	一意のセッション識別子
<code>transcript_path</code>	会話トランスクリプトファイルへのパス

フィールド	説明
<code>version</code>	Claude Code バージョン
<code>output_style.name</code>	現在の出力スタイルの名前
<code>vim.mode</code>	<code>vim モード</code> が有効な場合の現在の vim モード（ <code>NORMAL</code> または <code>INSERT</code> ）
<code>agent.name</code>	<code>--agent</code> フラグまたはエージェント設定が設定されている場合のエージェント名
<code>worktree.name</code>	アクティブな worktree の名前。 <code>--worktree</code> セッション中にのみ存在します
<code>worktree.path</code>	worktree ディレクトリへの絶対パス
<code>worktree.branch</code>	worktree の git ブランチ名（例： <code>"worktree-my-feature"</code> ）。フックベースの worktree では存在しません
<code>worktree.original_cwd</code>	worktree に入る前に Claude がいたディレクトリ
<code>worktree.original_branch</code>	worktree に入る前にチェックアウトされた git ブランチ。フックベースの worktree では存在しません

ステータスラインコマンドは stdin 経由でこの JSON 構造を受け取ります：

```
{
  "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"
}
}
```

存在しない可能性があるフィールド（JSON に存在しない）：

- **vim**：vim モードが有効な場合にのみ表示されます
- **agent**：`--agent` フラグまたはエージェント設定が設定されている場合にのみ表示されます
- **worktree**：`--worktree` セッション中にのみ表示されます。存在する場合、**branch** と **original_branch** もフックベースの worktree では存在しない可能性があります

null である可能性があるフィールド：

- **context_window.current_usage**：セッションの最初の API 呼び出しの前は **null**
- **context_window.used_percentage**、**context_window.remaining_percentage**：セッションの早期段階では **null** である可能性があります

スクリプトで条件付きアクセスと null 値のフォールバックデフォルトを使用して、欠落しているフィールドを処理します。

コンテキストウィンドウフィールド

context_window オブジェクトは、コンテキスト使用状況を追跡する 2 つの方法を提供します：

- **累積合計**（**total_input_tokens**、**total_output_tokens**）：セッション全体のすべてのトークンの合計。総消費量の追跡に便利です
- **現在の使用状況**（**current_usage**）：最新の API 呼び出しからのトークン数。実際のコンテキスト状態を反映しているため、正確なコンテキスト割合に使用します

`current_usage` オブジェクトには以下が含まれます：

- `input_tokens`：現在のコンテキストの入力トークン
- `output_tokens`：生成された出力トークン
- `cache_creation_input_tokens`：キャッシュに書き込まれたトークン
- `cache_read_input_tokens`：キャッシュから読み取られたトークン

`used_percentage` フィールドは入力トークンのみから計算されます：`input_tokens + cache_creation_input_tokens + cache_read_input_tokens` ÷ `output_tokens` は含まれません。

`current_usage` から手動でコンテキスト割合を計算する場合は、`used_percentage` と一致させるために同じ入力のための式を使用します。

`current_usage` オブジェクトはセッションの最初の API 呼び出しの前は `null` です。

例

これらの例は一般的なステータスラインパターンを示しています。任意の例を使用するには：

1. スクリプトを `~/.claude/statusline.sh`（または `.py` / `.js`）などのファイルに保存します
2. 実行可能にします：`chmod +x ~/.claude/statusline.sh`
3. [設定](#) にパスを追加します

Bash の例では `jq` を使用して JSON を解析します。Python と Node.js には組み込みの JSON 解析があります。

コンテキストウィンドウ使用状況

現在のモデルとコンテキストウィンドウ使用状況を視覚的なプログレスバーで表示します。各スクリプトは stdin から JSON を読み取り、`used_percentage` フィールドを抽出し、塗りつぶされたブロック (■) が使用状況を表す 10 文字のバーを構築します：



モデル名とパーセンテージ付きのプログレスバーを表示するステータスライン

```
#!/bin/bash
## stdin 全体を変数に読み取る
input=$(cat)

## jq でフィールドを抽出します。"// 0" はnull のフォールバックを提供します
MODEL=$(echo "$input" | jq -r '.model.display_name')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## プログレスバーを構築します：printf はスペースを作成し、tr はブロックに置き換えます
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 はstdin を 1 ステップで読み取り、解析します
data = json.load(sys.stdin)
model = data['model']['display_name']
## "or 0" は null 値を処理します
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)

## 文字列乗算がバーを構築します
filled = pct * 10 // 100
bar = '█' * filled + '░' * (10 - filled)

print(f"[{model}] {bar} {pct}%")
```

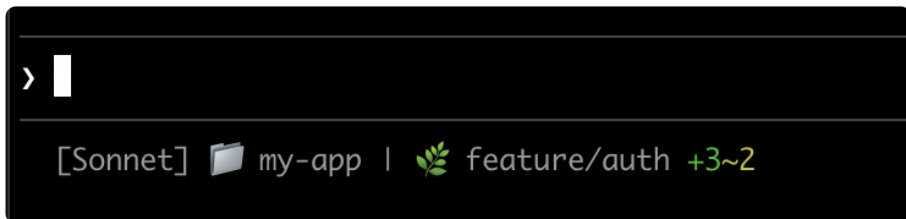
```
#!/usr/bin/env node
// Node.js はイベントで stdin を非同期に読み取ります
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  // オプショナルチェーン (?) は null フィールドを安全に処理します
  const pct = Math.floor(data.context_window?.used_percentage || 0);

  // String.repeat() がバーを構築します
  const filled = Math.floor(pct * 10 / 100);
  const bar = '█'.repeat(filled) + '░'.repeat(10 - filled);

  console.log(`[${model}] ${bar} ${pct}%`);
});
```

git ステータスと色

ステージングされたファイルと変更されたファイルのカラーコード化されたインジケータを使用して git ブランチを表示します。このスクリプトはターミナルの色に [ANSI エスケープコード](#) を使用します： `\033[32m` は緑、 `\033[33m` は黄、 `\033[0m` はデフォルトにリセットします。



