Claude Code

完整技術文件

By Anthropic

製作日期：2026-03-17

一個在終端機中運行的 AI 編碼工具，
理解您的程式碼庫，幫助您更快地編寫程式。

內容編譯自 code.claude.com/docs/zh-TW
如需最新版本，請造訪網站。

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

Table of Contents

Part 1: Getting Started

Claude Code 概述

> Claude Code 是一個代理編碼工具，可以讀取您的程式碼庫、編輯檔案、執行命令，並與您的開發工具整合。可在您的終端、IDE、桌面應用程式和瀏覽器中使用。

Claude Code 是一個由 AI 驅動的編碼助手，可幫助您建立功能、修復錯誤和自動化開發任務。它理解您的整個程式碼庫，並可以跨多個檔案和工具工作以完成任務。

開始使用

選擇您的環境以開始使用。大多數介面需要 Claude 訂閱或 Anthropic Console 帳戶。終端 CLI 和 VS Code 也支援第三方提供商。

Terminal

功能完整的 CLI，用於直接在終端中使用 Claude Code。編輯檔案、執行命令，並從命令列管理整個專案。

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.

然後在任何專案中啟動 Claude Code：

```
cd your-project
claude
```

首次使用時，系統會提示您登入。就這麼簡單！繼續快速入門 →

> **Tip:**
>
> 請參閱進階設定以了解安裝選項、手動更新或卸載說明。如果遇到問題，請訪問疑難排解。

VS Code

VS Code 擴充功能在編輯器中直接提供內聯差異、@-提及、計畫審查和對話歷史記錄。

- 安裝 VS Code
- 安裝 Cursor

或在擴充功能檢視中搜尋「Claude Code」（Mac 上為 `Cmd+Shift+X`，Windows/Linux 上為 `Ctrl+Shift+X`）。安裝後，開啟命令面板（`Cmd+Shift+P` / `Ctrl+Shift+P`），輸入「Claude Code」，然後選擇**在新標籤中開啟**。

開始使用 VS Code →

Desktop app

一個獨立應用程式，用於在 IDE 或終端外執行 Claude Code。以視覺方式審查差異、並排執行多個工作階段、排程重複任務，以及啟動雲端工作階段。

下載並安裝：

- macOS（Intel 和 Apple Silicon）
- Windows（x64）
- Windows ARM64（僅限遠端工作階段）

安裝後，啟動 Claude，登入，然後按一下**Code** 標籤以開始編碼。需要付費訂閱。

深入了解桌面應用程式 →

Web

在瀏覽器中執行 Claude Code，無需本機設定。啟動長時間執行的任務，並在完成後檢查，處理您本機沒有的儲存庫，或並行執行多個任務。可在桌面瀏覽器和 Claude iOS 應用程式上使用。

在 claude.ai/code 開始編碼。

開始在網路上使用 →

JetBrains

IntelliJ IDEA、PyCharm、WebStorm 和其他 JetBrains IDE 的外掛程式，具有互動式差異檢視和選擇內容共享。

從 JetBrains Marketplace 安裝 Claude Code 外掛程式，然後重新啟動 IDE。

開始使用 JetBrains →

您可以做什麼

以下是您可以使用 Claude Code 的一些方式：

Claude Code 處理佔用您一整天的繁瑣任務：為未測試的程式碼編寫測試、修復整個專案中的 lint 錯誤、解決合併衝突、更新依賴項和編寫發行說明。

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

用純文字描述您想要的內容。Claude Code 規劃方法、跨多個檔案編寫程式碼，並驗證其是否有效。

對於錯誤，貼上錯誤訊息或描述症狀。Claude Code 透過您的程式碼庫追蹤問題、識別根本原因並實施修復。請參閱常見工作流程以了解更多範例。

Claude Code 直接與 git 配合使用。它暫存變更、編寫提交訊息、建立分支並開啟拉取請求。

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

在 CI 中，您可以使用 GitHub Actions 或 GitLab CI/CD 自動化程式碼審查和問題分類。

Model Context Protocol (MCP) 是一個開放標準，用於將 AI 工具連接到外部資料來源。使用 MCP，Claude Code 可以讀取 Google Drive 中的設計文件、更新 Jira 中的票證、從 Slack 提取資料，或使用您自己的自訂工具。

`CLAUDE.md` 是您新增到專案根目錄的 markdown 檔案，Claude Code 在每個工作階段開始時讀取。使用它來設定編碼標準、架構決策、首選程式庫和審查檢查清單。Claude 也會在工作時建立自動記憶，儲存學習內容，例如建置命令和除錯見解，跨工作階段而無需您編寫任何內容。

建立自訂命令以封裝您的團隊可以共享的可重複工作流程，例如 `/review-pr` 或 `/deploy-staging`。

Hooks 可讓您在 Claude Code 操作之前或之後執行 shell 命令，例如在每次檔案編輯後自動格式化或在提交前執行 lint。

生成多個 Claude Code 代理，同時處理任務的不同部分。主導代理協調工作、指派子任務並合併結果。

對於完全自訂的工作流程，Agent SDK 可讓您建立由 Claude Code 的工具和功能驅動的自己的代理，並完全控制編排、工具存取和權限。

Claude Code 是可組合的，遵循 Unix 哲學。將日誌管道傳輸到其中、在 CI 中執行它，或將其與其他工具鏈接：

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

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

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

請參閱 CLI 參考以了解完整的命令和旗標集。

工作階段不受限於單一介面。當您的內容變更時，在環境之間移動工作：

- 離開您的辦公桌，使用遠端控制從您的手機或任何瀏覽器繼續工作
- 在網路或 iOS 應用程式上啟動長時間執行的任務，然後使用 `/teleport` 將其提取到終端
- 使用 `/desktop` 將終端工作階段交付給桌面應用程式以進行視覺差異審查
- 從團隊聊天路由任務：在 Slack 中提及 `@Claude` 並提供錯誤報告，然後取回拉取請求

在任何地方使用 Claude Code

每個介面都連接到相同的基礎 Claude Code 引擎，因此您的 CLAUDE.md 檔案、設定和 MCP servers 可在所有介面中使用。

除了上述終端、VS Code、JetBrains、桌面和網路環境外，Claude Code 還與 CI/CD、聊天和瀏覽器工作流程整合：

我想要…	最佳選項
從我的手機或其他裝置繼續本機工作階段	遠端控制
在本機啟動任務，在行動裝置上繼續	網路或 Claude iOS 應用程式

我想要…	最佳選項
自動化 PR 審查和問題分類	GitHub Actions 或 GitLab CI/CD
在每個 PR 上獲得自動程式碼審查	GitHub Code Review
將 Slack 中的錯誤報告路由到拉取請求	Slack
除錯即時網路應用程式	Chrome
為您自己的工作流程建立自訂代理	Agent SDK

後續步驟

安裝 Claude Code 後，這些指南可幫助您深入了解。

- 快速入門：逐步完成您的第一個真實任務，從探索程式碼庫到提交修復
- 儲存說明和記憶：使用 CLAUDE.md 檔案和自動記憶為 Claude 提供持久說明
- 常見工作流程和最佳實踐：從 Claude Code 獲得最大收益的模式
- 設定：為您的工作流程自訂 Claude Code
- 疑難排解：常見問題的解決方案
- code.claude.com：演示、定價和產品詳細資訊

快速入門

> 歡迎使用 Claude Code！

本快速入門指南將在幾分鐘內讓您使用 AI 驅動的編碼協助。完成後，您將了解如何使用 Claude Code 進行常見的開發任務。

開始前

確保您擁有：

- 已開啟的終端或命令提示字元
  - 如果您從未使用過終端，請查看終端指南
- 一個要使用的程式碼專案
- 一個 Claude 訂閱（Pro、Max、Teams 或 Enterprise）、Claude Console 帳戶，或透過支援的雲端提供商的存取權

> **Note:**
>
> 本指南涵蓋終端 CLI。Claude Code 也可在網頁、桌面應用程式、VS Code 和 JetBrains IDE、Slack 中使用，以及透過 GitHub Actions 和 GitLab 進行 CI/CD。請參閱所有介面。

步驟 1：安裝 Claude Code

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.

步驟 2：登入您的帳戶

Claude Code 需要帳戶才能使用。當您使用 `claude` 命令啟動互動式工作階段時，您需要登入：

```
claude
## 首次使用時系統會提示您登入
```

```
/login
## 按照提示使用您的帳戶登入
```

您可以使用以下任何帳戶類型登入：

- Claude Pro、Max、Teams 或 Enterprise（推薦）
- Claude Console（具有預付額度的 API 存取）。首次登入時，Console 中會自動為集中式成本追蹤建立一個「Claude Code」工作區。
- Amazon Bedrock、Google Vertex AI 或 Microsoft Foundry（企業雲端提供商）

登入後，您的認證將被儲存，您無需再次登入。若要稍後切換帳戶，請使用 `/login` 命令。

步驟 3：啟動您的第一個工作階段

在任何專案目錄中開啟您的終端並啟動 Claude Code：

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

您將看到 Claude Code 歡迎畫面，其中包含您的工作階段資訊、最近的對話和最新更新。輸入 `/help` 以查看可用命令，或輸入 `/resume` 以繼續之前的對話。

> **Tip:**
>
> 登入後（步驟 2），您的認證將儲存在您的系統上。在認證管理中了解更多。

步驟 4：提出您的第一個問題

讓我們從了解您的程式碼庫開始。嘗試以下命令之一：

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

Claude 將分析您的檔案並提供摘要。您也可以提出更具體的問題：

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

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

```
explain the folder structure
```

您也可以詢問 Claude 其自身的功能：

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

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

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

> **Note:**
>
> Claude Code 根據需要讀取您的專案檔案。您無需手動新增內容。

步驟 5：進行您的第一次程式碼變更

現在讓我們讓 Claude Code 進行一些實際的編碼。嘗試一個簡單的任務：

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

Claude Code 將：

1. 找到適當的檔案
2. 向您顯示建議的變更
3. 要求您的批准
4. 進行編輯

> **Note:**
>
> Claude Code 在修改檔案前始終要求許可。您可以批准個別變更或為工作階段啟用「全部接受」模式。

步驟 6：使用 Git 與 Claude Code

Claude Code 使 Git 操作變得對話式：

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

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

您也可以提示進行更複雜的 Git 操作：

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

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

```
help me resolve merge conflicts
```

步驟 7：修復錯誤或新增功能

Claude 擅長除錯和功能實現。

用自然語言描述您想要的內容：

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

或修復現有問題：

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

Claude Code 將：

- 定位相關程式碼
- 理解上下文
- 實現解決方案
- 如果可用，執行測試

步驟 8：測試其他常見工作流程

有許多方式可以與 Claude 合作：

重構程式碼

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

編寫測試

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

更新文件

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

程式碼審查

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

> **Tip:**
> 像與有幫助的同事交談一樣與 Claude 交談。描述您想要達成的目標，它將幫助您實現。

基本命令

以下是日常使用中最重要的命令：

命令	功能	範例
`claude`	啟動互動模式	`claude`
`claude "task"`	執行一次性任務	`claude "fix the build error"`
`claude -p "query"`	執行一次性查詢，然後退出	`claude -p "explain this function"`
`claude -c`	在目前目錄中繼續最近的對話	`claude -c`

命令	功能	範例
`claude -r`	恢復之前的對話	`claude -r`
`claude commit`	建立 Git 提交	`claude commit`
`/clear`	清除對話歷史	`/clear`
`/help`	顯示可用命令	`/help`
`exit` 或 Ctrl+C	退出 Claude Code	`exit`

請參閱 CLI 參考以取得完整的命令列表。

初學者的專業提示

如需更多資訊，請參閱最佳實踐和常見工作流程。

不要這樣做：‘修復錯誤’

試試這樣：‘修復登入錯誤，使用者輸入錯誤認證後看到空白畫面’

將複雜任務分解為步驟：

```
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
```

在進行變更之前，讓 Claude 了解您的程式碼：

```
analyze the database schema
```

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

- 按 `?` 查看所有可用的快捷鍵
- 使用 Tab 進行命令完成
- 按 ↑ 查看命令歷史
- 輸入 `/` 查看所有命令和 skills

接下來呢？

現在您已經學習了基礎知識，請探索更多進階功能：

- Claude Code 如何運作 了解代理迴圈、內建工具以及 Claude Code 如何與您的專案互動
- 最佳實踐 透過有效的提示和專案設定獲得更好的結果
- 常見工作流程 常見任務的逐步指南
- 擴展 Claude Code 使用 CLAUDE.md、skills、hooks、MCP 等進行自訂

獲取幫助

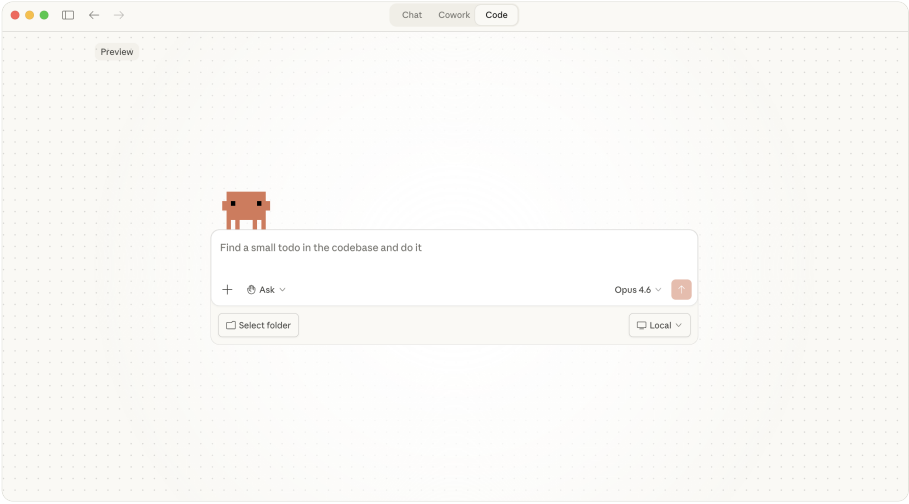
- **在 Claude Code 中**：輸入 `/help` 或詢問「how do I…」
- **文件**：您在這裡！瀏覽其他指南
- **社群**：加入我們的 Discord 以獲取提示和支援

開始使用桌面應用程式

> 在桌面上安裝 Claude Code 並開始您的第一個編碼會話

桌面應用程式為您提供具有圖形介面的 Claude Code：視覺化差異檢查、即時應用程式預覽、GitHub PR 監控與自動合併、使用 Git worktree 隔離的並行會話、排程任務，以及遠端執行任務的能力。無需終端機。

本頁面將引導您安裝應用程式並開始您的第一個會話。如果您已經設定完成，請參閱 使用 Claude Code Desktop 以取得完整參考。



Claude Code Desktop 介面顯示已選擇 Code 標籤，包含提示框、權限模式選擇器設定為’詢問權限’ 、模型選擇器、資料夾選擇器和本機環境選項

桌面應用程式有三個標籤：

- **Chat**：無檔案存取的一般對話，類似於 claude.ai。
- **Cowork**：一個自主背景代理，在雲端 VM 中處理任務，具有自己的環境。它可以獨立執行，同時您進行其他工作。
- **Code**：一個互動式編碼助手，可直接存取您的本機檔案。您可以即時檢查並批准每項變更。

Chat 和 Cowork 涵蓋在 Claude Desktop 支援文章 中。本頁面重點關注 **Code** 標籤。

> **Note:**
>
> Claude Code 需要 Pro、Max、Teams 或 Enterprise 訂閱。

安裝

Step 1: 下載應用程式

為您的平台下載 Claude。

- macOS 適用於 Intel 和 Apple Silicon 的通用版本
- Windows 適用於 x64 處理器

對於 Windows ARM64，在此下載。

目前不支援 Linux。

Step 2: 登入

從您的應用程式資料夾 (macOS) 或開始功能表 (Windows) 啟動 Claude。使用您的 Anthropic 帳戶登入。

Step 3: 開啟 Code 標籤

點擊頂部中央的 **Code** 標籤。如果點擊 Code 提示您升級，您需要先 訂閱付費方案。如果提示您線上登入，請完成登入並重新啟動應用程式。如果您看到 403 錯誤，請參閱 驗證疑難排解。

桌面應用程式包含 Claude Code。您無需單獨安裝 Node.js 或 CLI。若要從終端機使用 `claude`，請單獨安裝 CLI。請參閱 開始使用 CLI。

開始您的第一個會話

開啟 Code 標籤後，選擇一個專案並告訴 Claude 要做什麼。

Step 1: 選擇環境和資料夾

選擇 **Local** 以在您的機器上執行 Claude，直接使用您的檔案。點擊 **Select folder** 並選擇您的專案目錄。

> **Tip:**
>
> 從您熟悉的小型專案開始。這是查看 Claude Code 能做什麼的最快方式。在 Windows 上，必須安裝 Git 才能使本機會話正常運作。大多數 Mac 預設包含 Git。

您也可以選擇：

- **Remote**：在 Anthropic 的雲端基礎設施上執行會話，即使您關閉應用程式也會繼續。遠端會話使用與 Claude Code on the web 相同的基礎設施。
- **SSH**：透過 SSH 連接到遠端機器（您自己的伺服器、雲端 VM 或開發容器）。Claude Code 必須安裝在遠端機器上。

Step 2: 選擇模型

從傳送按鈕旁的下拉式選單中選擇模型。請參閱 models 以比較 Opus、Sonnet 和 Haiku。會話開始後無法變更模型。

Step 3: 告訴 Claude 要做什麼

輸入您想讓 Claude 做的事：

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

會話 是與 Claude 關於您的程式碼的對話。每個會話追蹤自己的上下文和變更，因此您可以處理多個任務而不會相互干擾。

Step 4: 檢查並接受變更

預設情況下，Code 標籤以 詢問權限模式 啟動，其中 Claude 提出變更並等待您的批准才能應用它們。您將看到：

1. 一個 差異檢視 顯示每個檔案中將發生的確切變更
2. 接受/拒絕按鈕以批准或拒絕每項變更
3. 當 Claude 處理您的請求時的即時更新

如果您拒絕變更，Claude 將詢問您希望如何以不同方式進行。在您接受之前，您的檔案不會被修改。

接下來呢？

您已進行了第一次編輯。如需 Desktop 可執行的所有操作的完整參考，請參閱 使用 Claude Code Desktop。以下是一些接下來要嘗試的事項。

中斷並引導。 您可以隨時中斷 Claude。如果它走錯了方向，點擊停止按鈕或輸入您的更正並按 **Enter**。Claude 停止正在進行的操作並根據您的輸入進行調整。您無需等待它完成或重新開始。

為 Claude 提供更多上下文。 在提示框中輸入 `@filename` 以將特定檔案拉入對話、使用附件按鈕附加影像和 PDF，或直接將檔案拖放到提示中。Claude 擁有的上下文越多，結果越好。請參閱 新增檔案和上下文。

使用 skills 執行可重複的任務。 輸入 `/` 或點擊 **+** → **Slash commands** 以瀏覽 內建命令、自訂 skills 和外掛 skills。Skills 是可重複使用的提示，您可以在需要時調用，例如程式碼檢查清單或部署步驟。

在提交前檢查變更。 Claude 編輯檔案後，會出現 `+12 -1` 指示器。點擊它以開啟 差異檢視，逐個檔案檢查修改，並在特定行上評論。Claude 會讀取您的評論並進行修訂。點擊 **Review code** 讓 Claude 自己評估差異並留下內聯建議。

調整您擁有的控制量。 您的 權限模式 控制平衡。詢問權限（預設）在每次編輯前需要批准。自動接受編輯會自動接受檔案編輯以加快迭代。Plan mode 讓 Claude 在不觸及任何檔案的情況下規劃方法，這在大型重構前很有用。

新增外掛以獲得更多功能。 點擊提示框旁的 **+** 按鈕並選擇 **Plugins** 以瀏覽和安裝 外掛，這些外掛新增 skills、代理、MCP servers 等。

預覽您的應用程式。 點擊 **Preview** 下拉式選單以直接在桌面中執行您的開發伺服器。Claude 可以檢視執行中的應用程式、測試端點、檢查日誌並對其看到的內容進行迭代。請參閱 預覽您的應用程式。

追蹤您的提取請求。 開啟 PR 後，Claude Code 監控 CI 檢查結果，並可以自動修復失敗或在所有檢查通過後合併 PR。請參閱 監控提取請求狀態。

將 Claude 放在排程上。 設定 排程任務 以定期自動執行 Claude：每天早上進行程式碼檢查、每週進行依賴項審計，或從您連接的工具中提取的簡報。

準備好時擴展。 從側邊欄開啟 並行會話 以同時處理多個任務，每個任務都在自己的 Git worktree 中。將 長期執行的工作發送到雲端，以便即使您關閉應用程式也會繼續，或 在網路或 IDE 中繼續會話（如果任務花費的時間比預期長）。連接外部工具，例如 GitHub、Slack 和 Linear，以整合您的工作流程。

來自 CLI？

Desktop 使用與 CLI 相同的引擎，但具有圖形介面。您可以在同一專案上同時執行兩者，它們共享配置（CLAUDE.md 檔案、MCP servers、hooks、skills 和設定）。如需功能、標誌等效項和 Desktop 中不可用的內容的完整比較，請參閱 CLI 比較。

接下來

- 使用 Claude Code Desktop：權限模式、並行會話、差異檢視、連接器和企業配置
- 疑難排解：常見錯誤和設定問題的解決方案

- 最佳實踐：撰寫有效提示和充分利用 Claude Code 的提示
- 常見工作流程：除錯、重構、測試等的教學

設定 Claude Code

> 在您的開發機器上安裝、驗證和開始使用 Claude Code。

系統需求

- **作業系統**：
  - macOS 13.0+
  - Windows 10 1809+ 或 Windows Server 2019+ (查看設定說明)
  - Ubuntu 20.04+
  - Debian 10+
  - Alpine Linux 3.19+ (需要額外的依賴項)
- **硬體**：4 GB+ RAM
- **網路**：需要網際網路連線 (查看 網路設定)
- **Shell**：在 Bash 或 Zsh 中效果最佳
- **位置**：Anthropic 支援的國家

額外的依賴項

- **ripgrep**：通常包含在 Claude Code 中。如果搜尋失敗，請查看 搜尋疑難排解。
- **Node.js 18+**：僅在 已棄用的 npm 安裝 時需要

安裝

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.

安裝程序完成後，導航到您的專案並啟動 Claude Code：

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

如果您在安裝過程中遇到任何問題，請查閱 疑難排解指南。

> **Tip:**
>
> 安裝後執行 `claude doctor` 以檢查您的安裝類型和版本。

平台特定設定

Windows：原生執行 Claude Code (需要 Git Bash) 或在 WSL 內執行。支援 WSL 1 和 WSL 2，但 WSL 1 的支援有限，不支援 Bash 工具沙箱等功能。

Alpine Linux 和其他 musl/uClibc 型發行版：

Alpine 和其他 musl/uClibc 型發行版上的原生安裝程式需要 `libgcc` 、 `libstdc++` 和 `ripgrep` 。使用您的發行版的套件管理員安裝這些，然後設定 `USE_BUILTIN_RIPGREP=0` 。

在 Alpine 上：

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

驗證

針對個人

1. **Claude Pro 或 Max 方案** (建議)：訂閱 Claude 的 Pro 或 Max 方案，以獲得統一的訂閱，包括 Claude Code 和網頁版 Claude。在一個地方管理您的帳戶，並使用您的 Claude.ai 帳戶登入。
2. **Claude Console**：透過 Claude Console 連線並完成 OAuth 程序。需要在 Anthropic Console 中有有效的帳單。系統會自動為使用情況追蹤和成本管理建立「Claude Code」工作區。您無法為 Claude Code 工作區建立 API 金鑰；它專門用於 Claude Code 使用。

針對團隊和組織

1. **Claude for Teams 或 Enterprise** (建議)：訂閱 Claude for Teams 或 Claude for Enterprise，以獲得集中式帳單、團隊管理以及對 Claude Code 和網頁版 Claude 的存取。團隊成員使用其 Claude.ai 帳戶登入。
2. **Claude Console 與團隊帳單**：設定共用的 Claude Console 組織，並使用團隊帳單。邀請團隊成員並指派角色以進行使用情況追蹤。
3. **雲端提供商**：設定 Claude Code 以使用 Amazon Bedrock、Google Vertex AI 或 Microsoft Foundry，以便與您現有的雲端基礎結構進行部署。

安裝特定版本

原生安裝程式接受特定版本號或發行頻道 (`latest` 或 `stable`)。您在安裝時選擇的頻道將成為自動更新的預設值。如需詳細資訊，請查看 設定發行頻道。

若要安裝最新版本 (預設)：

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
```

若要安裝穩定版本：

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
```

若要安裝特定版本號：

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
```

二進位完整性和程式碼簽署

- 所有平台的 SHA256 校驗和都發佈在發行版本清單中，目前位於 `https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json` (範例：將 `{VERSION}` 替換為 `2.0.30`)
- 簽署的二進位檔案分佈在以下平台上：
  - macOS：由’Anthropic PBC’簽署並由 Apple 公證
  - Windows：由’Anthropic, PBC’簽署

NPM 安裝 (已棄用)

NPM 安裝已棄用。盡可能使用 原生安裝 方法。若要將現有的 npm 安裝遷移到原生安裝，請執行 `claude install`。

全域 npm 安裝

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

> **Warning:**
>
> 請勿使用 `sudo npm install -g`，因為這可能導致權限問題和安全風險。 如果您遇到權限錯誤，請查看 疑難排解權限錯誤 以獲得建議的解決方案。

Windows 設定

選項 1：WSL 內的 Claude Code

- 支援 WSL 1 和 WSL 2
- WSL 2 支援 沙箱 以增強安全性。WSL 1 不支援沙箱。

選項 2：使用 Git Bash 在原生 Windows 上執行 Claude Code

- 需要 Git for Windows
- 對於可攜式 Git 安裝，請指定您的 `bash.exe` 的路徑：

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

更新 Claude Code

自動更新

Claude Code 會自動保持最新狀態，以確保您擁有最新的功能和安全修補程式。

- **更新檢查**：在啟動時和執行時定期執行
- **更新程序**：在背景自動下載和安裝
- **通知**：安裝更新時您會看到通知
- **套用更新**：更新在您下次啟動 Claude Code 時生效

> **Note:**
>
> Homebrew 和 WinGet 安裝不會自動更新。使用 `brew upgrade claude-code` 或 `winget upgrade Anthropic.ClaudeCode` 以手動更新。
>
> **已知問題**：Claude Code 可能會在新版本在這些套件管理員中可用之前通知您有更新。如果升級失敗，請稍候並稍後重試。

設定發行頻道

使用 `autoUpdatesChannel` 設定來設定 Claude Code 針對自動更新和 `claude update` 遵循的發行頻道：

- `"latest"` (預設)：在新功能發佈時立即接收
- `"stable"`：使用通常約一週舊的版本，跳過有重大迴歸的發行版本

透過 `/config` → **自動更新頻道** 進行設定，或將其新增到您的 settings.json 檔案：

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

對於企業部署，您可以使用 受管設定 在整個組織中強制執行一致的發行頻道。

停用自動更新

在您的 shell 或 settings.json 檔案 中設定 `DISABLE_AUTOUPDATER` 環境變數：

```
export DISABLE_AUTOUPDATER=1
```

手動更新

```
claude update
```

解除安裝 Claude Code

如果您需要解除安裝 Claude Code，請按照您的安裝方法的說明進行操作。

原生安裝

移除 Claude Code 二進位檔案和版本檔案：

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 安裝

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

WinGet 安裝

```
winget uninstall Anthropic.ClaudeCode
```

NPM 安裝

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

清理設定檔案 (選用)

> **Warning:**
> 移除設定檔案將刪除您的所有設定、允許的工具、MCP 伺服器設定和工作階段歷史記錄。

若要移除 Claude Code 設定和快取資料：

macOS、Linux、WSL：

```
## 移除使用者設定和狀態
rm -rf ~/.claude
rm ~/.claude.json

## 移除專案特定設定（從您的專案目錄執行）
rm -rf .claude
rm -f .mcp.json
```

Windows PowerShell：

```
## 移除使用者設定和狀態
Remove-Item -Path "$env:USERPROFILE\.claude" -Recurse -Force
Remove-Item -Path "$env:USERPROFILE\.claude.json" -Force

## 移除專案特定設定（從您的專案目錄執行）
Remove-Item -Path ".claude" -Recurse -Force
Remove-Item -Path ".mcp.json" -Force
```

Windows CMD：

```
REM 移除使用者設定和狀態
rmdir /s /q "%USERPROFILE%\.claude"
del "%USERPROFILE%\.claude.json"

REM 移除專案特定設定（從您的專案目錄執行）
rmdir /s /q ".claude"
del ".mcp.json"
```

Claude Code 如何運作

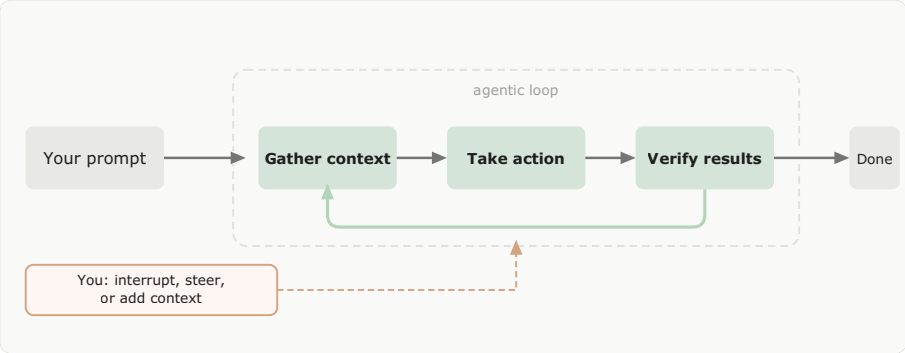
> 了解代理迴圈、內建工具，以及 Claude Code 如何與您的專案互動。

Claude Code 是一個在您的終端機中執行的代理助手。雖然它在編碼方面表現出色，但它可以幫助您從命令列執行的任何操作：撰寫文件、執行建置、搜尋檔案、研究主題等。

本指南涵蓋核心架構、內建功能，以及有效使用 Claude Code 的提示。如需逐步說明，請參閱常見工作流程。如需 skills、MCP 和 hooks 等擴充功能，請參閱擴充 Claude Code。

代理迴圈

當您給 Claude 一項任務時，它會經歷三個階段：**收集上下文**、**採取行動**和**驗證結果**。這些階段相互融合。Claude 始終使用工具，無論是搜尋檔案以了解您的程式碼、編輯以進行變更，還是執行測試以檢查其工作。



代理迴圈：您的提示導致 Claude 收集上下文、採取行動、驗證結果，並重複直到任務完成。您可以在任何時刻中斷。

迴圈會根據您的要求進行調整。關於您程式碼庫的問題可能只需要收集上下文。錯誤修復會反覆循環所有三個階段。重構可能涉及廣泛的驗證。Claude 根據從上一步學到的內容決定每一步需要什麼，將數十個操作鏈接在一起，並沿途進行課程修正。

您也是這個迴圈的一部分。您可以在任何時刻中斷以引導 Claude 朝不同方向發展、提供額外上下文，或要求它嘗試不同的方法。Claude 自主工作，但對您的輸入保持回應。

代理迴圈由兩個元件提供支援：推理的模型和執行的工具。Claude Code 充當 Claude 周圍的**代理工具**：它提供工具、上下文管理和執行環境，將語言模型轉變為能力強大的編碼代理。

模型

Claude Code 使用 Claude 模型來理解您的程式碼並推理任務。Claude 可以讀取任何語言的程式碼、理解元件如何連接，以及找出需要改變什麼來完成您的目標。對於複雜的任務，它將工作分解為步驟、執行它們，並根據學到的內容進行調整。

多個模型可用，具有不同的權衡。Sonnet 可以很好地處理大多數編碼任務。Opus 為複雜的架構決策提供更強的推理能力。在會話期間使用 `/model` 切換，或使用 `claude --model <name>` 開始。

當本指南說「Claude 選擇」或「Claude 決定」時，是模型在進行推理。

工具

工具是使 Claude Code 成為代理的原因。沒有工具，Claude 只能用文字回應。有了工具，Claude 可以採取行動：讀取您的程式碼、編輯檔案、執行命令、搜尋網路，以及與外部服務互動。每個工具使用都會返回資訊，反饋到迴圈中，告知 Claude 的下一個決定。

內建工具通常分為五個類別，每個類別代表不同類型的代理。

類別	Claude 可以做什麼
檔案操作	讀取檔案、編輯程式碼、建立新檔案、重新命名和重新組織
搜尋	按模式查找檔案、使用正規表達式搜尋內容、探索程式碼庫
執行	執行 shell 命令、啟動伺服器、執行測試、使用 git
網路	搜尋網路、擷取文件、查詢錯誤訊息
程式碼智慧	編輯後查看類型錯誤和警告、跳轉到定義、尋找參考（需要程式碼智慧外掛程式）

這些是主要功能。Claude 還具有用於生成 subagents、詢問您問題和其他協調任務的工具。請參閱Claude 可用的工具以取得完整清單。

Claude 根據您的提示和沿途學到的內容選擇使用哪些工具。當您說「修復失敗的測試」時，Claude 可能會：

1. 執行測試套件以查看失敗的內容
2. 讀取錯誤輸出
3. 搜尋相關的原始檔案
4. 讀取這些檔案以理解程式碼
5. 編輯檔案以修復問題

6. 再次執行測試以驗證

每個工具使用都會給 Claude 新的資訊，告知下一步。這就是代理迴圈的實際運作。

擴充基本功能： 內建工具是基礎。您可以使用 skills 擴充 Claude 知道的內容、使用 MCP 連接到外部服務、使用 hooks 自動化工作流程，以及使用 subagents 卸載任務。這些擴充形成了核心代理迴圈之上的一層。請參閱擴充 Claude Code 以獲得有關為您的需求選擇正確擴充的指導。

Claude 可以存取什麼

本指南著重於終端機。Claude Code 也在 VS Code、JetBrains IDE 和其他環境中執行。

當您在目錄中執行 `claude` 時，Claude Code 可以存取：

- **您的專案。** 您目錄和子目錄中的檔案，以及其他地方經您許可的檔案。
- **您的終端機。** 您可以執行的任何命令：建置工具、git、套件管理器、系統公用程式、指令碼。如果您可以從命令列執行，Claude 也可以。
- **您的 git 狀態。** 目前分支、未提交的變更和最近的提交歷史。
- **您的 CLAUDE.md。** 一個 markdown 檔案，您可以在其中儲存專案特定的指示、慣例和 Claude 應該在每個會話中知道的上下文。
- **自動記憶。** Claude 在您工作時自動儲存的學習內容，例如專案模式和您的偏好。MEMORY.md 的前 200 行在每個會話開始時載入。
- **您設定的擴充。** 用於外部服務的 MCP servers、用於工作流程的 skills、用於委派工作的 subagents，以及用於瀏覽器互動的 Claude in Chrome。

因為 Claude 看到您的整個專案，它可以跨越它工作。當您要求 Claude「修復身份驗證錯誤」時，它會搜尋相關檔案、讀取多個檔案以理解上下文、跨它們進行協調編輯、執行測試以驗證修復，以及在您要求時提交變更。這與只看到目前檔案的內聯程式碼助手不同。

環境和介面

上述代理迴圈、工具和功能在您使用 Claude Code 的任何地方都是相同的。改變的是程式碼執行的位置以及您與它互動的方式。

執行環境

Claude Code 在三個環境中執行，每個環境對程式碼執行位置有不同的權衡。

環境	程式碼執行位置	使用案例
本機	您的機器	預設。完全存取您的檔案、工具和環境

環境	程式碼執行位置	使用案例
雲端	Anthropic 管理的 VM	卸載任務、處理您本機沒有的儲存庫
遠端控制	您的機器，從瀏覽器控制	使用網路 UI，同時保持一切本機

介面

您可以透過終端機、桌面應用程式、IDE 擴充、claude.ai/code、遠端控制、Slack 和 CI/CD 管道存取 Claude Code。介面決定了您如何看到和與 Claude 互動，但底層的代理迴圈是相同的。請參閱在任何地方使用 Claude Code 以取得完整清單。

使用會話

Claude Code 在您工作時將您的對話儲存在本機。每條訊息、工具使用和結果都被儲存，這使得倒帶、恢復和分叉會話成為可能。在 Claude 進行程式碼變更之前，它還會快照受影響的檔案，以便您在需要時可以還原。

會話是獨立的。 每個新會話都以新的上下文視窗開始，沒有來自先前會話的對話歷史。Claude 可以使用自動記憶跨會話保留學習內容，您可以在 CLAUDE.md 中新增自己的持久指示。

跨分支工作

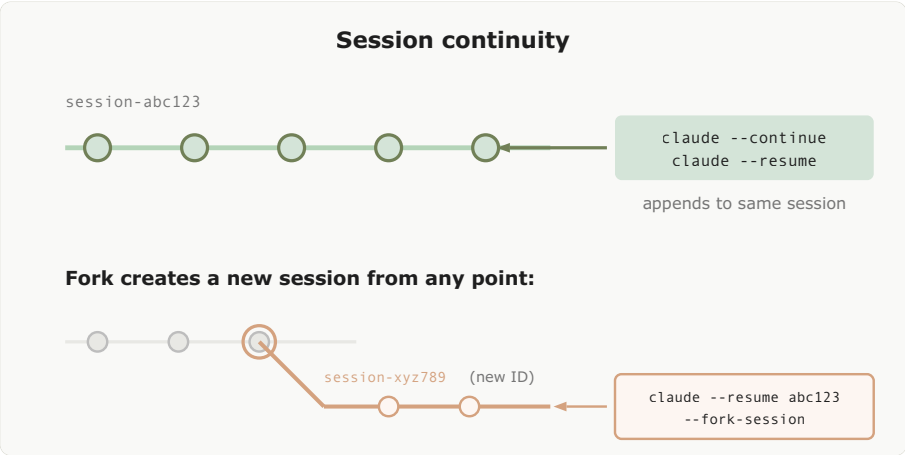
每個 Claude Code 對話都是一個與您目前目錄相關的會話。當您恢復時，您只會看到該目錄中的會話。

Claude 看到您目前分支的檔案。當您切換分支時，Claude 看到新分支的檔案，但您的對話歷史保持不變。Claude 記得您討論過的內容，即使在切換後也是如此。

由於會話與目錄相關，您可以使用 git worktrees 執行平行 Claude 會話，這會為個別分支建立單獨的目錄。

恢復或分叉會話

當您使用 `claude --continue` 或 `claude --resume` 恢復會話時，您使用相同的會話 ID 從中斷的地方繼續。新訊息附加到現有對話。您的完整對話歷史被還原，但會話範圍的許可不被還原。您需要重新批准這些。



會話連續性：恢復繼續相同的會話，分叉建立具有新 ID 的新分支。

要分支並嘗試不同的方法而不影響原始會話，請使用 `--fork-session` 旗標：

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

這會建立一個新的會話 ID，同時保留該點之前的對話歷史。原始會話保持不變。與恢復一樣，分叉的會話不會繼承會話範圍的許可。

在多個終端機中的相同會話：如果您在多個終端機中恢復相同的會話，兩個終端機都會寫入相同的會話檔案。來自兩者的訊息會交錯，就像兩個人在同一個筆記本中寫字。沒有任何東西會損壞，但對話變得混亂。每個終端機在會話期間只看到自己的訊息，但如果您稍後恢復該會話，您會看到所有交錯的內容。對於從相同起點進行的平行工作，使用 `--fork-session` 為每個終端機提供自己的乾淨會話。

上下文視窗

Claude 的上下文視窗保存您的對話歷史、檔案內容、命令輸出、CLAUDE.md、載入的 skills 和系統指示。當您工作時，上下文會填滿。Claude 會自動壓縮，但對話早期的指示可能會丟失。將持久規則放在 CLAUDE.md 中，並執行 `/context` 以查看什麼在佔用空間。

當上下文填滿時

Claude Code 在您接近限制時自動管理上下文。它首先清除較舊的工具輸出，然後在需要時總結對話。您的請求和關鍵程式碼片段被保留；對話早期的詳細指示可能會丟失。將持久規則放在 CLAUDE.md 中，而不是依賴對話歷史。

要控制在壓縮期間保留的內容，請在 CLAUDE.md 中新增「Compact Instructions」部分或使用焦點執行 `/compact`（例如 `/compact focus on the API changes`）。

執行 `/context` 以查看什麼在佔用空間。MCP servers 將工具定義新增到每個請求，因此幾個伺服器在您開始工作之前可能會消耗大量上下文。執行 `/mcp` 以檢查每個伺服器的成本。

使用 skills 和 subagents 管理上下文

除了壓縮，您可以使用其他功能來控制載入到上下文中的內容。

Skills 按需載入。Claude 在會話開始時看到 skill 描述，但完整內容只在使用 skill 時載入。對於您手動呼叫的 skills，設定 `disable-model-invocation: true` 以將描述保留在上下文之外，直到您需要它們。

Subagents 獲得自己的新上下文，完全與您的主要對話分開。他們的工作不會使您的上下文膨脹。完成後，他們返回摘要。這種隔離是 subagents 在長會話中有幫助的原因。

請參閱上下文成本以了解每個功能的成本，以及減少令牌使用以獲得管理上下文的提示。

使用檢查點和許可保持安全

Claude 有兩個安全機制：檢查點讓您撤銷檔案變更，許可控制 Claude 可以在不詢問的情況下執行的操作。

使用檢查點撤銷變更

每個檔案編輯都是可逆的。 在 Claude 編輯任何檔案之前，它會快照目前的內容。如果出現問題，按 `Esc` 兩次以倒帶到先前的狀態，或要求 Claude 撤銷。

檢查點是會話本機的，與 git 分開。它們只涵蓋檔案變更。影響遠端系統的操作（資料庫、API、部署）無法檢查點，這就是為什麼 Claude 在執行具有外部副作用的命令之前詢問。

控制 Claude 可以做什麼

按 `Shift+Tab` 循環通過許可模式：

- **預設**：Claude 在檔案編輯和 shell 命令之前詢問
- **自動接受編輯**：Claude 編輯檔案而不詢問，仍然詢問命令
- **Plan Mode**：Claude 僅使用唯讀工具，建立您可以在執行前批准的計畫

您也可以在 `.claude/settings.json` 中允許特定命令，以便 Claude 不會每次都詢問。這對於受信任的命令（如 `npm test` 或 `git status`）很有用。設定可以從組織範圍的政策範圍到個人偏好。請參閱許可以取得詳細資訊。

有效使用 Claude Code

這些提示可幫助您從 Claude Code 獲得更好的結果。

向 Claude Code 尋求幫助

Claude Code 可以教您如何使用它。提出問題，例如「我如何設定 hooks？」或「構造我的 CLAUDE.md 的最佳方式是什麼？」，Claude 會解釋。

內建命令也會引導您完成設定：

- `/init` 引導您為您的專案建立 CLAUDE.md
- `/agents` 幫助您設定自訂 subagents
- `/doctor` 診斷您的安裝的常見問題

這是一個對話

Claude Code 是對話式的。您不需要完美的提示。從您想要的開始，然後進行細化：

```
修復登入錯誤
```

[Claude 調查，嘗試一些東西]

```
這不太對。問題在於會話處理。
```

[Claude 調整方法]

當第一次嘗試不正確時，您不會重新開始。您進行迭代。

中斷和引導

您可以在任何時刻中斷 Claude。如果它走錯了路，只需輸入您的更正並按 Enter。Claude 將停止正在執行的操作，並根據您的輸入調整其方法。您不必等待它完成或重新開始。

預先具體

您的初始提示越精確，您需要的更正就越少。參考特定檔案、提及限制，並指出範例模式。

```
結帳流程對於具有過期卡的使用者已損壞。
檢查 src/payments/ 以查找問題，特別是令牌刷新。
先寫一個失敗的測試，然後修復它。
```

模糊的提示有效，但您會花更多時間引導。像上面這樣的具體提示通常在第一次嘗試時就成功。

給 Claude 一些東西來驗證

當 Claude 可以檢查自己的工作時，它表現得更好。包括測試案例、貼上預期 UI 的螢幕截圖，或定義您想要的輸出。

```
實現 validateEmail。測試案例：'user@example.com' → true，
'invalid' → false，'user@.com' → false。之後執行測試。
```

對於視覺工作，貼上設計的螢幕截圖，並要求 Claude 將其實現與其進行比較。

在實現之前進行探索

對於複雜的問題，將研究與編碼分開。使用計畫模式（按 `Shift+Tab` 兩次）首先分析程式碼庫：

```
讀取 src/auth/ 並理解我們如何處理會話。
然後為新增 OAuth 支援建立計畫。
```

檢查計畫，透過對話進行細化，然後讓 Claude 實現。這種兩階段方法比直接跳到程式碼產生更好的結果。

委派，不要指示

想像委派給一位能力強的同事。提供上下文和方向，然後相信 Claude 會找出詳細資訊：

```
結帳流程對於具有過期卡的使用者已損壞。
相關程式碼在 src/payments/ 中。您可以調查並修復它嗎？
```

您不需要指定要讀取哪些檔案或執行哪些命令。Claude 會找出來。

接下來是什麼

- 使用功能進行擴充 新增 Skills、MCP 連接和自訂命令
- 常見工作流程 典型任務的逐步指南

Part 2: Core Usage

擴展 Claude Code

> 了解何時使用 CLAUDE.md、Skills、subagents、hooks、MCP 和 plugins。

Claude Code 結合了一個能夠推理您程式碼的模型與內建工具，用於檔案操作、搜尋、執行和網路存取。內建工具涵蓋了大多數編碼任務。本指南涵蓋擴展層：您添加的功能，用於自訂 Claude 的知識、將其連接到外部服務，以及自動化工作流程。

> **Note:**
>
> 有關核心代理迴圈如何運作的資訊，請參閱Claude Code 如何運作。

初次使用 Claude Code？ 從CLAUDE.md開始了解專案約定。根據需要添加其他擴展。

概述

擴展插入代理迴圈的不同部分：

- **CLAUDE.md** 添加 Claude 在每個會話中看到的持久上下文
- **Skills** 添加可重複使用的知識和可調用的工作流程
- **MCP** 將 Claude 連接到外部服務和工具
- **Subagents** 在隔離的上下文中運行自己的迴圈，返回摘要
- **Agent teams** 協調多個獨立會話，具有共享任務和點對點訊息傳遞
- **Hooks** 完全在迴圈外運行作為確定性指令碼
- **Plugins** 和 **marketplaces** 打包和分發這些功能

Skills是最靈活的擴展。Skill 是一個包含知識、工作流程或指令的 markdown 檔案。您可以使用像 `/deploy` 這樣的命令調用 skills，或者 Claude 可以在相關時自動載入它們。Skills 可以在您目前的對話中運行，或通過 subagents 在隔離的上下文中運行。

將功能與您的目標相匹配

功能範圍從 Claude 在每個會話中看到的始終開啟的上下文，到您或 Claude 可以調用的按需功能，再到在特定事件上運行的背景自動化。下表顯示了可用的功能以及何時使用每一個。

功能	它的作用	何時使用	範例
CLAUDE.md	每次對話載入的持久上下文	專案約定、「始終執行 X」規則	「使用 pnpm，而不是 npm。在提交前運行測試。」
Skill	Claude 可以使用的指令、知識和工作流程	可重複使用的內容、參考文件、可重複的任務	`/deploy` 運行您的部署檢查清單；包含端點模式的 API 文件 skill
Subagent	返回摘要結果的隔離執行上下文	上下文隔離、並行任務、專門的工作者	讀取許多檔案但僅返回關鍵發現的研究任務
Agent teams	協調多個獨立的 Claude Code 會話	並行研究、新功能開發、使用競爭假設進行除錯	生成審查者以同時檢查安全性、效能和測試
MCP	連接到外部服務	外部資料或操作	查詢您的資料庫、發佈到 Slack、控制瀏覽器
Hook	在事件上運行的確定性指令碼	可預測的自動化，不涉及 LLM	在每次檔案編輯後運行 ESLint

Plugins 是打包層。Plugin 將 skills、hooks、subagents 和 MCP servers 捆綁到單個可安裝單元中。Plugin skills 是命名空間的（如 `/my-plugin:review`），因此多個 plugins 可以共存。當您想在多個儲存庫中重複使用相同的設置或通過 **marketplace** 分發給他人時，使用 plugins。

比較相似的功能

某些功能可能看起來相似。以下是如何區分它們。

Skill vs Subagent

Skills 和 subagents 解決不同的問題：

- **Skills** 是可重複使用的內容，您可以將其載入任何上下文
- **Subagents** 是與您的主要對話分開運行的隔離工作者

方面	Skill	Subagent
它是什麼	可重複使用的指令、知識或工作流程	具有自己上下文的隔離工作者

方面	Skill	Subagent
主要優勢	在上下文之間共享內容	上下文隔離。工作單獨進行，僅返回摘要
最適合	參考資料、可調用的工作流程	讀取許多檔案的任務、並行工作、專門的工作者

Skills 可以是參考或操作。 參考 skills 提供 Claude 在整個會話中使用的知識（如您的 API 風格指南）。操作 skills 告訴 Claude 執行特定操作（如運行您的部署工作流程的 `/deploy`）。

當您需要上下文隔離或您的上下文視窗變滿時，使用 subagent。Subagent 可能讀取數十個檔案或運行廣泛的搜尋，但您的主要對話僅接收摘要。由於 subagent 工作不消耗您的主要上下文，當您不需要中間工作保持可見時，這也很有用。自訂 subagents 可以有自己的指令，並可以預載 skills。

它們可以結合。 Subagent 可以預載特定 skills（`skills:` 欄位）。Skill 可以使用 `context: fork` 在隔離的上下文中運行。有關詳細資訊，請參閱 Skills。

CLAUDE.md vs Skill

兩者都存儲指令，但它們的載入方式和用途不同。

方面	CLAUDE.md	Skill
載入	每個會話，自動	按需
可以包含檔案	是，使用 `@path` 匯入	是，使用 `@path` 匯入
可以觸發工作流程	否	是，使用 `/<name>`
最適合	「始終執行 X」規則	參考資料、可調用的工作流程

如果 Claude 應該始終知道它，請將其放在 CLAUDE.md 中：編碼約定、構建命令、專案結構、「永遠不要執行 X」規則。

如果它是 Claude 有時需要的參考資料（API 文件、風格指南）或您使用 `/<name>` 觸發的工作流程（部署、審查、發佈），請將其放在 skill 中。

經驗法則： 保持 CLAUDE.md 在 200 行以下。如果它在增長，將參考內容移動到 skills 或拆分為 `.claude/rules/` 檔案。

CLAUDE.md vs Rules vs Skills

所有三者都存儲指令，但它們的載入方式不同：

方面	CLAUDE.md	`.claude/rules/`	Skill
載入	每個會話	每個會話，或在打開匹配檔案時	按需，在調用或相關時
範圍	整個專案	可以限定到檔案路徑	特定於任務
最適合	核心約定和構建命令	特定於語言或目錄的指南	參考資料、可重複的工作流程

使用 CLAUDE.md 用於每個會話需要的指令：構建命令、測試約定、專案架構。

使用 rules 保持 CLAUDE.md 專注。具有 `paths` frontmatter 的 rules 僅在 Claude 使用匹配檔案時載入，節省上下文。

使用 skills 用於 Claude 有時只需要的內容，如 API 文件或您使用 `/<name>` 觸發的部署檢查清單。

Subagent vs Agent team

兩者都並行化工作，但它們在架構上不同：

- **Subagents** 在您的會話內運行並將結果報告回您的主要上下文
- **Agent teams** 是相互通訊的獨立 Claude Code 會話

方面	Subagent	Agent team
上下文	自己的上下文視窗；結果返回給呼叫者	自己的上下文視窗；完全獨立
通訊	僅向主代理報告結果	隊友直接相互訊息傳遞
協調	主代理管理所有工作	具有自我協調的共享任務清單
最適合	只有結果重要的專注任務	需要討論和協作的複雜工作
令牌成本	較低：結果摘要回主上下文	較高：每個隊友是單獨的 Claude 實例

當您需要快速、專注的工作者時，使用 subagent：研究問題、驗證聲明、審查檔案。Subagent 執行工作並返回摘要。您的主要對話保持乾淨。

當隊友需要共享發現、相互質疑和獨立協調時，使用 agent team。Agent teams 最適合具有競爭假設的研究、並行程式碼審查，以及每個隊友擁有單獨部分的新功能開發。

轉換點： 如果您運行並行 subagents 但遇到上下文限制，或者您的 subagents 需要相互通訊，agent teams 是自然的下一步。

> **Note:**
> Agent teams 是實驗性的，預設情況下被禁用。有關設置和目前限制，請參閱 agent teams。

MCP vs Skill

MCP 將 Claude 連接到外部服務。Skills 擴展 Claude 的知識，包括如何有效地使用這些服務。

方面	MCP	Skill
它是什麼	連接到外部服務的協議	知識、工作流程和參考資料
提供	工具和資料存取	知識、工作流程、參考資料
範例	Slack 整合、資料庫查詢、瀏覽器控制	程式碼審查檢查清單、部署工作流程、API 風格指南

這些解決不同的問題，並且可以很好地協同工作：

MCP 給予 Claude 與外部系統互動的能力。沒有 MCP，Claude 無法查詢您的資料庫或發佈到 Slack。

Skills 給予 Claude 關於如何有效使用這些工具的知識，以及您可以使用 `/<name>` 觸發的工作流程。Skill 可能包括您的團隊資料庫架構和查詢模式，或具有您的團隊訊息格式規則的 `/post-to-slack` 工作流程。

範例：MCP 伺服器將 Claude 連接到您的資料庫。Skill 教導 Claude 您的資料模型、常見查詢模式，以及用於不同任務的表格。

了解功能如何分層

功能可以在多個級別定義：使用者範圍、每個專案、通過 plugins，或通過受管理的策略。您也可以在子目錄中嵌套 CLAUDE.md 檔案，或在 monorepo 的特定套件中放置 skills。當相同的功能存在於多個級別時，以下是它們的分層方式：

- **CLAUDE.md 檔案** 是累加的：所有級別同時對 Claude 的上下文貢獻內容。來自您的工作目錄及以上的檔案在啟動時載入；子目錄在您在其中工作時載入。當指令衝突時，Claude 使用判斷來協調它們，更具體的指令通常優先。請參閱 CLAUDE.md 檔案如何載入。

- **Skills 和 subagents** 按名稱覆蓋：當相同名稱存在於多個級別時，一個定義根據優先級獲勝（skills 為受管理 > 使用者 > 專案；subagents 為受管理 > CLI 標誌 > 專案 > 使用者 > plugin）。Plugin skills 是命名空間的，以避免衝突。請參閱 skill 發現 和 subagent 範圍。
- **MCP 伺服器** 按名稱覆蓋：本地 > 專案 > 使用者。請參閱 MCP 範圍。
- **Hooks** 合併：所有註冊的 hooks 為其匹配事件觸發，無論來源如何。請參閱 hooks。

結合功能

每個擴展解決不同的問題：CLAUDE.md 處理始終開啟的上下文，skills 處理按需知識和工作流程，MCP 處理外部連接，subagents 處理隔離，hooks 處理自動化。真實的設置根據您的工作流程結合它們。

例如，您可能使用 CLAUDE.md 用於專案約定、skill 用於您的部署工作流程、MCP 用於連接到您的資料庫，以及 hook 用於在每次編輯後運行 linting。每個功能處理它最擅長的事情。

模式	它如何運作	範例
Skill + MCP	MCP 提供連接；skill 教導 Claude 如何很好地使用它	MCP 連接到您的資料庫，skill 記錄您的架構和查詢模式
Skill + Subagent	Skill 生成 subagents 進行並行工作	`/audit` skill 啟動在隔離上下文中工作的安全性、效能和風格 subagents
CLAUDE.md + Skills	CLAUDE.md 保持始終開啟的規則；skills 保持按需載入的參考資料	CLAUDE.md 說’遵循我們的 API 約定’，skill 包含完整的 API 風格指南
Hook + MCP	Hook 通過 MCP 觸發外部操作	編輯後 hook 在 Claude 修改關鍵檔案時發送 Slack 通知

了解上下文成本

您添加的每個功能都消耗 Claude 的一些上下文。太多可能會填滿您的上下文視窗，但它也可能添加噪聲，使 Claude 效率降低；skills 可能無法正確觸發，或 Claude 可能會失去對您的約定的追蹤。了解這些權衡有助於您構建有效的設置。

按功能的上下文成本

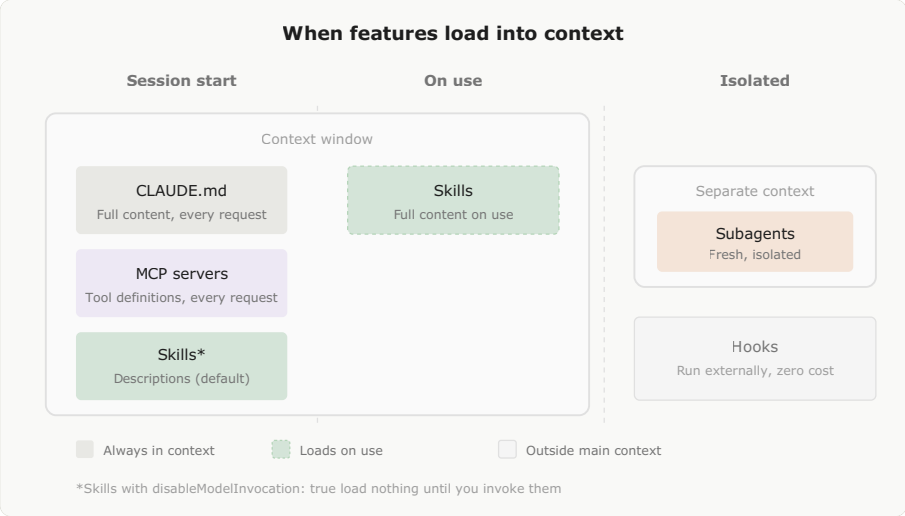
每個功能都有不同的載入策略和上下文成本：

功能	何時載入	什麼載入	上下文成本
CLAUDE.md	會話開始	完整內容	每個請求
Skills	會話開始 + 使用時	啟動時的描述，使用時的完整內容	低（每個請求的描述）*
MCP 伺服器	會話開始	所有工具定義和架構	每個請求
Subagents	生成時	具有指定 skills 的新鮮上下文	與主會話隔離
Hooks	觸發時	無（外部運行）	零，除非 hook 返回額外上下文

*預設情況下，skill 描述在會話開始時載入，以便 Claude 決定何時使用它們。在 skill 的 frontmatter 中設置 `disable-model-invocation: true` 以將其完全隱藏在 Claude 中，直到您手動調用它。這將 skills 的上下文成本降低到零，您只需自己觸發這些 skills。

了解功能如何載入

每個功能在您的會話中的不同點載入。下面的選項卡說明每個功能何時載入以及什麼進入上下文。



上下文載入：CLAUDE.md 和 MCP 在會話開始時載入並保留在每個請求中。Skills 在啟動時載入描述，在調用時載入完整內容。Subagents 獲得隔離的上下文。Hooks 外部運行。

CLAUDE.md

何時： 會話開始

什麼載入： 所有 CLAUDE.md 檔案的完整內容（受管理、使用者和專案級別）。

繼承： Claude 從您的工作目錄讀取 CLAUDE.md 檔案直到根目錄，並在訪問這些檔案時在子目錄中發現嵌套的檔案。有關詳細資訊，請參閱 CLAUDE.md 檔案如何載入。

> **Tip:**
> 保持 CLAUDE.md 在約 500 行以下。將參考資料移動到 skills，它們按需載入。

Skills

Skills 是 Claude 工具包中的額外功能。它們可以是參考資料（如 API 風格指南）或可調用的工作流程，您可以使用 `/<name>` 觸發（如 `/deploy`）。Claude Code 附帶捆綁的 skills，如 `/simplify`、`/batch` 和 `/debug`，開箱即用。您也可以創建自己的。Claude 在適當時使用 skills，或者您可以直接調用一個。

何時： 取決於 skill 的配置。預設情況下，描述在會話開始時載入，完整內容在使用時載入。對於僅使用者 skills（`disable-model-invocation: true`），在您調用它們之前不會載入任何內容。

什麼載入： 對於模型可調用的 skills，Claude 在每個請求中看到名稱和描述。當您使用 `/<name>` 調用 skill 或 Claude 自動載入它時，完整內容載入到您的對話中。

Claude 如何選擇 skills： Claude 將您的任務與 skill 描述相匹配，以決定哪些相關。如果描述模糊或重疊，Claude 可能載入錯誤的 skill 或錯過會有幫助的。要告訴 Claude 使用特定 skill，請使用 `/<name>` 調用它。具有 `disable-model-invocation: true` 的 Skills 對 Claude 不可見，直到您調用它們。

上下文成本： 低，直到使用。僅使用者 skills 在調用前成本為零。

在 subagents 中： Skills 在 subagents 中的工作方式不同。不是按需載入，傳遞給 subagent 的 skills 在啟動時完全預載入其上下文。Subagents 不從主會話繼承 skills；您必須明確指定它們。

> **Tip:**
> 對具有副作用的 skills 使用 `disable-model-invocation: true`。這節省上下文並確保只有您觸發它們。

MCP 伺服器

何時： 會話開始。

什麼載入： 來自連接伺服器的所有工具定義和 JSON 架構。

上下文成本： 工具搜尋（預設啟用）將 MCP 工具載入到上下文的 10%，並延遲其餘部分直到需要。

可靠性注意： MCP 連接可能在會話中途無聲地失敗。如果伺服器斷開連接，其工具會無警告地消失。Claude 可能嘗試使用不再存在的工具。如果您注意到 Claude 無法使用它之前可以存取的 MCP 工具，請使用 `/mcp` 檢查連接。

> **Tip:**
>
> 運行 `/mcp` 以查看每個伺服器的令牌成本。斷開您未主動使用的伺服器。

Subagents

何時： 按需，當您或 Claude 為任務生成一個時。

什麼載入： 新鮮、隔離的上下文，包含：

- 系統提示（與父級共享以提高快取效率）
- 代理 `skills:` 欄位中列出的 skills 的完整內容
- CLAUDE.md 和 git 狀態（從父級繼承）
- 主代理在提示中傳遞的任何上下文

上下文成本： 與主會話隔離。Subagents 不繼承您的對話歷史或調用的 skills。

> **Tip:**
>
> 對不需要您完整對話上下文的工作使用 subagents。它們的隔離防止膨脹您的主會話。

Hooks

何時： 觸發時。Hooks 在特定生命週期事件（如工具執行、會話邊界、提示提交、權限請求和壓縮）時觸發。有關完整清單，請參閱 Hooks。

什麼載入： 預設情況下無。Hooks 作為外部指令碼運行。

上下文成本： 零，除非 hook 返回添加為訊息到您的對話的輸出。

> **Tip:**
>
> Hooks 非常適合不需要影響 Claude 上下文的副作用（linting、logging）。

了解更多

每個功能都有自己的指南，包含設置指令、範例和配置選項。

- CLAUDE.md 存儲專案上下文、約定和指令
- Skills 給予 Claude 領域專業知識和可重複使用的工作流程
- Subagents 將工作卸載到隔離的上下文
- Agent teams 協調多個並行工作的會話
- MCP 將 Claude 連接到外部服務
- Hooks 使用 hooks 自動化工作流程
- Plugins 捆綁和共享功能集
- Marketplaces 託管和分發 plugin 集合

互動模式

> Claude Code 會話中鍵盤快捷鍵、輸入模式和互動功能的完整參考。

鍵盤快捷鍵

Note:

鍵盤快捷鍵可能因平台和終端而異。按 `?` 查看您環境中可用的快捷鍵。

macOS 使用者：Option/Alt 鍵快捷鍵（`Alt+B` 、 `Alt+F` 、 `Alt+Y` 、 `Alt+M` 、 `Alt+P`）需要在終端中將 Option 配置為 Meta：

- **iTerm2**：設定 → 設定檔 → 鍵 → 將左/右 Option 鍵設定為「Esc+」
- **Terminal.app**：設定 → 設定檔 → 鍵盤 → 勾選「使用 Option 作為 Meta 鍵」
- **VS Code**：設定 → 設定檔 → 鍵 → 將左/右 Option 鍵設定為「Esc+」

詳見終端配置。

一般控制

快捷鍵	說明	上下文
`Ctrl+C`	取消目前輸入或生成	標準中斷
`Ctrl+F`	終止所有背景代理。在 3 秒內按兩次以確認	背景代理控制
`Ctrl+D`	退出 Claude Code 會話	EOF 信號
`Ctrl+G`	在預設文字編輯器中開啟	在預設文字編輯器中編輯您的提示或自訂回應
`Ctrl+L`	清除終端螢幕	保留對話歷史
`Ctrl+O`	切換詳細輸出	顯示詳細的工具使用和執行情況
`Ctrl+R`	反向搜尋命令歷史	以互動方式搜尋先前的命令

快捷鍵	說明	上下文
`Ctrl+V` 或 `Cmd+V`（iTerm2）或 `Alt+V`（Windows）	從剪貼簿貼上影像	貼上影像或影像檔案的路徑
`Ctrl+B`	背景執行工作	將 bash 命令和代理放在背景執行。Tmux 使用者按兩次
`Ctrl+T`	切換工作清單	在終端狀態區域中顯示或隱藏工作清單
`Left/Right arrows`	在對話框標籤之間循環	在權限對話框和選單中的標籤之間導航
`Up/Down arrows`	導航命令歷史	回憶先前的輸入
`Esc` + `Esc`	回溯或摘要	將程式碼和/或對話恢復到先前的點，或從選定的訊息進行摘要
`Shift+Tab` 或 `Alt+M`（某些配置）	切換權限模式	在自動接受模式、Plan Mode 和正常模式之間切換。
`Option+P`（macOS）或 `Alt+P`（Windows/Linux）	切換模型	在不清除提示的情況下切換模型
`Option+T`（macOS）或 `Alt+T`（Windows/Linux）	切換擴展思考	啟用或停用擴展思考模式。首先執行 `/terminal-setup` 以啟用此快捷鍵

文字編輯

快捷鍵	說明	上下文
`Ctrl+K`	刪除到行尾	儲存已刪除的文字以供貼上
`Ctrl+U`	刪除整行	儲存已刪除的文字以供貼上
`Ctrl+Y`	貼上已刪除的文字	貼上使用 `Ctrl+K` 或 `Ctrl+U` 刪除的文字

快捷鍵	說明	上下文
`Alt+Y`（在 `Ctrl+Y` 之後）	循環貼上歷史	貼上後，循環瀏覽先前刪除的文字。在 macOS 上需要將 Option 設定為 Meta
`Alt+B`	將游標向後移動一個單字	單字導航。在 macOS 上需要將 Option 設定為 Meta
`Alt+F`	將游標向前移動一個單字	單字導航。在 macOS 上需要將 Option 設定為 Meta

主題和顯示

快捷鍵	說明	上下文
`Ctrl+T`	切換程式碼區塊的語法醒目提示	僅在 `/theme` 選擇器選單內有效。控制 Claude 回應中的程式碼是否使用語法著色

> **Note:**
>
> 語法醒目提示僅在 Claude Code 的原生版本中可用。

多行輸入

方法	快捷鍵	上下文
快速逃脫	`\` + `Enter`	適用於所有終端
macOS 預設	`Option+Enter`	macOS 上的預設
Shift+Enter	`Shift+Enter`	在 iTerm2、WezTerm、Ghostty、Kitty 中開箱即用
控制序列	`Ctrl+J`	多行的換行符
貼上模式	直接貼上	適用於程式碼區塊、日誌

> **Tip:**

> Shift+Enter 在 iTerm2、WezTerm、Ghostty 和 Kitty 中無需配置即可使用。對於其他終端（VS Code、Alacritty、Zed、Warp），執行 `/terminal-setup` 以安裝繫結。

快速命令

快捷鍵	說明	備註
`/` 在開始	命令或 skill	請參閱內建命令和 skills
`!` 在開始	Bash 模式	直接執行命令並將執行輸出新增到會話
`@`	檔案路徑提及	觸發檔案路徑自動完成

內建命令

在 Claude Code 中輸入 `/` 以查看所有可用命令，或輸入 `/` 後跟任何字母以篩選。並非所有命令對每個使用者都可見。有些取決於您的平台、計畫或環境。例如，`/desktop` 僅在 macOS 和 Windows 上出現，`/upgrade` 和 `/privacy-settings` 僅在 Pro 和 Max 計畫上可用，而 `/terminal-setup` 在您的終端原生支援其快捷鍵時隱藏。

Claude Code 還附帶捆綁的 skills，例如 `/simplify`、`/batch` 和 `/debug`，當您輸入 `/` 時會與內建命令一起出現。若要建立您自己的命令，請參閱 skills。

在下表中，`<arg>` 表示必需的引數，`[arg]` 表示可選的引數。

命令	用途
`/add-dir <path>`	將新的工作目錄新增到目前會話
`/agents`	管理 agent 配置
`/btw <question>`	提出快速側面問題，無需新增到對話
`/chrome`	配置Chrome 中的 Claude 設定
`/clear`	清除對話歷史並釋放上下文。別名：`/reset`、`/new`
`/compact [instructions]`	壓縮對話，可選擇焦點指示

命令	用途
`/config`	開啟設定介面以調整主題、模型、輸出樣式和其他偏好設定。別名：`/settings`
`/context`	將目前上下文使用情況視覺化為彩色網格
`/copy`	將最後一個助手回應複製到剪貼簿。當存在程式碼區塊時，顯示互動式選擇器以選擇個別區塊或完整回應
`/cost`	顯示 token 使用統計資訊。詳見成本追蹤指南以了解訂閱特定詳細資訊
`/desktop`	在 Claude Code 桌面應用程式中繼續目前會話。僅限 macOS 和 Windows。別名：`/app`
`/diff`	開啟互動式差異檢視器，顯示未提交的變更和每個回合的差異。使用左/右箭頭在目前 git 差異和個別 Claude 回合之間切換，使用上/下瀏覽檔案
`/doctor`	診斷並驗證您的 Claude Code 安裝和設定
`/exit`	退出 CLI。別名：`/quit`
`/export [filename]`	將目前對話匯出為純文字。使用檔案名稱時，直接寫入該檔案。不使用時，開啟對話框以複製到剪貼簿或儲存到檔案
`/extra-usage`	配置額外使用量以在達到速率限制時繼續工作
`/fast [on|off]`	切換 fast mode 開啟或關閉
`/feedback [report]`	提交有關 Claude Code 的意見反應。別名：`/bug`
`/fork [name]`	在此點建立目前對話的分支
`/help`	顯示說明和可用命令
`/hooks`	管理工具事件的 hook 配置
`/ide`	管理 IDE 整合並顯示狀態
`/init`	使用 `CLAUDE.md` 指南初始化專案
`/insights`	生成分析您的 Claude Code 會話的報告，包括專案區域、互動模式和摩擦點
`/install-github-app`	為存放庫設定 Claude GitHub Actions 應用程式。引導您選擇存放庫並配置整合

命令	用途
`/install-slack-app`	安裝 Claude Slack 應用程式。開啟瀏覽器以完成 OAuth 流程
`/keybindings`	開啟或建立您的快捷鍵配置檔案
`/login`	登入您的 Anthropic 帳戶
`/logout`	登出您的 Anthropic 帳戶
`/mcp`	管理 MCP server 連線和 OAuth 驗證
`/memory`	編輯 `CLAUDE.md` 記憶檔案、啟用或停用 auto-memory，以及檢視自動記憶項目
`/mobile`	顯示 QR 碼以下載 Claude 行動應用程式。別名：`/ios`、`/android`
`/model [model]`	選擇或變更 AI 模型。對於支援的模型，使用左/右箭頭調整努力等級。變更立即生效，無需等待目前回應完成
`/passes`	與朋友分享免費一週的 Claude Code。僅在您的帳戶符合條件時可見
`/permissions`	檢視或更新權限。別名：`/allowed-tools`
`/plan`	直接從提示進入 Plan Mode
`/plugin`	管理 Claude Code plugins
`/pr-comments [PR]`	從 GitHub pull request 擷取並顯示評論。自動偵測目前分支的 PR，或傳遞 PR URL 或編號。需要 `gh` CLI
`/privacy-settings`	檢視和更新您的隱私設定。僅適用於 Pro 和 Max 計畫訂閱者
`/release-notes`	檢視完整變更日誌，最新版本最接近您的提示
`/reload-plugins`	重新載入所有作用中的 plugins 以套用待處理變更，無需重新啟動。報告已載入的內容並記錄需要重新啟動的任何變更
`/remote-control`	使此會話可從 claude.ai 進行遠端控制。別名：`/rc`
`/remote-env`	為 teleport 會話配置預設遠端環境
`/rename [name]`	重新命名目前會話。不使用名稱時，從對話歷史自動生成

命令	用途
`/resume [session]`	按 ID 或名稱繼續對話，或開啟會話選擇器。別名：`/continue`
`/review`	已棄用。改為安裝 `code-review` plugin：`claude plugin install code-review@claude-code-marketplace`
`/rewind`	回溯對話和/或程式碼到先前的點，或從選定的訊息進行摘要。詳見 checkpointing。別名：`/checkpoint`
`/sandbox`	切換 sandbox mode。僅在支援的平台上可用
`/security-review`	分析目前分支上的待處理變更以查找安全漏洞。檢查 git 差異並識別注入、驗證問題和資料洩露等風險
`/skills`	列出可用的 skills
`/stats`	視覺化每日使用情況、會話歷史、連勝和模型偏好
`/status`	開啟設定介面（狀態標籤），顯示版本、模型、帳戶和連線性
`/statusline`	配置 Claude Code 的狀態行。描述您想要的內容，或不帶引數執行以從您的 shell 提示自動配置
`/stickers`	訂購 Claude Code 貼紙
`/tasks`	列出並管理背景工作
`/terminal-setup`	為 Shift+Enter 和其他快捷鍵配置終端快捷鍵。僅在需要它的終端中可見，例如 VS Code、Alacritty 或 Warp
`/theme`	變更色彩主題。包括淺色和深色變體、色盲無障礙（daltonized）主題和使用終端色彩調色板的 ANSI 主題
`/upgrade`	開啟升級頁面以切換到更高的計畫層級
`/usage`	顯示計畫使用限制和速率限制狀態
`/vim`	在 Vim 和正常編輯模式之間切換

MCP prompts

MCP servers 可以公開顯示為命令的 prompts。這些使用格式 `/mcp__<server>__<prompt>`，並從連線的 servers 動態發現。詳見 MCP prompts。

Vim 編輯器模式

使用 `/vim` 命令啟用 vim 風格編輯，或透過 `/config` 永久配置。

模式切換

命令	動作	來自模式
`Esc`	進入 NORMAL 模式	INSERT
`i`	在游標前插入	NORMAL
`I`	在行首插入	NORMAL
`a`	在游標後插入	NORMAL
`A`	在行尾插入	NORMAL
`o`	在下方開啟行	NORMAL
`O`	在上方開啟行	NORMAL

導航（NORMAL 模式）

命令	動作
`h` / `j` / `k` / `l`	向左/下/上/右移動
`w`	下一個單字
`e`	單字結尾
`b`	上一個單字
`0`	行首
`$`	行尾
`^`	第一個非空白字元
`gg`	輸入開始
`G`	輸入結尾
`f{char}`	跳到下一個字元出現
`F{char}`	跳到上一個字元出現

命令	動作
`t{char}`	跳到下一個字元出現之前
`T{char}`	跳到上一個字元出現之後
`;`	重複最後一個 f/F/t/T 動作
`,`	反向重複最後一個 f/F/t/T 動作

> **Note:**
> 在 vim 正常模式中，如果游標在輸入的開始或結尾，無法進一步移動，箭頭鍵將導航命令歷史。

編輯（NORMAL 模式）

命令	動作
`x`	刪除字元
`dd`	刪除行
`D`	刪除到行尾
`dw` / `de` / `db`	刪除單字/到結尾/向後
`cc`	變更行
`C`	變更到行尾
`cw` / `ce` / `cb`	變更單字/到結尾/向後
`yy` / `Y`	複製（yank）行
`yw` / `ye` / `yb`	複製單字/到結尾/向後
`p`	在游標後貼上
`P`	在游標前貼上
`>>`	縮排行
`<<`	取消縮排行
`J`	合併行

命令	動作
`.`	重複最後一個變更

文字物件（NORMAL 模式）

文字物件與運算子（如 `d`、`c` 和 `y`）搭配使用：

命令	動作
`iw` / `aw`	內部/周圍單字
`iW` / `aW`	內部/周圍 WORD（空白分隔）
`i"` / `a"`	內部/周圍雙引號
`i'` / `a'`	內部/周圍單引號
`i(` / `a(`	內部/周圍括號
`i[` / `a[`	內部/周圍方括號
`i{` / `a{`	內部/周圍大括號

命令歷史

Claude Code 維護目前會話的命令歷史：

- 輸入歷史按工作目錄儲存
- 當您執行 `/clear` 以啟動新會話時，輸入歷史會重設。先前會話的對話會保留，可以繼續。
- 使用上/下箭頭導航（請參閱上面的鍵盤快捷鍵）
- **注意**：歷史擴展（`!`）預設停用

使用 Ctrl+R 反向搜尋

按 `Ctrl+R` 以互動方式搜尋您的命令歷史：

1. **開始搜尋**：按 `Ctrl+R` 啟動反向歷史搜尋
2. **輸入查詢**：輸入文字以在先前的命令中搜尋。搜尋詞在匹配結果中醒目提示
3. **導航匹配**：再次按 `Ctrl+R` 以循環瀏覽較舊的匹配
4. **接受匹配**：
    - 按 `Tab` 或 `Esc` 以接受目前匹配並繼續編輯

- 按 `Enter` 以接受並立即執行命令

5. **取消搜尋**：
   - 按 `Ctrl+C` 以取消並恢復您的原始輸入
   - 在空搜尋上按 `Backspace` 以取消

搜尋顯示匹配的命令，搜尋詞醒目提示，因此您可以找到並重複使用先前的輸入。

背景 bash 命令

Claude Code 支援在背景執行 bash 命令，允許您在長時間執行的程序執行時繼續工作。

背景執行的工作原理

當 Claude Code 在背景執行命令時，它以非同步方式執行命令並立即傳回背景工作 ID。Claude Code 可以在命令在背景繼續執行時回應新的提示。

若要在背景執行命令，您可以：

- 提示 Claude Code 在背景執行命令
- 按 Ctrl+B 將常規 Bash 工具呼叫移到背景。（Tmux 使用者必須按 Ctrl+B 兩次，因為 tmux 的前綴鍵。）

主要功能：

- 輸出被緩衝，Claude 可以使用 TaskOutput 工具擷取它
- 背景工作有唯一的 ID 用於追蹤和輸出擷取
- 當 Claude Code 退出時，背景工作會自動清理

若要停用所有背景工作功能，請將 `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` 環境變數設定為 `1`。詳見環境變數。

常見的背景執行命令：

- 建置工具（webpack、vite、make）
- 套件管理器（npm、yarn、pnpm）
- 測試執行器（jest、pytest）
- 開發伺服器
- 長時間執行的程序（docker、terraform）

使用 `!` 前綴的 Bash 模式

透過在輸入前加上 `!` 直接執行 bash 命令，無需透過 Claude：

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

Bash 模式：

- 將命令及其輸出新增到對話上下文
- 顯示即時進度和輸出
- 支援相同的 `Ctrl+B` 背景執行以用於長時間執行的命令
- 不需要 Claude 解釋或批准命令
- 支援基於歷史的自動完成：輸入部分命令並按 **Tab** 以從目前專案中的先前 `!` 命令完成
- 在空提示上使用 `Escape`、`Backspace` 或 `Ctrl+U` 退出

這對於快速 shell 操作同時維護對話上下文很有用。

提示建議

當您首次開啟會話時，灰色的範例命令會出現在提示輸入中以幫助您開始。Claude Code 從您的專案的 git 歷史中選擇此項，因此它反映您最近一直在處理的檔案。

Claude 回應後，建議會根據您的對話歷史繼續出現，例如多部分請求的後續步驟或工作流程的自然延續。

- 按 **Tab** 以接受建議，或按 **Enter** 以接受並提交
- 開始輸入以關閉它

建議作為背景請求執行，重複使用父對話的 prompt cache，因此額外成本最少。當 cache 冷時，Claude Code 會跳過建議生成以避免不必要的成本。

建議在對話的第一個回合之後、在非互動模式中以及在 Plan Mode 中自動跳過。

若要完全停用提示建議，請設定環境變數或在 `/config` 中切換設定：