モデル、ディレクトリ、git ブランチ、ステージングされたファイルと変更されたファイルのカラーコード化されたインジケータを表示するステータスライン

各スクリプトは現在のディレクトリが git リポジトリであるかどうかを確認し、ステージングされたファイルと変更されたファイルをカウントし、カラーコード化されたインジケータを表示します：

```
#!/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}`);
  }
});
```

コストと期間の追跡

セッションの API コストと経過時間を追跡します。 `cost.total_cost_usd` フィールドは現在のセッションのすべての API 呼び出しのコストを累積します。

`cost.total_duration_ms` フィールドはセッション開始からの総経過時間を測定し、

`cost.total_api_duration_ms` は API レスポンスを待つのに費やされた時間のみを追跡します。

各スクリプトはコストを通貨としてフォーマットし、ミリ秒を分と秒に変換します：

> █

[Sonnet] 💰 \$0.08 | 🕒 7m 3s

モデル名、セッションコスト、期間を表示するステータスライン

```
#!/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")
```

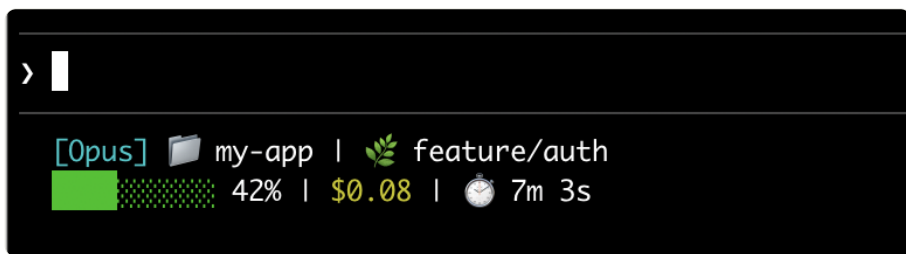
```
#!/usr/bin/env node
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const cost = data.cost?.total_cost_usd || 0;
  const durationMs = data.cost?.total_duration_ms || 0;

  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`);
});
```

複数行を表示する

スクリプトは複数の行を出力して、より豊かなディスプレイを作成できます。各 `echo` ステートメントはステータス領域に別の行を生成します。



最初の行にモデル名、ディレクトリ、git ブランチを表示し、2 番目の行にコンテキスト使用状況プログレスバー、コスト、期間を表示する複数行ステータスライン

この例は複数のテクニックを組み合わせています：閾値ベースの色（70% 未満は緑、70～89% は黄、90% 以上は赤）、プログレスバー、git ブランチ情報。各 `print` または `echo` ステートメントは別の行を作成します：

```
#!/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'

## コンテキスト使用状況に基づいてバーの色を選択します
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,
e, 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);

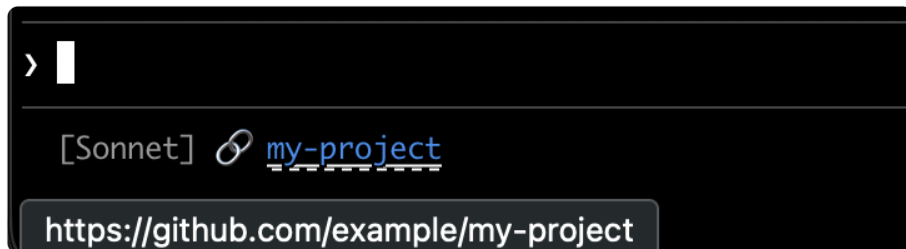
  const mins = Math.floor(durationMs / 60000);
  const secs = Math.floor((durationMs % 60000) / 1000);

  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`);
});
```

クリック可能なリンク

この例は GitHub リポジトリへのクリック可能なリンクを作成します。git リモート URL を読み取り、SSH 形式を `sed` で HTTPS に変換し、リポジトリ名を OSC 8 エスケープコードでラップします。Cmd (macOS) または Ctrl (Windows/Linux) を押しながらクリックして、ブラウザでリンクを開きます。



GitHub リポジトリへのクリック可能なリンクを表示するステータスライン

各スクリプトは git リモート URL を取得し、SSH 形式を HTTPS に変換し、リポジトリ名を OSC 8 エスケープコードでラップします。Bash バージョンは `printf '%b'` を使用します。これはバックスラッシュエスケープを `echo -e` よりもシェル全体でより確実に解釈します：

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')

## git SSH URL を HTTPS に変換します
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 "$REMOTE")
    # OSC 8 形式: \e]8;;URL\a その後 TEXT その後 \e]8;;\a
    # printf %b はシェル全体でエスケープシーケンスを確実に解釈します
    printf '%b' "[${MODEL}] 
```

```
#!/usr/bin/env python3
import json, sys, subprocess, re, os

data = json.load(sys.stdin)
model = data['model']['display_name']

## git リモート URL を取得します
try:
    remote = subprocess.check_output(
        ['git', 'remote', 'get-url', 'origin'],
        stderr=subprocess.DEVNULL, text=True
    ).strip()
    # SSH を HTTPS 形式に変換します
    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 エスケープシーケンス
    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();
    // SSH を HTTPS 形式に変換します
    remote = remote.replace(/^git@github\.com:/, 'https://github.com/').replac
e(/\.git$/, '');
    const repoName = path.basename(remote);
    // OSC 8 エスケープシーケンス
    const link = `\x1b]8;;${remote}\x07${repoName}\x1b]8;;\x07`;
    console.log(`[${model}] 🔗 ${link}`);
  } catch {
    console.log(`[${model}]`);
  }
});
```

高コストな操作をキャッシュする

ステータスラインスクリプトはアクティブなセッション中に頻繁に実行されます。`git status` や `git diff` などのコマンドは、特に大規模なリポジトリでは遅くなる可能性があります。この例は git 情報を一時ファイルにキャッシュし、5 秒ごとにのみ更新します。

`/tmp/statusline-git-cache` のような安定した固定ファイル名をキャッシュファイルに使用します。各ステータスライン呼び出しは新しいプロセスとして実行されるため、`$`、`os.getpid()`、`process.pid` のようなプロセススペースの識別子は毎回異なる値を生成し、キャッシュは再利用されません。

各スクリプトは git コマンドを実行する前に、キャッシュファイルが欠落しているか 5 秒より古いかを確認します：