```
export CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION=false
```

使用 /btw 的側面問題

使用 `/btw` 提出有關您目前工作的快速問題，無需新增到對話歷史。當您想要快速答案但不想雜亂主要上下文或使 Claude 偏離長時間執行的工作時，這很有用。

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

側面問題可以完全看到目前對話，因此您可以詢問 Claude 已經讀過的程式碼、它之前做出的決定或會話中的任何其他內容。問題和答案是短暫的：它們出現在可關閉的覆蓋層中，永遠不會進入對話歷史。

- **在 Claude 工作時可用**：即使 Claude 正在處理回應時，您也可以執行 `/btw`。側面問題獨立執行，不會中斷主要回合。
- **無工具存取**：側面問題僅從已在上下文中的內容回答。Claude 在回答側面問題時無法讀取檔案、執行命令或搜尋。
- **單一回應**：沒有後續回合。如果您需要來回往返，請改用正常提示。
- **低成本**：側面問題重複使用父對話的 prompt cache，因此額外成本最少。

按 **Space**、**Enter** 或 **Escape** 以關閉答案並返回提示。

`/btw` 是 subagent 的反面：它看到您的完整對話但沒有工具，而 subagent 有完整工具但以空上下文開始。使用 `/btw` 詢問 Claude 已經從此會話中知道的內容；使用 subagent 去發現新的東西。

工作清單

在處理複雜的多步驟工作時，Claude 會建立工作清單以追蹤進度。工作會在終端的狀態區域中出現，指示器顯示待處理、進行中或完成的內容。

- 按 `Ctrl+T` 以切換工作清單檢視。顯示一次最多 10 個工作
- 若要查看所有工作或清除它們，直接詢問 Claude：「show me all tasks」或「clear all tasks」
- 工作在上下文壓縮中持續，幫助 Claude 在較大的專案上保持組織
- 若要在會話之間共享工作清單，請設定 `CLAUDE_CODE_TASK_LIST_ID` 以使用 `~/.claude/tasks/` 中的命名目錄：`CLAUDE_CODE_TASK_LIST_ID=my-project claude`
- 若要還原到先前的 TODO 清單，請設定 `CLAUDE_CODE_ENABLE_TASKS=false`。

PR 審查狀態

在處理具有開啟 pull request 的分支時，Claude Code 在頁尾顯示可點擊的 PR 連結（例如「PR #446」）。連結有一個彩色底線，指示審查狀態：

- 綠色：已批准
- 黃色：待審查
- 紅色：要求變更

- 灰色：草稿
- 紫色：已合併

`Cmd+click`（Mac）或 `Ctrl+click`（Windows/Linux）連結以在瀏覽器中開啟 pull request。狀態每 60 秒自動更新。

> **Note:**
>
> PR 狀態需要安裝並驗證 `gh` CLI（`gh auth login`）。

另請參閱

- Skills - 自訂提示和工作流程
- Checkpointing - 回溯 Claude 的編輯並恢復先前的狀態
- CLI 參考 - 命令列旗標和選項
- 設定 - 配置選項
- 記憶管理 - 管理 CLAUDE.md 檔案

Claude Code 最佳實踐

> 從配置環境到跨平行會話擴展，充分利用 Claude Code 的提示和模式。

Claude Code 是一個代理式編碼環境。與等待回答問題的聊天機器人不同，Claude Code 可以讀取您的文件、運行命令、進行更改，並在您觀看、重定向或完全離開時自主解決問題。

這改變了您的工作方式。與其自己編寫代碼並要求 Claude 審查，不如描述您想要的內容，讓 Claude 找出如何構建它。Claude 會探索、規劃和實施。

但這種自主性仍然伴隨著學習曲線。Claude 在您需要理解的某些約束條件下工作。

本指南涵蓋了在 Anthropic 內部團隊和在各種代碼庫、語言和環境中使用 Claude Code 的工程師中已被證明有效的模式。有關代理循環如何在幕後工作的信息，請參閱 Claude Code 如何工作。

大多數最佳實踐都基於一個約束：Claude 的 context window 填滿得很快，隨著填滿，性能會下降。

Claude 的 context window 保存您的整個對話，包括每條消息、Claude 讀取的每個文件和每個命令輸出。但是，這可能會很快填滿。單個調試會話或代碼庫探索可能會生成並消耗數萬個令牌。

這很重要，因為隨著 context 填滿，LLM 性能會下降。當 context window 即將滿時，Claude 可能會開始「遺忘」早期的指令或犯更多錯誤。context window 是最重要的資源來管理。使用 自定義狀態行 持續跟蹤 context 使用情況，並查看 減少令牌使用 以了解減少令牌使用的策略。

給 Claude 一種驗證其工作的方式

> **Tip:**
>
> 包括測試、截圖或預期輸出，以便 Claude 可以檢查自己。這是您可以做的最高槓桿的事情。

當 Claude 能夠驗證自己的工作時，例如運行測試、比較截圖和驗證輸出，Claude 的表現會大幅提高。

沒有明確的成功標準，它可能會產生看起來正確但實際上不起作用的東西。您成為唯一的反饋循環，每個錯誤都需要您的關注。

策略	之前	之後
提供驗證標準	「實現一個驗證電子郵件地址的函數」	「編寫一個 validateEmail 函數。示例測試用例：user@example.com 為真，invalid 為假，user@.com 為假。實施後運行測試」
以視覺方式驗證 UI 更改	「使儀表板看起來更好」	「[粘貼截圖] 實施此設計。對結果進行截圖並與原始設計進行比較。列出差異並修復它們」
解決根本原因，而不是症狀	「構建失敗」	「構建失敗，出現此錯誤：[粘貼錯誤]。修復它並驗證構建成功。解決根本原因，不要抑制錯誤」

UI 更改可以使用 Claude Chrome 擴展 進行驗證。它在您的瀏覽器中打開新標籤頁，測試 UI，並迭代直到代碼工作。

您的驗證也可以是測試套件、linter 或檢查輸出的 Bash 命令。投資使您的驗證堅如磐石。

先探索，然後規劃，然後編碼

> **Tip:**
> 將研究和規劃與實施分開，以避免解決錯誤的問題。

讓 Claude 直接跳到編碼可能會產生解決錯誤問題的代碼。使用 Plan Mode 將探索與執行分開。

推薦的工作流程有四個階段：

Step 1: 探索

進入 Plan Mode。Claude 讀取文件並回答問題，不進行任何更改。

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

Step 2: 規劃

要求 Claude 創建詳細的實施計劃。

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

按 `Ctrl+G` 在文本編輯器中打開計劃進行直接編輯，然後 Claude 再繼續。

Step 3: 實施

切換回正常模式，讓 Claude 編碼，根據其計劃進行驗證。

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

Step 4: 提交

要求 Claude 使用描述性消息進行提交並創建 PR。

```
commit with a descriptive message and open a PR
```

Plan Mode 很有用，但也增加了開銷。

對於範圍明確且修復很小的任務（如修復拼寫錯誤、添加日誌行或重命名變量），直接要求 Claude 執行。

當您對方法不確定、更改修改多個文件或您不熟悉被修改的代碼時，規劃最有用。如果您可以用一句話描述 diff，請跳過計劃。

在提示中提供具體的上下文

> **Tip:**
> 您的指令越精確，您需要的更正就越少。

Claude 可以推斷意圖，但無法讀心術。參考特定文件、提及約束條件並指出示例模式。

策略	之前	之後
限定任務範圍。 指定哪個文件、什麼場景和測試偏好。	「為 foo.py 添加測試」	「為 foo.py 編寫測試，涵蓋用戶已登出的邊界情況。避免使用 mocks。」
指向來源。 指導 Claude 到可以回答問題的來源。	「為什麼 ExecutionFactory 有這樣奇怪的 api？」	「查看 ExecutionFactory 的 git 歷史記錄並總結其 api 是如何形成的」
參考現有模式。 指向代碼庫中的模式。	「添加日曆小部件」	「查看主頁上現有小部件的實施方式以了解模式。HotDogWidget.php 是一個很好的例子。按照模式實施一個新的日曆小部件，讓用戶選擇月份並向前/向後分頁以選擇年份。從頭開始構建，除了代碼庫中已使用的庫外，不使用其他庫。」
描述症狀。 提供症狀、可能的位置以及「修復」的樣子。	「修復登錄錯誤」	「用戶報告會話超時後登錄失敗。檢查 src/auth/ 中的身份驗證流程，特別是令牌刷新。編寫一個失敗的測試來重現問題，然後修復它」

當您在探索並可以承受改正時，模糊的提示可能很有用。像 `「您會改進此文件中的什麼？」` 這樣的提示可以表面您沒有想到要詢問的內容。

提供豐富的內容

> **Tip:**
>
> 使用 `@` 參考文件、粘貼截圖/圖像或直接管道數據。

您可以通過多種方式向 Claude 提供豐富的數據：

- **使用 `@` 參考文件**，而不是描述代碼的位置。Claude 在回應前讀取文件。
- **直接粘貼圖像**。複製/粘貼或將圖像拖放到提示中。
- **提供 URL** 用於文檔和 API 參考。使用 `/permissions` 將常用域名列入白名單。
- **通過運行 `cat error.log | claude` 管道數據**，直接發送文件內容。

- **讓 Claude 獲取它需要的內容**。告訴 Claude 使用 Bash 命令、MCP 工具或通過讀取文件自己拉取上下文。

配置您的環境

一些設置步驟使 Claude Code 在所有會話中的效果顯著提高。有關擴展功能的完整概述和何時使用每個功能，請參閱 擴展 Claude Code。

編寫有效的 CLAUDE.md

> **Tip:**
>
> 運行 `/init` 根據您當前的項目結構生成一個啟動 CLAUDE.md 文件，然後隨著時間推移進行改進。

CLAUDE.md 是一個特殊文件，Claude 在每次對話開始時都會讀取。包括 Bash 命令、代碼風格和工作流規則。這給 Claude 提供了它無法從代碼中推斷出的持久上下文。

`/init` 命令分析您的代碼庫以檢測構建系統、測試框架和代碼模式，為您提供堅實的基礎進行改進。

CLAUDE.md 文件沒有必需的格式，但要保持簡短和易於閱讀。例如：

```
## 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 在每個會話中加載，因此只包括廣泛適用的內容。對於僅在某些時候相關的域知識或工作流，請改用 skills。Claude 按需加載它們，不會使每次對話都變得臃腫。

保持簡潔。對於每一行，問自己：「刪除這一行會導致 Claude 犯錯誤嗎？」 如果不會，刪除它。臃腫的 CLAUDE.md 文件會導致 Claude 忽略您的實際指令！

✅包括	❌排除
Claude 無法猜測的 Bash 命令	Claude 可以通過讀取代碼找出的任何內容

✅包括	❌排除
與默認值不同的代碼風格規則	Claude 已經知道的標準語言約定
測試指令和首選測試運行器	詳細的 API 文檔（改為鏈接到文檔）
存儲庫禮儀（分支命名、PR 約定）	經常變化的信息
特定於您項目的架構決策	長篇解釋或教程
開發人員環境怪癖（必需的環境變量）	文件逐個描述代碼庫
常見陷阱或非顯而易見的行為	自明的實踐，如「編寫乾淨代碼」

如果 Claude 儘管有反對規則仍然不斷做您不想要的事情，該文件可能太長，規則被遺漏了。如果 Claude 詢問您在 CLAUDE.md 中回答的問題，措辭可能不明確。像對待代碼一樣對待 CLAUDE.md：當事情出錯時進行審查，定期修剪，並通過觀察 Claude 的行為是否實際改變來測試更改。

您可以通過添加強調（例如「IMPORTANT」或「YOU MUST」）來調整指令以改進遵守。將文件簽入 git，以便您的團隊可以貢獻。該文件的價值隨著時間的推移而複合。

CLAUDE.md 文件可以使用 `@path/to/import` 語法導入其他文件：

```
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
```

您可以將 CLAUDE.md 文件放在多個位置：

- **主文件夾（`~/.claude/CLAUDE.md`）**：適用於所有 Claude 會話
- **項目根目錄（`./CLAUDE.md`）**：簽入 git 以與您的團隊共享
- **父目錄**：對於 monorepos 很有用，其中 `root/CLAUDE.md` 和 `root/foo/CLAUDE.md` 都會自動拉入
- **子目錄**：當在這些目錄中的文件上工作時，Claude 按需拉入子 CLAUDE.md 文件

配置權限

> **Tip:**

> 使用 `/permissions` 將安全命令列入白名單或使用 `/sandbox` 進行操作系統級隔離。這減少了中斷，同時讓您保持控制。

默認情況下，Claude Code 請求可能修改您的系統的操作的權限：文件寫入、Bash 命令、MCP 工具等。這是安全的但很繁瑣。在第十次批准後，您實際上不是在審查，而是在點擊。有兩種方法可以減少這些中斷：

- **權限白名單**：允許您知道安全的特定工具（如 `npm run lint` 或 `git commit`）
- **沙箱**：啟用操作系統級隔離，限制文件系統和網絡訪問，允許 Claude 在定義的邊界內更自由地工作

或者，使用 `--dangerously-skip-permissions` 繞過所有權限檢查以進行包含的工作流，如修復 lint 錯誤或生成樣板。

> **Warning:**
>
> 讓 Claude 運行任意命令可能導致數據丟失、系統損壞或通過提示注入進行數據滲漏。僅在沒有互聯網訪問的沙箱中使用 `--dangerously-skip-permissions`。

閱讀更多關於 配置權限 和 啟用沙箱。

使用 CLI 工具

> **Tip:**
>
> 告訴 Claude Code 在與外部服務交互時使用 CLI 工具，如 `gh`、`aws`、`gcloud` 和 `sentry-cli`。

CLI 工具是與外部服務交互的最 context 高效的方式。如果您使用 GitHub，請安裝 `gh` CLI。Claude 知道如何使用它來創建問題、打開拉取請求和讀取評論。沒有 `gh`，Claude 仍然可以使用 GitHub API，但未經身份驗證的請求經常會達到速率限制。

Claude 也很擅長學習它不知道的 CLI 工具。嘗試像 `Use 'foo-cli-tool --help' to learn about foo tool, then use it to solve A, B, C.` 這樣的提示。

連接 MCP servers

> **Tip:**
>
> 運行 `claude mcp add` 以連接外部工具，如 Notion、Figma 或您的數據庫。

使用 MCP servers，您可以要求 Claude 從問題跟蹤器實施功能、查詢數據庫、分析監控數據、集成來自 Figma 的設計並自動化工作流。

設置 hooks

> **Tip:**
>
> 使用 hooks 進行必須每次發生且沒有例外的操作。

Hooks 在 Claude 工作流中的特定點自動運行腳本。與建議性的 CLAUDE.md 指令不同，hooks 是確定性的，保證操作發生。

Claude 可以為您編寫 hooks。嘗試像 「編寫一個在每次文件編輯後運行 eslint 的 hook」 或 「編寫一個阻止寫入遷移文件夾的 hook。」 這樣的提示。運行 `/hooks` 進行交互式配置，或直接編輯 `.claude/settings.json`。

創建 skills

> **Tip:**
>
> 在 `.claude/skills/` 中創建 `SKILL.md` 文件，為 Claude 提供域知識和可重用工作流。

Skills 使用特定於您的項目、團隊或域的信息擴展 Claude 的知識。Claude 在相關時自動應用它們，或者您可以使用 `/skill-name` 直接調用它們。

通過將目錄與 `SKILL.md` 添加到 `.claude/skills/` 來創建 skill：

```
---
name: api-conventions
description: REST API design conventions for our services
---
## API Conventions
- Use kebab-case for URL paths
- Use camelCase for JSON properties
- Always include pagination for list endpoints
- Version APIs in the URL path (/v1/, /v2/)
```

Skills 也可以定義您直接調用的可重複工作流：

```
---
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
```

運行 `/fix-issue 1234` 來調用它。對於具有您想手動觸發的副作用的工作流，使用 `disable-model-invocation: true`。

創建自定義 subagents

> **Tip:**
>
> 在 `.claude/agents/` 中定義專門的助手，Claude 可以委派給它們進行隔離的任務。

Subagents 在自己的 context 中運行，具有自己的一組允許的工具。它們對於讀取許多文件或需要專門關注而不會使主對話變得混亂的任務很有用。

```
---
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.
```

明確告訴 Claude 使用 subagents：「使用 subagent 審查此代碼以查找安全問題。」

安裝 plugins

> **Tip:**
>
> 運行 `/plugin` 瀏覽市場。Plugins 無需配置即可添加 skills、工具和集成。

Plugins 將 skills、hooks、subagents 和 MCP servers 捆綁到來自社區和 Anthropic 的單個可安裝單元中。如果您使用類型化語言，請安裝 代碼智能 plugin 以為 Claude 提供精確的符號導航和編輯後的自動錯誤檢測。

有關在 skills、subagents、hooks 和 MCP 之間選擇的指導，請參閱 擴展 Claude Code。

有效溝通

您與 Claude Code 溝通的方式會顯著影響結果的質量。

詢問代碼庫問題

> **Tip:**
>
> 詢問 Claude 您會問資深工程師的問題。

當加入新代碼庫時，使用 Claude Code 進行學習和探索。您可以詢問 Claude 與詢問另一位工程師相同類型的問題：

- 日誌記錄如何工作？
- 我如何創建新的 API 端點？
- `foo.rs` 第 134 行的 `async move { ... }` 做什麼？
- `CustomerOnboardingFlowImpl` 處理哪些邊界情況？
- 為什麼此代碼在第 333 行調用 `foo()` 而不是 `bar()` ？

以這種方式使用 Claude Code 是一個有效的入職工作流程，改進了入職時間並減少了對其他工程師的負擔。無需特殊提示：直接提出問題。

讓 Claude 採訪您

> **Tip:**
>
> 對於較大的功能，讓 Claude 先採訪您。從最小的提示開始，並要求 Claude 使用 `AskUserQuestion` 工具採訪您。

Claude 會詢問您可能還沒有考慮的事情，包括技術實施、UI/UX、邊界情況和權衡。

```
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.
```

規格完成後，開始新會話以執行它。新會話具有完全專注於實施的乾淨 context，您有一個書面規格可供參考。

管理您的會話

對話是持久的和可逆的。利用這一點！

及早且經常改正方向

> **Tip:**
>
> 一旦您注意到 Claude 偏離軌道，立即改正。

最好的結果來自緊密的反饋循環。儘管 Claude 有時會在第一次嘗試時完美地解決問題，但快速改正通常會更快地產生更好的解決方案。

- `Esc`：使用 `Esc` 鍵在中途停止 Claude。Context 被保留，所以您可以重定向。
- `Esc + Esc` **或** `/rewind`：按 `Esc` 兩次或運行 `/rewind` 打開倒帶菜單並恢復之前的對話和代碼狀態，或從選定的消息進行總結。
- `「撤銷那個」`：讓 Claude 恢復其更改。
- `/clear`：在不相關的任務之間重置 context。具有不相關 context 的長會話可能會降低性能。

如果您在一個會話中對同一問題改正了 Claude 超過兩次，context 就會被失敗的方法所污染。運行 `/clear` 並使用更具體的提示重新開始，該提示包含您學到的內容。具有更好提示的乾淨會話幾乎總是優於具有累積改正的長會話。

積極管理 context

> **Tip:**
>
> 在不相關的任務之間運行 `/clear` 以重置 context。

當您接近 context 限制時，Claude Code 會自動壓縮對話歷史記錄，這保留了重要的代碼和決策，同時釋放空間。

在長會話期間，Claude 的 context window 可能會充滿不相關的對話、文件內容和命令。這可能會降低性能，有時會分散 Claude 的注意力。

- 在任務之間頻繁使用 `/clear` 以完全重置 context window
- 當自動壓縮觸發時，Claude 總結最重要的內容，包括代碼模式、文件狀態和關鍵決策
- 為了更好地控制，運行 `/compact <instructions>`，如 `/compact Focus on the API changes`
- 要僅壓縮對話的一部分，使用 `Esc + Esc` 或 `/rewind`，選擇消息檢查點，然後選擇 **從此處進行總結**。這會壓縮該點之後的消息，同時保持早期 context 完整。
- 在 CLAUDE.md 中使用像 `「壓縮時，始終保留完整的修改文件列表和任何測試命令」` 這樣的指令自定義壓縮行為，以確保關鍵 context 在總結中存活。

- 對於不需要留在 context 中的快速問題，使用 `/btw`。答案出現在可關閉的覆蓋層中，永遠不會進入對話歷史記錄，所以您可以檢查詳細信息而不會增加 context。

使用 subagents 進行調查

> **Tip:**
>
> 使用 `「使用 subagents 調查 X」` 委派研究。他們在單獨的 context 中探索，為實施保持您的主對話乾淨。

由於 context 是您的基本約束，subagents 是可用的最強大的工具之一。當 Claude 研究代碼庫時，它讀取許多文件，所有這些都會消耗您的 context。Subagents 在單獨的 context windows 中運行並報告回摘要：

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

subagent 探索代碼庫、讀取相關文件並報告發現，所有這些都不會使您的主對話變得混亂。

您也可以在 Claude 實施某些內容後使用 subagents 進行驗證：

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

使用檢查點倒帶

> **Tip:**
>
> Claude 進行的每個操作都會創建一個檢查點。您可以將對話、代碼或兩者恢復到任何之前的檢查點。

Claude 在更改前自動檢查點。雙擊 `Escape` 或運行 `/rewind` 打開倒帶菜單。您可以僅恢復對話、僅恢復代碼、恢復兩者或從選定的消息進行總結。有關詳細信息，請參閱 Checkpointing。

與其仔細規劃每一步，不如告訴 Claude 嘗試一些冒險的事情。如果不起作用，倒帶並嘗試不同的方法。檢查點在會話之間持續，所以您可以關閉終端並稍後仍然倒帶。

> **Warning:**

檢查點僅跟蹤 Claude 進行的更改，不跟蹤外部進程。這不是 git 的替代品。

恢復對話

Tip:

運行 `claude --continue` 以從中斷的地方繼續，或 `--resume` 以從最近的會話中選擇。

Claude Code 在本地保存對話。當任務跨越多個會話時，您不必重新解釋 context：

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

使用 `/rename` 給會話起描述性名稱，如 `「oauth-migration」` 或 `「debugging-memory-leak」`，以便您稍後可以找到它們。像對待分支一樣對待會話：不同的工作流可以有單獨的、持久的 contexts。

自動化和擴展

一旦您對一個 Claude 有效，通過平行會話、非交互模式和扇出模式將您的輸出乘以倍數。

到目前為止，一切都假設一個人、一個 Claude 和一個對話。但 Claude Code 水平擴展。本節中的技術展示了您如何完成更多工作。

運行非交互模式

Tip:

在 CI、pre-commit hooks 或腳本中使用 `claude -p "prompt"`。添加 `--output-format stream-json` 以獲得流式 JSON 輸出。

使用 `claude -p "your prompt"`，您可以非交互地運行 Claude，不需要會話。非交互模式是您將 Claude 集成到 CI 管道、pre-commit hooks 或任何自動化工作流中的方式。輸出格式讓您以編程方式解析結果：純文本、JSON 或流式 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
```

運行多個 Claude 會話

> **Tip:**
> 並行運行多個 Claude 會話以加快開發、運行隔離的實驗或啟動複雜的工作流。

有三種主要方式來運行平行會話：

- Claude Code 桌面應用：以視覺方式管理多個本地會話。每個會話都有自己的隔離 worktree。
- Claude Code 在網絡上：在 Anthropic 的安全雲基礎設施上在隔離的 VM 中運行。
- Agent teams：多個會話的自動協調，具有共享任務、消息和團隊領導。

除了並行化工作外，多個會話還支持質量聚焦的工作流。新鮮的 context 改進代碼審查，因為 Claude 不會偏向於它剛剛編寫的代碼。

例如，使用 Writer/Reviewer 模式：

會話 A（Writer）	會話 B（Reviewer）
`實施我們 API 端點的速率限制器`	
	`審查 @src/middleware/rateLimiter.ts 中的速率限制器實施。查找邊界情況、競態條件和與我們現有中間件模式的一致性。`
`這是審查反饋：[會話 B 輸出]。解決這些問題。`	

您可以對測試做類似的事情：讓一個 Claude 編寫測試，然後另一個編寫代碼來通過它們。

跨文件扇出

> **Tip:**
> 循環遍歷任務，為每個任務調用 `claude -p` 。使用 `--allowedTools` 為批量操作限定權限。

對於大型遷移或分析，您可以在許多平行 Claude 調用中分配工作：

Step 1: 生成任務列表

讓 Claude 列出所有需要遷移的文件（例如，`列出所有 2,000 個需要遷移的 Python 文件`）

Step 2: 編寫腳本以循環遍歷列表

```
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: 在幾個文件上測試，然後大規模運行

根據前 2-3 個文件出現的問題改進您的提示，然後在完整集合上運行。`--allowedTools` 標誌限制 Claude 可以做什麼，這在您無人值守運行時很重要。

您也可以將 Claude 集成到現有的數據/處理管道中：

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

在開發期間使用 `--verbose` 進行調試，在生產中關閉它。

避免常見的失敗模式

這些是常見的錯誤。及早識別它們可以節省時間：

- **廚房水槽會話。** 您從一個任務開始，然後詢問 Claude 不相關的事情，然後回到第一個任務。Context 充滿了不相關的信息。 > **修復**：在不相關的任務之間使用 `/clear` 。
- **一次又一次地改正。** Claude 做錯了什麼，您改正它，它仍然是錯的，您再次改正。Context 被失敗的方法所污染。 > **修復**：在兩次失敗的改正後，`/clear` 並編寫一個更好的初始提示，包含您學到的內容。

- **過度指定的 CLAUDE.md。** 如果您的 CLAUDE.md 太長，Claude 會忽略其中的一半，因為重要的規則在噪音中丟失了。 > **修復**：無情地修剪。如果 Claude 已經在沒有指令的情況下正確地做某事，刪除它或將其轉換為 hook。
- **信任然後驗證的差距。** Claude 產生看起來合理的實施，但不處理邊界情況。 > **修復**：始終提供驗證（測試、腳本、截圖）。如果您無法驗證它，不要發布它。
- **無限探索。** 您要求 Claude「調查」某些內容而不限定範圍。Claude 讀取數百個文件，填滿 context。 > **修復**：狹隘地限定調查範圍或使用 subagents，以便探索不會消耗您的主 context。

培養您的直覺

本指南中的模式不是一成不變的。它們是通常效果很好的起點，但可能不是每種情況的最優選擇。

有時您_應該_讓 context 累積，因為您深入一個複雜的問題，歷史很有價值。有時您應該跳過規劃，讓 Claude 找出答案，因為任務是探索性的。有時模糊的提示正是您想要的，因為您想在限制它之前看到 Claude 如何解釋問題。

注意什麼有效。當 Claude 產生出色的輸出時，注意您做了什麼：提示結構、您提供的 context、您所在的模式。當 Claude 遇到困難時，問為什麼。Context 太嘈雜了嗎？提示太模糊了嗎？任務對於一次通過來說太大了嗎？

隨著時間的推移，您將培養沒有指南可以捕捉的直覺。您將知道何時具體以及何時開放，何時規劃以及何時探索，何時清除 context 以及何時讓它累積。

相關資源

- Claude Code 如何工作：代理循環、工具和 context 管理
- 擴展 Claude Code：skills、hooks、MCP、subagents 和 plugins
- 常見工作流：調試、測試、PR 等的分步配方
- CLAUDE.md：存儲項目約定和持久 context

常見工作流程

> 使用 Claude Code 探索程式碼庫、修復錯誤、重構、測試和其他日常任務的逐步指南。

本頁涵蓋日常開發的實用工作流程：探索陌生程式碼、除錯、重構、編寫測試、建立 PR 和管理會話。每個部分都包含您可以適應自己專案的範例提示。如需更高層級的模式和提示，請參閱最佳實踐。

了解新的程式碼庫

快速取得程式碼庫概覽

假設您剛加入一個新專案，需要快速了解其結構。

Step 1: 導航到專案根目錄

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

Step 2: 啟動 Claude Code

```
claude
```

Step 3: 要求高層級概覽

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

Step 4: 深入探討特定元件

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

```
what are the key data models?
```

```
how is authentication handled?
```

> **Tip:**
>
> 提示：
>
> - 從廣泛的問題開始，然後縮小到特定領域
> - 詢問專案中使用的編碼慣例和模式
> - 要求專案特定術語的詞彙表

尋找相關程式碼

假設您需要找到與特定功能相關的程式碼。

Step 1: 要求 Claude 尋找相關檔案

```
find the files that handle user authentication
```

Step 2: 取得元件如何互動的背景資訊

```
how do these authentication files work together?
```

Step 3: 了解執行流程

```
trace the login process from front-end to database
```

> **Tip:**
>
> 提示：
>
> - 明確說明您要尋找的內容
> - 使用專案中的領域語言
> - 為您的語言安裝程式碼智能外掛，以便 Claude 進行精確的’前往定義’和’尋找參考’導航

有效地修復錯誤

假設您遇到了錯誤訊息，需要找到並修復其來源。

Step 1: 與 Claude 分享錯誤

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

Step 2: 要求修復建議

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

Step 3: 應用修復

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

> **Tip:**
>
> 提示：
>
> - 告訴 Claude 重現問題的命令並取得堆疊追蹤
> - 提及重現錯誤的任何步驟
> - 讓 Claude 知道錯誤是間歇性的還是持續的

重構程式碼

假設您需要更新舊程式碼以使用現代模式和實踐。

Step 1: 識別用於重構的舊版程式碼

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

Step 2: 取得重構建議

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

Step 3: 安全地應用變更

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

Step 4: 驗證重構

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

> **Tip:**
>
> 提示：
>
> - 要求 Claude 解釋現代方法的優點
> - 在需要時要求變更保持向後相容性
> - 以小的、可測試的增量進行重構

使用專門的 subagents

假設您想使用專門的 AI subagents 來更有效地處理特定任務。

Step 1: 檢視可用的 subagents

```
/agents
```

這會顯示所有可用的 subagents 並讓您建立新的。

Step 2: 自動使用 subagents

Claude Code 自動將適當的任務委派給專門的 subagents：

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

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

Step 3: 明確要求特定的 subagents

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

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

Step 4: 為您的工作流程建立自訂 subagents

```
/agents
```

然後選擇「建立新 subagent」並按照提示定義：

- 描述 subagent 目的的唯一識別碼（例如 `code-reviewer`、`api-designer`）。
- Claude 何時應使用此代理
- 它可以存取哪些工具
- 描述代理角色和行為的系統提示

> **Tip:**
>
> 提示：
>
> - 在 `.claude/agents/` 中建立專案特定的 subagents 以供團隊共享
> - 使用描述性的 `description` 欄位來啟用自動委派
> - 限制工具存取權限為每個 subagent 實際需要的內容
> - 查看subagents 文件以取得詳細範例

使用 Plan Mode 進行安全的程式碼分析

Plan Mode 指示 Claude 通過使用唯讀操作分析程式碼庫來建立計畫，非常適合探索程式碼庫、規劃複雜變更或安全地檢查程式碼。在 Plan Mode 中，Claude 使用 `AskUserQuestion` 在提出計畫之前收集需求並澄清您的目標。

何時使用 Plan Mode

- **多步驟實現**：當您的功能需要編輯許多檔案時
- **程式碼探索**：當您想在更改任何內容之前徹底研究程式碼庫時
- **互動式開發**：當您想與 Claude 迭代方向時

如何使用 Plan Mode

在會話期間開啟 Plan Mode

您可以在會話期間使用 **Shift+Tab** 循環切換權限模式來切換到 Plan Mode。

如果您處於 Normal Mode，**Shift+Tab** 首先切換到 Auto-Accept Mode，在終端底部顯示 `▶▶ accept edits on`。隨後的 **Shift+Tab** 將切換到 Plan Mode，顯示 `⏸ plan mode on`。

在 Plan Mode 中啟動新會話

要在 Plan Mode 中啟動新會話，請使用 `--permission-mode plan` 標誌：

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

在 Plan Mode 中執行「無頭」查詢

您也可以使用 `-p` 直接在 Plan Mode 中執行查詢（即在「無頭模式」中）：

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

範例：規劃複雜的重構

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

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

Claude 分析當前實現並建立全面的計畫。使用後續問題進行細化：

```
What about backward compatibility?
```

```
How should we handle database migration?
```

> **Tip:**
>
> 按 `Ctrl+G` 在您的預設文字編輯器中開啟計畫，您可以在 Claude 繼續之前直接編輯它。

將 Plan Mode 設定為預設值

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

查看設定文件以取得更多設定選項。

使用測試

假設您需要為未涵蓋的程式碼新增測試。

Step 1: 識別未測試的程式碼

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

Step 2: 產生測試框架

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

Step 3: 新增有意義的測試案例

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

Step 4: 執行並驗證測試

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

Claude 可以產生遵循您專案現有模式和慣例的測試。要求測試時，請明確說明您想驗證的行為。Claude 檢查您現有的測試檔案以匹配已在使用的風格、框架和斷言模式。

為了獲得全面的涵蓋範圍，要求 Claude 識別您可能遺漏的邊界情況。Claude 可以分析您的程式碼路徑並建議測試錯誤條件、邊界值和容易忽略的意外輸入。

建立 pull requests

您可以直接要求 Claude 建立 pull requests（「為我的變更建立 pr」），或逐步引導 Claude 完成：

Step 1: 總結您的變更

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

Step 2: 產生 pull request

```
create a pr
```

Step 3: 檢查並細化

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

當您使用 `gh pr create` 建立 PR 時，會話會自動連結到該 PR。您稍後可以使用 `claude --from-pr <number>` 繼續它。