```
#!/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 # 秒

cache_is_stale() {
    [ ! -f "$CACHE_FILE" ] || \
    # stat -f %m は macOS、stat -c %Y は 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 # 秒

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; // 秒

  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 設定

Windows では、Claude Code はステータスラインコマンドを Git Bash 経由で実行します。そのシェルから PowerShell を呼び出すことができます：

```

{
  "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]"
}

```

または Bash スクリプトを直接実行します：

```
{
  "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]"
```

ヒント

- **モック入力でテストする**：`echo '{"model": {"display_name": "Opus"}, "context_window": {"used_percentage": 25}}' | ./statusline.sh`
- **出力を短く保つ**：ステータスバーの幅は限られているため、長い出力は切り詰められたり、不適切にラップされたりする可能性があります
- **遅い操作をキャッシュする**：スクリプトはアクティブなセッション中に頻繁に実行されるため、`git status` などのコマンドはラグを引き起こす可能性があります。これを処理する方法については、[キャッシング例](#)を参照してください。

[ccstatusline](#) や [starship-claude](#) などのコミュニティプロジェクトは、テーマと追加機能を備えた事前構築設定を提供します。

トラブルシューティング

ステータスラインが表示されない

- スクリプトが実行可能であることを確認します：
`chmod +x ~/.claude/statusline.sh`
- スクリプトが stdout に出力し、stderr に出力していないことを確認します
- スクリプトを手動で実行して、出力を生成することを確認します
- `disableAllHooks` が設定で `true` に設定されている場合、ステータスラインも無効になります。この設定を削除するか、`false` に設定して再度有効にします。

- `claude --debug` を実行して、セッションの最初のステータスラインの呼び出しからの終了コードと `stderr` をログに記録します
- Claude に設定ファイルを読み取り、`statusLine` コマンドを直接実行するよう依頼して、エラーを表示します

ステータスラインが `--` または空の値を表示する

- フィールドは最初の API レスポンスが完了する前は `null` である可能性があります
- スクリプトで `null` 値を `// 0` のようなフォールバックで処理します
- 複数のメッセージの後でも値が空のままの場合は、Claude Code を再起動します

コンテキスト割合が予期しない値を表示する

- 累積合計ではなく、正確なコンテキスト状態に `used_percentage` を使用します
- `total_input_tokens` と `total_output_tokens` はセッション全体で累積され、コンテキストウィンドウサイズを超える可能性があります
- コンテキスト割合は各計算のタイミングが異なるため、`/context` 出力と異なる場合があります

OSC 8 リンクがクリック可能でない

- ターミナルが OSC 8 ハイパーリンクをサポートしていることを確認します (iTerm2、Kitty、WezTerm)
- Terminal.app はクリック可能なリンクをサポートしていません
- SSH と tmux セッションは設定に応じて OSC シーケンスをストリップする可能性があります
- エスケープシーケンスが `\e[8;;` のようなりテラルテキストとして表示される場合は、`echo -e` の代わりに `printf '%b'` を使用して、より確実なエスケープ処理を行います

エスケープシーケンスでの表示の不具合

- 複雑なエスケープシーケンス (ANSI 色、OSC 8 リンク) は、他の UI 更新と重複する場合、出力が破損する可能性があります
- 破損したテキストが表示される場合は、スクリプトをプレーンテキスト出力に簡略化してみてください
- エスケープコード付きの複数行ステータスラインは、単一行のプレーンテキストよりもレンダリングの問題が発生しやすくなります

スクリプトエラーまたはハング

- ゼロ以外のコードで終了するか、出力を生成しないスクリプトは、ステータスラインを空白にします

- 遅いスクリプトは、完了するまでステータスラインの更新をブロックします。古い出力を避けるため、スクリプトを高速に保ちます。
- 遅いスクリプトの実行中に新しい更新がトリガーされた場合、実行中のスクリプトはキャンセルされます
- 設定する前に、モック入力を使用してスクリプトを独立してテストします

通知がステータスラインの行を共有する

- MCP サーバーエラー、自動更新、トークン警告などのシステム通知は、ステータスラインと同じ行の右側に表示されます
- 詳細モードを有効にすると、この領域にトークンカウンターが追加されます
- 狭いターミナルでは、これらの通知がステータスラインの出力を切り詰める可能性があります

Part 13: Troubleshooting & Changelog

トラブルシューティング

Claude Code のインストールと使用に関する一般的な問題の解決策を見つけます。

インストール問題のトラブルシューティング

Tip:

ターミナルを完全にスキップしたい場合は、[Claude Code デスクトップアプリ](#)を使用して、グラフィカルインターフェイスを通じて Claude Code をインストールして使用できます。
[macOS](#)または[Windows](#)用にダウンロードして、コマンドラインセットアップなしでコーディングを開始できます。

表示されているエラーメッセージまたは症状を見つけてください：

表示内容	解決策
<code>command not found: claude</code> または <code>'claude' is not recognized</code>	PATH を修正する
<code>syntax error near unexpected token '<'</code>	インストールスクリプトが HTML を返す
<code>curl: (56) Failure writing output to destination</code>	スクリプトをダウンロードしてから実行する
Linux でのインストール中に <code>Killed</code>	低メモリサーバーにスワップスペースを追加する
<code>TLS connect error</code> または <code>SSL/TLS secure channel</code>	CA 証明書を更新する
<code>Failed to fetch version</code> またはダウンロードサーバーに到達できない	ネットワークとプロキシ設定を確認する

表示内容	解決策
<code>irm is not recognized</code> または <code>&& is not valid</code>	シェルに適切なコマンドを使用する
<code>Claude Code on Windows requires git-bash</code>	Git Bash をインストールまたは設定する
<code>Error loading shared library</code>	システムに適切なバイナリバリエーションをインストールする
Linux での <code>Illegal instruction</code>	アーキテクチャの不一致
macOS での <code>dyld: cannot load</code> または <code>Abort trap</code>	バイナリの互換性
<code>Invoke-Expression: Missing argument in parameter list</code>	インストールスクリプトが HTML を返す
<code>App unavailable in region</code>	Claude Code はお客様の国では利用できません。 サポートされている国 を参照してください。
<code>unable to get local issuer certificate</code>	企業 CA 証明書を設定する
<code>OAuth error</code> または <code>403 Forbidden</code>	認証を修正する