> **Tip:**
> 在提交前檢查 Claude 產生的 PR，並要求 Claude 突出顯示潛在的風險或考慮事項。

處理文件

假設您需要為程式碼新增或更新文件。

Step 1: 識別未記錄的程式碼

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

Step 2: 產生文件

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

Step 3: 檢查並增強

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

Step 4: 驗證文件

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

> **Tip:**
>
> 提示：
>
> - 指定您想要的文件風格（JSDoc、docstrings 等）
> - 要求文件中的範例
> - 要求公開 API、介面和複雜邏輯的文件

使用影像

假設您需要在程式碼庫中使用影像，並希望 Claude 幫助分析影像內容。

Step 1: 將影像新增到對話中

您可以使用以下任何方法：

1. 將影像拖放到 Claude Code 視窗中
2. 複製影像並使用 ctrl+v 將其貼到 CLI 中（不要使用 cmd+v）
3. 向 Claude 提供影像路徑。例如，「分析此影像：/path/to/your/image.png」

Step 2: 要求 Claude 分析影像

```
What does this image show?
```

```
Describe the UI elements in this screenshot
```

```
Are there any problematic elements in this diagram?
```

Step 3: 使用影像作為背景資訊

```
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: 從視覺內容取得程式碼建議

```
Generate CSS to match this design mockup
```

```
What HTML structure would recreate this component?
```

> **Tip:**
>
> 提示：
>
> - 當文字描述不清楚或繁瑣時，使用影像
> - 包含錯誤、UI 設計或圖表的螢幕截圖以獲得更好的背景資訊
> - 您可以在對話中使用多個影像
> - 影像分析適用於圖表、螢幕截圖、模型等
> - 當 Claude 參考影像時（例如 `[Image #1]`），`Cmd+Click`（Mac）或 `Ctrl+Click`（Windows/Linux）連結以在您的預設檢視器中開啟影像

參考檔案和目錄

使用 @ 快速包含檔案或目錄，無需等待 Claude 讀取它們。

Step 1: 參考單個檔案

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

這會在對話中包含檔案的完整內容。

Step 2: 參考目錄

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

這提供了帶有檔案資訊的目錄清單。

Step 3: 參考 MCP 資源

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

這使用 @server:resource 格式從連接的 MCP 伺服器取得資料。查看MCP 資源以取得詳細資訊。

> **Tip:**
>
> 提示：
>
> - 檔案路徑可以是相對的或絕對的
> - @ 檔案參考在檔案的目錄和父目錄中新增 `CLAUDE.md` 到背景資訊
> - 目錄參考顯示檔案清單，而不是內容
> - 您可以在單個訊息中參考多個檔案（例如「@file1.js and @file2.js」）

使用擴展思考（Thinking Mode）

擴展思考預設啟用，為 Claude 提供空間在回應前逐步推理複雜問題。此推理在詳細模式中可見，您可以使用 `Ctrl+O` 切換。

此外，Opus 4.6 引入了自適應推理：不是固定的思考令牌預算，而是模型根據您的努力級別設定動態分配思考。擴展思考和自適應推理一起工作，讓您控制 Claude 在回應前推理的深度。

擴展思考對於複雜的架構決策、具有挑戰性的錯誤、多步驟實現規劃和評估不同方法之間的權衡特別有價值。

> **Note:**
>
> 「think」、「think hard」和「think more」等短語被解釋為常規提示指令，不分配思考令牌。

設定 Thinking Mode

思考預設啟用，但您可以調整或停用它。

範圍	如何設定	詳細資訊
努力級別	在 `/model` 中調整或設定 `CLAUDE_CODE_EFFORT_LEVEL`	控制 Opus 4.6 和 Sonnet 4.6 的思考深度：低、中、高。查看調整努力級別
`ultrathink` **關鍵字**	在您的提示中的任何地方包含「ultrathink」	在 Opus 4.6 和 Sonnet 4.6 上為該輪設定努力為高。對於不需要永久更改努力設定的一次性任務很有用
切換快捷鍵	按 `Option+T`（macOS）或 `Alt+T`（Windows/Linux）	切換當前會話的思考開/關（所有模型）。可能需要終端設定來啟用 Option 鍵快捷鍵
全域預設值	使用 `/config` 切換 Thinking Mode	在所有專案中設定您的預設值（所有模型）。儲存為 `~/.claude/settings.json` 中的 `alwaysThinkingEnabled`
限制令牌預算	設定 `MAX_THINKING_TOKENS` 環境變數	將思考預算限制為特定數量的令牌（在 Opus 4.6 上被忽略，除非設定為 0）。範例：`export MAX_THINKING_TOKENS=10000`

要檢視 Claude 的思考過程，按 `Ctrl+O` 切換詳細模式，並查看顯示為灰色斜體文字的內部推理。

擴展思考如何工作

擴展思考控制 Claude 在回應前執行多少內部推理。更多思考提供更多空間來探索解決方案、分析邊界情況和自我糾正錯誤。

使用 Opus 4.6，思考使用自適應推理：模型根據您選擇的努力級別（低、中、高）動態分配思考令牌。這是調整速度和推理深度之間權衡的推薦方式。

使用其他模型，思考使用固定預算，最多 31,999 個令牌來自您的輸出預算。您可以使用 `MAX_THINKING_TOKENS` 環境變數限制此，或通過 `/config` 或 `Option+T` / `Alt+T` 切換完全停用思考。

`MAX_THINKING_TOKENS` 在 Opus 4.6 和 Sonnet 4.6 上被忽略，因為自適應推理控制思考深度。一個例外：設定 `MAX_THINKING_TOKENS=0` 仍會在任何模型上完全停用思考。要停用自適應思考並恢復為固定思考預算，請設定 `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`。查看環境變數。

> **Warning:**
> 您需要為所有使用的思考令牌付費，即使 Claude 4 模型顯示摘要思考

繼續之前的對話

啟動 Claude Code 時，您可以繼續之前的會話：

- `claude --continue` 繼續當前目錄中最近的對話
- `claude --resume` 開啟對話選擇器或按名稱繼續
- `claude --from-pr 123` 繼續連結到特定 pull request 的會話

從活躍會話內，使用 `/resume` 切換到不同的對話。

會話按專案目錄儲存。`/resume` 選擇器顯示來自同一 git 儲存庫的會話，包括 worktrees。

命名您的會話

給會話描述性名稱以便稍後找到它們。這是在處理多個任務或功能時的最佳實踐。

Step 1: 命名當前會話

在會話期間使用 `/rename` 給它一個易記的名稱：

```
/rename auth-refactor
```

您也可以從選擇器重新命名任何會話：執行 `/resume`，導航到會話，然後按 `R`。

Step 2: 稍後按名稱繼續

從命令列：

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

或從活躍會話內：

```
/resume auth-refactor
```

使用會話選擇器

`/resume` 命令（或 `claude --resume` 不帶引數）開啟具有以下功能的互動式會話選擇器：

選擇器中的鍵盤快捷鍵：

快捷鍵	動作
`↑` / `↓`	在會話之間導航
`→` / `←`	展開或摺疊分組的會話
`Enter`	選擇並繼續突出顯示的會話
`P`	預覽會話內容
`R`	重新命名突出顯示的會話
`/`	搜尋以篩選會話
`A`	在當前目錄和所有專案之間切換
`B`	篩選為來自您當前 git 分支的會話
`Esc`	退出選擇器或搜尋模式

會話組織：

選擇器顯示帶有有用中繼資料的會話：

- 會話名稱或初始提示
- 自上次活動以來經過的時間
- 訊息計數
- Git 分支（如果適用）

分叉的會話（使用 `/rewind` 或 `--fork-session` 建立）在其根會話下分組，使找到相關對話更容易。

> **Tip:**
>
> 提示：
>
> - **盡早命名會話**：在開始處理不同任務時使用 `/rename` ——稍後找到「payment-integration」比「explain this function」容易得多

- 使用 `--continue` 快速存取當前目錄中最近的對話
- 當您知道需要哪個會話時，使用 `--resume session-name`
- 當您需要瀏覽和選擇時，使用 `--resume`（不帶名稱）
- 對於指令碼，使用 `claude --continue --print "prompt"` 以非互動模式繼續
- 在選擇器中按 `P` 在繼續前預覽會話
- 繼續的對話以與原始對話相同的模型和設定開始

它如何工作：

1. **對話儲存**：所有對話都自動在本地儲存，包含完整的訊息歷史記錄
2. **訊息反序列化**：繼續時，整個訊息歷史記錄被恢復以保持背景資訊
3. **工具狀態**：前一個對話中的工具使用和結果被保留
4. **背景資訊恢復**：對話以所有先前背景資訊完整繼續

使用 Git worktrees 執行平行 Claude Code 會話

同時處理多個任務時，您需要每個 Claude 會話都有自己的程式碼庫副本，以便變更不會衝突。Git worktrees 通過建立單獨的工作目錄來解決此問題，每個目錄都有自己的檔案和分支，同時共享相同的儲存庫歷史記錄和遠端連接。這意味著您可以讓 Claude 在一個 worktree 中處理功能，同時在另一個 worktree 中修復錯誤，而不會相互干擾。

使用 `--worktree`（`-w`）標誌建立隔離的 worktree 並在其中啟動 Claude。您傳遞的值成為 worktree 目錄名稱和分支名稱：

```
## 在名為「feature-auth」的 worktree 中啟動 Claude
## 使用新分支建立 .claude/worktrees/feature-auth/
claude --worktree feature-auth

## 在單獨的 worktree 中啟動另一個會話
claude --worktree bugfix-123
```

如果您省略名稱，Claude 會自動產生一個隨機名稱：

```
## 自動產生名稱，如「bright-running-fox」
claude --worktree
```

Worktrees 建立在 `<repo>/.claude/worktrees/<name>` 並從預設遠端分支分支。worktree 分支名為 `worktree-<name>`。

您也可以在會話期間要求 Claude「在 worktree 中工作」或「啟動 worktree」，它會自動建立一個。

Subagent worktrees

Subagents 也可以使用 worktree 隔離來並行工作而不會衝突。要求 Claude「為您的代理使用 worktrees」或在自訂 subagent 中設定它，方法是在代理的 frontmatter 中新增 `isolation: worktree`。每個 subagent 都獲得自己的 worktree，在 subagent 完成而沒有變更時自動清理。

Worktree 清理

當您退出 worktree 會話時，Claude 根據您是否進行了變更來處理清理：

- **沒有變更**：worktree 及其分支會自動移除
- **存在變更或提交**：Claude 提示您保留或移除 worktree。保留會保留目錄和分支，以便您稍後可以返回。移除會刪除 worktree 目錄及其分支，丟棄所有未提交的變更和提交

要在 Claude 會話外清理 worktrees，請使用手動 worktree 管理。

> **Tip:**
>
> 將 `.claude/worktrees/` 新增到您的 `.gitignore` 以防止 worktree 內容在您的主儲存庫中顯示為未追蹤的檔案。

手動管理 worktrees

為了更好地控制 worktree 位置和分支設定，直接使用 Git 建立 worktrees。當您需要簽出特定現有分支或將 worktree 放在儲存庫外時，這很有用。

```
## 使用新分支建立 worktree
git worktree add ../project-feature-a -b feature-a

## 使用現有分支建立 worktree
git worktree add ../project-bugfix bugfix-123

## 在 worktree 中啟動 Claude
cd ../project-feature-a && claude

## 完成時清理
git worktree list
git worktree remove ../project-feature-a
```

在官方 Git worktree 文件中了解更多。

> **Tip:**
>
> 記住根據您的專案設定在每個新 worktree 中初始化您的開發環境。根據您的堆疊，這可能包括執行依賴項安裝（`npm install`、`yarn`）、設定虛擬環境或遵循您的專案標準設定過程。

非 git 版本控制

Worktree 隔離預設使用 git。對於其他版本控制系統（如 SVN、Perforce 或 Mercurial），設定 WorktreeCreate 和 WorktreeRemove hooks 以提供自訂 worktree 建立和清理邏輯。設定後，當您使用 `--worktree` 時，這些 hooks 會取代預設 git 行為。

對於具有共享任務和訊息的平行會話的自動協調，請參閱代理團隊。

在 Claude 需要您注意時獲得通知

當您啟動長時間執行的任務並切換到另一個視窗時，您可以設定桌面通知，以便在 Claude 完成或需要您的輸入時知道。這使用 `Notification` hook 事件，每當 Claude 等待權限、閒置並準備好新提示或完成身份驗證時觸發。

Step 1: 開啟 hooks 選單

輸入 `/hooks` 並從事件清單中選擇 `Notification`。

Step 2: 設定匹配器

選擇 `+ Match all (no filter)` 以在所有通知類型上觸發。要僅針對特定事件通知，請選擇 `+ Add new matcher…` 並輸入以下值之一：

匹配器	觸發時機
`permission_prompt`	Claude 需要您批准工具使用
`idle_prompt`	Claude 完成並等待您的下一個提示
`auth_success`	身份驗證完成
`elicitation_dialog`	Claude 在問您一個問題

Step 3: 新增您的通知命令

選擇 `+ Add new hook…` 並輸入您的作業系統的命令：

macOS

使用 `osascript` 通過 AppleScript 觸發原生 macOS 通知：

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

Linux

使用 `notify-send`，它在大多數帶有通知守護程式的 Linux 桌面上預先安裝：

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

Windows (PowerShell)

使用 PowerShell 通過 .NET 的 Windows Forms 顯示原生訊息框：

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

Step 4: 儲存到使用者設定

選擇 `User settings` 以在所有專案中應用通知。

如需完整的逐步說明和 JSON 設定範例，請參閱使用 hooks 自動化工作流程。如需完整的事件架構和通知類型，請參閱通知參考。

將 Claude 用作 unix 風格的實用程式

將 Claude 新增到您的驗證過程

假設您想將 Claude Code 用作 linter 或程式碼審查者。

將 Claude 新增到您的建置指令碼：

```
// 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:**
>
> 提示：
>
> - 在您的 CI/CD 管道中使用 Claude 進行自動程式碼審查
> - 自訂提示以檢查與您的專案相關的特定問題
> - 考慮為不同類型的驗證建立多個指令碼

管道進入、管道輸出

假設您想將資料管道輸入 Claude，並以結構化格式取回資料。

通過 Claude 管道資料：

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

> **Tip:**
>
> 提示：
>
> - 使用管道將 Claude 整合到現有 shell 指令碼中
> - 與其他 Unix 工具結合以實現強大的工作流程
> - 考慮使用 –output-format 以獲得結構化輸出

控制輸出格式

假設您需要 Claude 的輸出採用特定格式，特別是在將 Claude Code 整合到指令碼或其他工具時。

Step 1: 使用文字格式（預設）

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

這只輸出 Claude 的純文字回應（預設行為）。

Step 2: 使用 JSON 格式

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

這輸出包含中繼資料（包括成本和持續時間）的訊息的 JSON 陣列。

Step 3: 使用串流 JSON 格式

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

這在 Claude 處理請求時實時輸出一系列 JSON 物件。每個訊息都是有效的 JSON 物件，但如果連接，整個輸出不是有效的 JSON。

> **Tip:**
>
> 提示：
>
> - 對於簡單整合（您只需要 Claude 的回應），使用 `--output-format text`
> - 當您需要完整的對話日誌時，使用 `--output-format json`

- 對於每個對話輪次的實時輸出，使用 `--output-format stream-json`

詢問 Claude 其功能

Claude 內建存取其文件，可以回答有關其自身功能和限制的問題。

範例問題

```
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 根據文件提供對這些問題的答案。如需可執行的範例和實踐演示，請參閱上面的特定工作流程部分。

Tip:

提示：

- Claude 始終可以存取最新的 Claude Code 文件，無論您使用的版本如何
- 提出具體問題以獲得詳細答案
- Claude 可以解釋複雜的功能，如 MCP 整合、企業設定和進階工作流程

後續步驟

- 最佳實踐 從 Claude Code 中獲得最大收益的模式
- Claude Code 如何工作 了解代理迴圈和背景資訊管理
- 擴展 Claude Code 新增 skills、hooks、MCP、subagents 和外掛
- 參考實現 複製我們的開發容器參考實現

Part 3: Commands & Reference

CLI 參考

> Claude Code 命令列介面的完整參考，包括命令和旗標。

CLI 命令

您可以使用這些命令來啟動工作階段、管道內容、繼續對話和管理更新：

命令	描述	範例
`claude`	啟動互動式工作階段	`claude`
`claude "query"`	使用初始提示啟動互動式工作階段	`claude "explain this project"`
`claude -p "query"`	透過 SDK 查詢，然後退出	`claude -p "explain this function"`
`cat file | claude -p "query"`	處理管道內容	`cat logs.txt | claude -p "explain"`
`claude -c`	在目前目錄中繼續最近的對話	`claude -c`
`claude -c -p "query"`	透過 SDK 繼續	`claude -c -p "Check for type errors"`
`claude -r "<session>" "query"`	按 ID 或名稱繼續工作階段	`claude -r "auth-refactor" "Finish this PR"`
`claude update`	更新到最新版本	`claude update`
`claude auth login`	登入您的 Anthropic 帳戶。使用 `--email` 預先填入您的電子郵件地址，使用 `--sso` 強制進行 SSO 驗證	`claude auth login --email user@example.com --sso`
`claude auth logout`	從您的 Anthropic 帳戶登出	`claude auth logout`

命令	描述	範例
`claude auth status`	以 JSON 格式顯示驗證狀態。使用 `--text` 以人類可讀的格式輸出。如果已登入則以代碼 0 退出，如果未登入則以代碼 1 退出	`claude auth status`
`claude agents`	列出所有已設定的 subagents，按來源分組	`claude agents`
`claude mcp`	設定 Model Context Protocol (MCP) 伺服器	請參閱 Claude Code MCP 文件。
`claude remote-control`	啟動 Remote Control 工作階段，以在本機執行時從 Claude.ai 或 Claude 應用程式控制 Claude Code。請參閱 Remote Control 以了解旗標	`claude remote-control`

CLI 旗標

使用這些命令列旗標自訂 Claude Code 的行為：

旗標	描述	範例
`--add-dir`	新增額外的工作目錄供 Claude 存取（驗證每個路徑是否存在為目錄）	`claude --add-dir ../apps ../lib`
`--agent`	為目前工作階段指定代理程式（覆蓋 `agent` 設定）	`claude --agent my-custom-agent`
`--agents`	透過 JSON 動態定義自訂 subagents（請參閱下方的格式）	`claude --agents '{"reviewer":{"description":"Reviews code","prompt":"You are a code reviewer"}}'`
`--allow-dangerously-skip-permissions`	啟用權限略過作為選項，但不立即啟動它。允許與 `--permission-mode` 組合（謹慎使用）	`claude --permission-mode plan --allow-dangerously-skip-permissions`

旗標	描述	範例
`--allowedTools`	無需提示權限即可執行的工具。請參閱 權限規則語法 以了解模式匹配。若要限制可用的工具，請改用 `--tools`	`"Bash(git log *)" "Bash(git diff *)" "Read"`
`--append-system-prompt`	將自訂文字附加到預設系統提示的末尾	`claude --append-system-prompt "Always use TypeScript"`
`--append-system-prompt-file`	從檔案載入額外的系統提示文字並附加到預設提示	`claude --append-system-prompt-file ./extra-rules.txt`
`--betas`	要包含在 API 請求中的 Beta 標頭（僅限 API 金鑰使用者）	`claude --betas interleaved-thinking`
`--chrome`	啟用 Chrome 瀏覽器整合，用於網頁自動化和測試	`claude --chrome`
`--continue`, `-c`	在目前目錄中載入最近的對話	`claude --continue`
`--dangerously-skip-permissions`	略過所有權限提示（謹慎使用）	`claude --dangerously-skip-permissions`
`--debug`	啟用偵錯模式，可選擇類別篩選（例如，`"api,hooks"` 或 `"!statsig,!file"`）	`claude --debug "api,mcp"`
`--disable-slash-commands`	為此工作階段停用所有 skills 和命令	`claude --disable-slash-commands`
`--disallowedTools`	從模型的上下文中移除且無法使用的工具	`"Bash(git log *)" "Bash(git diff *)" "Edit"`
`--fallback-model`	當預設模型過載時啟用自動回退到指定的模型（僅限列印模式）	`claude -p --fallback-model sonnet "query"`
`--fork-session`	繼續時，建立新的工作階段 ID 而不是重複使用原始 ID（與 `--resume` 或 `--continue` 搭配使用）	`claude --resume abc123 --fork-session`

旗標	描述	範例
`--from-pr`	繼續連結到特定 GitHub PR 的工作階段。接受 PR 編號或 URL。透過 `gh pr create` 建立時會自動連結工作階段	`claude --from-pr 123`
`--ide`	如果恰好有一個有效的 IDE 可用，則在啟動時自動連線到 IDE	`claude --ide`
`--init`	執行初始化 hooks 並啟動互動模式	`claude --init`
`--init-only`	執行初始化 hooks 並退出（無互動工作階段）	`claude --init-only`
`--include-partial-messages`	在輸出中包含部分串流事件（需要 `--print` 和 `--output-format=stream-json`）	`claude -p --output-format stream-json --include-partial-messages "query"`
`--input-format`	為列印模式指定輸入格式（選項：`text`、`stream-json`）	`claude -p --output-format json --input-format stream-json`
`--json-schema`	在代理程式完成其工作流程後取得符合 JSON Schema 的驗證 JSON 輸出（僅限列印模式，請參閱 結構化輸出）	`claude -p --json-schema '{"type":"object","properties":{...}}' "query"`
`--maintenance`	執行維護 hooks 並退出	`claude --maintenance`
`--max-budget-usd`	在停止前在 API 呼叫上花費的最大美元金額（僅限列印模式）	`claude -p --max-budget-usd 5.00 "query"`
`--max-turns`	限制代理程式轉數（僅限列印模式）。達到限制時以錯誤退出。預設無限制	`claude -p --max-turns 3 "query"`
`--mcp-config`	從 JSON 檔案或字串載入 MCP 伺服器（以空格分隔）	`claude --mcp-config ./mcp.json`

旗標	描述	範例
`--model`	使用最新模型的別名（`sonnet` 或 `opus`）或模型的完整名稱為目前工作階段設定模型	`claude --model claude-sonnet-4-6`
`--no-chrome`	為此工作階段停用 Chrome 瀏覽器整合	`claude --no-chrome`
`--no-session-persistence`	停用工作階段持久性，使工作階段不會儲存到磁碟且無法繼續（僅限列印模式）	`claude -p --no-session-persistence "query"`
`--output-format`	為列印模式指定輸出格式（選項：`text`、`json`、`stream-json`）	`claude -p "query" --output-format json`
`--permission-mode`	以指定的權限模式開始	`claude --permission-mode plan`
`--permission-prompt-tool`	指定 MCP 工具以在非互動模式中處理權限提示	`claude -p --permission-prompt-tool mcp_auth_tool "query"`
`--plugin-dir`	為此工作階段僅從目錄載入外掛程式（可重複）	`claude --plugin-dir ./my-plugins`
`--print`, `-p`	列印回應而不進入互動模式（請參閱 Agent SDK 文件以了解程式設計用法詳細資訊）	`claude -p "query"`
`--remote`	在 claude.ai 上建立新的網頁工作階段，並提供工作描述	`claude --remote "Fix the login bug"`
`--resume`, `-r`	按 ID 或名稱繼續特定工作階段，或顯示互動式選擇器以選擇工作階段	`claude --resume auth-refactor`
`--session-id`	為對話使用特定的工作階段 ID（必須是有效的 UUID）	`claude --session-id "550e8400-e29b-41d4-a716-446655440000"`
`--setting-sources`	要載入的設定來源的逗號分隔清單（`user`、`project`、`local`）	`claude --setting-sources user,project`

旗標	描述	範例
`--settings`	設定 JSON 檔案的路徑或要載入其他設定的 JSON 字串	`claude --settings ./settings.json`
`--strict-mcp-config`	僅使用 `--mcp-config` 中的 MCP 伺服器，忽略所有其他 MCP 設定	`claude --strict-mcp-config --mcp-config ./mcp.json`
`--system-prompt`	用自訂文字取代整個系統提示	`claude --system-prompt "You are a Python expert"`
`--system-prompt-file`	從檔案載入系統提示，取代預設提示	`claude --system-prompt-file ./custom-prompt.txt`
`--teleport`	在本機終端中繼續 網頁工作階段	`claude --teleport`
`--teammate-mode`	設定 agent team 隊友的顯示方式：`auto`（預設）、`in-process` 或 `tmux`。請參閱 設定 agent teams	`claude --teammate-mode in-process`
`--tools`	限制 Claude 可以使用的內建工具。使用 `""` 停用全部，`"default"` 表示全部，或工具名稱如 `"Bash,Edit,Read"`	`claude --tools "Bash,Edit,Read"`
`--verbose`	啟用詳細記錄，顯示完整的逐轉輸出	`claude --verbose`
`--version`, `-v`	輸出版本號	`claude -v`
`--worktree`, `-w`	在隔離的 git worktree 中啟動 Claude，位於 `<repo>/.claude/worktrees/<name>`。如果未指定名稱，則會自動產生	`claude -w feature-auth`

Tip:

`--output-format json` 旗標對於指令碼和自動化特別有用，允許您以程式設計方式解析 Claude 的回應。

Agents 旗標格式

`--agents` 旗標接受定義一個或多個自訂 subagents 的 JSON 物件。每個 subagent 需要一個唯一的名稱（作為金鑰）和一個具有以下欄位的定義物件：

欄位	必需	描述
`description`	是	何時應叫用 subagent 的自然語言描述
`prompt`	是	指導 subagent 行為的系統提示
`tools`	否	subagent 可以使用的特定工具陣列，例如 `["Read", "Edit", "Bash"]`。如果省略，則繼承所有工具。支援 `Agent(agent_type)` 語法
`disallowedTools`	否	要為此 subagent 明確拒絕的工具名稱陣列
`model`	否	要使用的模型別名：`sonnet`、`opus`、`haiku` 或 `inherit`。如果省略，預設為 `inherit`
`skills`	否	要預先載入到 subagent 上下文中的 skill 名稱陣列
`mcpServers`	否	此 subagent 的 MCP servers 陣列。每個項目是伺服器名稱字串或 `{name: config}` 物件
`maxTurns`	否	subagent 停止前的最大代理程式轉數

範例：

```
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."
  }
}'
```

如需有關建立和使用 subagents 的更多詳細資訊，請參閱 subagents 文件。

系統提示旗標

Claude Code 提供四個旗標用於自訂系統提示。所有四個都在互動和非互動模式中運作。

旗標	行為	使用案例
`--system-prompt`	**取代**整個預設提示	完全控制 Claude 的行為和指示
`--system-prompt-file`	**取代**為檔案內容	從檔案載入提示以實現可重現性和版本控制
`--append-system-prompt`	**附加**到預設提示	新增特定指示，同時保持預設 Claude Code 行為
`--append-system-prompt-file`	**附加**檔案內容到預設提示	從檔案載入額外指示，同時保持預設值

何時使用每一個：

- `--system-prompt`：當您需要完全控制 Claude 的系統提示時使用。這會移除所有預設 Claude Code 指示，為您提供一個空白的狀態。

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

- `--system-prompt-file`：當您想從檔案載入自訂提示時使用，對於團隊一致性或版本控制的提示範本很有用。

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

- `--append-system-prompt`：當您想新增特定指示，同時保持 Claude Code 的預設功能完整時使用。這是大多數使用案例的最安全選項。

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

- `--append-system-prompt-file`：當您想從檔案附加指示，同時保持 Claude Code 的預設值時使用。對於版本控制的新增很有用。

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

`--system-prompt` 和 `--system-prompt-file` 互斥。附加旗標可與任一取代旗標一起使用。

對於大多數使用案例，建議使用 `--append-system-prompt` 或 `--append-system-prompt-file`，因為它們保留 Claude Code 的內建功能，同時新增您的自訂需求。僅當您需要完全控制系統提示時，才使用 `--system-prompt` 或 `--system-prompt-file`。

另請參閱

- Chrome 擴充功能 - 瀏覽器自動化和網頁測試
- 互動模式 - 快捷鍵、輸入模式和互動功能
- 快速入門指南 - Claude Code 入門
- 常見工作流程 - 進階工作流程和模式
- 設定 - 設定選項
- Agent SDK 文件 - 程式設計用法和整合

自訂鍵盤快捷鍵

使用快捷鍵配置檔案在 Claude Code 中自訂鍵盤快捷鍵。

> **Note:**
>
> 可自訂的鍵盤快捷鍵需要 Claude Code v2.1.18 或更新版本。使用 `claude --version` 檢查您的版本。

Claude Code 支援可自訂的鍵盤快捷鍵。執行 `/keybindings` 以在 `~/.claude/keybindings.json` 建立或開啟您的配置檔案。

配置檔案

快捷鍵配置檔案是一個包含 `bindings` 陣列的物件。每個區塊指定一個上下文和一個按鍵組合到動作的對應。

> **Note:**
>
> 快捷鍵檔案的變更會自動偵測並套用，無需重新啟動 Claude Code。

欄位	說明
`$schema`	選用的 JSON Schema URL，用於編輯器自動完成
`$docs`	選用的文件 URL
`bindings`	按上下文分組的繫結區塊陣列

此範例在聊天上下文中將 `Ctrl+E` 繫結到開啟外部編輯器，並取消繫結 `Ctrl+U`：

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

上下文

每個繫結區塊指定一個**上下文**，其中快捷鍵適用：

上下文	說明
`Global`	在應用程式的任何地方適用
`Chat`	主聊天輸入區域
`Autocomplete`	自動完成選單已開啟
`Settings`	設定選單（僅限 Escape 關閉）
`Confirmation`	權限和確認對話框
`Tabs`	標籤導覽元件
`Help`	說明選單可見
`Transcript`	文字記錄檢視器
`HistorySearch`	歷史記錄搜尋模式 (Ctrl+R)
`Task`	背景工作正在執行
`ThemePicker`	主題選擇器對話框
`Attachments`	影像/附件欄導覽

上下文	說明
`Footer`	頁尾指示器導覽（工作、團隊、差異）
`MessageSelector`	回溯和摘要對話框訊息選擇
`DiffDialog`	差異檢視器導覽
`ModelPicker`	模型選擇器努力程度
`Select`	通用選擇/清單元件
`Plugin`	Plugin 對話框（瀏覽、探索、管理）

可用動作

動作遵循 `namespace:action` 格式，例如 `chat:submit` 用於傳送訊息，或 `app:toggleTodos` 用於顯示工作清單。每個上下文都有特定的可用動作。

應用程式動作

在 `Global` 上下文中可用的動作：

動作	預設值	說明
`app:interrupt`	Ctrl+C	取消目前操作
`app:exit`	Ctrl+D	結束 Claude Code
`app:toggleTodos`	Ctrl+T	切換工作清單可見性
`app:toggleTranscript`	Ctrl+O	切換詳細文字記錄

歷史記錄動作

用於導覽命令歷史記錄的動作：

動作	預設值	說明
`history:search`	Ctrl+R	開啟歷史記錄搜尋
`history:previous`	Up	上一個歷史記錄項目
`history:next`	Down	下一個歷史記錄項目

聊天動作

在 `Chat` 上下文中可用的動作：

動作	預設值	說明
`chat:cancel`	Escape	取消目前輸入
`chat:cycleMode`	Shift+Tab*	循環權限模式
`chat:modelPicker`	Cmd+P / Meta+P	開啟模型選擇器
`chat:thinkingToggle`	Cmd+T / Meta+T	切換延伸思考
`chat:submit`	Enter	提交訊息
`chat:undo`	Ctrl+_	復原上一個動作
`chat:externalEditor`	Ctrl+G	在外部編輯器中開啟
`chat:stash`	Ctrl+S	暫存目前提示
`chat:imagePaste`	Ctrl+V (Windows 上為 Alt+V)	貼上影像

*在沒有 VT 模式的 Windows 上（Node <24.2.0/<22.17.0、Bun <1.2.23），預設為 Meta+M。

自動完成動作

在 `Autocomplete` 上下文中可用的動作：

動作	預設值	說明
`autocomplete:accept`	Tab	接受建議
`autocomplete:dismiss`	Escape	關閉選單
`autocomplete:previous`	Up	上一個建議
`autocomplete:next`	Down	下一個建議

確認動作

在 `Confirmation` 上下文中可用的動作：

動作	預設值	說明
`confirm:yes`	Y, Enter	確認動作

動作	預設值	說明
`confirm:no`	N, Escape	拒絕動作
`confirm:previous`	Up	上一個選項
`confirm:next`	Down	下一個選項
`confirm:nextField`	Tab	下一個欄位
`confirm:previousField`	(未繫結)	上一個欄位
`confirm:cycleMode`	Shift+Tab	循環權限模式
`confirm:toggleExplanation`	Ctrl+E	切換權限說明

權限動作

在 `Confirmation` 上下文中可用於權限對話框的動作：

動作	預設值	說明
`permission:toggleDebug`	Ctrl+D	切換權限偵錯資訊

文字記錄動作

在 `Transcript` 上下文中可用的動作：

動作	預設值	說明
`transcript:toggleShowAll`	Ctrl+E	切換顯示所有內容
`transcript:exit`	Ctrl+C, Escape	結束文字記錄檢視

歷史記錄搜尋動作

在 `HistorySearch` 上下文中可用的動作：

動作	預設值	說明
`historySearch:next`	Ctrl+R	下一個符合項目
`historySearch:accept`	Escape, Tab	接受選擇
`historySearch:cancel`	Ctrl+C	取消搜尋
`historySearch:execute`	Enter	執行選定的命令

工作動作

在 `Task` 上下文中可用的動作：

動作	預設值	說明
`task:background`	Ctrl+B	背景執行目前工作

主題動作

在 `ThemePicker` 上下文中可用的動作：

動作	預設值	說明
`theme:toggleSyntaxHighlighting`	Ctrl+T	切換語法醒目提示

說明動作

在 `Help` 上下文中可用的動作：

動作	預設值	說明
`help:dismiss`	Escape	關閉說明選單

標籤動作

在 `Tabs` 上下文中可用的動作：

動作	預設值	說明
`tabs:next`	Tab, Right	下一個標籤
`tabs:previous`	Shift+Tab, Left	上一個標籤

附件動作

在 `Attachments` 上下文中可用的動作：

動作	預設值	說明
`attachments:next`	Right	下一個附件
`attachments:previous`	Left	上一個附件
`attachments:remove`	Backspace, Delete	移除選定的附件
`attachments:exit`	Down, Escape	結束附件欄

頁尾動作

在 `Footer` 上下文中可用的動作：

動作	預設值	說明
`footer:next`	Right	下一個頁尾項目
`footer:previous`	Left	上一個頁尾項目
`footer:openSelected`	Enter	開啟選定的頁尾項目
`footer:clearSelection`	Escape	清除頁尾選擇

訊息選擇器動作

在 `MessageSelector` 上下文中可用的動作：

動作	預設值	說明
`messageSelector:up`	Up, K, Ctrl+P	在清單中向上移動
`messageSelector:down`	Down, J, Ctrl+N	在清單中向下移動
`messageSelector:top`	Ctrl+Up, Shift+Up, Meta+Up, Shift+K	跳至頂部
`messageSelector:bottom`	Ctrl+Down, Shift+Down, Meta+Down, Shift+J	跳至底部
`messageSelector:select`	Enter	選擇訊息

差異動作

在 `DiffDialog` 上下文中可用的動作：

動作	預設值	說明
`diff:dismiss`	Escape	關閉差異檢視器
`diff:previousSource`	Left	上一個差異來源
`diff:nextSource`	Right	下一個差異來源
`diff:previousFile`	Up	差異中的上一個檔案

動作	預設值	說明
`diff:nextFile`	Down	差異中的下一個檔案
`diff:viewDetails`	Enter	檢視差異詳細資訊
`diff:back`	(特定於上下文)	在差異檢視器中返回

模型選擇器動作

在 `ModelPicker` 上下文中可用的動作：

動作	預設值	說明
`modelPicker:decreaseEffort`	Left	降低努力程度
`modelPicker:increaseEffort`	Right	提高努力程度

選擇動作

在 `Select` 上下文中可用的動作：

動作	預設值	說明
`select:next`	Down, J, Ctrl+N	下一個選項
`select:previous`	Up, K, Ctrl+P	上一個選項
`select:accept`	Enter	接受選擇
`select:cancel`	Escape	取消選擇

Plugin 動作

在 `Plugin` 上下文中可用的動作：

動作	預設值	說明
`plugin:toggle`	Space	切換 plugin 選擇
`plugin:install`	I	安裝選定的 plugins

設定動作

在 `Settings` 上下文中可用的動作：

動作	預設值	說明
`settings:search`	/	進入搜尋模式
`settings:retry`	R	重試載入使用量資料（發生錯誤時）

按鍵組合語法

修飾鍵

使用 `+` 分隔符搭配修飾鍵：

- `ctrl` 或 `control` - Control 鍵
- `alt`、`opt` 或 `option` - Alt/Option 鍵
- `shift` - Shift 鍵
- `meta`、`cmd` 或 `command` - Meta/Command 鍵

例如：

```
ctrl+k          單一鍵搭配修飾鍵
shift+tab       Shift + Tab
meta+p          Command/Meta + P
ctrl+shift+c    多個修飾鍵
```

大寫字母

獨立的大寫字母表示 Shift。例如，`K` 等同於 `shift+k`。這對於 vim 風格的繫結很有用，其中大寫和小寫鍵有不同的含義。

搭配修飾鍵的大寫字母（例如 `ctrl+K`）被視為風格上的，**不**表示 Shift — `ctrl+K` 與 `ctrl+k` 相同。

和弦

和弦是由空格分隔的按鍵組合序列：