問題がリストに記載されていない場合は、これらの診断手順を実行してください。

インストール問題のデバッグ

ネットワーク接続を確認する

インストーラーは storage.googleapis.com からダウンロードします。到達可能であることを確認してください：

```
curl -sI https://storage.googleapis.com
```

これが失敗する場合、ネットワークが接続をブロックしている可能性があります。一般的な原因：

- Google Cloud Storage をブロックしている企業ファイアウォールまたはプロキシ
- 地域的なネットワーク制限：VPN または別のネットワークを試してください
- TLS/SSL の問題：システムの CA 証明書を更新するか、`HTTPS_PROXY` が設定されているかどうかを確認してください

企業プロキシの背後にいる場合は、インストール前に `HTTPS_PROXY` と `HTTP_PROXY` をプロキシのアドレスに設定してください。プロキシ URL がわからない場合は IT チームに問い合わせるか、ブラウザのプロキシ設定を確認してください。

この例は両方のプロキシ変数を設定してから、プロキシを通じてインストーラーを実行します：

```
export HTTP_PROXY=http://proxy.example.com:8080
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

PATH を確認する

インストールが成功したが、`claude` を実行するときに `command not found` または `not recognized` エラーが表示される場合、インストールディレクトリが PATH に含まれていません。シェルは PATH にリストされているディレクトリ内のプログラムを検索し、インストーラーは macOS/Linux では `~/local/bin/claude` に、Windows では `%USERPROFILE%\local\bin\claude.exe` に `claude` を配置します。

PATH にインストールディレクトリが含まれているかどうかを確認するには、PATH エントリをリストして `local/bin` でフィルタリングしてください：

macOS/Linux

```
echo $PATH | tr ':' '\n' | grep local/bin
```

出力がない場合、ディレクトリが見つかりません。シェル設定に追加してください：

```
## Zsh (macOS デフォルト)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc

## Bash (Linux デフォルト)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

または、ターミナルを閉じて再度開いてください。

修正が機能したことを確認してください：

```
claude --version
```

Windows PowerShell

```
$env:PATH -split ';' | Select-String 'local\bin'
```

出力がない場合は、インストールディレクトリをユーザー PATH に追加してください：

```
$currentPath = [Environment]::GetEnvironmentVariable('PATH', 'User')
[Environment]::SetEnvironmentVariable('PATH', "$currentPath;$env:USERPROFILE\local\bin", 'User')
```

変更を有効にするためにターミナルを再起動してください。

修正が機能したことを確認してください：

```
claude --version
```

Windows CMD

```
echo %PATH% | findstr /i "local\bin"
```

出力がない場合は、システム設定を開き、環境変数に移動して、`%USERPROFILE%\local\bin` をユーザー PATH 変数に追加してください。ターミナルを再起動してください。

修正が機能したことを確認してください：

```
claude --version
```

競合するインストールを確認する

複数の Claude Code インストールはバージョンの不一致または予期しない動作を引き起こす可能性があります。インストールされているものを確認してください：

macOS/Linux

PATH に見つかったすべての `claude` バイナリをリストします：

```
which -a claude
```

ネイティブインストーラーと npm バージョンが存在するかどうかを確認してください：

```
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"
```

複数のインストールが見つかった場合は、1つだけを保持してください。`~/.local/bin/claude` でのネイティブインストールが推奨されます。余分なインストールを削除してください：

npm グローバルインストールをアンインストールします：

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

macOS で Homebrew インストールを削除します：

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

ディレクトリのアクセス許可を確認する

インストーラーは `~/.local/bin/` と `~/.claude/` への書き込みアクセスが必要です。インストールがアクセス許可エラーで失敗する場合は、これらのディレクトリが書き込み可能かどうかを確認してください：

```
test -w ~/.local/bin && echo "writable" || echo "not writable"
test -w ~/.claude && echo "writable" || echo "not writable"
```

いずれかのディレクトリが書き込み可能でない場合は、インストールディレクトリを作成し、ユーザーを所有者として設定してください：

```
sudo mkdir -p ~/.local/bin
sudo chown -R $(whoami) ~/.local
```

バイナリが機能することを確認する

`claude` がインストールされているがスタートアップでクラッシュまたはハングする場合は、これらのチェックを実行して原因を絞り込んでください。

バイナリが存在し、実行可能であることを確認してください：

```
ls -la $(which claude)
```

Linux では、不足している共有ライブラリを確認してください。`ldd` が不足しているライブラリを表示する場合は、システムパッケージをインストールする必要があるかもしれません。Alpine Linux および他の musl ベースのディストリビューションについては、[Alpine Linux セットアップ](#)を参照してください。

```
ldd $(which claude) | grep "not found"
```

バイナリが実行できることを確認するための簡単なサニティチェックを実行してください：

```
claude --version
```

一般的なインストール問題

これらは最も頻繁に発生するインストール問題とその解決策です。

インストールスクリプトがシェルスクリプトの代わりに HTML を返す

インストールコマンドを実行するときに、次のいずれかのエラーが表示される場合があります：

```
bash: line 1: syntax error near unexpected token `<'
bash: line 1: `<!DOCTYPE html>'
```

PowerShell では、同じ問題は次のように表示されます：

```
Invoke-Expression: Missing argument in parameter list.
```

これは、インストール URL がインストールスクリプトの代わりに HTML ページを返したことを意味します。HTML ページに「App unavailable in region」と表示されている場合、Claude Code はお客様の国では利用できません。[サポートされている国](#)を参照してください。