```
ctrl+k ctrl+s   按 Ctrl+K，放開，然後按 Ctrl+S
```

特殊鍵

- `escape` 或 `esc` - Escape 鍵
- `enter` 或 `return` - Enter 鍵

- `tab` - Tab 鍵
- `space` - 空格鍵
- `up` 、 `down` 、 `left` 、 `right` - 方向鍵
- `backspace` 、 `delete` - 刪除鍵

取消繫結預設快捷鍵

將動作設定為 `null` 以取消繫結預設快捷鍵：

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

保留的快捷鍵

這些快捷鍵無法重新繫結：

快捷鍵	原因
Ctrl+C	硬編碼的中斷/取消
Ctrl+D	硬編碼的結束

終端機衝突

某些快捷鍵可能與終端機多工器衝突：

快捷鍵	衝突
Ctrl+B	tmux 前綴（按兩次以傳送）
Ctrl+A	GNU screen 前綴
Ctrl+Z	Unix 程序暫停 (SIGTSTP)

Vim 模式互動

啟用 vim 模式時（`/vim`），快捷鍵和 vim 模式獨立運作：

- **Vim 模式**在文字輸入層級處理輸入（游標移動、模式、動作）
- **快捷鍵**在元件層級處理動作（切換待辦事項、提交等）
- vim 模式中的 Escape 鍵從 INSERT 切換到 NORMAL 模式；它不會觸發 `chat:cancel`
- 大多數 Ctrl+鍵快捷鍵通過 vim 模式傳遞到快捷鍵系統
- 在 vim NORMAL 模式中，`?` 顯示說明選單（vim 行為）

驗證

Claude Code 驗證您的快捷鍵並顯示以下警告：

- 解析錯誤（無效的 JSON 或結構）
- 無效的上下文名稱
- 保留快捷鍵衝突
- 終端機多工器衝突
- 同一上下文中的重複繫結

執行 `/doctor` 以查看任何快捷鍵警告。

Part 4: Configuration

設定權限

> 使用細粒度權限規則、模式和受管理原則來控制 Claude Code 可以存取和執行的操作。

Claude Code 支援細粒度權限，讓您可以精確指定代理允許執行和不允許執行的操作。權限設定可以簽入版本控制並分發給組織中的所有開發人員，也可以由個別開發人員自訂。

權限系統

Claude Code 使用分層權限系統來平衡功能和安全性：

工具類型	範例	需要批准	“是，不要再問”行為
唯讀	檔案讀取、Grep	否	不適用
Bash 命令	Shell 執行	是	每個專案目錄和命令永久有效
檔案修改	Edit/write 檔案	是	直到工作階段結束

管理權限

您可以使用 `/permissions` 檢視和管理 Claude Code 的工具權限。此 UI 列出所有權限規則及其來源的 settings.json 檔案。

- **Allow** 規則讓 Claude Code 使用指定的工具，無需手動批准。
- **Ask** 規則在 Claude Code 嘗試使用指定工具時提示確認。
- **Deny** 規則防止 Claude Code 使用指定的工具。

規則按順序評估：**deny -> ask -> allow**。第一個符合的規則獲勝，因此 deny 規則始終優先。

權限模式

Claude Code 支援多種權限模式來控制工具的批准方式。在您的設定檔案中設定 `defaultMode`：

模式	描述
`default`	標準行為：在首次使用每個工具時提示權限
`acceptEdits`	自動接受工作階段的檔案編輯權限
`plan`	Plan Mode：Claude 可以分析但不能修改檔案或執行命令
`dontAsk`	自動拒絕工具，除非透過 `/permissions` 或 `permissions.allow` 規則預先批准
`bypassPermissions`	跳過所有權限提示（需要安全環境，請參閱下方警告）

> **Warning:**
>
> `bypassPermissions` 模式會停用所有權限檢查。僅在隔離環境（如容器或虛擬機）中使用此模式，其中 Claude Code 無法造成損害。管理員可以透過在受管理設定中將 `disableBypassPermissionsMode` 設定為 `"disable"` 來防止此模式。

權限規則語法

權限規則遵循格式 `Tool` 或 `Tool(specifier)`。

符合工具的所有使用

若要符合工具的所有使用，請使用不帶括號的工具名稱：

規則	效果
`Bash`	符合所有 Bash 命令
`WebFetch`	符合所有網頁擷取請求
`Read`	符合所有檔案讀取

`Bash(*)` 等同於 `Bash` 並符合所有 Bash 命令。

使用指定符進行細粒度控制

在括號中新增指定符以符合特定工具使用：

規則	效果
`Bash(npm run build)`	符合確切命令 `npm run build`

規則	效果
`Read(./.env)`	符合讀取目前目錄中的 `.env` 檔案
`WebFetch(domain:example.com)`	符合對 example.com 的擷取請求

萬用字元模式

Bash 規則支援使用 `*` 的 glob 模式。萬用字元可以出現在命令中的任何位置。此設定允許 npm 和 git commit 命令，同時阻止 git push：

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

`*` 前的空格很重要：`Bash(ls *)` 符合 `ls -la` 但不符合 `lsof`，而 `Bash(ls*)` 兩者都符合。舊版 `:*` 後綴語法等同於 `*` 但已棄用。

工具特定的權限規則

Bash

Bash 權限規則支援使用 `*` 的萬用字元符合。萬用字元可以出現在命令中的任何位置，包括開頭、中間或結尾：

- `Bash(npm run build)` 符合確切的 Bash 命令 `npm run build`
- `Bash(npm run test *)` 符合以 `npm run test` 開頭的 Bash 命令
- `Bash(npm *)` 符合任何以 `npm` 開頭的命令
- `Bash(* install)` 符合任何以 `install` 結尾的命令

- `Bash(git * main)` 符合命令如 `git checkout main`、`git merge main`

當 `*` 出現在末尾且前面有空格時（如 `Bash(ls *)`），它會強制執行字邊界，要求前綴後面跟著空格或字串結尾。例如，`Bash(ls *)` 符合 `ls -la` 但不符合 `lsof`。相比之下，`Bash(ls*)` 沒有空格會同時符合 `ls -la` 和 `lsof`，因為沒有字邊界限制。

> **Tip:**
>
> Claude Code 知道 shell 運算子（如 `&&`），所以前綴符合規則如 `Bash(safe-cmd *)` 不會給它執行命令 `safe-cmd && other-cmd` 的權限。

> **Warning:**
>
> 嘗試限制命令引數的 Bash 權限模式很脆弱。例如，`Bash(curl http://github.com/ *)` 旨在將 curl 限制為 GitHub URL，但不會符合以下變化：
>
> - URL 前的選項：`curl -X GET http://github.com/...`
> - 不同的協定：`curl https://github.com/...`
> - 重新導向：`curl -L http://bit.ly/xyz`（重新導向到 github）
> - 變數：`URL=http://github.com && curl $URL`
> - 額外空格：`curl http://github.com`
>
> 為了更可靠的 URL 篩選，請考慮：
>
> - **限制 Bash 網路工具**：使用 deny 規則阻止 `curl`、`wget` 和類似命令，然後使用 WebFetch 工具搭配 `WebFetch(domain:github.com)` 權限以允許的網域
> - **使用 PreToolUse hooks**：實作一個 hook 來驗證 Bash 命令中的 URL 並阻止不允許的網域
> - 透過 CLAUDE.md 指示 Claude Code 關於您允許的 curl 模式
>
> 請注意，單獨使用 WebFetch 不會防止網路存取。如果允許 Bash，Claude 仍然可以使用 `curl`、`wget` 或其他工具來存取任何 URL。

Read 和 Edit

`Edit` 規則適用於所有編輯檔案的內建工具。Claude 會盡力嘗試將 `Read` 規則應用於所有讀取檔案的內建工具，如 Grep 和 Glob。

Read 和 Edit 規則都遵循 gitignore 規格，具有四種不同的模式類型：

模式	意義	範例	符合
`//path`	來自檔案系統根目錄的**絕對**路徑	`Read(//Users/alice/secrets/**)`	`/Users/alice/secrets/**`

模式	意義	範例	符合
`~/path`	來自**主目錄**的路徑	`Read(~/Documents/*.pdf)`	`/Users/alice/Documents/*.pdf`
`/path`	**相對於專案根目錄**的路徑	`Edit(/src/**/*.ts)`	`<project root>/src/**/*.ts`
`path` 或 `./path`	**相對於目前目錄**的路徑	`Read(*.env)`	`<cwd>/*.env`

> **Warning:**
>
> 像 `/Users/alice/file` 這樣的模式不是絕對路徑。它相對於專案根目錄。使用 `//Users/alice/file` 表示絕對路徑。

範例：

- `Edit(/docs/**)`：編輯 `<project>/docs/` 中的檔案（不是 `/docs/` 也不是 `<project>/.claude/docs/`）
- `Read(~/.zshrc)`：讀取您主目錄的 `.zshrc`
- `Edit(//tmp/scratch.txt)`：編輯絕對路徑 `/tmp/scratch.txt`
- `Read(src/**)`：從 `<current-directory>/src/` 讀取

> **Note:**
>
> 在 gitignore 模式中，`*` 符合單一目錄中的檔案，而 `**` 遞迴符合目錄。若要允許所有檔案存取，請使用不帶括號的工具名稱：`Read`、`Edit` 或 `Write`。

WebFetch

- `WebFetch(domain:example.com)` 符合對 example.com 的擷取請求

MCP

- `mcp__puppeteer` 符合由 `puppeteer` 伺服器提供的任何工具（在 Claude Code 中設定的名稱）
- `mcp__puppeteer__*` 萬用字元語法，也符合來自 `puppeteer` 伺服器的所有工具
- `mcp__puppeteer__puppeteer_navigate` 符合由 `puppeteer` 伺服器提供的 `puppeteer_navigate` 工具

Agent（subagents）

使用 `Agent(AgentName)` 規則來控制 Claude 可以使用哪些 subagents：

- `Agent(Explore)` 符合 Explore subagent
- `Agent(Plan)` 符合 Plan subagent
- `Agent(my-custom-agent)` 符合名為 `my-custom-agent` 的自訂 subagent

將這些規則新增到您設定中的 `deny` 陣列，或使用 `--disallowedTools` CLI 旗標來停用特定代理。若要停用 Explore 代理：

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

使用 hooks 擴展權限

Claude Code hooks 提供了一種方式來註冊自訂 shell 命令，以在執行時執行權限評估。當 Claude Code 進行工具呼叫時，PreToolUse hooks 在權限系統之前執行，hook 輸出可以決定是否批准或拒絕工具呼叫，以取代權限系統。

工作目錄

根據預設，Claude 可以存取啟動它的目錄中的檔案。您可以擴展此存取：

- **在啟動期間**：使用 `--add-dir <path>` CLI 引數
- **在工作階段期間**：使用 `/add-dir` 命令
- **持久設定**：新增到設定檔案中的 `additionalDirectories`

其他目錄中的檔案遵循與原始工作目錄相同的權限規則：它們變成可讀的而無需提示，檔案編輯權限遵循目前的權限模式。

權限如何與沙箱互動

權限和沙箱是互補的安全層：

- **權限**控制 Claude Code 可以使用哪些工具以及它可以存取哪些檔案或網域。它們適用於所有工具（Bash、Read、Edit、WebFetch、MCP 和其他）。
- **沙箱**提供作業系統級別的強制執行，限制 Bash 工具的檔案系統和網路存取。它僅適用於 Bash 命令及其子程序。

使用兩者進行深度防禦：

- 權限 deny 規則阻止 Claude 甚至嘗試存取受限資源
- 沙箱限制防止 Bash 命令到達定義邊界外的資源，即使提示注入繞過 Claude 的決策制定
- 沙箱中的檔案系統限制使用 Read 和 Edit deny 規則，而不是單獨的沙箱設定
- 網路限制結合 WebFetch 權限規則與沙箱的 `allowedDomains` 清單

受管理設定

對於需要集中控制 Claude Code 設定的組織，管理員可以部署無法被使用者或專案設定覆蓋的受管理設定。這些原則設定遵循與一般設定檔案相同的格式，可以透過 MDM/OS 級別原則、受管理設定檔案或伺服器管理的設定傳遞。請參閱設定檔案以了解傳遞機制和檔案位置。

僅受管理的設定

某些設定僅在受管理設定中有效：

設定	描述
`disableBypassPermissionsMode`	設定為 `"disable"` 以防止 `bypassPermissions` 模式和 `--dangerously-skip-permissions` 旗標
`allowManagedPermissionRulesOnly`	當為 `true` 時，防止使用者和專案設定定義 `allow`、`ask` 或 `deny` 權限規則。僅套用受管理設定中的規則
`allowManagedHooksOnly`	當為 `true` 時，防止載入使用者、專案和外掛 hooks。僅允許受管理 hooks 和 SDK hooks
`allowManagedMcpServersOnly`	當為 `true` 時，僅尊重受管理設定中的 `allowedMcpServers`。`deniedMcpServers` 仍然從所有來源合併。請參閱受管理 MCP 設定
`blockedMarketplaces`	市場來源的封鎖清單。在下載前檢查被封鎖的來源，因此它們永遠不會接觸檔案系統。請參閱受管理市場限制

設定	描述
`sandbox.network.allowManagedDomainsOnly`	當為 `true` 時，僅尊重來自受管理設定的 `allowedDomains` 和 `WebFetch(domain:...)` allow 規則。非允許的網域會自動被阻止，無需提示使用者。被拒絕的網域仍然從所有來源合併
`strictKnownMarketplaces`	控制使用者可以新增哪些外掛市場。請參閱受管理市場限制
`allow_remote_sessions`	當為 `true` 時，允許使用者啟動遠端控制和網頁工作階段。預設為 `true`。設定為 `false` 以防止遠端工作階段存取

設定優先順序

權限規則遵循與所有其他 Claude Code 設定相同的設定優先順序：

1. **受管理設定**：無法被任何其他級別覆蓋，包括命令列引數
2. **命令列引數**：臨時工作階段覆蓋
3. **本機專案設定**（`.claude/settings.local.json`）
4. **共用專案設定**（`.claude/settings.json`）
5. **使用者設定**（`~/.claude/settings.json`）

如果工具在任何級別被拒絕，沒有其他級別可以允許它。例如，受管理設定 deny 無法被 `--allowedTools` 覆蓋，`--disallowedTools` 可以新增超出受管理設定定義的限制。

如果權限在使用者設定中被允許但在專案設定中被拒絕，專案設定優先，權限被阻止。

範例設定

此儲存庫包含常見部署情境的入門設定設定。使用這些作為起點並根據您的需求進行調整。

另請參閱

- Settings：完整設定參考，包括權限設定表
- Sandboxing：Bash 命令的作業系統級別檔案系統和網路隔離
- Authentication：設定使用者對 Claude Code 的存取
- Security：安全防護措施和最佳實踐
- Hooks：自動化工作流程並擴展權限評估

Claude 如何記住您的專案

> 使用 CLAUDE.md 檔案為 Claude 提供持久指令，並讓 Claude 透過自動記憶自動累積學習。

每個 Claude Code 工作階段都以全新的 context window 開始。兩個機制可以跨工作階段傳遞知識：

- **CLAUDE.md 檔案**：您編寫的指令，為 Claude 提供持久的上下文
- **自動記憶**：Claude 根據您的更正和偏好自己編寫的筆記

本頁涵蓋如何：

- 編寫和組織 CLAUDE.md 檔案
- 使用 `.claude/rules/` 將規則範圍限定於特定檔案類型
- 配置自動記憶，讓 Claude 自動記筆記
- 疑難排解指令未被遵循的情況

CLAUDE.md 與自動記憶

Claude Code 有兩個互補的記憶系統。兩者都在每次對話開始時載入。Claude 將它們視為上下文，而不是強制配置。您的指令越具體和簡潔，Claude 遵循它們的一致性就越高。

	CLAUDE.md 檔案	自動記憶
誰編寫	您	Claude
包含內容	指令和規則	學習和模式
範圍	專案、使用者或組織	每個工作樹
載入到	每個工作階段	每個工作階段（前 200 行）
用於	編碼標準、工作流程、專案架構	建置命令、除錯見解、Claude 發現的偏好

當您想引導 Claude 的行為時，使用 CLAUDE.md 檔案。自動記憶讓 Claude 從您的更正中學習，無需手動操作。

Subagents 也可以維護自己的自動記憶。詳見 subagent 配置。

CLAUDE.md 檔案

CLAUDE.md 檔案是 markdown 檔案，為專案、您的個人工作流程或整個組織提供 Claude 持久指令。您以純文字編寫這些檔案；Claude 在每個工作階段開始時讀取它們。

選擇 CLAUDE.md 檔案的位置

CLAUDE.md 檔案可以位於多個位置，每個位置有不同的範圍。更具體的位置優先於更廣泛的位置。

範圍	位置	目的	使用案例	共享對象
受管理的政策	• macOS：`/Library/Application Support/ClaudeCode/CLAUDE.md` • Linux 和 WSL：`/etc/claude-code/CLAUDE.md` • Windows：`C:\Program Files\ClaudeCode\CLAUDE.md`	由 IT/DevOps 管理的組織範圍指令	公司編碼標準、安全政策、合規要求	組織中的所有使用者
專案指令	`./CLAUDE.md` 或 `./.claude/CLAUDE.md`	專案的團隊共享指令	專案架構、編碼標準、常見工作流程	透過原始碼控制的團隊成員
使用者指令	`~/.claude/CLAUDE.md`	所有專案的個人偏好	程式碼樣式偏好、個人工具快捷方式	僅您（所有專案）

工作目錄上方目錄層級中的 CLAUDE.md 檔案在啟動時完整載入。子目錄中的 CLAUDE.md 檔案在 Claude 讀取這些目錄中的檔案時按需載入。詳見 CLAUDE.md 檔案如何載入以了解完整的解析順序。

對於大型專案，您可以使用 專案規則將指令分解為主題特定的檔案。規則讓您將指令範圍限定於特定檔案類型或子目錄。

設定專案 CLAUDE.md

專案 CLAUDE.md 可以儲存在 `./CLAUDE.md` 或 `./.claude/CLAUDE.md` 中。建立此檔案並新增適用於任何在專案上工作的人的指令：建置和測試命令、編碼標準、架構決策、命名慣例和常見工作流程。這些指令透過版本控制與您的團隊共享，因此請專注於專案級別的標準，而不是個人偏好。

> **Tip:**

> 執行 `/init` 自動產生起始 CLAUDE.md。Claude 分析您的程式碼庫並建立一個檔案，其中包含它發現的建置命令、測試指令和專案慣例。如果 CLAUDE.md 已存在，`/init` 會建議改進而不是覆蓋它。從那裡進行細化，新增 Claude 不會自己發現的指令。

編寫有效的指令

CLAUDE.md 檔案在每個工作階段開始時載入到 context window 中，與您的對話一起消耗 tokens。因為它們是上下文而不是強制配置，您編寫指令的方式會影響 Claude 遵循它們的可靠性。具體、簡潔、結構良好的指令效果最好。

大小：目標是每個 CLAUDE.md 檔案少於 200 行。較長的檔案消耗更多上下文並降低遵循度。如果您的指令變得很大，請使用 匯入或 `.claude/rules/` 檔案進行分割。

結構：使用 markdown 標題和項目符號來分組相關指令。Claude 掃描結構的方式與讀者相同：組織良好的部分比密集的段落更容易遵循。

具體性：編寫具體到足以驗證的指令。例如：

- 「使用 2 空格縮排」而不是「正確格式化程式碼」
- 「提交前執行 `npm test`」而不是「測試您的變更」
- 「API 處理程式位於 `src/api/handlers/`」而不是「保持檔案組織」

一致性：如果兩個規則相互矛盾，Claude 可能會任意選擇一個。定期檢查您的 CLAUDE.md 檔案、子目錄中的巢狀 CLAUDE.md 檔案和 `.claude/rules/`，以移除過時或衝突的指令。在 monorepos 中，使用 `claudeMdExcludes` 跳過與您的工作無關的其他團隊的 CLAUDE.md 檔案。

匯入其他檔案

CLAUDE.md 檔案可以使用 `@path/to/import` 語法匯入其他檔案。匯入的檔案會展開並在啟動時與參考它們的 CLAUDE.md 一起載入到上下文中。

允許相對和絕對路徑。相對路徑相對於包含匯入的檔案解析，而不是工作目錄。匯入的檔案可以遞迴匯入其他檔案，最大深度為五跳。

要引入 README、package.json 和工作流程指南，請在 CLAUDE.md 中的任何位置使用 `@` 語法參考它們：

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

## Additional Instructions
- git workflow @docs/git-instructions.md
```

對於您不想簽入的個人偏好，從您的主目錄匯入檔案。匯入位於共享 CLAUDE.md 中，但它指向的檔案保留在您的機器上：

```
## Individual Preferences
- @~/.claude/my-project-instructions.md
```

> **Warning:**
>
> Claude Code 第一次在專案中遇到外部匯入時，會顯示一個核准對話框，列出檔案。如果您拒絕，匯入將保持禁用狀態，對話框不會再次出現。

如需更結構化的指令組織方法，請參閱 `.claude/rules/`。

CLAUDE.md 檔案如何載入

Claude Code 透過從您目前的工作目錄向上走目錄樹來讀取 CLAUDE.md 檔案，檢查沿途的每個目錄。這意味著如果您在 `foo/bar/` 中執行 Claude Code，它會從 `foo/bar/CLAUDE.md` 和 `foo/CLAUDE.md` 載入指令。

Claude 也會在您目前工作目錄下的子目錄中發現 CLAUDE.md 檔案。它們不是在啟動時載入，而是在 Claude 讀取這些子目錄中的檔案時包含。

如果您在大型 monorepo 中工作，其他團隊的 CLAUDE.md 檔案被拾取，請使用 `claudeMdExcludes` 跳過它們。

從其他目錄載入

`--add-dir` 旗標讓 Claude 存取主工作目錄外的其他目錄。預設情況下，不會載入這些目錄中的 CLAUDE.md 檔案。

要也從其他目錄載入 CLAUDE.md 檔案，包括 `CLAUDE.md`、`.claude/CLAUDE.md` 和 `.claude/rules/*.md`，請設定 `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD` 環境變數：

```
CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1 claude --add-dir ../shared-config
```

使用 `.claude/rules/` 組織規則

對於較大的專案，您可以使用 `.claude/rules/` 目錄將指令組織成多個檔案。這使指令保持模組化，更容易讓團隊維護。規則也可以 範圍限定於特定檔案路徑，因此它們只在 Claude 處理匹配檔案時載入到上下文中，減少雜訊並節省上下文空間。

> **Note:**
>
> 規則在每個工作階段或開啟匹配檔案時載入到上下文中。對於不需要始終在上下文中的任務特定指令，請改用 skills，它們只在您呼叫它們或 Claude 確定它們與您的提示相關時載入。

設定規則

在您的專案的 `.claude/rules/` 目錄中放置 markdown 檔案。每個檔案應涵蓋一個主題，具有描述性檔案名稱，如 `testing.md` 或 `api-design.md`。所有 `.md` 檔案都被遞迴發現，因此您可以將規則組織到子目錄中，如 `frontend/` 或 `backend/`：

```
your-project/
├── .claude/
│   ├── CLAUDE.md           # Main project instructions
│   └── rules/
│       ├── code-style.md   # Code style guidelines
│       ├── testing.md      # Testing conventions
│       └── security.md     # Security requirements
```

沒有 `paths` frontmatter 的規則在啟動時載入，優先級與 `.claude/CLAUDE.md` 相同。

路徑特定規則

規則可以使用帶有 `paths` 欄位的 YAML frontmatter 範圍限定於特定檔案。這些條件規則只在 Claude 處理與指定模式匹配的檔案時適用。

```
---
paths:
  - "src/api/**/*.ts"
---

## API Development Rules

- All API endpoints must include input validation
- Use the standard error response format
- Include OpenAPI documentation comments
```

沒有 `paths` 欄位的規則無條件載入並適用於所有檔案。路徑範圍規則在 Claude 讀取與模式匹配的檔案時觸發，而不是在每次工具使用時。

在 `paths` 欄位中使用 glob 模式按副檔名、目錄或任何組合匹配檔案：

模式	匹配
`**/*.ts`	任何目錄中的所有 TypeScript 檔案
`src/**/*`	`src/` 目錄下的所有檔案
`*.md`	專案根目錄中的 Markdown 檔案
`src/components/*.tsx`	特定目錄中的 React 元件

您可以指定多個模式並使用大括號展開在一個模式中匹配多個副檔名：

```
---
paths:
  - "src/**/*.{ts,tsx}"
  - "lib/**/*.ts"
  - "tests/**/*.test.ts"
---
```

使用符號連結跨專案共享規則

`.claude/rules/` 目錄支援符號連結，因此您可以維護一組共享規則並將它們連結到多個專案中。符號連結被解析並正常載入，並且循環符號連結被檢測並妥善處理。

此範例連結共享目錄和個別檔案：

```
ln -s ~/shared-claude-rules .claude/rules/shared
ln -s ~/company-standards/security.md .claude/rules/security.md
```

使用者級別規則

`~/.claude/rules/` 中的個人規則適用於您機器上的每個專案。使用它們來設定不是專案特定的偏好：

```
~/.claude/rules/
├── preferences.md    # Your personal coding preferences
└── workflows.md      # Your preferred workflows
```

使用者級別規則在專案規則之前載入，給予專案規則更高的優先級。

為大型團隊管理 CLAUDE.md

對於在團隊中部署 Claude Code 的組織，您可以集中指令並控制載入哪些 CLAUDE.md 檔案。

部署組織範圍的 CLAUDE.md

組織可以部署一個集中管理的 CLAUDE.md，適用於機器上的所有使用者。此檔案無法由個別設定排除。

Step 1: 在受管理的政策位置建立檔案

- macOS：`/Library/Application Support/ClaudeCode/CLAUDE.md`
- Linux 和 WSL：`/etc/claude-code/CLAUDE.md`
- Windows：`C:\Program Files\ClaudeCode\CLAUDE.md`

Step 2: 使用您的配置管理系統進行部署

使用 MDM、群組原則、Ansible 或類似工具在開發人員機器上分發檔案。詳見 受管理的設定以了解其他組織範圍的配置選項。

排除特定的 CLAUDE.md 檔案

在大型 monorepos 中，祖先 CLAUDE.md 檔案可能包含與您的工作無關的指令。`claudeMdExcludes` 設定讓您按路徑或 glob 模式跳過特定檔案。

此範例排除頂級 CLAUDE.md 和父資料夾中的規則目錄。將其新增到 `.claude/settings.local.json`，以便排除保留在您的機器本地：

```
{
  "claudeMdExcludes": [
    "**/monorepo/CLAUDE.md",
    "/home/user/monorepo/other-team/.claude/rules/**"
  ]
}
```

模式使用 glob 語法與絕對檔案路徑匹配。您可以在任何 設定層配置 `claudeMdExcludes`：使用者、專案、本地或受管理的政策。陣列跨層合併。

受管理的政策 CLAUDE.md 檔案無法排除。這確保組織範圍的指令始終適用，無論個別設定如何。

自動記憶

自動記憶讓 Claude 跨工作階段累積知識，無需您編寫任何內容。Claude 在工作時為自己保存筆記：建置命令、除錯見解、架構筆記、程式碼樣式偏好和工作流程習慣。Claude 不會每個工作階段都保存內容。它根據資訊在未來對話中是否有用來決定值得記住的內容。

> **Note:**
>
> 自動記憶需要 Claude Code v2.1.59 或更新版本。使用 `claude --version` 檢查您的版本。

啟用或停用自動記憶

自動記憶預設為開啟。要切換它，在工作階段中開啟 `/memory` 並使用自動記憶切換，或在您的專案設定中設定 `autoMemoryEnabled`：

```
{
  "autoMemoryEnabled": false
}
```

要透過環境變數停用自動記憶，請設定 `CLAUDE_CODE_DISABLE_AUTO_MEMORY=1`。

儲存位置

每個專案在 `~/.claude/projects/<project>/memory/` 獲得自己的記憶目錄。`<project>` 路徑源自 git 儲存庫，因此同一儲存庫內的所有工作樹和子目錄共享一個自動記憶目錄。在 git 儲存庫外，改用專案根目錄。

要將自動記憶儲存在不同位置，請在您的使用者或本地設定中設定 `autoMemoryDirectory`：

```
{
  "autoMemoryDirectory": "~/my-custom-memory-dir"
}
```

此設定從政策、本地和使用者設定接受。它不從專案設定（`.claude/settings.json`）接受，以防止共享專案將自動記憶寫入重定向到敏感位置。

目錄包含 `MEMORY.md` 進入點和可選的主題檔案：

```
~/.claude/projects/<project>/memory/
├── MEMORY.md          # Concise index, loaded into every session
├── debugging.md       # Detailed notes on debugging patterns
├── api-conventions.md # API design decisions
└── ...                # Any other topic files Claude creates
```

`MEMORY.md` 充當記憶目錄的索引。Claude 在您的工作階段中讀取和寫入此目錄中的檔案，使用 `MEMORY.md` 追蹤儲存的內容。

自動記憶是機器本地的。同一 git 儲存庫內的所有工作樹和子目錄共享一個自動記憶目錄。檔案不在機器或雲端環境之間共享。

它如何運作

`MEMORY.md` 的前 200 行在每次對話開始時載入。第 200 行之外的內容在工作階段開始時不載入。Claude 透過將詳細筆記移到單獨的主題檔案中來保持 `MEMORY.md` 簡潔。

此 200 行限制僅適用於 `MEMORY.md`。CLAUDE.md 檔案無論長度如何都完整載入，儘管較短的檔案會產生更好的遵循度。

主題檔案如 `debugging.md` 或 `patterns.md` 在啟動時不載入。Claude 在需要資訊時使用其標準檔案工具按需讀取它們。

Claude 在您的工作階段中讀取和寫入記憶檔案。當您在 Claude Code 介面中看到「Writing memory」或「Recalled memory」時，Claude 正在主動更新或讀取 `~/.claude/projects/<project>/memory/`。

審計和編輯您的記憶

自動記憶檔案是純 markdown，您可以隨時編輯或刪除。執行 `/memory` 以在工作階段中瀏覽和開啟記憶檔案。

使用 `/memory` 檢視和編輯

`/memory` 命令列出在您目前工作階段中載入的所有 CLAUDE.md 和規則檔案，讓您切換自動記憶開啟或關閉，並提供開啟自動記憶資料夾的連結。選擇任何檔案以在您的編輯器中開啟它。

當您要求 Claude 記住某些內容時，例如「始終使用 pnpm，而不是 npm」或「記住 API 測試需要本地 Redis 實例」，Claude 會將其保存到自動記憶。要改為將指令新增到 CLAUDE.md，請直接要求 Claude，例如「將此新增到 CLAUDE.md」，或透過 `/memory` 自己編輯檔案。

疑難排解記憶問題

這些是 CLAUDE.md 和自動記憶最常見的問題，以及除錯步驟。

Claude 不遵循我的 CLAUDE.md

CLAUDE.md 是上下文，不是強制。Claude 讀取它並嘗試遵循它，但沒有嚴格遵循的保證，特別是對於模糊或衝突的指令。

要除錯：

- 執行 `/memory` 驗證您的 CLAUDE.md 檔案被載入。如果檔案未列出，Claude 看不到它。
- 檢查相關的 CLAUDE.md 是否位於為您的工作階段載入的位置（請參閱 選擇 CLAUDE.md 檔案的位置）。
- 使指令更具體。「使用 2 空格縮排」比「正確格式化程式碼」效果更好。
- 尋找跨 CLAUDE.md 檔案的衝突指令。如果兩個檔案為相同行為提供不同的指導，Claude 可能會任意選擇一個。

> **Tip:**
>
> 使用 `InstructionsLoaded` hook 記錄確切載入的指令檔案、何時載入以及為什麼。這對於除錯路徑特定規則或子目錄中的延遲載入檔案很有用。

我不知道自動記憶保存了什麼

執行 `/memory` 並選擇自動記憶資料夾以瀏覽 Claude 保存的內容。一切都是純 markdown，您可以讀取、編輯或刪除。

我的 CLAUDE.md 太大了

超過 200 行的檔案消耗更多上下文，可能會降低遵循度。將詳細內容移到使用 `@path` 匯入參考的單獨檔案（請參閱 匯入其他檔案），或將您的指令分割到 `.claude/rules/` 檔案中。

指令似乎在 `/compact` 後丟失

CLAUDE.md 完全在壓縮中倖存。在 `/compact` 後，Claude 從磁碟重新讀取您的 CLAUDE.md 並將其新鮮重新注入到工作階段中。如果指令在壓縮後消失，它只在對話中給出，未寫入 CLAUDE.md。將其新增到 CLAUDE.md 以使其在工作階段中持久化。

詳見 編寫有效的指令以了解大小、結構和具體性的指導。

相關資源

- Skills：封裝按需載入的可重複工作流程
- 設定：使用設定檔案配置 Claude Code 行為
- 管理工作階段：管理上下文、恢復對話和執行平行工作階段
- Subagent 記憶：讓 subagents 維護自己的自動記憶

Claude Code 設定

> 使用全域和專案層級設定以及環境變數來設定 Claude Code。

Claude Code 提供多種設定選項，可根據您的需求配置其行為。您可以在使用互動式 REPL 時執行 `/config` 命令來設定 Claude Code，這會開啟一個標籤式設定介面，您可以在其中查看狀態資訊並修改設定選項。

設定範圍

Claude Code 使用**範圍系統**來決定設定的適用位置和共享對象。了解範圍可幫助您決定如何為個人使用、團隊協作或企業部署配置 Claude Code。

可用的範圍

範圍	位置	影響對象	與團隊共享？
Managed	伺服器管理的設定、plist / 登錄或系統層級 `managed-settings.json`	機器上的所有使用者	是（由 IT 部署）
User	`~/.claude/` 目錄	您，跨所有專案	否
Project	儲存庫中的 `.claude/`	此儲存庫上的所有協作者	是（提交到 git）
Local	`.claude/settings.local.json`	您，僅在此儲存庫中	否（gitignored）

何時使用各個範圍

Managed 範圍用於：

- 必須在整個組織範圍內強制執行的安全政策
- 無法覆蓋的合規要求
- 由 IT/DevOps 部署的標準化配置

User 範圍最適合：

- 您想在任何地方使用的個人偏好設定（主題、編輯器設定）
- 您在所有專案中使用的工具和 plugins

- API 金鑰和身份驗證（安全儲存）

Project 範圍最適合：

- 團隊共享的設定（權限、hooks、MCP servers）
- 整個團隊應該擁有的 plugins
- 跨協作者標準化工具

Local 範圍最適合：

- 特定專案的個人覆蓋
- 在與團隊共享之前測試配置
- 對其他人不適用的機器特定設定

範圍如何互動

當相同的設定在多個範圍中配置時，更具體的範圍優先：

1. **Managed**（最高）- 無法被任何東西覆蓋
2. **命令列引數** - 臨時工作階段覆蓋
3. **Local** - 覆蓋專案和使用者設定
4. **Project** - 覆蓋使用者設定
5. **User**（最低）- 當沒有其他東西指定設定時適用

例如，如果使用者設定中允許某個權限，但專案設定中拒絕該權限，則專案設定優先，該權限被阻止。

哪些功能使用範圍

範圍適用於許多 Claude Code 功能：

功能	使用者位置	專案位置	Local 位置
Settings	`~/.claude/settings.json`	`.claude/settings.json`	`.claude/settings.local.json`
Subagents	`~/.claude/agents/`	`.claude/agents/`	—
MCP servers	`~/.claude.json`	`.mcp.json`	`~/.claude.json`（每個專案）
Plugins	`~/.claude/settings.json`	`.claude/settings.json`	`.claude/settings.local.json`

功能	使用者位置	專案位置	Local 位置
CLAUDE.md	`~/.claude/CLAUDE.md`	`CLAUDE.md` 或 `.claude/CLAUDE.md`	—

設定檔案

`settings.json` 檔案是我們用於透過分層設定來配置 Claude Code 的官方機制：

- **使用者設定**在 `~/.claude/settings.json` 中定義，適用於所有專案。
- **專案設定**儲存在您的專案目錄中：
  - `.claude/settings.json` 用於簽入原始碼控制並與您的團隊共享的設定
  - `.claude/settings.local.json` 用於未簽入的設定，適用於個人偏好設定和實驗。Claude Code 在建立時會將 `.claude/settings.local.json` 配置為 git 忽略。
- **Managed 設定**：對於需要集中控制的組織，Claude Code 支援多種 managed 設定的傳遞機制。所有機制都使用相同的 JSON 格式，無法被使用者或專案設定覆蓋：
  - **伺服器管理的設定**：透過 Claude.ai 管理員主控台從 Anthropic 的伺服器傳遞。請參閱伺服器管理的設定。
  - **MDM/OS 層級政策**：透過 macOS 和 Windows 上的原生裝置管理傳遞：
  - macOS：`com.anthropic.claudecode` managed preferences 網域（透過 Jamf、Kandji 或其他 MDM 工具中的設定檔案部署）
  - Windows：`HKLM\SOFTWARE\Policies\ClaudeCode` 登錄機碼，其中包含 JSON 的 `Settings` 值（REG_SZ 或 REG_EXPAND_SZ）（透過群組原則或 Intune 部署）
  - Windows（使用者層級）：`HKCU\SOFTWARE\Policies\ClaudeCode`（最低政策優先順序，僅在沒有管理員層級來源時使用）
  - **檔案型**：`managed-settings.json` 和 `managed-mcp.json` 部署到系統目錄：
  - macOS：`/Library/Application Support/ClaudeCode/`
  - Linux 和 WSL：`/etc/claude-code/`
  - Windows：`C:\Program Files\ClaudeCode\`

  請參閱 managed 設定 和 Managed MCP 配置 以取得詳細資訊。

> **Note:**

> Managed 部署也可以使用 `strictKnownMarketplaces` 限制 **plugin marketplace 新增**。如需詳細資訊，請參閱 Managed marketplace 限制。

- **其他配置**儲存在 `~/.claude.json` 中。此檔案包含您的偏好設定（主題、通知設定、編輯器模式）、OAuth 工作階段、MCP server 配置（用於使用者和 local 範圍）、每個專案的狀態（允許的工具、信任設定）和各種快取。專案範圍的 MCP servers 分別儲存在 `.mcp.json` 中。

> **Note:**
>
> Claude Code 會自動建立設定檔案的時間戳記備份，並保留最近五個備份以防止資料遺失。

```
{
  "$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": [
    "歡迎來到 Acme Corp！請在 docs.acme.com 查看我們的程式碼指南",
    "提醒：所有 PR 都需要程式碼審查",
    "新的安全政策已生效"
  ]
}
```

上面範例中的 `$schema` 行指向 Claude Code 設定的官方 JSON 架構。將其新增到您的 `settings.json` 可在 VS Code、Cursor 和任何其他支援 JSON 架構驗證的編輯器中啟用自動完成和內聯驗證。