それ以外の場合、これはネットワークの問題、地域的なルーティング、または一時的なサービス中断が原因で発生する可能性があります。

解決策：

1. 別のインストール方法を使用する：

macOS または Linux では、Homebrew 経由でインストールしてください：

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

Windows では、WinGet 経由でインストールしてください：

```
winget install Anthropic.ClaudeCode
```

1. **数分後に再試行する**：この問題は一時的なことが多いです。待ってから元のコマンドを再度試してください。

インストール後に `command not found: claude`

インストールが完了しましたが、`claude` が機能しません。正確なエラーはプラットフォームによって異なります：

プラットフォーム	エラーメッセージ
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>

これは、インストールディレクトリがシェル検索パスに含まれていないことを意味します。各プラットフォームでの修正については、[PATHを確認する](#)を参照してください。

`curl: (56) Failure writing output to destination`

`curl ... | bash` コマンドはスクリプトをダウンロードし、パイプ（`|`）を使用して Bash に直接実行するために渡します。このエラーは、スクリプトのダウンロードが完了する前に接続が切断されたことを意味します。一般的な原因には、ネットワーク中断、ダウンロードがストリーム中にブロックされた、またはシステムリソースの制限が含まれます。

解決策：

1. **ネットワークの安定性を確認する**：Claude Code バイナリは Google Cloud Storage でホストされています。到達可能であることをテストしてください：

```
curl -fsSL https://storage.googleapis.com -o /dev/null
```

コマンドが静かに完了する場合、接続は問題なく、問題は一時的である可能性があります。インストールコマンドを再試行してください。エラーが表示される場合、ネットワークがダウンロードをブロックしている可能性があります。

1. 別のインストール方法を試す：

macOS または Linux：

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

Windows：

```
winget install Anthropic.ClaudeCode
```

TLS または SSL 接続エラー

`curl: (35) TLS connect error`、`schannel: next InitializeSecurityContext failed`、または PowerShell の `Could not establish trust relationship for the SSL/TLS secure channel` などのエラーは、TLS ハンドシェイク失敗を示します。

解決策：

1. システム CA 証明書を更新する：

Ubuntu/Debian：

```
sudo apt-get update && sudo apt-get install ca-certificates
```

macOS (Homebrew 経由)：

```
brew install ca-certificates
```

1. Windows では、インストーラーを実行する前に PowerShell で TLS 1.2 を有効にする：

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12  
irm https://claude.ai/install.ps1 | iex
```

1. プロキシまたはファイアウォールの干渉を確認する：TLS 検査を実行する企業プロキシは、`unable to get local issuer certificate` を含むこれらのエラーを引き起こ

可能性があります。`NODE_EXTRA_CA_CERTS` を企業 CA 証明書バンドルに設定してください：

```
export NODE_EXTRA_CA_CERTS=/path/to/corporate-ca.pem
```

証明書ファイルがない場合は IT チームに問い合わせてください。直接接続でも試して、プロキシが原因であることを確認できます。

`Failed to fetch version from storage.googleapis.com`

インストーラーがダウンロードサーバーに到達できませんでした。これは通常、`storage.googleapis.com` がネットワークでブロックされていることを意味します。

解決策：

1. 接続を直接テストする：

```
curl -sI https://storage.googleapis.com
```

1. **プロキシの背後にいる場合**、`HTTPS_PROXY` を設定して、インストーラーがそれを通じてルーティングできるようにしてください。詳細については、[プロキシ設定](#)を参照してください。

```
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

1. **制限されたネットワーク上にいる場合**、別のネットワークまたは VPN を試すか、別のインストール方法を使用してください：

macOS または Linux：

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

Windows：

```
winget install Anthropic.ClaudeCode
```

Windows : irm または && が認識されない

'irm' is not recognized または The token '&&' is not valid が表示される場合、シェルに対して間違ったコマンドを実行しています。

- **irm が認識されない** : PowerShell ではなく CMD にいます。2つのオプションがあります :

スタートメニューで「PowerShell」を検索して PowerShell を開き、元のインストールコマンドを実行してください :

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

または CMD にとどまり、代わりに CMD インストーラーを使用してください :

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

- **&& が有効でない** : PowerShell にいますが、CMD インストーラーコマンドを実行しました。PowerShell インストーラーを使用してください :

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

低メモリ Linux サーバーでインストールが強制終了される

VPS またはクラウドインスタンスでインストール中に **Killed** が表示される場合 :

```
Setting up Claude Code...  
Installing Claude Code native build latest...  
bash: line 142: 34803 Killed "$binary_path" install ${TARGET:+"$TARGET"}
```

Linux OOM キラーがプロセスを終了しました。システムがメモリ不足になったためです。Claude Code には少なくとも 4 GB の利用可能な RAM が必要です。

解決策 :

1. **サーバーの RAM が限定されている場合、スワップスペースを追加する**。スワップはディスク領域をオーバーフローメモリとして使用し、物理 RAM が少ない場合でもインストールを完了できます。
2 GB スワップファイルを作成して有効にしてください :

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```

その後、インストールを再試行してください：

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

1. **インストール前に他のプロセスを閉じてメモリを解放してください。**
2. **可能であれば、より大きなインスタンスを使用する。** Claude Code には少なくとも 4 GB の RAM が必要です。

Docker でインストールがハングする

Docker コンテナに Claude Code をインストールするときに、root として `/` にインストールするとハングが発生する可能性があります。

解決策：

1. **インストーラーを実行する前に作業ディレクトリを設定する。** `/` から実行すると、インストーラーはファイルシステム全体をスキャンし、過度なメモリ使用を引き起こします。`WORKDIR` を設定すると、スキャンが小さなディレクトリに制限されます：

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

1. **Docker メモリ制限を増やす** (Docker Desktop を使用している場合)：

```
docker build --memory=4g .
```

Windows：Claude Desktop が `claude` CLI コマンドをオーバーライドする

古いバージョンの Claude Desktop をインストールした場合、`WindowsApps` ディレクトリに `Claude.exe` を登録して、Claude Code CLI よりも PATH の優先度を高くする可能性があります。`claude` を実行するとデスクトップアプリが開きます。

Claude Desktop を最新バージョンに更新して、この問題を修正してください。

Windows : 「Claude Code on Windows requires git-bash」

ネイティブ Windows の Claude Code には、Git Bash を含む[Git for Windows](#)が必要です。

Git がインストールされていない場合、git-scm.com/downloads/winからダウンロードしてインストールしてください。セットアップ中に「Add to PATH」を選択してください。インストール後、ターミナルを再起動してください。