可用的設定

`settings.json` 支援多個選項：

金鑰	描述	範例
`apiKeyHelper`	自訂指令碼，在 `/bin/sh` 中執行，以產生驗證值。此值將作為 `X-Api-Key` 和 `Authorization: Bearer` 標頭傳送以進行模型請求	`/bin/generate_temp_api_key.sh`
`cleanupPeriodDays`	非作用中超過此期間的工作階段在啟動時被刪除。設定為 `0` 會立即刪除所有工作階段。（預設值：30 天）	`20`
`companyAnnouncements`	在啟動時顯示給使用者的公告。如果提供多個公告，它們將隨機循環。	`["歡迎來到 Acme Corp！請在 docs.acme.com 查看我們的程式碼指南"]`
`env`	將應用於每個工作階段的環境變數	`{"FOO": "bar"}`
`attribution`	自訂 git 提交和提取請求的歸屬。請參閱歸屬設定	`{"commit": "🤖 Generated with Claude Code", "pr": ""}`
`includeCoAuthoredBy`	**已棄用**：改用 `attribution`。是否在 git 提交和提取請求中包含 `co-authored-by Claude` 署名（預設值：`true`）	`false`
`includeGitInstructions`	在 Claude 的系統提示中包含內建提交和 PR 工作流程指示（預設值：`true`）。設定為 `false` 以移除這些指示，例如在使用您自己的 git 工作流程 skills 時。`CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS` 環境變數在設定時優先於此設定	`false`
`permissions`	請參閱下表以了解權限的結構。	
`hooks`	配置自訂命令以在生命週期事件執行。請參閱 hooks 文件 以了解格式	請參閱 hooks
`disableAllHooks`	停用所有 hooks 和任何自訂狀態行	`true`

金鑰	描述	範例
`allowManagedHooksOnly`	（Managed 設定僅限）防止載入使用者、專案和 plugin hooks。僅允許 managed hooks 和 SDK hooks。請參閱 Hook 配置	`true`
`allowedHttpHookUrls`	HTTP hooks 可能針對的 URL 模式的允許清單。支援 `*` 作為萬用字元。設定時，具有不符合 URL 的 hooks 被阻止。未定義 = 無限制，空陣列 = 阻止所有 HTTP hooks。陣列跨設定來源合併。請參閱 Hook 配置	`["https://hooks.example.com/*"]`
`httpHookAllowedEnvVars`	HTTP hooks 可能插入到標頭中的環境變數名稱的允許清單。設定時，每個 hook 的有效 `allowedEnvVars` 是與此清單的交集。未定義 = 無限制。陣列跨設定來源合併。請參閱 Hook 配置	`["MY_TOKEN", "HOOK_SECRET"]`
`allowManagedPermissionRulesOnly`	（Managed 設定僅限）防止使用者和專案設定定義 `allow`、`ask` 或 `deny` 權限規則。僅適用 managed 設定中的規則。請參閱 Managed 專用設定	`true`
`allowManagedMcpServersOnly`	（Managed 設定僅限）僅尊重 managed 設定中的 `allowedMcpServers`。`deniedMcpServers` 仍從所有來源合併。使用者仍可新增 MCP servers，但僅適用管理員定義的允許清單。請參閱 Managed MCP 配置	`true`
`model`	覆蓋 Claude Code 使用的預設模型	`"claude-sonnet-4-6"`
`availableModels`	限制使用者可透過 `/model`、`--model`、Config 工具或 `ANTHROPIC_MODEL` 選擇的模型。不影響預設選項。請參閱限制模型選擇	`["sonnet", "haiku"]`

金鑰	描述	範例
`modelOverrides`	將 Anthropic 模型 ID 對應到提供者特定的模型 ID，例如 Bedrock 推論設定檔 ARN。每個模型選擇器項目在呼叫提供者 API 時使用其對應的值。請參閱覆蓋每個版本的模型 ID	`{"claude-opus-4-6": "arn:aws:bedrock:..."}`
`otelHeadersHelper`	指令碼以產生動態 OpenTelemetry 標頭。在啟動時和定期執行（請參閱動態標頭）	`/bin/generate_otel_headers.sh`
`statusLine`	配置自訂狀態行以顯示上下文。請參閱 `statusLine` 文件	`{"type": "command", "command": "~/.claude/statusline.sh"}`
`fileSuggestion`	為 `@` 檔案自動完成配置自訂指令碼。請參閱檔案建議設定	`{"type": "command", "command": "~/.claude/file-suggestion.sh"}`
`respectGitignore`	控制 `@` 檔案選擇器是否尊重 `.gitignore` 模式。當為 `true`（預設值）時，符合 `.gitignore` 模式的檔案被排除在建議之外	`false`
`outputStyle`	配置輸出樣式以調整系統提示。請參閱輸出樣式文件	`"Explatory"`
`forceLoginMethod`	使用 `claudeai` 限制登入到 Claude.ai 帳戶，`console` 限制登入到 Claude Console（API 使用計費）帳戶	`claudeai`
`forceLoginOrgUUID`	指定組織的 UUID 以在登入期間自動選擇它，繞過組織選擇步驟。需要設定 `forceLoginMethod`	`"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"`
`enableAllProjectMcpServers`	自動批准專案 `.mcp.json` 檔案中定義的所有 MCP servers	`true`
`enabledMcpjsonServers`	要批准的 `.mcp.json` 檔案中特定 MCP servers 的清單	`["memory", "github"]`

金鑰	描述	範例
`disabledMcpjsonServers`	要拒絕的 `.mcp.json` 檔案中特定 MCP servers 的清單	`["filesystem"]`
`allowedMcpServers`	在 managed-settings.json 中設定時，使用者可以配置的 MCP servers 的允許清單。未定義 = 無限制，空陣列 = 鎖定。適用於所有範圍。拒絕清單優先。請參閱 Managed MCP 配置	`[{ "serverName": "github" }]`
`deniedMcpServers`	在 managed-settings.json 中設定時，明確阻止的 MCP servers 的拒絕清單。適用於所有範圍，包括 managed servers。拒絕清單優先於允許清單。請參閱 Managed MCP 配置	`[{ "serverName": "filesystem" }]`
`strictKnownMarketplaces`	在 managed-settings.json 中設定時，使用者可以新增的 plugin marketplaces 的允許清單。未定義 = 無限制，空陣列 = 鎖定。僅適用於 marketplace 新增。請參閱 Managed marketplace 限制	`[{ "source": "github", "repo": "acme-corp/plugins" }]`
`blockedMarketplaces`	（Managed 設定僅限）marketplace 來源的阻止清單。在下載前檢查被阻止的來源，因此它們永遠不會接觸檔案系統。請參閱 Managed marketplace 限制	`[{ "source": "github", "repo": "untrusted/plugins" }]`
`pluginTrustMessage`	（Managed 設定僅限）自訂訊息附加到安裝前顯示的 plugin 信任警告。使用此選項新增組織特定的上下文，例如確認來自您內部 marketplace 的 plugins 已獲得批准。	`"來自我們 marketplace 的所有 plugins 都已獲得 IT 批准"`
`awsAuthRefresh`	修改 `.aws` 目錄的自訂指令碼（請參閱進階認證配置）	`aws sso login --profile myprofile`
`awsCredentialExport`	輸出包含 AWS 認證的 JSON 的自訂指令碼（請參閱進階認證配置）	`/bin/generate_aws_grant.sh`

金鑰	描述	範例
`alwaysThinkingEnabled`	預設為所有工作階段啟用延伸思考。通常透過 `/config` 命令而不是直接編輯來配置	`true`
`plansDirectory`	自訂計畫檔案的儲存位置。路徑相對於專案根目錄。預設值：`~/.claude/plans`	`"./plans"`
`showTurnDuration`	在回應後顯示回合持續時間訊息（例如「Cooked for 1m 6s」）。設定為 `false` 以隱藏這些訊息	`true`
`spinnerVerbs`	自訂在微調器和回合持續時間訊息中顯示的動作動詞。將 `mode` 設定為 `"replace"` 以僅使用您的動詞，或 `"append"` 以將其新增到預設值	`{"mode": "append", "verbs": ["Pondering", "Crafting"]}`
`language`	配置 Claude 的首選回應語言（例如 `"japanese"`、`"spanish"`、`"french` Claude 預設將以此語言回應	`"japanese"`
`autoUpdatesChannel`	遵循更新的發行頻道。使用 `"stable"` 以取得通常約一週舊的版本並跳過具有主要迴歸的版本，或 `"latest"`（預設值）以取得最新版本	`"stable"`
`spinnerTipsEnabled`	在 Claude 工作時在微調器中顯示提示。設定為 `false` 以停用提示（預設值：`true`）	`false`
`spinnerTipsOverride`	使用自訂字串覆蓋微調器提示。`tips`：提示字串陣列。`excludeDefault`：如果為 `true`，僅顯示自訂提示；如果為 `false` 或不存在，自訂提示與內建提示合併	`{ "excludeDefault": true, "tips": ["使用我們的內部工具 X"] }`
`terminalProgressBarEnabled`	啟用終端進度條，在 Windows Terminal 和 iTerm2 等支援的終端中顯示進度（預設值：`true`）	`false`
`prefersReducedMotion`	減少或停用 UI 動畫（微調器、閃爍、閃光效果）以提高可訪問性	`true`

金鑰	描述	範例
`fastModePerSessionOptIn`	當為 `true` 時，快速模式不會跨工作階段持續。每個工作階段都以快速模式關閉開始，需要使用者使用 `/fast` 啟用它。使用者的快速模式偏好設定仍會儲存。請參閱要求每個工作階段的選擇加入	`true`
`teammateMode`	agent team 隊友的顯示方式：`auto`（在 tmux 或 iTerm2 中選擇分割窗格，否則為進程內）、`in-process` 或 `tmux`。請參閱設定 agent teams	`"in-process"`

權限設定

金鑰	描述	範例
`allow`	允許工具使用的權限規則陣列。請參閱下面的權限規則語法以了解模式匹配詳細資訊	`[ "Bash(git diff *)" ]`
`ask`	要求在工具使用時確認的權限規則陣列。請參閱下面的權限規則語法	`[ "Bash(git push *)" ]`
`deny`	拒絕工具使用的權限規則陣列。使用此選項從 Claude Code 存取中排除敏感檔案。請參閱權限規則語法和 Bash 權限限制	`[ "WebFetch", "Bash(curl *)", "Read(./.env)", "Read(./secrets/**)" ]`
`additionalDirectories`	Claude 有權存取的其他工作目錄	`[ "../docs/" ]`
`defaultMode`	開啟 Claude Code 時的預設權限模式	`"acceptEdits"`
`disableBypassPermissionsMode`	設定為 `"disable"` 以防止啟用 `bypassPermissions` 模式。這會停用 `--dangerously-skip-permissions` 命令列旗標。請參閱 managed 設定	`"disable"`

權限規則語法

權限規則遵循 `Tool` 或 `Tool(specifier)` 的格式。規則按順序評估：首先是拒絕規則，然後是詢問，最後是允許。第一個符合的規則獲勝。

快速範例：

規則	效果
`Bash`	符合所有 Bash 命令
`Bash(npm run *)`	符合以 `npm run` 開頭的命令
`Read(./.env)`	符合讀取 `.env` 檔案
`WebFetch(domain:example.com)`	符合對 example.com 的提取請求

如需完整的規則語法參考，包括萬用字元行為、Read、Edit、WebFetch、MCP 和 Agent 規則的工具特定模式，以及 Bash 模式的安全限制，請參閱權限規則語法。

Sandbox 設定

配置進階 sandboxing 行為。Sandboxing 將 bash 命令與您的檔案系統和網路隔離。請參閱 Sandboxing 以了解詳細資訊。

金鑰	描述	範例
`enabled`	啟用 bash sandboxing（macOS、Linux 和 WSL2）。預設值：false	`true`
`autoAllowBashIfSandboxed`	在 sandboxed 時自動批准 bash 命令。預設值：true	`true`
`excludedCommands`	應在 sandbox 外執行的命令	`["git", "docker"]`
`allowUnsandboxedCommands`	允許命令透過 `dangerouslyDisableSandbox` 參數在 sandbox 外執行。當設定為 `false` 時，`dangerouslyDisableSandbox` 逃脫艙完全停用，所有命令必須 sandboxed（或在 `excludedCommands` 中）。對於需要嚴格 sandboxing 的企業政策很有用。預設值：true	`false`

金鑰	描述	範例
`filesystem.allowWrite`	sandboxed 命令可以寫入的其他路徑。陣列跨所有設定範圍合併：使用者、專案和 managed 路徑合併，不替換。也與 `Edit(...)` 允許權限規則中的路徑合併。請參閱下面的路徑前綴。	`["//tmp/build", "~/.kube"]`
`filesystem.denyWrite`	sandboxed 命令無法寫入的路徑。陣列跨所有設定範圍合併。也與 `Edit(...)` 拒絕權限規則中的路徑合併。	`["//etc", "//usr/local/bin"]`
`filesystem.denyRead`	sandboxed 命令無法讀取的路徑。陣列跨所有設定範圍合併。也與 `Read(...)` 拒絕權限規則中的路徑合併。	`["~/.aws/credentials"]`
`network.allowUnixSockets`	sandbox 中可存取的 Unix socket 路徑（用於 SSH 代理等）	`["~/.ssh/agent-socket"]`
`network.allowAllUnixSockets`	允許 sandbox 中的所有 Unix socket 連線。預設值：false	`true`
`network.allowLocalBinding`	允許繫結到 localhost 連接埠（僅限 macOS）。預設值：false	`true`
`network.allowedDomains`	允許出站網路流量的網域陣列。支援萬用字元（例如 `*.example.com`）。	`["github.com", "*.npmjs.org"]`
`network.allowManagedDomainsOnly`	（Managed 設定僅限）僅尊重 managed 設定中的 `allowedDomains` 和 `WebFetch(domain:...)` 允許規則。來自使用者、專案和 local 設定的網域被忽略。非允許的網域自動被阻止，不提示使用者。拒絕的網域仍從所有來源受尊重。預設值：false	`true`
`network.httpProxyPort`	如果您想帶上自己的代理，則使用的 HTTP 代理連接埠。如果未指定，Claude 將執行自己的代理。	`8080`

金鑰	描述	範例
`network.socksProxyPort`	如果您想帶上自己的代理，則使用的 SOCKS5 代理連接埠。如果未指定，Claude 將執行自己的代理。	`8081`
`enableWeakerNestedSandbox`	為無特權 Docker 環境啟用較弱的 sandbox（僅限 Linux 和 WSL2）。**降低安全性。** 預設值：false	`true`
`enableWeakerNetworkIsolation`	（僅限 macOS）允許在 sandbox 中存取系統 TLS 信任服務（`com.apple.trustd.agent`）。對於 Go 型工具（如 `gh`、`gcloud` 和 `terraform`）在使用 `httpProxyPort` 與 MITM 代理和自訂 CA 時驗證 TLS 憑證是必需的。**降低安全性**，開啟潛在的資料洩露路徑。預設值：false	`true`

Sandbox 路徑前綴

`filesystem.allowWrite`、`filesystem.denyWrite` 和 `filesystem.denyRead` 中的路徑支援這些前綴：

前綴	含義	範例
`//`	從檔案系統根目錄的絕對路徑	`//tmp/build` 變成 `/tmp/build`
`~/`	相對於主目錄	`~/.kube` 變成 `$HOME/.kube`
`/`	相對於設定檔案的目錄	`/build` 變成 `$SETTINGS_DIR/build`
`./` 或無前綴	相對路徑（由 sandbox 執行時解析）	`./output`

配置範例：

```
{
  "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
    }
  }
}
```

檔案系統和網路限制可以透過兩種合併在一起的方式配置：

- **`sandbox.filesystem` 設定**（如上所示）：在 OS 層級 sandbox 邊界控制路徑。這些限制適用於所有子流程命令（例如 `kubectl`、`terraform`、`npm`），而不僅僅是 Claude 的檔案工具。
- **權限規則**：使用 `Edit` 允許/拒絕規則控制 Claude 的檔案工具存取，`Read` 拒絕規則阻止讀取，`WebFetch` 允許/拒絕規則控制網路網域。這些規則中的路徑也合併到 sandbox 配置中。

歸屬設定

Claude Code 將歸屬新增到 git 提交和提取請求。這些分別配置：

- 提交預設使用 git trailers（如 `Co-Authored-By`），可以自訂或停用
- 提取請求描述是純文字

金鑰	描述
`commit`	git 提交的歸屬，包括任何 trailers。空字串隱藏提交歸屬
`pr`	提取請求描述的歸屬。空字串隱藏提取請求歸屬

預設提交歸屬：

```
🤖 Generated with [Claude Code](https://claude.com/claude-code)

   Co-Authored-By: Claude Sonnet 4.6 <noreply@anthropic.com>
```

預設提取請求歸屬：

```
🤖 Generated with [Claude Code](https://claude.com/claude-code)
```

範例：

```
{
  "attribution": {
    "commit": "Generated with AI\n\nCo-Authored-By: AI <ai@example.com>",
    "pr": ""
  }
}
```

> **Note:**
>
> `attribution` 設定優先於已棄用的 `includeCoAuthoredBy` 設定。若要隱藏所有歸屬，請將 `commit` 和 `pr` 設定為空字串。

檔案建議設定

為 `@` 檔案路徑自動完成配置自訂命令。內建檔案建議使用快速檔案系統遍歷，但大型 monorepos 可能受益於專案特定的索引，例如預先建立的檔案索引或自訂工具。

```
{
  "fileSuggestion": {
    "type": "command",
    "command": "~/.claude/file-suggestion.sh"
  }
}
```

該命令使用與 hooks 相同的環境變數執行，包括 `CLAUDE_PROJECT_DIR`。它透過 stdin 接收包含 `query` 欄位的 JSON：

```
{"query": "src/comp"}
```

將換行符分隔的檔案路徑輸出到 stdout（目前限制為 15）：

```
src/components/Button.tsx
src/components/Modal.tsx
src/components/Form.tsx
```

範例：

```
#!/bin/bash
query=$(cat | jq -r '.query')
your-repo-file-index --query "$query" | head -20
```

Hook 配置

這些設定控制允許執行哪些 hooks 以及 HTTP hooks 可以存取的內容。`allowManagedHooksOnly` 設定只能在 managed 設定中配置。URL 和環境變數允許清單可以在任何設定層級設定，並跨來源合併。

當 `allowManagedHooksOnly` 為 `true` 時的行為：

- 載入 managed hooks 和 SDK hooks
- 使用者 hooks、專案 hooks 和 plugin hooks 被阻止

限制 HTTP hook URL：

限制 HTTP hooks 可以針對的 URL。支援 `*` 作為匹配的萬用字元。定義陣列時，針對不符合 URL 的 HTTP hooks 被無聲地阻止。

```
{
  "allowedHttpHookUrls": ["https://hooks.example.com/*", "http://localhost:*"]
}
```

限制 HTTP hook 環境變數：

限制 HTTP hooks 可以插入到標頭值中的環境變數名稱。每個 hook 的有效 `allowedEnvVars` 是其自己清單與此設定的交集。

```
{
  "httpHookAllowedEnvVars": ["MY_TOKEN", "HOOK_SECRET"]
}
```

設定優先順序

設定按優先順序順序應用。從最高到最低：

1. **Managed 設定**（伺服器管理、MDM/OS 層級政策 或 managed 設定）
   - 由 IT 透過伺服器傳遞、MDM 設定檔案、登錄政策或 managed 設定檔案部署的政策
   - 無法被任何其他層級覆蓋，包括命令列引數
   - 在 managed 層級內，優先順序為：伺服器管理 > MDM/OS 層級政策 > `managed-settings.json` > HKCU 登錄（僅限 Windows）。僅使用一個 managed 來源；來源不合併。
2. **命令列引數**
   - 特定工作階段的臨時覆蓋
3. **Local 專案設定**（`.claude/settings.local.json`）
   - 個人專案特定設定
4. **共享專案設定**（`.claude/settings.json`）
   - 原始碼控制中的團隊共享專案設定
5. **使用者設定**（`~/.claude/settings.json`）
   - 個人全域設定

此階層確保組織政策始終被強制執行，同時仍允許團隊和個人自訂其體驗。

例如，如果您的使用者設定允許 `Bash(npm run *)`，但專案的共享設定拒絕它，則專案設定優先，命令被阻止。

> **Note:**

> **陣列設定跨範圍合併。** 當相同的陣列值設定（例如 `sandbox.filesystem.allowWrite` 或 `permissions.allow`）出現在多個範圍中時，陣列被**連接和去重**，而不是替換。這意味著較低優先順序的範圍可以新增項目而不覆蓋由較高優先順序範圍設定的項目，反之亦然。例如，如果 managed 設定將 `allowWrite` 設定為 `["//opt/company-tools"]`，使用者新增 `["~/.kube"]`，則最終配置中包含兩個路徑。

驗證作用中的設定

在 Claude Code 內執行 `/status` 以查看哪些設定來源是作用中的以及它們來自何處。輸出顯示每個配置層（managed、使用者、專案）及其來源，例如 `Enterprise managed settings (remote)`、`Enterprise managed settings (plist)`、`Enterprise managed settings (HKLM)` 或 `Enterprise managed settings (file)`。如果設定檔案包含錯誤，`/status` 會報告問題，以便您可以修復它。

關於配置系統的要點

- **記憶體檔案（`CLAUDE.md`）**：包含 Claude 在啟動時載入的指示和上下文
- **設定檔案（JSON）**：配置權限、環境變數和工具行為
- **Skills**：可以使用 `/skill-name` 叫用或由 Claude 自動載入的自訂提示
- **MCP servers**：使用其他工具和整合擴展 Claude Code
- **優先順序**：較高層級的配置（Managed）覆蓋較低層級的配置（User/Project）
- **繼承**：設定被合併，更具體的設定新增到或覆蓋更廣泛的設定

系統提示

Claude Code 的內部系統提示未發佈。若要新增自訂指示，請使用 `CLAUDE.md` 檔案或 `--append-system-prompt` 旗標。

排除敏感檔案

若要防止 Claude Code 存取包含敏感資訊（如 API 金鑰、機密和環境檔案）的檔案，請在您的 `.claude/settings.json` 檔案中使用 `permissions.deny` 設定：

```
{
  "permissions": {
    "deny": [
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(./secrets/**)",
      "Read(./config/credentials.json)",
      "Read(./build)"
    ]
  }
}
```

這取代了已棄用的 `ignorePatterns` 配置。符合這些模式的檔案被排除在檔案發現和搜尋結果之外，這些檔案上的讀取操作被拒絕。

Subagent 配置

Claude Code 支援可在使用者和專案層級配置的自訂 AI subagents。這些 subagents 儲存為具有 YAML frontmatter 的 Markdown 檔案：

- **使用者 subagents**：`~/.claude/agents/` - 在所有專案中可用
- **專案 subagents**：`.claude/agents/` - 特定於您的專案，可與您的團隊共享

Subagent 檔案定義具有自訂提示和工具權限的專門 AI 助手。在 subagents 文件中了解更多關於建立和使用 subagents 的資訊。

Plugin 配置

Claude Code 支援 plugin 系統，可讓您使用 skills、agents、hooks 和 MCP servers 擴展功能。Plugins 透過 marketplaces 分發，可以在使用者和儲存庫層級配置。

Plugin 設定

`settings.json` 中的 plugin 相關設定：

```
{
  "enabledPlugins": {
    "formatter@acme-tools": true,
    "deployer@acme-tools": true,
    "analyzer@security-plugins": false
  },
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": "github",
      "repo": "acme-corp/claude-plugins"
    }
  }
}
```

`enabledPlugins`

控制啟用哪些 plugins。格式：`"plugin-name@marketplace-name": true/false`

範圍：

- **使用者設定**（`~/.claude/settings.json`）：個人 plugin 偏好設定
- **專案設定**（`.claude/settings.json`）：與團隊共享的專案特定 plugins
- **Local 設定**（`.claude/settings.local.json`）：每台機器的覆蓋（未提交）

範例：

```
{
  "enabledPlugins": {
    "code-formatter@team-tools": true,
    "deployment-tools@team-tools": true,
    "experimental-features@personal": false
  }
}
```

`extraKnownMarketplaces`

定義應為儲存庫提供的其他 marketplaces。通常在儲存庫層級設定中使用，以確保團隊成員有權存取所需的 plugin 來源。

當儲存庫包含 `extraKnownMarketplaces` 時：

1. 當團隊成員信任資料夾時，系統會提示他們安裝 marketplace
2. 然後系統會提示團隊成員安裝來自該 marketplace 的 plugins
3. 使用者可以跳過不需要的 marketplaces 或 plugins（儲存在使用者設定中）
4. 安裝尊重信任邊界並需要明確同意

範例：

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/claude-plugins"
      }
    },
    "security-plugins": {
      "source": {
        "source": "git",
        "url": "https://git.example.com/security/plugins.git"
      }
    }
  }
}
```

Marketplace 來源類型：

- `github`：GitHub 儲存庫（使用 `repo`）
- `git`：任何 git URL（使用 `url`）
- `directory`：本機檔案系統路徑（使用 `path`，僅用於開發）
- `hostPattern`：正規表達式模式以符合 marketplace 主機（使用 `hostPattern`）

`strictKnownMarketplaces`

Managed 設定僅限：控制使用者可以新增哪些 plugin marketplaces。此設定只能在 managed 設定中配置，為管理員提供對 marketplace 來源的嚴格控制。

Managed 設定檔案位置：

- **macOS：**`/Library/Application Support/ClaudeCode/managed-settings.json`

- **Linux 和 WSL**：`/etc/claude-code/managed-settings.json`
- **Windows**：`C:\Program Files\ClaudeCode\managed-settings.json`

主要特徵：

- 僅在 managed 設定（`managed-settings.json`）中可用
- 無法被使用者或專案設定覆蓋（最高優先順序）
- 在網路/檔案系統操作之前強制執行（被阻止的來源永遠不會執行）
- 對來源規格使用精確匹配（包括 git 來源的 `ref`、`path`），除了 `hostPattern`，它使用正規表達式匹配

允許清單行為：

- `undefined`（預設值）：無限制 - 使用者可以新增任何 marketplace
- 空陣列 `[]`：完全鎖定 - 使用者無法新增任何新 marketplaces
- 來源清單：使用者只能新增完全符合的 marketplaces

所有支援的來源類型：

允許清單支援七種 marketplace 來源類型。大多數來源使用精確匹配，而 `hostPattern` 使用正規表達式匹配 marketplace 主機。

1. **GitHub 儲存庫**：

```
{ "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": "marketp
lace" }
```

欄位：`repo`（必需）、`ref`（可選：分支/標籤/SHA）、`path`（可選：子目錄）

1. **Git 儲存庫**：

```
{ "source": "git", "url": "https://gitlab.example.com/tools/plugins.git" }
{ "source": "git", "url": "https://bitbucket.org/acme-corp/plugins.git", "ref": "p
roduction" }
{ "source": "git", "url": "ssh://git@git.example.com/plugins.git", "ref": "v3.1",
"path": "approved" }
```

欄位：`url`（必需）、`ref`（可選：分支/標籤/SHA）、`path`（可選：子目錄）

1. **基於 URL 的 marketplaces**：

```
{ "source": "url", "url": "https://plugins.example.com/marketplace.json" }
{ "source": "url", "url": "https://cdn.example.com/marketplace.json", "headers":
{ "Authorization": "Bearer ${TOKEN}" } }
```

欄位：`url`（必需）、`headers`（可選：用於驗證存取的 HTTP 標頭）

> **Note:**
>
> 基於 URL 的 marketplaces 僅下載 `marketplace.json` 檔案。它們不從伺服器下載 plugin 檔案。基於 URL 的 marketplaces 中的 Plugins 必須使用外部來源（GitHub、npm 或 git URL），而不是相對路徑。對於具有相對路徑的 plugins，請改用基於 Git 的 marketplace。請參閱故障排除以了解詳細資訊。

1. **NPM 套件**：

```
{ "source": "npm", "package": "@acme-corp/claude-plugins" }
{ "source": "npm", "package": "@acme-corp/approved-marketplace" }
```

欄位：`package`（必需，支援範圍套件）

1. **檔案路徑**：

```
{ "source": "file", "path": "/usr/local/share/claude/acme-marketplace.json" }
{ "source": "file", "path": "/opt/acme-corp/plugins/marketplace.json" }
```

欄位：`path`（必需：marketplace.json 檔案的絕對路徑）

1. **目錄路徑**：

```
{ "source": "directory", "path": "/usr/local/share/claude/acme-plugins" }
{ "source": "directory", "path": "/opt/acme-corp/approved-marketplaces" }
```

欄位：`path`（必需：包含 `.claude-plugin/marketplace.json` 的目錄的絕對路徑）

1. **主機模式匹配**：

```
{ "source": "hostPattern", "hostPattern": "^github\\.example\\.com$" }
{ "source": "hostPattern", "hostPattern": "^gitlab\\.internal\\.example\\.com$" }
```

欄位：`hostPattern`（必需：正規表達式模式以匹配 marketplace 主機）

當您想允許來自特定主機的所有 marketplaces 而不列舉每個儲存庫時，請使用主機模式匹配。這對於具有內部 GitHub Enterprise 或 GitLab 伺服器的組織很有用，開發人員在其中建立自己的 marketplaces。

按來源類型的主機提取：

- `github`：始終匹配 `github.com`
- `git`：從 URL 提取主機名稱（支援 HTTPS 和 SSH 格式）
- `url`：從 URL 提取主機名稱
- `npm`、`file`、`directory`：不支援主機模式匹配

配置範例：

範例：僅允許特定 marketplaces：

```
{
  "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"
    }
  ]
}
```

範例 - 停用所有 marketplace 新增：

```
{
  "strictKnownMarketplaces": []
}
```

範例：允許來自內部 git 伺服器的所有 marketplaces：

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

精確匹配要求：

Marketplace 來源必須**完全符合**才能允許使用者的新增。對於基於 git 的來源（`github` 和 `git`），這包括所有可選欄位：

- `repo` 或 `url` 必須完全符合
- `ref` 欄位必須完全符合（或兩者都未定義）
- `path` 欄位必須完全符合（或兩者都未定義）

不符合的來源範例：

```
// 這些是不同的來源：
{ "source": "github", "repo": "acme-corp/plugins" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main" }

// 這些也是不同的：
{ "source": "github", "repo": "acme-corp/plugins", "path": "marketplace" }
{ "source": "github", "repo": "acme-corp/plugins" }
```

與 `extraKnownMarketplaces` 的比較：

方面	`strictKnownMarketplaces`	`extraKnownMarketplaces`
目的	組織政策強制執行	團隊便利
設定檔案	僅 `managed-settings.json`	任何設定檔案
行為	阻止非允許清單新增	自動安裝遺失的 marketplaces

方面	`strictKnownMarketplaces`	`extraKnownMarketplaces`
何時強制執行	在網路/檔案系統操作之前	在使用者信任提示之後
可以被覆蓋	否（最高優先順序）	是（由較高優先順序設定）
來源格式	直接來源物件	具有巢狀來源的命名 marketplace
使用案例	合規、安全限制	上線、標準化

格式差異：

`strictKnownMarketplaces` 使用直接來源物件：

```
{
  "strictKnownMarketplaces": [
    { "source": "github", "repo": "acme-corp/plugins" }
  ]
}
```

`extraKnownMarketplaces` 需要命名 marketplaces：

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": { "source": "github", "repo": "acme-corp/plugins" }
    }
  }
}
```

重要注意事項：

- 限制在任何網路請求或檔案系統操作之前檢查
- 被阻止時，使用者會看到清晰的錯誤訊息，指示來源被 managed 政策阻止
- 限制僅適用於新增 NEW marketplaces；先前安裝的 marketplaces 保持可存取
- Managed 設定具有最高優先順序，無法被覆蓋

請參閱 Managed marketplace 限制以了解面向使用者的文件。

管理 plugins

使用 `/plugin` 命令以互動方式管理 plugins：

- 瀏覽來自 marketplaces 的可用 plugins
- 安裝/卸載 plugins
- 啟用/停用 plugins
- 檢視 plugin 詳細資訊（提供的命令、agents、hooks）
- 新增/移除 marketplaces

在 plugins 文件中了解更多關於 plugin 系統的資訊。

環境變數

Claude Code 支援以下環境變數來控制其行為：

> **Note:**
>
> 所有環境變數也可以在 `settings.json` 中配置。這是自動為每個工作階段設定環境變數或為整個團隊或組織推出一組環境變數的有用方式。

變數	目的	
`ANTHROPIC_API_KEY`	作為 `X-Api-Key` 標頭傳送的 API 金鑰，通常用於 Claude SDK（對於互動式使用，執行 `/login`）	
`ANTHROPIC_AUTH_TOKEN`	`Authorization` 標頭的自訂值（您在此設定的值將以 `Bearer` 為前綴）	
`ANTHROPIC_CUSTOM_HEADERS`	要新增到請求的自訂標頭（`Name: Value` 格式，多個標頭用換行符分隔）	
`ANTHROPIC_DEFAULT_HAIKU_MODEL`	請參閱模型配置	
`ANTHROPIC_DEFAULT_OPUS_MODEL`	請參閱模型配置	
`ANTHROPIC_DEFAULT_SONNET_MODEL`	請參閱模型配置	

變數	目的	
`ANTHROPIC_FOUNDRY_API_KEY`	Microsoft Foundry 驗證的 API 金鑰（請參閱 Microsoft Foundry）	
`ANTHROPIC_FOUNDRY_BASE_URL`	Foundry 資源的完整基本 URL（例如 `https://my-resource.services.ai.azure.com/anthropic`）。`ANTHROPIC_FOUNDRY_RESOURCE` 的替代方案（請參閱 Microsoft Foundry）	
`ANTHROPIC_FOUNDRY_RESOURCE`	Foundry 資源名稱（例如 `my-resource`）。如果未設定 `ANTHROPIC_FOUNDRY_BASE_URL`，則為必需（請參閱 Microsoft Foundry）	
`ANTHROPIC_MODEL`	要使用的模型設定名稱（請參閱模型配置）	
`ANTHROPIC_SMALL_FAST_MODEL`	[已棄用] Haiku 級模型用於背景工作的名稱	
`ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION`	使用 Bedrock 時覆蓋 Haiku 級模型的 AWS 區域	
`AWS_BEARER_TOKEN_BEDROCK`	Bedrock API 金鑰用於驗證（請參閱 Bedrock API 金鑰）	
`BASH_DEFAULT_TIMEOUT_MS`	長時間執行 bash 命令的預設逾時	
`BASH_MAX_OUTPUT_LENGTH`	bash 輸出中的最大字元數，超過此數量後會進行中間截斷	
`BASH_MAX_TIMEOUT_MS`	模型可以為長時間執行 bash 命令設定的最大逾時	

變數	目的	
`CLAUDE_AUTOCOMPACT_PCT_OVERRIDE`	設定自動壓縮觸發的上下文容量百分比（1-100）。預設情況下，自動壓縮在約 95% 容量時觸發。使用較低的值（如 `50`）以更早壓縮。高於預設閾值的值無效。適用於主要對話和 subagents。此百分比與狀態行中可用的 `context_window.used_percentage` 欄位對齊	
`CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR`	在每個 Bash 命令後返回原始工作目錄	
`CLAUDE_CODE_ACCOUNT_UUID`	已驗證使用者的帳戶 UUID。由 SDK 呼叫者使用以同步提供帳戶資訊，避免早期遙測事件缺少帳戶中繼資料的競爭條件。需要同時設定 `CLAUDE_CODE_USER_EMAIL` 和 `CLAUDE_CODE_ORGANIZATION_UUID`	
`CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD`	設定為 `1` 以從使用 `--add-dir` 指定的目錄載入 CLAUDE.md 檔案。預設情況下，其他目錄不載入記憶體檔案	`1`
`CLAUDE_CODE_API_KEY_HELPER_TTL_MS`	應刷新認證的間隔（以毫秒為單位）（使用 `apiKeyHelper` 時）	
`CLAUDE_CODE_CLIENT_CERT`	用於 mTLS 驗證的用戶端憑證檔案的路徑	
`CLAUDE_CODE_CLIENT_KEY`	用於 mTLS 驗證的用戶端私密金鑰檔案的路徑	
`CLAUDE_CODE_CLIENT_KEY_PASSPHRASE`	加密 CLAUDE_CODE_CLIENT_KEY 的密碼（可選）	
`CLAUDE_CODE_DISABLE_1M_CONTEXT`	設定為 `1` 以停用 1M 上下文視窗支援。設定時，1M 模型變體在模型選擇器中不可用。對於具有合規要求的企業環境很有用	

變數	目的	
`CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING`	設定為 `1` 以停用 Opus 4.6 和 Sonnet 4.6 的自適應推理。停用時，這些模型回退到由 `MAX_THINKING_TOKENS` 控制的固定思考預算	
`CLAUDE_CODE_DISABLE_AUTO_MEMORY`	設定為 `1` 以停用自動記憶。設定為 `0` 以在逐步推出期間強制啟用自動記憶。停用時，Claude 不建立或載入自動記憶檔案	
`CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS`	設定為 `1` 以從 Claude 的系統提示中移除內建提交和 PR 工作流程指示。在使用您自己的 git 工作流程 skills 時很有用。設定時優先於 `includeGitInstructions` 設定	
`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS`	設定為 `1` 以停用所有背景工作功能，包括 Bash 和 subagent 工具上的 `run_in_background` 參數、自動背景化和 Ctrl+B 快捷鍵	
`CLAUDE_CODE_DISABLE_CRON`	設定為 `1` 以停用排程工作。`/loop` skill 和 cron 工具變得不可用，任何已排程的工作停止觸發，包括已在工作階段中執行的工作	
`CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS`	設定為 `1` 以停用 Anthropic API 特定的 `anthropic-beta` 標頭。如果在使用具有第三方提供者的 LLM 閘道時遇到「Unexpected value(s) for the `anthropic-beta` header」之類的問題，請使用此選項	
`CLAUDE_CODE_DISABLE_FAST_MODE`	設定為 `1` 以停用快速模式	
`CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY`	設定為 `1` 以停用「Claude 表現如何？」工作階段品質調查。在使用第三方提供者或停用遙測時也會停用。請參閱工作階段品質調查	

變數	目的	
`CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC`	等同於設定 `DISABLE_AUTOUPDATER`、`DISABLE_BUG_COMMAND`、`DISABLE_ERROR_REPORTING` 和 `DISABLE_TELEMETRY`	
`CLAUDE_CODE_DISABLE_TERMINAL_TITLE`	設定為 `1` 以停用基於對話上下文的自動終端標題更新	
`CLAUDE_CODE_EFFORT_LEVEL`	為支援的模型設定努力級別。值：`low`、`medium`、`high`。較低的努力更快且更便宜，較高的努力提供更深入的推理。在 Opus 4.6 和 Sonnet 4.6 上支援。請參閱調整努力級別	
`CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION`	設定為 `false` 以停用提示建議（`/config` 中的「提示建議」切換）。這些是 Claude 回應後出現在您的提示輸入中的灰色預測。請參閱提示建議	
`CLAUDE_CODE_ENABLE_TASKS`	設定為 `false` 以暫時還原為先前的 TODO 清單而不是工作追蹤系統。預設值：`true`。請參閱工作清單	
`CLAUDE_CODE_ENABLE_TELEMETRY`	設定為 `1` 以啟用用於指標和日誌的 OpenTelemetry 資料收集。在配置 OTel 匯出器之前需要。請參閱監控	
`CLAUDE_CODE_EXIT_AFTER_STOP_DELAY`	查詢迴圈變為閒置後自動退出前等待的時間（以毫秒為單位）。對於使用 SDK 模式的自動化工作流程和指令碼很有用	
`CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS`	設定為 `1` 以啟用 agent teams。Agent teams 是實驗性的，預設停用	
`CLAUDE_CODE_FILE_READ_MAX_OUTPUT_TOKENS`	覆蓋檔案讀取的預設令牌限制。當您需要完整讀取較大的檔案時很有用	
`CLAUDE_CODE_HIDE_ACCOUNT_INFO`	設定為 `1` 以從 Claude Code UI 隱藏您的電子郵件地址和組織名稱。在串流或錄製時很有用	

變數	目的	
`CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL`	跳過 IDE 擴展的自動安裝	
`CLAUDE_CODE_MAX_OUTPUT_TOKENS`	為大多數請求設定最大輸出令牌數。預設值：32,000。最大值：64,000。增加此值會減少在自動壓縮觸發之前可用的有效上下文視窗。	
`CLAUDE_CODE_ORGANIZATION_UUID`	已驗證使用者的組織 UUID。由 SDK 呼叫者使用以同步提供帳戶資訊。需要同時設定 `CLAUDE_CODE_ACCOUNT_UUID` 和 `CLAUDE_CODE_USER_EMAIL`	
`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS`	刷新動態 OpenTelemetry 標頭的間隔（以毫秒為單位）（預設值：1740000 / 29 分鐘）。請參閱動態標頭	
`CLAUDE_CODE_PLAN_MODE_REQUIRED`	自動設定為 `true` 在需要計畫批准的 agent team 隊友上。唯讀：在生成隊友時由 Claude Code 設定。請參閱要求隊友的計畫批准	
`CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS`	安裝或更新 plugins 時 git 操作的逾時（以毫秒為單位）（預設值：120000）。對於大型儲存庫或緩慢的網路連線，增加此值。請參閱 Git 操作逾時	
`CLAUDE_CODE_PROXY_RESOLVES_HOSTS`	設定為 `true` 以允許代理執行 DNS 解析而不是呼叫者。對於代理應處理主機名稱解析的環境選擇加入	
`CLAUDE_CODE_SHELL`	覆蓋自動 shell 偵測。當您的登入 shell 與您的首選工作 shell 不同時很有用（例如 `bash` 與 `zsh`）	
`CLAUDE_CODE_SHELL_PREFIX`	命令前綴以包裝所有 bash 命令（例如用於日誌或審計）。範例：`/path/to/logger.sh` 將執行 `/path/to/logger.sh <command>`	

變數	目的	
`CLAUDE_CODE_SIMPLE`	設定為 `1` 以使用最小系統提示和僅 Bash、檔案讀取和檔案編輯工具執行。停用 MCP 工具、附件、hooks 和 CLAUDE.md 檔案	
`CLAUDE_CODE_SKIP_BEDROCK_AUTH`	跳過 Bedrock 的 AWS 驗證（例如在使用 LLM 閘道時）	
`CLAUDE_CODE_SKIP_FOUNDRY_AUTH`	跳過 Microsoft Foundry 的 Azure 驗證（例如在使用 LLM 閘道時）	
`CLAUDE_CODE_SKIP_VERTEX_AUTH`	跳過 Vertex 的 Google 驗證（例如在使用 LLM 閘道時）	
`CLAUDE_CODE_SUBAGENT_MODEL`	請參閱模型配置	
`CLAUDE_CODE_TASK_LIST_ID`	跨工作階段共享工作清單。在多個 Claude Code 實例中設定相同的 ID 以協調共享工作清單。請參閱工作清單	
`CLAUDE_CODE_TEAM_NAME`	此隊友所屬的 agent team 的名稱。在 agent team 成員上自動設定	
`CLAUDE_CODE_TMPDIR`	覆蓋用於內部臨時檔案的臨時目錄。Claude Code 將 `/claude/` 附加到此路徑。預設值：Unix/macOS 上的 `/tmp`，Windows 上的 `os.tmpdir()`	
`CLAUDE_CODE_USER_EMAIL`	已驗證使用者的電子郵件地址。由 SDK 呼叫者使用以同步提供帳戶資訊。需要同時設定 `CLAUDE_CODE_ACCOUNT_UUID` 和 `CLAUDE_CODE_ORGANIZATION_UUID`	
`CLAUDE_CODE_USE_BEDROCK`	使用 Bedrock	
`CLAUDE_CODE_USE_FOUNDRY`	使用 Microsoft Foundry	

變數	目的	
`CLAUDE_CODE_USE_VERTEX`	使用 Vertex	
`CLAUDE_CONFIG_DIR`	自訂 Claude Code 儲存其配置和資料檔案的位置	
`DISABLE_AUTOUPDATER`	設定為 `1` 以停用自動更新。	
`DISABLE_BUG_COMMAND`	設定為 `1` 以停用 `/bug` 命令	
`DISABLE_COST_WARNINGS`	設定為 `1` 以停用成本警告訊息	
`DISABLE_ERROR_REPORTING`	設定為 `1` 以選擇退出 Sentry 錯誤報告	
`DISABLE_INSTALLATION_CHECKS`	設定為 `1` 以停用安裝警告。僅在手動管理安裝位置時使用，因為這可能會掩蓋標準安裝的問題	
`DISABLE_NON_ESSENTIAL_MODEL_CALLS`	設定為 `1` 以停用非關鍵路徑（如風味文字）的模型呼叫	
`DISABLE_PROMPT_CACHING`	設定為 `1` 以停用所有模型的提示快取（優先於每個模型的設定）	
`DISABLE_PROMPT_CACHING_HAIKU`	設定為 `1` 以停用 Haiku 模型的提示快取	
`DISABLE_PROMPT_CACHING_OPUS`	設定為 `1` 以停用 Opus 模型的提示快取	
`DISABLE_PROMPT_CACHING_SONNET`	設定為 `1` 以停用 Sonnet 模型的提示快取	
`DISABLE_TELEMETRY`	設定為 `1` 以選擇退出 Statsig 遙測（請注意 Statsig 事件不包含使用者資料，如程式碼、檔案路徑或 bash 命令）	
`ENABLE_CLAUDEAI_MCP_SERVERS`	設定為 `false` 以停用 Claude Code 中的 claude.ai MCP servers。對於已登入的使用者預設啟用	

變數	目的	
`ENABLE_TOOL_SEARCH`	控制 MCP 工具搜尋。值：`auto`（預設值，在 10% 上下文時啟用）、`auto:N`（自訂閾值，例如 `auto:5` 表示 5%）、`true`（始終開啟）、`false`（停用）	
`FORCE_AUTOUPDATE_PLUGINS`	設定為 `true` 以強制 plugin 自動更新，即使主自動更新器透過 `DISABLE_AUTOUPDATER` 停用	
`HTTP_PROXY`	為網路連線指定 HTTP 代理伺服器	
`HTTPS_PROXY`	為網路連線指定 HTTPS 代理伺服器	
`IS_DEMO`	設定為 `true` 以啟用演示模式：從 UI 隱藏電子郵件和組織、跳過上線和隱藏內部命令。對於串流或錄製工作階段很有用	
`MAX_MCP_OUTPUT_TOKENS`	MCP 工具回應中允許的最大令牌數。當輸出超過 10,000 個令牌時，Claude Code 顯示警告（預設值：25000）	
`MAX_THINKING_TOKENS`	覆蓋延伸思考令牌預算。思考預設在最大預算（31,999 個令牌）啟用。使用此選項限制預算（例如 `MAX_THINKING_TOKENS=10000`）或完全停用思考（`MAX_THINKING_TOKENS=0`）。對於 Opus 4.6，思考深度由努力級別控制，此變數被忽略，除非設定為 `0` 以停用思考。	
`MCP_CLIENT_SECRET`	需要預先配置認證的 MCP servers 的 OAuth 用戶端機密。在新增具有 `--client-secret` 的伺服器時避免互動式提示	
`MCP_OAUTH_CALLBACK_PORT`	OAuth 重定向回呼的固定連接埠，作為在新增具有預先配置認證的 MCP server 時 `--callback-port` 的替代方案	

變數	目的	
`MCP_TIMEOUT`	MCP 伺服器啟動的逾時（以毫秒為單位）	
`MCP_TOOL_TIMEOUT`	MCP 工具執行的逾時（以毫秒為單位）	
`NO_PROXY`	將直接發出請求的網域和 IP 清單，繞過代理	
`SLASH_COMMAND_TOOL_CHAR_BUDGET`	覆蓋為 Skill 工具顯示的 skill 中繼資料的字元預算。預算在上下文視窗的 2% 動態縮放，回退為 16,000 個字元。為了向後相容性保留舊名稱	
`USE_BUILTIN_RIPGREP`	設定為 `0` 以使用系統安裝的 `rg` 而不是 Claude Code 包含的 `rg`	
`VERTEX_REGION_CLAUDE_3_5_HAIKU`	使用 Vertex AI 時覆蓋 Claude 3.5 Haiku 的區域	
`VERTEX_REGION_CLAUDE_3_7_SONNET`	使用 Vertex AI 時覆蓋 Claude 3.7 Sonnet 的區域	
`VERTEX_REGION_CLAUDE_4_0_OPUS`	使用 Vertex AI 時覆蓋 Claude 4.0 Opus 的區域	
`VERTEX_REGION_CLAUDE_4_0_SONNET`	使用 Vertex AI 時覆蓋 Claude 4.0 Sonnet 的區域	
`VERTEX_REGION_CLAUDE_4_1_OPUS`	使用 Vertex AI 時覆蓋 Claude 4.1 Opus 的區域	

Claude 可用的工具

Claude Code 可以存取一組強大的工具，可幫助它理解和修改您的程式碼庫：

工具	描述	需要權限
Agent	生成具有自己上下文視窗的 subagent 以處理工作	否
AskUserQuestion	詢問多選問題以收集要求或澄清歧義	否

工具	描述	需要權限
Bash	在您的環境中執行 shell 命令。請參閱 Bash 工具行為	是
CronCreate	在目前工作階段內排程重複或一次性提示（Claude 退出時消失）。請參閱排程工作	否
CronDelete	按 ID 取消排程工作	否
CronList	列出工作階段中的所有排程工作	否
Edit	對特定檔案進行有針對性的編輯	是
EnterPlanMode	切換到計畫模式以在編碼前設計方法	否
EnterWorktree	建立隔離的 git worktree 並切換到其中	否
ExitPlanMode	提出計畫以供批准並退出計畫模式	是
ExitWorktree	退出 worktree 工作階段並返回原始目錄	否
Glob	根據模式匹配查找檔案	否
Grep	在檔案內容中搜尋模式	否
ListMcpResourcesTool	列出連接的 MCP servers 公開的資源	否
LSP	透過語言伺服器的程式碼智慧。在檔案編輯後自動報告型別錯誤和警告。也支援導航操作：跳轉到定義、尋找參考、取得型別資訊、列出符號、尋找實現、追蹤呼叫階層。需要 code intelligence plugin 及其語言伺服器二進位檔	否
NotebookEdit	修改 Jupyter notebook 儲存格	是
Read	讀取檔案的內容	否
ReadMcpResourceTool	按 URI 讀取特定 MCP 資源	否
Skill	在主要對話中執行 skill	是

工具	描述	需要權限
TaskCreate	在工作清單中建立新工作	否
TaskGet	檢索特定工作的完整詳細資訊	否
TaskList	列出所有工作及其目前狀態	否
TaskOutput	檢索背景工作的輸出	否
TaskStop	按 ID 終止執行中的背景工作	否
TaskUpdate	更新工作狀態、依賴項、詳細資訊或刪除工作	否
TodoWrite	管理工作階段工作檢查清單。在非互動式模式和 Agent SDK 中可用；互動式工作階段改用 TaskCreate、TaskGet、TaskList 和 TaskUpdate	否
ToolSearch	當啟用 tool search 時搜尋並載入延遲工具	否
WebFetch	從指定的 URL 提取內容	是
WebSearch	執行網路搜尋	是
Write	建立或覆蓋檔案	是

權限規則可以使用 `/allowed-tools` 或在權限設定中配置。另請參閱工具特定權限規則。

Bash 工具行為

Bash 工具使用以下持續性行為執行 shell 命令：

- **工作目錄持續**：當 Claude 變更工作目錄（例如 `cd /path/to/dir`）時，後續 Bash 命令將在該目錄中執行。您可以使用 `CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR=1` 在每個命令後重設為專案目錄。
- **環境變數不持續**：在一個 Bash 命令中設定的環境變數（例如 `export MY_VAR=value`）在後續 Bash 命令中**不可用**。每個 Bash 命令在新的 shell 環境中執行。

若要在 Bash 命令中提供環境變數，您有**三個選項**：

選項 1：在啟動 Claude Code 前啟用環境（最簡單的方法）

在啟動 Claude Code 前在您的終端中啟用您的虛擬環境：

```
conda activate myenv
## 或：source /path/to/venv/bin/activate
claude
```

這適用於 shell 環境，但在 Claude 的 Bash 命令中設定的環境變數不會在命令之間持續。

選項 2：在啟動 Claude Code 前設定 CLAUDE_ENV_FILE（持續環境設定）

匯出包含您的環境設定的 shell 指令碼的路徑：

```
export CLAUDE_ENV_FILE=/path/to/env-setup.sh
claude
```

其中 `/path/to/env-setup.sh` 包含：

```
conda activate myenv
## 或：source /path/to/venv/bin/activate
## 或：export MY_VAR=value
```

Claude Code 將在每個 Bash 命令前來源此檔案，使環境在所有命令中持續。

選項 3：使用 SessionStart hook（專案特定配置）

在 `.claude/settings.json` 中配置：

```
{
  "hooks": {
    "SessionStart": [{
      "matcher": "startup",
      "hooks": [{
        "type": "command",
        "command": "echo 'conda activate myenv' >> \"$CLAUDE_ENV_FILE\""
      }]
    }]
  }
}
```

hook 寫入 `$CLAUDE_ENV_FILE`，然後在每個 Bash 命令前來源。這對於團隊共享的專案配置很理想。

請參閱 SessionStart hooks 以了解有關選項 3 的更多詳細資訊。

使用 hooks 擴展工具

您可以使用 Claude Code hooks 在任何工具執行前或執行後執行自訂命令。

例如，您可以在 Claude 修改 Python 檔案後自動執行 Python 格式化程式，或透過阻止對某些路徑的 Write 操作來防止修改生產配置檔案。

另請參閱

- 權限：權限系統、規則語法、工具特定模式和 managed 政策
- 驗證：設定使用者對 Claude Code 的存取
- 故障排除：常見配置問題的解決方案

模型配置

> 了解 Claude Code 模型配置，包括模型別名如 `opusplan`

可用模型

對於 Claude Code 中的 `model` 設定，您可以配置以下任一項：

- 一個**模型別名**
- 一個**模型名稱**
  - Anthropic API：完整的**模型名稱**
  - Bedrock：推論設定檔 ARN
  - Foundry：部署名稱
  - Vertex：版本名稱

模型別名

模型別名提供了一種便捷的方式來選擇模型設定，無需記住確切的版本號：

模型別名	行為
`default`	推薦的模型設定，取決於您的帳戶類型
`sonnet`	使用最新的 Sonnet 模型（目前為 Sonnet 4.6）進行日常編碼任務
`opus`	使用最新的 Opus 模型（目前為 Opus 4.6）進行複雜推理任務
`haiku`	使用快速高效的 Haiku 模型進行簡單任務
`sonnet[1m]`	使用 Sonnet 搭配100 萬個 token 的 context window進行長時間會話
`opusplan`	特殊模式，在 plan mode 期間使用 `opus`，然後在執行時切換到 `sonnet`

別名始終指向最新版本。若要固定到特定版本，請使用完整模型名稱（例如 `claude-opus-4-6`）或設定相應的環境變數，如 `ANTHROPIC_DEFAULT_OPUS_MODEL`。

設定您的模型

您可以透過多種方式配置模型，按優先順序列出：

1. **在會話期間** - 使用 `/model <alias|name>` 在會話中途切換模型

2. **在啟動時** - 使用 `claude --model <alias|name>` 啟動
3. **環境變數** - 設定 `ANTHROPIC_MODEL=<alias|name>`
4. **設定** - 使用 `model` 欄位在設定檔中永久配置。

使用範例：

```
## 使用 Opus 啟動
claude --model opus

## 在會話期間切換到 Sonnet
/model sonnet
```

設定檔範例：

```
{
    "permissions": {
        ...
    },
    "model": "opus"
}
```

限制模型選擇

企業管理員可以在受管理或政策設定中使用 `availableModels` 來限制使用者可以選擇的模型。

設定 `availableModels` 後，使用者無法透過 `/model`、`--model` 旗標、Config 工具或 `ANTHROPIC_MODEL` 環境變數切換到清單中沒有的模型。

```
{
  "availableModels": ["sonnet", "haiku"]
}
```

預設模型行為

模型選擇器中的「預設」選項不受 `availableModels` 影響。它始終保持可用，並代表系統的執行時預設值基於使用者的訂閱層級。

即使使用 `availableModels: []`，使用者仍然可以使用其層級的預設模型來使用 Claude Code。

控制使用者執行的模型

若要完全控制模型體驗，請將 `availableModels` 與 `model` 設定一起使用：

- **availableModels**：限制使用者可以切換到的內容
- **model**：設定明確的模型覆蓋，優先於預設值

此範例確保所有使用者執行 Sonnet 4.6，並且只能在 Sonnet 和 Haiku 之間選擇：

```
{
  "model": "sonnet",
  "availableModels": ["sonnet", "haiku"]
}
```

合併行為

當 `availableModels` 在多個層級設定時（例如使用者設定和專案設定），陣列會被合併並去重。若要強制執行嚴格的允許清單，請在受管理或政策設定中設定 `availableModels`，這些設定具有最高優先順序。

特殊模型行為

`default` 模型設定

`default` 的行為取決於您的帳戶類型：

- **Max 和 Team Premium**：預設為 Opus 4.6
- **Pro 和 Team Standard**：預設為 Sonnet 4.6
- **Enterprise**：Opus 4.6 可用但不是預設值

如果您達到 Opus 的使用閾值，Claude Code 可能會自動回退到 Sonnet。

`opusplan` 模型設定

`opusplan` 模型別名提供了一種自動化的混合方法：

- **在 plan mode 中** - 使用 `opus` 進行複雜推理和架構決策
- **在執行模式中** - 自動切換到 `sonnet` 進行程式碼生成和實現

這為您提供了兩全其美的方案：Opus 優越的推理能力用於規劃，Sonnet 的效率用於執行。

調整努力等級

努力等級控制自適應推理，根據任務複雜性動態分配思考。較低的努力對於直接的任務更快且更便宜，而較高的努力為複雜問題提供更深入的推理。

有三個等級可用：**low**、**medium** 和 **high**。Opus 4.6 對於 Max 和 Team 訂閱者預設為中等努力。

設定努力：

- **在 `/model` 中**：選擇模型時使用左/右箭頭鍵調整努力滑塊
- **環境變數**：設定 `CLAUDE_CODE_EFFORT_LEVEL=low|medium|high`
- **設定**：在設定檔中設定 `effortLevel`

Opus 4.6 和 Sonnet 4.6 支援努力。當選擇支援的模型時，努力滑塊會出現在 `/model` 中。目前的努力等級也會顯示在標誌和微調器旁邊（例如「with low effort」），因此您可以確認哪個設定處於活動狀態，而無需開啟 `/model`。

若要在 Opus 4.6 和 Sonnet 4.6 上禁用自適應推理並恢復到先前的固定思考預算，請設定 `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`。禁用時，這些模型使用由 `MAX_THINKING_TOKENS` 控制的固定預算。請參閱環境變數。

擴展 context

Opus 4.6 和 Sonnet 4.6 支援100 萬個 token 的 context window，用於具有大型程式碼庫的長時間會話。

> **Note:**
> 1M context window 目前處於測試版。功能、定價和可用性可能會變更。

擴展 context 適用於：

- **API 和按使用量付費的使用者**：完全存取 1M context
- **Pro、Max、Teams 和 Enterprise 訂閱者**：可透過額外使用量啟用

若要完全禁用 1M context，請設定 `CLAUDE_CODE_DISABLE_1M_CONTEXT=1`。這會從模型選擇器中移除 1M 模型變體。請參閱環境變數。

選擇 1M 模型不會立即改變計費。您的會話使用標準費率，直到超過 200K tokens 的 context。超過 200K tokens 後，請求按長 context 定價收費，並具有專用的速率限制。對於訂閱者，超過 200K 的 tokens 按額外使用量計費，而不是透過訂閱。

如果您的帳戶支援 1M context，該選項會出現在最新版本 Claude Code 的模型選擇器（`/model`）中。如果您看不到它，請嘗試重新啟動您的會話。

您也可以將 `[1m]` 後綴與模型別名或完整模型名稱一起使用：

```
## 使用 sonnet[1m] 別名
/model sonnet[1m]

## 或將 [1m] 附加到完整模型名稱
/model claude-sonnet-4-6[1m]
```

檢查您目前的模型

您可以透過多種方式查看您目前使用的模型：

1. 在狀態行中（如果已配置）
2. 在 `/status` 中，它也會顯示您的帳戶資訊。

環境變數

您可以使用以下環境變數，這些變數必須是完整的**模型名稱**（或您的 API 提供者的等效項），以控制別名對應到的模型名稱。

環境變數	描述
`ANTHROPIC_DEFAULT_OPUS_MODEL`	用於 `opus` 的模型，或在 Plan Mode 活動時用於 `opusplan` 的模型。
`ANTHROPIC_DEFAULT_SONNET_MODEL`	用於 `sonnet` 的模型，或在 Plan Mode 不活動時用於 `opusplan` 的模型。
`ANTHROPIC_DEFAULT_HAIKU_MODEL`	用於 `haiku` 的模型，或背景功能
`CLAUDE_CODE_SUBAGENT_MODEL`	用於 subagents 的模型

注意：`ANTHROPIC_SMALL_FAST_MODEL` 已棄用，改為使用 `ANTHROPIC_DEFAULT_HAIKU_MODEL`。

為第三方部署固定模型

透過 Bedrock、Vertex AI 或 Foundry 部署 Claude Code 時，在向使用者推出前固定模型版本。

不固定模型時，Claude Code 使用模型別名（`sonnet`、`opus`、`haiku`），這些別名解析為最新版本。當 Anthropic 發佈新模型時，其帳戶未啟用新版本的使用者將無聲地中斷。

> **Warning:**
>
> 在初始設定中將所有三個模型環境變數設定為特定版本 ID。跳過此步驟意味著 Claude Code 更新可能會在您沒有任何操作的情況下破壞您的使用者。

對您的提供者使用以下環境變數搭配版本特定的模型 ID：

提供者	範例
Bedrock	`export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-6-v1'`
Vertex AI	`export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'`
Foundry	`export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'`

對 `ANTHROPIC_DEFAULT_SONNET_MODEL` 和 `ANTHROPIC_DEFAULT_HAIKU_MODEL` 應用相同的模式。有關所有提供者的目前和舊版模型 ID，請參閱模型概述。若要將使用者升級到新模型版本，請更新這些環境變數並重新部署。

> **Note:**
>
> 使用第三方提供者時，`settings.availableModels` 允許清單仍然適用。篩選與模型別名（`opus`、`sonnet`、`haiku`）相符，而不是提供者特定的模型 ID。

按版本覆蓋模型 ID

上述家族級環境變數為每個家族別名配置一個模型 ID。如果您需要將同一家族中的多個版本對應到不同的提供者 ID，請改用 `modelOverrides` 設定。

`modelOverrides` 將個別 Anthropic 模型 ID 對應到 Claude Code 發送到您提供者 API 的提供者特定字串。當使用者在 `/model` 選擇器中選擇對應的模型時，Claude Code 使用您配置的值而不是內建預設值。

這讓企業管理員可以將每個模型版本路由到特定的 Bedrock 推論設定檔 ARN、Vertex AI 版本名稱或 Foundry 部署名稱，以進行治理、成本分配或區域路由。

在您的設定檔中設定 `modelOverrides`：

```
{
  "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"
  }
}
```

鍵必須是 Anthropic 模型 ID，如模型概述中所列。對於日期模型 ID，請包含日期後綴，完全如其所示。未知的鍵會被忽略。

覆蓋會取代支援 `/model` 選擇器中每個條目的內建模型 ID。在 Bedrock 上，覆蓋優先於 Claude Code 在啟動時自動發現的任何推論設定檔。您直接透過 `ANTHROPIC_MODEL`、`--model` 或 `ANTHROPIC_DEFAULT_*_MODEL` 環境變數提供的值會按原樣傳遞給提供者，不會由 `modelOverrides` 轉換。

`modelOverrides` 與 `availableModels` 一起運作。允許清單針對 Anthropic 模型 ID 進行評估，而不是覆蓋值，因此 `availableModels` 中的條目（如 `"opus"`）即使在 Opus 版本對應到 ARN 時也會繼續相符。

Prompt caching 配置

Claude Code 自動使用 prompt caching 來優化效能並降低成本。您可以全域或針對特定模型層級禁用 prompt caching：

環境變數	描述
`DISABLE_PROMPT_CACHING`	設定為 `1` 以禁用所有模型的 prompt caching（優先於每個模型的設定）
`DISABLE_PROMPT_CACHING_HAIKU`	設定為 `1` 以僅禁用 Haiku 模型的 prompt caching
`DISABLE_PROMPT_CACHING_SONNET`	設定為 `1` 以僅禁用 Sonnet 模型的 prompt caching
`DISABLE_PROMPT_CACHING_OPUS`	設定為 `1` 以僅禁用 Opus 模型的 prompt caching

這些環境變數為您提供對 prompt caching 行為的細粒度控制。全域 `DISABLE_PROMPT_CACHING` 設定優先於模型特定的設定，允許您在需要時快速禁用所有快取。每個模型的設定對於選擇性控制很有用，例如在調試特定模型或使用可能具有不同快取實現的雲端提供者時。

輸出樣式

> 將 Claude Code 適配用於軟體工程以外的用途

輸出樣式允許您將 Claude Code 用作任何類型的代理，同時保留其核心功能，例如執行本地指令碼、讀取/寫入檔案和追蹤待辦事項。

內建輸出樣式

Claude Code 的**預設**輸出樣式是現有的系統提示，旨在幫助您有效地完成軟體工程任務。

還有兩種額外的內建輸出樣式，專注於教您了解程式碼庫和 Claude 的運作方式：

- **Explanatory**：在幫助您完成軟體工程任務的同時提供教育性的「Insights」。幫助您理解實現選擇和程式碼庫模式。
- **Learning**：協作式的邊做邊學模式，Claude 不僅會在編碼時分享「Insights」，還會要求您自己貢獻小的、策略性的程式碼片段。Claude Code 將在您的程式碼中添加 `TODO(human)` 標記供您實現。

輸出樣式的工作原理

輸出樣式直接修改 Claude Code 的系統提示。

- 所有輸出樣式都排除了高效輸出的指令（例如簡潔回應）。
- 自訂輸出樣式排除了編碼指令（例如使用測試驗證程式碼），除非 `keep-coding-instructions` 為真。
- 所有輸出樣式都在系統提示的末尾添加了自己的自訂指令。
- 所有輸出樣式都會在對話期間觸發提醒，讓 Claude 遵守輸出樣式指令。

變更您的輸出樣式

執行 `/config` 並選擇**輸出樣式**以從選單中選擇樣式。您的選擇會儲存到本地專案層級的 `.claude/settings.local.json`。

若要在不使用選單的情況下設定樣式，請直接編輯設定檔中的 `outputStyle` 欄位：

```
{
  "outputStyle": "Explanatory"
}
```

由於輸出樣式是在工作階段開始時在系統提示中設定的，變更將在您下次啟動新工作階段時生效。這使系統提示在整個對話中保持穩定，以便 prompt caching 可以降低延遲和成本。

建立自訂輸出樣式

自訂輸出樣式是具有 frontmatter 和將添加到系統提示的文字的 Markdown 檔案：

```
---
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...]
```

您可以在使用者層級 (`~/.claude/output-styles`) 或專案層級 (`.claude/output-styles`) 儲存這些檔案。

Frontmatter

輸出樣式檔案支援 frontmatter 以指定中繼資料：

Frontmatter	用途	預設
`name`	輸出樣式的名稱，如果不是檔案名稱	繼承自檔案名稱
`description`	輸出樣式的描述，在 `/config` 選擇器中顯示	無

Frontmatter	用途	預設
`keep-coding-instructions`	是否保留 Claude Code 系統提示中與編碼相關的部分。	false

與相關功能的比較

輸出樣式 vs. CLAUDE.md vs. –append-system-prompt

輸出樣式完全「關閉」Claude Code 預設系統提示中特定於軟體工程的部分。CLAUDE.md 和 `--append-system-prompt` 都不會編輯 Claude Code 的預設系統提示。CLAUDE.md 將內容添加為 Claude Code 預設系統提示_之後_的使用者訊息。`--append-system-prompt` 將內容附加到系統提示。

輸出樣式 vs. Agents

輸出樣式直接影響主代理迴圈，僅影響系統提示。Agents 被呼叫以處理特定任務，可以包括其他設定，例如要使用的模型、可用的工具以及有關何時使用代理的一些上下文。

輸出樣式 vs. Skills

輸出樣式修改 Claude 的回應方式（格式、語氣、結構），一旦選擇就始終處於活動狀態。Skills 是特定於任務的提示，您可以使用 `/skill-name` 呼叫或 Claude 在相關時自動載入。使用輸出樣式來實現一致的格式設定偏好；使用 skills 來實現可重複使用的工作流程和任務。

使用快速模式加快回應速度

> 在 Claude Code 中切換快速模式，以獲得更快的 Opus 4.6 回應。

Note:

快速模式處於研究預覽階段。該功能、定價和可用性可能會根據反饋而改變。

快速模式是 Claude Opus 4.6 的高速配置，使模型速度提升 2.5 倍，但每個 token 的成本更高。當您需要速度進行互動式工作（如快速迭代或實時調試）時，使用 `/fast` 切換開啟，當成本比延遲更重要時，切換關閉。

快速模式不是不同的模型。它使用相同的 Opus 4.6，但採用不同的 API 配置，優先考慮速度而非成本效率。您獲得相同的品質和功能，只是回應速度更快。

Note:

快速模式需要 Claude Code v2.1.36 或更新版本。使用 `claude --version` 檢查您的版本。

需要了解的事項：

- 使用 `/fast` 在 Claude Code CLI 中切換快速模式。也可在 Claude Code VS Code 擴充功能中透過 `/fast` 使用。
- Opus 4.6 快速模式定價起價為 $30/150 MTok。快速模式在 2 月 16 日太平洋時間晚上 11:59 之前享受所有方案 50% 折扣。
- 適用於訂閱方案（Pro/Max/Team/Enterprise）上的所有 Claude Code 使用者和 Claude Console。
- 對於訂閱方案（Pro/Max/Team/Enterprise）上的 Claude Code 使用者，快速模式僅透過額外使用提供，不包含在訂閱速率限制中。

本頁涵蓋如何切換快速模式、其成本權衡、何時使用、要求、每個工作階段選擇加入和速率限制行為。

切換快速模式

透過以下任一方式切換快速模式：

- 輸入 `/fast` 並按 Tab 鍵切換開啟或關閉

- 在您的使用者設定檔案中設定 `"fastMode": true`

預設情況下，快速模式在工作階段之間保持。管理員可以配置快速模式在每個工作階段重設。詳見要求每個工作階段選擇加入。

為了獲得最佳成本效率，請在工作階段開始時啟用快速模式，而不是在對話中途切換。詳見了解成本權衡。

當您啟用快速模式時：

- 如果您使用不同的模型，Claude Code 會自動切換到 Opus 4.6
- 您會看到確認訊息：“Fast mode ON”
- 快速模式啟用時，提示旁會出現一個小的 `↯` 圖示
- 隨時再次執行 `/fast` 以檢查快速模式是否開啟或關閉

當您再次使用 `/fast` 關閉快速模式時，您仍保持在 Opus 4.6 上。模型不會還原到您之前的模型。要切換到不同的模型，請使用 `/model`。

了解成本權衡

快速模式的每個 token 定價高於標準 Opus 4.6：

模式	輸入 (MTok)	輸出 (MTok)
Opus 4.6 上的快速模式 (<200K)	$30	$150
Opus 4.6 上的快速模式 (>200K)	$60	$225

快速模式與 1M token 擴展上下文視窗相容。

當您在對話中途切換到快速模式時，您需要為整個對話上下文支付完整的快速模式未快取輸入 token 價格。這比從一開始就啟用快速模式的成本更高。

決定何時使用快速模式

快速模式最適合用於回應延遲比成本更重要的互動式工作：

- 快速迭代程式碼變更
- 實時調試工作階段
- 時間敏感的工作，有緊迫的截止日期

標準模式更適合：

- 速度不那麼重要的長期自主任務
- 批次處理或 CI/CD 管道

- 成本敏感的工作負載

快速模式與努力等級

快速模式和努力等級都會影響回應速度，但方式不同：

設定	效果
快速模式	相同的模型品質、更低的延遲、更高的成本
較低的努力等級	較少的思考時間、更快的回應、在複雜任務上可能品質較低

您可以結合兩者：使用快速模式搭配較低的努力等級，在直接任務上獲得最大速度。

要求

快速模式需要以下所有條件：

- **第三方雲端提供商上不可用**：快速模式在 Amazon Bedrock、Google Vertex AI 或 Microsoft Azure Foundry 上不可用。快速模式可透過 Anthropic Console API 和使用額外使用的 Claude 訂閱方案取得。
- **啟用額外使用**：您的帳戶必須啟用額外使用，這允許超出您方案包含使用量的計費。對於個人帳戶，請在您的 Console 計費設定中啟用此功能。對於 Teams 和 Enterprise，管理員必須為組織啟用額外使用。

> **Note:**
>
> 快速模式使用直接計費到額外使用，即使您的方案上還有剩餘使用量。這意味著快速模式 token 不計入您方案的包含使用量，並從第一個 token 開始按快速模式費率計費。

- **Teams 和 Enterprise 的管理員啟用**：快速模式預設對 Teams 和 Enterprise 組織禁用。管理員必須明確啟用快速模式，使用者才能存取它。

> **Note:**
>
> 如果您的管理員尚未為您的組織啟用快速模式，`/fast` 命令將顯示「Fast mode has been disabled by your organization.」

為您的組織啟用快速模式

管理員可以在以下位置啟用快速模式：

- **Console**（API 客戶）：Claude Code 偏好設定

- **Claude AI**（Teams 和 Enterprise）：管理員設定 > Claude Code

另一個完全禁用快速模式的選項是設定 `CLAUDE_CODE_DISABLE_FAST_MODE=1`。詳見環境變數。

要求每個工作階段選擇加入

預設情況下，快速模式在工作階段之間保持：如果使用者啟用快速模式，它會在未來工作階段中保持開啟。Teams 或 Enterprise 方案上的管理員可以透過在受管設定或伺服器受管設定中將 `fastModePerSessionOptIn` 設定為 `true` 來防止這種情況。這會導致每個工作階段以快速模式關閉開始，要求使用者使用 `/fast` 明確啟用它。

```
{
  "fastModePerSessionOptIn": true
}
```

這對於控制使用者執行多個並行工作階段的組織中的成本很有用。使用者在需要速度時仍可以使用 `/fast` 啟用快速模式，但它會在每個新工作階段開始時重設。使用者的快速模式偏好設定仍會保存，因此移除此設定會還原預設的持久行為。

處理速率限制

快速模式與標準 Opus 4.6 有不同的速率限制。當您達到快速模式速率限制或用完額外使用額度時：

1. 快速模式自動回退到標準 Opus 4.6
2. `↯` 圖示變灰以指示冷卻期
3. 您以標準速度和定價繼續工作
4. 冷卻期過期時，快速模式自動重新啟用

要手動禁用快速模式而不是等待冷卻期，請再次執行 `/fast`。

研究預覽

快速模式是研究預覽功能。這意味著：

- 該功能可能會根據反饋而改變
- 可用性和定價可能會改變
- 底層 API 配置可能會演變

透過您通常的 Anthropic 支援管道報告問題或反饋。

另請參閱

- 模型配置：切換模型和調整努力等級
- 有效管理成本：追蹤 token 使用量並降低成本
- 狀態行配置：顯示模型和上下文資訊

有效管理成本

> 追蹤 token 使用情況、設定團隊支出限制，並透過上下文管理、模型選擇、延伸思考設定和預處理 hooks 來降低 Claude Code 成本。

Claude Code 在每次互動時都會消耗 token。成本因程式碼庫大小、查詢複雜性和對話長度而異。平均成本為每位開發人員每天 $6，90% 的使用者每日成本保持在 $12 以下。

對於團隊使用，Claude Code 按 API token 消耗量計費。平均而言，Claude Code 使用 Sonnet 4.6 時的成本約為每位開發人員每月 $100-200，但根據使用者執行的執行個體數量以及他們是否在自動化中使用它，成本差異很大。