Git がすでにインストールされているが、Claude Code がそれを見つけられない場合は、[settings.json ファイル](#)でパスを設定してください：

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

Git がどこか別の場所にインストールされている場合は、PowerShell で `where.exe git` を実行してパスを見つけ、そのディレクトリから `bin\\bash.exe` パスを使用してください。

Linux : インストールされた間違ったバイナリバリエーション (musl/glibc の不一致)

インストール後に `libstdc++.so.6` または `libgcc_s.so.1` などの不足している共有ライブラリに関するエラーが表示される場合、インストーラーがシステムに対して間違ったバイナリバリエーションをダウンロードした可能性があります。

```
Error loading shared library libstdc++.so.6: No such file or directory
```

これは、musl クロスコンパイルパッケージがインストールされている glibc ベースのシステムで発生する可能性があり、インストーラーがシステムを musl として誤検出します。

解決策：

1. システムが使用している libc を確認する：

```
ldd /bin/ls | head -1
```

`linux-vdso.so` または `/lib/x86_64-linux-gnu/` への参照が表示される場合、`glibc` を使用しています。`musl` が表示される場合、`musl` を使用しています。

1. `glibc` を使用しているが `musl` バイナリを取得した場合、インストールを削除して再インストールしてください。GCS バケット

`https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json` から正しいバイナリ

を手動でダウンロードすることもできます。`ldd /bin/ls` と `ls /lib/libc.musl*` の出力を含む [GitHub issue](#) を提出してください。

2. 実際に `musl` を使用している場合 (Alpine Linux)、必要なパッケージをインストールしてください：

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

Linux での `Illegal instruction`

インストーラーが OOM `Killed` メッセージの代わりに `Illegal instruction` を出力する場合、ダウンロードされたバイナリが CPU アーキテクチャと一致しません。これは、ARM サーバーが x86 バイナリを受け取った場合、または必要な命令セットがない古い CPU で一般的に発生します。

```
bash: line 142: 2238232 Illegal instruction   "$binary_path" install ${TARGET:+"$TARGET"}
```

解決策：

1. アーキテクチャを確認する：

```
uname -m
```

`x86_64` は 64 ビット Intel/AMD を意味し、`aarch64` は ARM64 を意味します。バイナリが一致しない場合は、出力を含む [GitHub issue](#) を提出してください。

1. アーキテクチャの問題が解決されている間、別のインストール方法を試す：

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

macOS での `dyld: cannot load`

インストール中に `dyld: cannot load` または `Abort trap: 6` が表示される場合、バイナリは macOS バージョンまたはハードウェアと互換性がありません。

```
dyld: cannot load 'claude-2.1.42-darwin-x64' (load command 0x80000034 is unknown)
Abort trap: 6
```

解決策：

1. **macOS バージョンを確認する**：Claude Code には macOS 13.0 以降が必要です。Apple メニューを開き、「このマックについて」を選択してバージョンを確認してください。
2. **古いバージョンを使用している場合は macOS を更新する**。バイナリは古い macOS バージョンがサポートしていないロードコマンドを使用しています。
3. **別のインストール方法として Homebrew を試す**：

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

Windows インストール問題：WSL でのエラー

WSL で次の問題が発生する可能性があります：

OS/プラットフォーム検出の問題：インストール中にエラーが発生する場合、WSL が Windows `npm` を使用している可能性があります。試してください：

- インストール前に `npm config set os linux` を実行する
- `npm install -g @anthropic-ai/claude-code --force --no-os-check` でインストールする。`sudo` を使用しないでください。

Node が見つからないエラー：`claude` を実行するときに `exec: node: not found` が表示される場合、WSL 環境が Windows インストールの Node.js を使用している可能性があります。`which npm` と `which node` で確認できます。これらは `/usr/` で始まる Linux パスではなく `/mnt/c/` を指す必要があります。これを修正するには、Linux ディストリビューションのパッケージマネージャーまたは `nvm` 経由で Node をインストールしてみてください。

nvm バージョンの競合：WSL と Windows の両方に `nvm` がインストールされている場合、WSL で Node バージョンを切り替えるときにバージョンの競合が発生する可能性があります。これは WSL がデフォルトで Windows PATH をインポートするため、WSL インストールよりも Windows `nvm/npm` が優先されるためです。

この問題は以下で識別できます：

- `which npm` と `which node` を実行する - `/mnt/c/` で始まる Windows パスを指す場合、Windows バージョンが使用されています
- WSL で `nvm` を使用してノードバージョンを切り替えた後、機能が破損する

この問題を解決するには、Linux PATH を修正して、Linux ノード/npm バージョンが優先されるようにしてください：

主要な解決策：nvm がシェルに適切にロードされていることを確認する

最も一般的な原因は、`nvm` が非対話型シェルにロードされていないことです。シェル設定ファイル（`~/.bashrc`、`~/.zshrc` など）に以下を追加してください：

```
## nvm が存在する場合はロードする
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

または現在のセッションで直接実行してください：

```
source ~/.nvm/nvm.sh
```

代替案：PATH の順序を調整する

`nvm` が適切にロードされているが、Windows パスが依然として優先される場合、シェル設定で Linux パスを PATH の先頭に明示的に追加できます：

```
export PATH="$HOME/.nvm/versions/node/$(node -v)/bin:$PATH"
```

Warning:

`appendWindowsPath = false` で Windows PATH インポートを無効にすることは避けてください。これは WSL から Windows 実行可能ファイルを呼び出す機能を破損させます。同様に、Windows 開発に使用する場合は、Windows から Node.js をアンインストールすることは避けてください。

WSL2 サンドボックスセットアップ

サンドボックスは WSL2 でサポートされていますが、追加のパッケージをインストールする必要があります。 `/sandbox` を実行するときに「Sandbox requires socat and bubblewrap」というエラーが表示される場合は、依存関係をインストールしてください：

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

WSL1 はサンドボックスをサポートしていません。「Sandboxing requires WSL2」が表示される場合は、WSL2 にアップグレードするか、サンドボックスなしで Claude Code を実行する必要があります。

インストール中のアクセス許可エラー

ネイティブインストーラーがアクセス許可エラーで失敗する場合、ターゲットディレクトリが書き込み可能でない可能性があります。[ディレクトリのアクセス許可を確認する](#)を参照してください。

以前に npm でインストールしており、npm 固有のアクセス許可エラーが発生している場合は、ネイティブインストーラーに切り替えてください：

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

アクセス許可と認証

これらのセクションでは、ログイン失敗、トークンの問題、およびアクセス許可プロンプトの動作に対処します。

繰り返されるアクセス許可プロンプト

同じコマンドを繰り返し承認する必要がある場合は、`/permissions` コマンドを使用して、特定のツールが承認なしで実行されるようにできます。[アクセス許可ドキュメント](#)を参照してください。

認証の問題

認証の問題が発生している場合：

1. `/logout` を実行して完全にサインアウトしてください
2. Claude Code を閉じてください
3. `claude` で再起動して、認証プロセスを再度完了してください

ブラウザがログイン中に自動的に開かない場合は、`c` を押して OAuth URL をクリップボードにコピーしてから、手動でブラウザに貼り付けてください。

OAuth エラー：無効なコード

`OAuth error: Invalid code. Please make sure the full code was copied` が表示される場合、ログインコードが期限切れになったか、コピー貼り付け中に切り詰められました。

解決策：

- ・ブラウザが開いた後、すぐにログインを完了するために Enter キーを押して再試行してください
- ・ブラウザが自動的に開かない場合は、`c` を入力して完全な URL をコピーしてください
- ・リモート/SSH セッションを使用している場合、ブラウザが間違ったマシンで開く可能性があります。ターミナルに表示されている URL をコピーして、ローカルブラウザで開いてください。

ログイン後の 403 Forbidden

ログイン後に `API Error: 403 {"error":{"type":"forbidden","message":"Request not allowed"}}` が表示される場合：

- ・ **Claude Pro/Max ユーザー**：claude.ai/settings でサブスクリプションがアクティブであることを確認してください
- ・ **Console ユーザー**：管理者によって「Claude Code」または「Developer」ロールが割り当てられていることを確認してください
- ・ **プロキシの背後**：企業プロキシは API リクエストに干渉する可能性があります。[ネットワーク設定](#) でプロキシセットアップを参照してください。

WSL2 での OAuth ログイン失敗

WSL2 でのブラウザベースのログインは、WSL が Windows ブラウザを開くことができない場合に失敗する可能性があります。`BROWSER` 環境変数を設定してください：

```
export BROWSER="/mnt/c/Program Files/Google/Chrome/Application/chrome.exe"
claude
```

または、URL を手動でコピーしてください：ログインプロンプトが表示されたら、**c** を押して OAuth URL をコピーしてから、Windows ブラウザに貼り付けてください。

「ログインしていません」またはトークンの期限切れ

Claude Code がセッション後に再度ログインするよう求める場合、OAuth トークンが期限切れになった可能性があります。

/login を実行して再認証してください。これが頻繁に発生する場合は、システムクロックが正確であることを確認してください。トークン検証は正確なタイムスタンプに依存しています。

設定ファイルの場所

Claude Code は複数の場所に設定を保存します：

ファイル	目的
<code>~/.claude/settings.json</code>	ユーザー設定（アクセス許可、hooks、モデルオーバーライド）
<code>.claude/settings.json</code>	プロジェクト設定（ソース管理にチェックイン）
<code>.claude/settings.local.json</code>	ローカルプロジェクト設定（コミットされない）
<code>~/.claude.json</code>	グローバル状態（テーマ、OAuth、MCP サーバー）
<code>.mcp.json</code>	プロジェクト MCP サーバー（ソース管理にチェックイン）
<code>managed-mcp.json</code>	管理対象 MCP サーバー
管理対象設定	管理対象設定 （サーバー管理、MDM/OS レベルのポリシー、またはファイルベース）

Windows では、`~` はユーザーホームディレクトリ（`C:\Users\YourName` など）を指します。

これらのファイルの設定の詳細については、[設定](#)と[MCP](#)を参照してください。

設定をリセットする

Claude Code をデフォルト設定にリセットするには、設定ファイルを削除できます：

```
## すべてのユーザー設定と状態をリセットする
rm ~/.claude.json
rm -rf ~/.claude/

## プロジェクト固有の設定をリセットする
rm -rf .claude/
rm .mcp.json
```

Warning:

これにより、すべての設定、MCP サーバー設定、およびセッション履歴が削除されます。

パフォーマンスと安定性

これらのセクションでは、リソース使用、応答性、および検索動作に関連する問題に対処します。

CPU またはメモリ使用率が高い

Claude Code はほとんどの開発環境で動作するように設計されていますが、大規模なコードベースを処理するときに大量のリソースを消費する可能性があります。パフォーマンスの問題が発生している場合：

1. `/compact` を定期的を使用してコンテキストサイズを削減してください
2. 主要なタスク間で Claude Code を閉じて再起動してください
3. 大規模なビルドディレクトリを `.gitignore` ファイルに追加することを検討してください

コマンドがハングまたはフリーズする

Claude Code が応答しないように見える場合：

1. Ctrl+C を押して現在の操作をキャンセルしてください
2. 応答しない場合は、ターミナルを閉じて再起動する必要があるかもしれません

検索と発見の問題

Search ツール、`@file` メンション、カスタムエージェント、およびカスタムスキルが機能していない場合は、システム `ripgrep` をインストールしてください：

```
## 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
```

その後、`環境`で `USE_BUILTIN_RIPGREP=0` を設定してください。

WSL での遅い、または不完全な検索結果

[WSL でのファイルシステム間での作業](#)時のディスク読み取りパフォーマンスペナルティにより、WSL で Claude Code を使用するとき Search ツールを使用して予想より少ないマッチが返される可能性があります。検索は依然として機能しますが、ネイティブファイルシステムよりも少ない結果を返します。

Note:

この場合、`/doctor` は Search を OK として表示します。

解決策：

1. **より具体的な検索を送信する**：検索するファイル数を減らすために、ディレクトリまたはファイルタイプを指定してください：「auth-service パッケージで JWT 検証ロジックを検索する」または「JS ファイルで md5 ハッシュの使用を見つける」。

2. **プロジェクトを Linux ファイルシステムに移動する**：可能であれば、プロジェクトが Windows ファイルシステム（`/mnt/c/`）ではなく Linux ファイルシステム（`/home/`）に配置されていることを確認してください。
3. **ネイティブ Windows を使用する**：WSL ではなく Windows でネイティブに Claude Code を実行することを検討してください。ファイルシステムのパフォーマンスが向上します。

IDE 統合の問題

Claude Code が IDE に接続しない場合、または IDE ターミナル内で予期しない動作をする場合は、以下の解決策を試してください。

WSL2 で JetBrains IDE が検出されない

WSL2 で Claude Code を使用していて、JetBrains IDE を使用している場合に「No available IDEs detected」エラーが表示される場合、これは WSL2 のネットワーク設定または Windows ファイアウォールが接続をブロックしている可能性があります。

WSL2 ネットワークモード

WSL2 はデフォルトで NAT ネットワークを使用しており、IDE 検出を妨げる可能性があります。2つのオプションがあります：

オプション 1：Windows ファイアウォールを設定する（推奨）

1. WSL2 IP アドレスを見つけてください：

```
wsl hostname -I
## 出力例：172.21.123.45
```

1. 管理者として PowerShell を開き、ファイアウォールルールを作成してください：

```
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
```

ステップ 1 の WSL2 サブネットに基づいて IP 範囲を調整してください。

1. IDE と Claude Code の両方を再起動してください

オプション 2：ミラーリングネットワークに切り替える

Windows ユーザーディレクトリの `.wslconfig` に追加してください：

```
[wsl2]
```

```
networkingMode=mirrored
```

その後、PowerShell から `wsl --shutdown` で WSL を再起動してください。

Note:

これらのネットワーク問題は WSL2 にのみ影響します。WSL1 はホストのネットワークを直接使用し、これらの設定は必要ありません。

JetBrains の追加設定のヒントについては、[JetBrains IDE ガイド](#)を参照してください。

Windows IDE 統合の問題を報告する

Windows で IDE 統合の問題が発生している場合は、次の情報を含む[issue を作成](#)してください：

- 環境タイプ：ネイティブ Windows（Git Bash）または WSL1/WSL2
- WSL ネットワークモード（該当する場合）：NAT またはミラーリング
- IDE 名とバージョン
- Claude Code 拡張機能/プラグインバージョン
- シェルタイプ：Bash、Zsh、PowerShell など

JetBrains IDE ターミナルで Escape キーが機能しない

JetBrains ターミナルで Claude Code を使用していて、`Esc` キーがエージェントを中断しない場合、これは JetBrains のデフォルトショートカットとのキーバインディングの競合が原因である可能性があります。

この問題を修正するには：

1. 設定 → ツール → ターミナルに移動してください
2. 次のいずれかを実行してください：
 - 「Move focus to the editor with Escape」をオフにする、または
 - 「Configure terminal keybindings」をクリックして「Switch focus to Editor」ショートカットを削除する
3. 変更を適用してください

これにより、`Esc` キーが Claude Code 操作を適切に中断できるようになります。

Markdown フォーマットの問題

Claude Code は、生成された Markdown ファイルにコードフェンスの言語タグが不足していることがあり、GitHub、エディター、およびドキュメントツールでの構文強調表示と読みやすさに影響を与える可能性があります。

コードブロックの言語タグが不足している

生成された Markdown でこのようなコードブロックに気付いた場合：

```
```  

function example() {
 return "hello";
}
```text
```

適切にタグ付けされたブロックの代わりに：

```
```javascript  
function example() {
 return "hello";
}
```text
```

解決策：

1. **Claude に言語タグを追加するよう依頼する**：「Add appropriate language tags to all code blocks in this markdown file.」とリクエストしてください。
2. **後処理 hooks を使用する**：不足している言語タグを検出して追加するための自動フォーマット hooks を設定してください。例については、[編集後のコード自動フォーマット](#)を参照してください。
3. **手動検証**：Markdown ファイルを生成した後、適切なコードブロックフォーマットを確認し、必要に応じて修正をリクエストしてください。

一貫性のないスペースとフォーマット

生成された Markdown に過度な空行または一貫性のないスペースがある場合：

解決策：

1. **フォーマット修正をリクエストする**：Claude に「Fix spacing and formatting issues in this markdown file.」とリクエストしてください。

2. **フォーマットツールを使用する**： `prettier` またはカスタムフォーマットスクリプトなどの Markdown フォーマッターを実行する hooks を設定してください。
3. **フォーマット設定を指定する**：プロンプトまたはプロジェクト [メモリ](#) ファイルにフォーマット要件を含めてください。

Markdown フォーマットの問題を減らす

フォーマットの問題を最小化するには：

- **リクエストで明示的にする**：「properly formatted markdown with language-tagged code blocks」を求めてください
- **プロジェクト規約を使用する**： `CLAUDE.md` で推奨される Markdown スタイルを文書化してください
- **検証 hooks を設定する**：後処理 hooks を使用して、一般的なフォーマットの問題を自動的に検証および修正してください

さらにヘルプを得る

ここに記載されていない問題が発生している場合：

1. Claude Code 内で `/bug` コマンドを使用して、Anthropic に直接問題を報告してください
2. [GitHub リポジトリ](#) で既知の問題を確認してください
3. `/doctor` を実行して問題を診断してください。以下をチェックします：
 - インストールタイプ、バージョン、および検索機能
 - 自動更新ステータスと利用可能なバージョン
 - 無効な設定ファイル（不正な形式の JSON、不正な型）
 - MCP サーバー設定エラー
 - キーバインディング設定の問題
 - コンテキスト使用警告（大規模な CLAUDE.md ファイル、高い MCP トークン使用、到達不可能なアクセス許可ルール）
 - プラグインとエージェントのロードエラー
4. Claude に直接その機能と特性について質問してください - Claude はドキュメントへの組み込みアクセスを持っています