本頁涵蓋如何追蹤您的成本、管理團隊成本和減少 token 使用。

追蹤您的成本

使用 `/cost` 命令

> **Note:**
>
> `/cost` 命令顯示 API token 使用情況，適用於 API 使用者。Claude Max 和 Pro 訂閱者的使用情況已包含在其訂閱中，因此 `/cost` 資料與計費無關。訂閱者可以使用 `/stats` 來檢視使用模式。

`/cost` 命令為您目前的工作階段提供詳細的 token 使用統計資訊：

```
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
```

管理團隊成本

使用 Claude API 時，您可以在 Claude Code 工作區支出上設定工作區支出限制。管理員可以在主控台中檢視成本和使用情況報告。

> **Note:**

當您首次使用 Claude Console 帳戶驗證 Claude Code 時，系統會自動為您建立一個名為「Claude Code」的工作區。此工作區為您的組織中所有 Claude Code 使用情況提供集中式成本追蹤和管理。您無法為此工作區建立 API 金鑰；它專門用於 Claude Code 驗證和使用。

在 Bedrock、Vertex 和 Foundry 上，Claude Code 不會從您的雲端傳送指標。若要取得成本指標，多家大型企業報告使用LiteLLM，這是一個開源工具，可幫助公司按金鑰追蹤支出。此專案與 Anthropic 無關，且尚未進行安全審計。

速率限制建議

為團隊設定 Claude Code 時，請根據您的組織規模考慮這些每位使用者的 Token Per Minute (TPM) 和 Request Per Minute (RPM) 建議：

團隊規模	每位使用者 TPM	每位使用者 RPM
1-5 位使用者	200k-300k	5-7
5-20 位使用者	100k-150k	2.5-3.5
20-50 位使用者	50k-75k	1.25-1.75
50-100 位使用者	25k-35k	0.62-0.87
100-500 位使用者	15k-20k	0.37-0.47
500+ 位使用者	10k-15k	0.25-0.35

例如，如果您有 200 位使用者，您可能會為每位使用者請求 20k TPM，或總共 400 萬 TPM (200*20,000 = 400 萬)。

隨著團隊規模增長，每位使用者的 TPM 會減少，因為在較大的組織中，傾向於較少的使用者同時使用 Claude Code。這些速率限制適用於組織層級，而不是每個個別使用者，這意味著當其他人未主動使用該服務時，個別使用者可以暫時消耗超過其計算份額的資源。

Note:

如果您預期會出現異常高的並行使用情況（例如與大型群組進行的即時培訓課程），您可能需要更高的每位使用者 TPM 配置。

Agent 團隊 token 成本

Agent 團隊會產生多個 Claude Code 執行個體，每個都有自己的上下文視窗。Token 使用量會隨著活躍隊友數量和每個隊友執行時間的長短而擴展。

為了保持 agent 團隊成本可控：

- 為隊友使用 Sonnet。它為協調任務平衡了功能和成本。
- 保持團隊規模小。每位隊友執行自己的上下文視窗，因此 token 使用量大致與團隊規模成正比。
- 保持產生提示的焦點。隊友會自動載入 CLAUDE.md、MCP 伺服器和技能，但產生提示中的所有內容都會從一開始就新增到其上下文中。
- 工作完成時清理團隊。活躍的隊友即使閒置也會繼續消耗 token。
- Agent 團隊預設為停用。在您的settings.json或環境中設定 `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` 以啟用它們。請參閱啟用 agent 團隊。

減少 token 使用

Token 成本隨上下文大小而擴展：Claude 處理的上下文越多，您使用的 token 就越多。Claude Code 透過 prompt caching（減少重複內容（如系統提示）的成本）和 auto-compact（在接近上下文限制時總結對話歷史記錄）自動優化成本。

以下策略可幫助您保持上下文較小並降低每條訊息的成本。

主動管理上下文

使用 `/cost` 檢查您目前的 token 使用情況，或設定您的狀態行以持續顯示它。

- **在任務之間清除**：切換到不相關的工作時，使用 `/clear` 重新開始。過時的上下文會在後續的每條訊息上浪費 token。在清除之前使用 `/rename` 以便稍後輕鬆找到工作階段，然後使用 `/resume` 返回到它。
- **新增自訂壓縮指示**：`/compact Focus on code samples and API usage` 告訴 Claude 在總結期間要保留什麼。

您也可以在 CLAUDE.md 中自訂壓縮行為：

```
## Compact instructions

When you are using compact, please focus on test output and code changes
```

選擇正確的模型

Sonnet 能很好地處理大多數編碼任務，成本低於 Opus。為複雜的架構決策或多步驟推理保留 Opus。使用 `/model` 在工作階段中途切換模型，或在 `/config` 中設定預設值。對於簡單的 subagent 任務，在您的subagent 設定中指定 `model: haiku`。

減少 MCP 伺服器開銷

每個 MCP 伺服器都會將工具定義新增到您的上下文中，即使在閒置時也是如此。執行 `/context` 以查看消耗空間的內容。

- **在可用時偏好 CLI 工具**：`gh`、`aws`、`gcloud` 和 `sentry-cli` 等工具比 MCP 伺服器更具上下文效率，因為它們不會新增持久的工具定義。Claude 可以直接執行 CLI 命令，無需開銷。
- **停用未使用的伺服器**：執行 `/mcp` 以查看已設定的伺服器，並停用任何您未主動使用的伺服器。
- **工具搜尋是自動的**：當 MCP 工具描述超過您的上下文視窗的 10% 時，Claude Code 會自動延遲它們，並透過工具搜尋按需載入工具。由於延遲的工具只在實際使用時進入上下文，較低的閾值意味著較少的閒置工具定義消耗空間。使用 `ENABLE_TOOL_SEARCH=auto:<N>` 設定較低的閾值（例如，`auto:5` 在工具超過您的上下文視窗的 5% 時觸發）。

為型別化語言安裝程式碼智慧外掛

程式碼智慧外掛為 Claude 提供精確的符號導航，而不是基於文字的搜尋，在探索不熟悉的程式碼時減少不必要的檔案讀取。單一「前往定義」呼叫取代了可能需要的 grep 後跟讀取多個候選檔案。已安裝的語言伺服器也會在編輯後自動報告型別錯誤，因此 Claude 無需執行編譯器即可捕捉錯誤。

將處理卸載到 hooks 和技能

自訂hooks可以在 Claude 看到資料之前對其進行預處理。Claude 不是讀取 10,000 行日誌檔案來尋找錯誤，hook 可以 grep `ERROR` 並僅返回匹配的行，將上下文從數萬個 token 減少到數百個。

技能可以為 Claude 提供領域知識，因此它不必進行探索。例如，「codebase-overview」技能可以描述您的專案架構、關鍵目錄和命名慣例。當 Claude 呼叫該技能時，它會立即獲得此上下文，而不是花費 token 讀取多個檔案來理解結構。

例如，此 PreToolUse hook 篩選測試輸出以僅顯示失敗：

settings.json

將此新增到您的settings.json以在每個 Bash 命令之前執行 hook：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/.claude/hooks/filter-test-output.sh"
          }
        ]
      }
    ]
  }
}
```

filter-test-output.sh

hook 呼叫此指令碼，該指令碼檢查命令是否為測試執行器並修改它以僅顯示失敗：

```
#!/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\",\"permissionDeci
sion\":\"allow\",\"updatedInput\":{\"command\":\"$filtered_cmd\"}}}"
else
  echo "{}"
fi
```

將指示從 CLAUDE.md 移至技能

您的CLAUDE.md檔案在工作階段開始時載入到上下文中。如果它包含特定工作流程的詳細指示（例如 PR 審查或資料庫遷移），即使您在進行不相關的工作時，這些 token 也會存在。技能僅在呼叫時按需載入，因此將專門指示移至技能可以保持您的基本上下文較小。目標是透過僅包含必要內容來將 CLAUDE.md 保持在約 500 行以下。

調整延伸思考

延伸思考預設為啟用，預算為 31,999 個 token，因為它可以顯著改善複雜規劃和推理任務的效能。但是，思考 token 會作為輸出 token 計費，因此對於不需要深度推理的較簡單任務，您可以透過在 `/model` 中降低 Opus 4.6 的努力等級、在 `/config` 中停用思考或降低預算（例如，`MAX_THINKING_TOKENS=8000`）來降低成本。

將詳細操作委派給 subagents

執行測試、擷取文件或處理日誌檔案可能會消耗大量上下文。將這些委派給subagents，以便詳細輸出保留在 subagent 的上下文中，而只有摘要返回到您的主要對話。

管理 agent 團隊成本

當隊友在 plan mode 中執行時，Agent 團隊使用的 token 大約是標準工作階段的 7 倍，因為每位隊友維護自己的上下文視窗並作為單獨的 Claude 執行個體執行。保持團隊任務小且自成一體，以限制每位隊友的 token 使用。有關詳細資訊，請參閱agent 團隊。

撰寫具體提示

模糊的請求（例如「改進此程式碼庫」）會觸發廣泛掃描。具體的請求（例如「在 auth.ts 中的登入函式中新增輸入驗證」）讓 Claude 能夠以最少的檔案讀取高效地工作。

有效處理複雜任務

對於較長或更複雜的工作，這些習慣有助於避免因走錯方向而浪費的 token：

- **對複雜任務使用 plan mode**：按 Shift+Tab 進入plan mode，然後再進行實施。Claude 探索程式碼庫並提出一個方法供您批准，防止當初始方向錯誤時進行昂貴的返工。
- **及早糾正方向**：如果 Claude 開始朝著錯誤的方向前進，按 Escape 立即停止。使用 `/rewind` 或雙擊 Escape 將對話和程式碼恢復到先前的 checkpoint。
- **提供驗證目標**：在您的提示中包含測試案例、貼上螢幕截圖或定義預期輸出。當 Claude 可以驗證自己的工作時，它會在您需要請求修復之前捕捉問題。
- **增量測試**：寫一個檔案、測試它，然後繼續。這會在問題便宜時及早捕捉問題。

背景 token 使用

Claude Code 即使在閒置時也會為某些背景功能使用 token：

- **對話總結**：為 `claude --resume` 功能總結先前對話的背景工作
- **命令處理**：某些命令（例如 `/cost`）可能會產生檢查狀態的請求

這些背景程序即使沒有主動互動也會消耗少量 token（通常每個工作階段不到 $0.04）。

瞭解 Claude Code 行為的變化

Claude Code 定期接收可能改變功能工作方式的更新，包括成本報告。執行 `claude --version` 以檢查您目前的版本。如有具體計費問題，請透過您的主控台帳戶聯絡 Anthropic 支援。對於團隊部署，請從小型試點群組開始，以在更廣泛的推出前建立使用模式。

Part 5: Extensibility

Hooks 參考

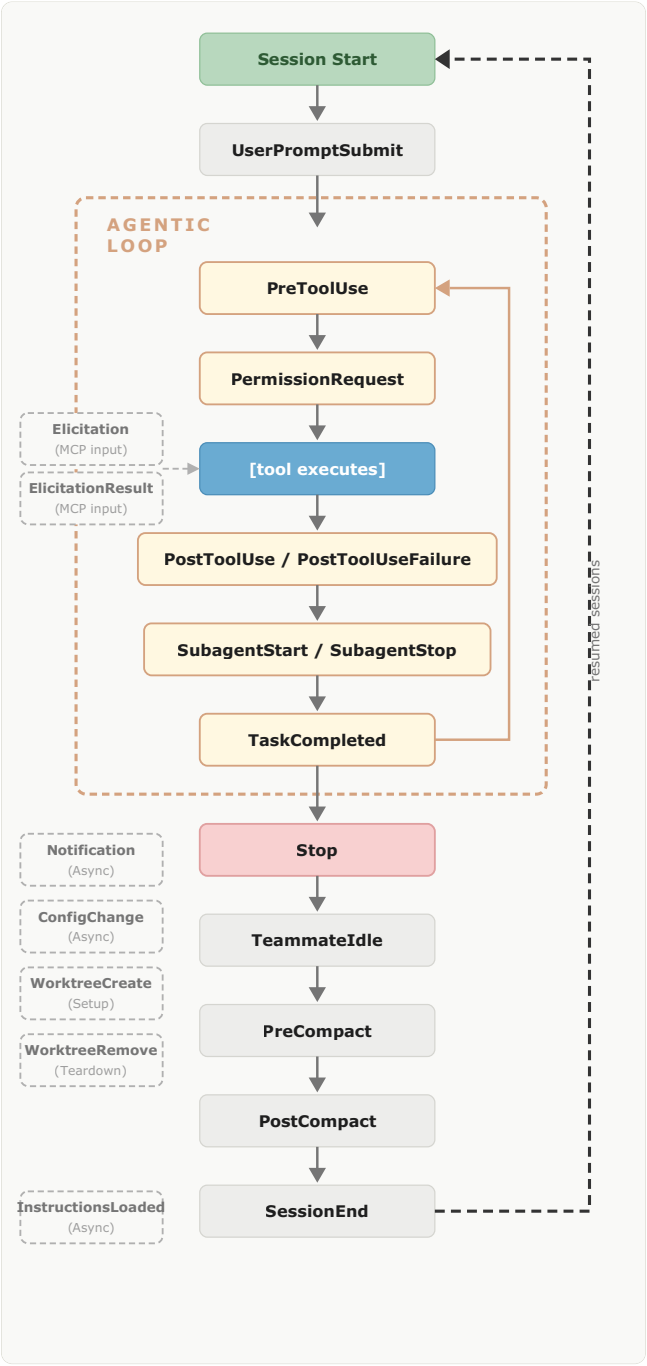
> Claude Code hook 事件、配置架構、JSON 輸入/輸出格式、退出代碼、非同步 hooks、HTTP hooks、提示 hooks 和 MCP 工具 hooks 的參考。

> **Tip:**
>
> 如需快速入門指南和範例，請參閱 使用 hooks 自動化工作流程。

Hooks 是使用者定義的 shell 命令、HTTP 端點或 LLM 提示，在 Claude Code 生命週期中的特定時間點自動執行。使用此參考來查詢事件架構、配置選項、JSON 輸入/輸出格式，以及非同步 hooks、HTTP hooks 和 MCP 工具 hooks 等進階功能。如果您是第一次設定 hooks，請改為從 指南 開始。

Hook 生命週期

Hooks 在 Claude Code 工作階段期間的特定時間點觸發。當事件觸發且匹配器匹配時，Claude Code 會將有關該事件的 JSON 上下文傳遞給您的 hook 處理程式。對於命令 hooks，輸入會到達 stdin。對於 HTTP hooks，它會作為 POST 請求正文到達。您的處理程式可以檢查輸入、採取行動，並可選擇性地返回決定。某些事件每個工作階段觸發一次，而其他事件在代理迴圈內重複觸發：


Session Start
UserPromptSubmit
AGENTIC LOOP
PreToolUse
PermissionRequest
Elicitation
(MCP input)
[tool executes]
ElicitationResult
(MCP input)
PostToolUse / PostToolUseFailure
SubagentStart / SubagentStop
TaskCompleted
Stop
Notification
(Async)
ConfigChange
(Async)
TeammateIdle
WorktreeCreate
(Setup)
PreCompact
WorktreeRemove
(Teardown)
PostCompact
InstructionsLoaded
(Async)
SessionEnd
resumed sessions

Hook 生命週期圖表，顯示從 SessionStart 到代理迴圈再到 SessionEnd 的 hooks 序列，WorktreeCreate、WorktreeRemove 和 InstructionsLoaded 作為獨立非同步事件

下表總結了每個事件何時觸發。Hook 事件 部分記錄了每個事件的完整輸入架構和決定控制選項。

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 `.claude/rules/*.md` 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 `--worktree` or `isolation: "worktree"`. 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

Event	When it fires
`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

Hook 如何解析

為了了解這些部分如何組合在一起，請考慮此 `PreToolUse` hook，它會阻止破壞性 shell 命令。該 hook 在每次 Bash 工具呼叫之前執行 `block-rm.sh`：

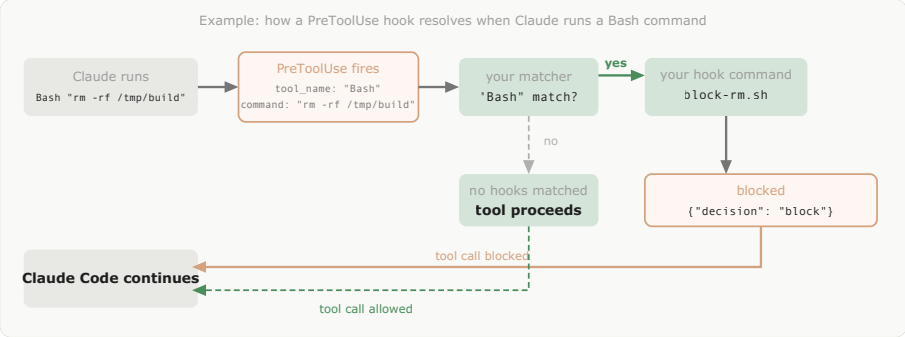
```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": ".claude/hooks/block-rm.sh"
          }
        ]
      }
    ]
  }
}
```

該指令碼從 stdin 讀取 JSON 輸入，提取命令，如果包含 `rm -rf`，則返回 `permissionDecision` 為 `"deny"`：

```
#!/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
```

現在假設 Claude Code 決定執行 `Bash "rm -rf /tmp/build"`。以下是發生的情況：



Hook 解析流程：PreToolUse 事件觸發，匹配器檢查 Bash 匹配，hook 處理程式執行，結果返回到 Claude Code

Step 1: 事件觸發

`PreToolUse` 事件觸發。Claude Code 將工具輸入作為 JSON 在 stdin 上發送到 hook：

```
{ "tool_name": "Bash", "tool_input": { "command": "rm -rf /tmp/build" }, ... }
```

Step 2: 匹配器檢查

匹配器 `"Bash"` 與工具名稱匹配，因此 `block-rm.sh` 執行。如果您省略匹配器或使用 `"*"`，hook 會在事件的每次出現時執行。只有當定義了匹配器且不匹配時，hooks 才會跳過。

Step 3: Hook 處理程式執行

該指令碼從輸入中提取 `"rm -rf /tmp/build"` 並找到 `rm -rf`，因此它將決定列印到 stdout：

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Destructive command blocked by hook"
  }
}
```

如果命令是安全的（例如 `npm test`），指令碼會改為執行 `exit 0`，這告訴 Claude Code 允許工具呼叫而無需進一步操作。

Step 4: Claude Code 根據結果採取行動

Claude Code 讀取 JSON 決定，阻止工具呼叫，並向 Claude 顯示原因。

下面的 配置 部分記錄了完整架構，每個 hook 事件 部分記錄了您的命令接收的輸入以及它可以返回的輸出。

配置

Hooks 在 JSON 設定檔中定義。配置有三個嵌套層級：

1. 選擇要回應的 hook 事件，例如 `PreToolUse` 或 `Stop`
2. 新增 匹配器群組 以篩選何時觸發，例如’僅適用於 Bash 工具’
3. 定義一個或多個 hook 處理程式 以在匹配時執行

有關完整的逐步說明和註解範例，請參閱上面的 Hook 如何解析。

> **Note:**
>
> 此頁面為每個層級使用特定術語：**hook 事件** 表示生命週期點，**匹配器群組** 表示篩選器，**hook 處理程式** 表示執行的 shell 命令、HTTP 端點、提示或代理。’ Hook’ 本身指的是一般功能。

Hook 位置

您定義 hook 的位置決定了其範圍：

位置	範圍	可共享
`~/.claude/settings.json`	您的所有專案	否，本機限定
`.claude/settings.json`	單一專案	是，可提交到儲存庫
`.claude/settings.local.json`	單一專案	否，gitignored
受管理的原則設定	組織範圍	是，由管理員控制
Plugin `hooks/hooks.json`	啟用外掛時	是，與外掛一起打包
Skill 或 agent frontmatter	元件活躍時	是，在元件檔案中定義

有關設定檔解析的詳細資訊，請參閱 settings。企業管理員可以使用 `allowManagedHooksOnly` 來阻止使用者、專案和外掛 hooks。請參閱 Hook 配置。

匹配器模式

`matcher` 欄位是一個正規表達式字串，用於篩選 hooks 何時觸發。使用 `"*"`、`""` 或完全省略 `matcher` 以匹配所有出現。每個事件類型在不同的欄位上匹配：

事件	匹配器篩選的內容	範例匹配器值
`PreToolUse`、`PostToolUse`、`PostToolUseFailure`、`PermissionRequest`	工具名稱	`Bash`、`Edit|Write`、`mcp__.*`
`SessionStart`	工作階段如何開始	`startup`、`resume`、`clear`、`compact`
`SessionEnd`	工作階段為何結束	`clear`、`logout`、`prompt_input_exit`、`bypass_permissions_disabled`、`other`

事件	匹配器篩選的內容	範例匹配器值
`Notification`	通知類型	`permission_prompt`、`idle_prompt`、`auth_success`、`elicitation_dialog`
`SubagentStart`	代理類型	`Bash`、`Explore`、`Plan` 或自訂代理名稱
`PreCompact`	觸發壓縮的原因	`manual`、`auto`
`SubagentStop`	代理類型	與 `SubagentStart` 相同的值
`ConfigChange`	配置來源	`user_settings`、`project_settings`、`local_settings`、`policy_settings`、`skills`
`UserPromptSubmit`、`Stop`、`TeammateIdle`、`TaskCompleted`、`WorktreeCreate`、`WorktreeRemove`、`InstructionsLoaded`	不支援匹配器	總是在每次出現時觸發

匹配器是正規表達式，因此 `Edit|Write` 匹配任一工具，`Notebook.*` 匹配任何以 Notebook 開頭的工具。匹配器針對 Claude Code 在 stdin 上發送給您的 hook 的 JSON 輸入 中的欄位執行。對於工具事件，該欄位是 `tool_name`。每個 hook 事件 部分列出了該事件的完整匹配器值集和輸入架構。

此範例僅在 Claude 寫入或編輯檔案時執行 linting 指令碼：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/lint-check.sh"
          }
        ]
      }
    ]
  }
}
```

`UserPromptSubmit`、`Stop`、`TeammateIdle`、`TaskCompleted`、`WorktreeCreate`、`WorktreeRemove` 和 `InstructionsLoaded` 不支援匹配器，總是在每次出現時觸發。如果您將 `matcher` 欄位新增到這些事件，它會被無聲地忽略。

匹配 MCP 工具

MCP 伺服器工具在工具事件（`PreToolUse`、`PostToolUse`、`PostToolUseFailure`、`PermissionRequest`）中顯示為常規工具，因此您可以像匹配任何其他工具名稱一樣匹配它們。

MCP 工具遵循命名模式 `mcp__<server>__<tool>`，例如：

- `mcp__memory__create_entities`：Memory 伺服器的建立實體工具
- `mcp__filesystem__read_file`：Filesystem 伺服器的讀取檔案工具
- `mcp__github__search_repositories`：GitHub 伺服器的搜尋工具

使用正規表達式模式來針對特定 MCP 工具或工具群組：

- `mcp__memory__.*` 匹配來自 `memory` 伺服器的所有工具
- `mcp__.*__write.*` 匹配來自任何伺服器的任何包含「write」的工具

此範例記錄所有 memory 伺服器操作並驗證來自任何 MCP 伺服器的寫入操作：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__memory__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Memory operation initiated' >> ~/mcp-operations.log"
          }
        ]
      },
      {
        "matcher": "mcp__.*__write.*",
        "hooks": [
          {
            "type": "command",
            "command": "/home/user/scripts/validate-mcp-write.py"
          }
        ]
      }
    ]
  }
}
```

Hook 處理程式欄位

內部 `hooks` 陣列中的每個物件都是一個 hook 處理程式：當匹配器匹配時執行的 shell 命令、HTTP 端點、LLM 提示或代理。有四種類型：

- **命令 hooks**（`type: "command"`）：執行 shell 命令。您的指令碼在 stdin 上接收事件的 JSON 輸入，並通過退出代碼和 stdout 傳回結果。
- **HTTP hooks**（`type: "http"`）：將事件的 JSON 輸入作為 HTTP POST 請求發送到 URL。端點通過使用與命令 hooks 相同的 JSON 輸出格式 的回應正文傳回結果。
- **提示 hooks**（`type: "prompt"`）：將提示發送到 Claude 模型進行單輪評估。模型返回 yes/no 決定作為 JSON。請參閱 基於提示的 hooks。
- **代理 hooks**（`type: "agent"`）：生成一個可以使用 Read、Grep 和 Glob 等工具來驗證條件的 subagent，然後返回決定。請參閱 基於代理的 hooks。

通用欄位

這些欄位適用於所有 hook 類型：

欄位	必需	描述
`type`	是	`"command"`、`"http"`、`"prompt"` 或 `"agent"`
`timeout`	否	取消前的秒數。預設值：命令 600、提示 30、代理 60
`statusMessage`	否	hook 執行時顯示的自訂微調訊息
`once`	否	如果為 `true`，每個工作階段只執行一次，然後被移除。僅限 skills，不是代理。請參閱 Skills 和代理中的 Hooks

命令 hook 欄位

除了 通用欄位 外，命令 hooks 還接受這些欄位：

欄位	必需	描述
`command`	是	要執行的 shell 命令
`async`	否	如果為 `true`，在背景執行而不阻止。請參閱 在背景執行 hooks

HTTP hook 欄位

除了 通用欄位 外，HTTP hooks 還接受這些欄位：

欄位	必需	描述
`url`	是	要發送 POST 請求的 URL

欄位	必需	描述
`headers`	否	其他 HTTP 標頭作為鍵值對。值支援使用 `$VAR_NAME` 或 `${VAR_NAME}` 語法的環境變數插值。只有列在 `allowedEnvVars` 中的變數才會被解析
`allowedEnvVars`	否	可能被插值到標頭值中的環境變數名稱清單。對未列出的變數的參考會被替換為空字串。任何環境變數插值都需要此項

Claude Code 將 hook 的 JSON 輸入 作為 POST 請求正文發送，`Content-Type: application/json`。回應正文使用與命令 hooks 相同的 JSON 輸出格式。

錯誤處理與命令 hooks 不同：非 2xx 回應、連線失敗和逾時都會產生非阻止性錯誤，允許執行繼續。要阻止工具呼叫或拒絕權限，請返回 2xx 回應，其 JSON 正文包含 `decision: "block"` 或具有 `permissionDecision: "deny"` 的 `hookSpecificOutput`。

此範例將 `PreToolUse` 事件發送到本機驗證服務，使用來自 `MY_TOKEN` 環境變數的令牌進行驗證：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/pre-tool-use",
            "timeout": 30,
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}
```

> **Note:**
>
> HTTP hooks 必須通過直接編輯設定 JSON 來配置。`/hooks` 互動式選單僅支援新增命令 hooks。

提示和代理 hook 欄位

除了 通用欄位 外，提示和代理 hooks 還接受這些欄位：

欄位	必需	描述
`prompt`	是	要發送到模型的提示文字。使用 `$ARGUMENTS` 作為 hook 輸入 JSON 的佔位符
`model`	否	用於評估的模型。預設為快速模型

所有匹配的 hooks 並行執行，相同的處理程式會自動去重。命令 hooks 按命令字串去重，HTTP hooks 按 URL 去重。處理程式在目前目錄中執行，使用 Claude Code 的環境。在遠端網路環境中，`$CLAUDE_CODE_REMOTE` 環境變數設定為 `"true"`，在本機 CLI 中未設定。

按路徑參考指令碼

使用環境變數按相對於專案或外掛根目錄的路徑參考 hook 指令碼，無論 hook 執行時的工作目錄如何：

- `$CLAUDE_PROJECT_DIR`：專案根目錄。用引號包裝以處理包含空格的路徑。
- `${CLAUDE_PLUGIN_ROOT}`：外掛的根目錄，用於與 plugin 一起打包的指令碼。

專案指令碼

此範例使用 `$CLAUDE_PROJECT_DIR` 在任何 `Write` 或 `Edit` 工具呼叫後從專案的 `.claude/hooks/` 目錄執行樣式檢查器：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/check-style.sh"
          }
        ]
      }
    ]
  }
}
```

外掛指令碼

在 `hooks/hooks.json` 中定義外掛 hooks，帶有可選的頂層 `description` 欄位。啟用外掛時，其 hooks 會與您的使用者和專案 hooks 合併。

此範例執行與外掛一起打包的格式化指令碼：

```
{
  "description": "Automatic code formatting",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format.sh",
            "timeout": 30
          }
        ]
      }
    ]
  }
}
```

有關建立外掛 hooks 的詳細資訊，請參閱 外掛元件參考。

Skills 和代理中的 Hooks

除了設定檔和外掛外，hooks 還可以使用 frontmatter 直接在 skills 和 subagents 中定義。這些 hooks 的範圍限於元件的生命週期，只有在該元件活躍時才執行。

支援所有 hook 事件。對於 subagents，`Stop` hooks 會自動轉換為 `SubagentStop`，因為這是 subagent 完成時觸發的事件。

Hooks 使用與基於設定的 hooks 相同的配置格式，但範圍限於元件的生命週期，並在完成時清理。

此 skill 定義了一個 `PreToolUse` hook，在每個 `Bash` 命令之前執行安全驗證指令碼：

```
---
name: secure-operations
description: Perform operations with security checks
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/security-check.sh"
---
```

代理在其 YAML frontmatter 中使用相同的格式。

`/hooks` 選單

在 Claude Code 中輸入 `/hooks` 以開啟互動式 hooks 管理器，您可以在其中查看、新增和刪除 hooks，而無需直接編輯設定檔。有關逐步說明，請參閱指南中的 設定您的第一個 hook。

選單中的每個 hook 都標有括號前綴，指示其來源：

- `[User]`：來自 `~/.claude/settings.json`
- `[Project]`：來自 `.claude/settings.json`
- `[Local]`：來自 `.claude/settings.local.json`
- `[Plugin]`：來自外掛的 `hooks/hooks.json`，唯讀

停用或移除 hooks

要移除 hook，請從設定 JSON 檔案中刪除其項目，或使用 `/hooks` 選單並選擇 hook 以刪除它。

要暫時停用所有 hooks 而不移除它們，請在設定檔中設定 `"disableAllHooks": true` 或使用 `/hooks` 選單中的切換。沒有辦法在保留 hook 在配置中的同時停用單個 hook。

`disableAllHooks` 設定遵循受管理的設定階層。如果管理員已通過受管理的原則設定配置了 hooks，則在使用者、專案或本機設定中設定的 `disableAllHooks` 無法停用這些受管理的 hooks。只有在受管理的設定層級設定的 `disableAllHooks` 才能停用受管理的 hooks。

對設定檔中 hooks 的直接編輯不會立即生效。Claude Code 在啟動時捕獲 hooks 的快照，並在整個工作階段中使用它。這可防止惡意或意外的 hook 修改在工作階段中途生效，而無需您的審查。如果 hooks 在外部被修改，Claude Code 會警告您，並要求在 `/hooks` 選單中進行審查，然後變更才能應用。

Hook 輸入和輸出

命令 hooks 通過 stdin 接收 JSON 資料，並通過退出代碼、stdout 和 stderr 傳回結果。HTTP hooks 接收相同的 JSON 作為 POST 請求正文，並通過 HTTP 回應正文傳回結果。本部分涵蓋所有事件通用的欄位和行為。每個事件在 Hook 事件 下的部分包括其特定的輸入架構和決定控制選項。

通用輸入欄位

除了每個 hook 事件 部分中記錄的事件特定欄位外，所有 hook 事件都接收這些欄位作為 JSON。對於命令 hooks，此 JSON 通過 stdin 到達。對於 HTTP hooks，它作為 POST 請求正文到達。

欄位	描述
`session_id`	目前工作階段識別碼
`transcript_path`	對話 JSON 的路徑
`cwd`	叫用 hook 時的目前工作目錄
`permission_mode`	目前 權限模式：`"default"`、`"plan"`、`"acceptEdits"`、`"dontAsk"` 或 `"bypassPermissions"`
`hook_event_name`	觸發的事件名稱

使用 `--agent` 執行或在 subagent 內執行時，包括兩個額外欄位：

欄位	描述
`agent_id`	Subagent 的唯一識別碼。僅當 hook 在 subagent 呼叫內觸發時出現。使用此項來區分 subagent hook 呼叫和主執行緒呼叫。
`agent_type`	代理名稱（例如 `"Explore"` 或 `"security-reviewer"`）。當工作階段使用 `--agent` 或 hook 在 subagent 內觸發時出現。對於 subagents，subagent 的類型優先於工作階段的 `--agent` 值。

例如，Bash 命令的 `PreToolUse` hook 在 stdin 上接收此內容：

```
{
  "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` 和 `tool_input` 欄位是事件特定的。每個 hook 事件 部分記錄了該事件的額外欄位。

退出代碼輸出

來自您的 hook 命令的退出代碼告訴 Claude Code 該操作是應該進行、被阻止還是被忽略。

退出 0 表示成功。Claude Code 解析 stdout 以查找 JSON 輸出欄位。JSON 輸出僅在退出 0 時處理。對於大多數事件，stdout 僅在詳細模式（`Ctrl+O`）中顯示。例外是 `UserPromptSubmit` 和 `SessionStart`，其中 stdout 被新增為 Claude 可以看到和作用的上下文。

退出 2 表示阻止性錯誤。Claude Code 忽略 stdout 和其中的任何 JSON。相反，stderr 文字被反饋給 Claude 作為錯誤訊息。效果取決於事件：`PreToolUse` 阻止工具呼叫，`UserPromptSubmit` 拒絕提示，等等。有關完整清單，請參閱 退出代碼 2 行為。

任何其他退出代碼 是非阻止性錯誤。stderr 在詳細模式（`Ctrl+O`）中顯示，執行繼續。

例如，一個 hook 命令指令碼，阻止危險的 Bash 命令：

```
#!/bin/bash
## 從 stdin 讀取 JSON 輸入，檢查命令
command=$(jq -r '.tool_input.command' < /dev/stdin)

if [[ "$command" == rm* ]]; then
  echo "Blocked: rm commands are not allowed" >&2
  exit 2  # 阻止性錯誤：工具呼叫被阻止
fi

exit 0  # 成功：工具呼叫進行
```

每個事件的退出代碼 2 行為

退出代碼 2 是 hook 發出「停止，不要這樣做」的方式。效果取決於事件，因為某些事件代表可以被阻止的操作（例如尚未發生的工具呼叫），而其他事件代表已經發生或無法防止的事情。

Hook 事件	可以阻止？	退出 2 時發生的情況
`PreToolUse`	是	阻止工具呼叫
`PermissionRequest`	是	拒絕權限
`UserPromptSubmit`	是	阻止提示處理並清除提示
`Stop`	是	防止 Claude 停止，繼續對話
`SubagentStop`	是	防止 subagent 停止
`TeammateIdle`	是	防止隊友閒置（隊友繼續工作）
`TaskCompleted`	是	防止任務被標記為已完成
`ConfigChange`	是	阻止配置變更生效（除了 `policy_settings`）
`PostToolUse`	否	向 Claude 顯示 stderr（工具已執行）
`PostToolUseFailure`	否	向 Claude 顯示 stderr（工具已失敗）

Hook 事件	可以阻止？	退出 2 時發生的情況
`Notification`	否	僅向使用者顯示 stderr
`SubagentStart`	否	僅向使用者顯示 stderr
`SessionStart`	否	僅向使用者顯示 stderr
`SessionEnd`	否	僅向使用者顯示 stderr
`PreCompact`	否	僅向使用者顯示 stderr
`WorktreeCreate`	是	任何非零退出代碼都會導致 worktree 建立失敗
`WorktreeRemove`	否	失敗僅在偵錯模式中記錄
`InstructionsLoaded`	否	退出代碼被忽略

HTTP 回應處理

HTTP hooks 使用 HTTP 狀態代碼和回應正文，而不是退出代碼和 stdout：

- **2xx 帶空正文**：成功，等同於退出代碼 0 且無輸出
- **2xx 帶純文字正文**：成功，文字被新增為上下文
- **2xx 帶 JSON 正文**：成功，使用與命令 hooks 相同的 JSON 輸出 架構進行解析
- **非 2xx 狀態**：非阻止性錯誤，執行繼續
- **連線失敗或逾時**：非阻止性錯誤，執行繼續

與命令 hooks 不同，HTTP hooks 無法僅通過狀態代碼發出阻止性錯誤信號。要阻止工具呼叫或拒絕權限，請返回 2xx 回應，其 JSON 正文包含適當的決定欄位。

JSON 輸出

退出代碼讓您允許或阻止，但 JSON 輸出提供更細粒度的控制。與其以代碼 2 退出來阻止，不如以 0 退出並將 JSON 物件列印到 stdout。Claude Code 從該 JSON 讀取特定欄位以控制行為，包括 決定控制 以阻止、允許或升級給使用者。

> **Note:**
>
> 您必須為每個 hook 選擇一種方法，而不是兩種：要麼單獨使用退出代碼進行信號傳遞，要麼以 0 退出並列印 JSON 以進行結構化控制。Claude Code 僅在退出 0 時處理 JSON。如果您退出 2，任何 JSON 都會被忽略。

您的 hook 的 stdout 必須僅包含 JSON 物件。如果您的 shell 設定檔在啟動時列印文字，它可能會干擾 JSON 解析。請參閱故障排除指南中的 JSON 驗證失敗。

JSON 物件支援三種欄位：

- **通用欄位**，如 `continue`，在所有事件中工作。這些列在下表中。
- **頂層 `decision` 和 `reason`** 由某些事件用於阻止或提供反饋。
- `hookSpecificOutput` 是一個嵌套物件，用於需要更豐富控制的事件。它需要一個設定為事件名稱的 `hookEventName` 欄位。

欄位	預設	描述
`continue`	`true`	如果為 `false`，Claude 在 hook 執行後完全停止處理。優先於任何事件特定的決定欄位
`stopReason`	無	hook 執行後向使用者顯示的訊息，當 `continue` 為 `false` 時。不向 Claude 顯示
`suppressOutput`	`false`	如果為 `true`，隱藏詳細模式輸出中的 stdout
`systemMessage`	無	向使用者顯示的警告訊息

要無論事件類型如何都完全停止 Claude：

```
{ "continue": false, "stopReason": "Build failed, fix errors before continuing" }
```

決定控制

並非每個事件都支援阻止或通過 JSON 控制行為。支援的事件各自使用不同的欄位集來表達該決定。在編寫 hook 之前，使用此表作為快速參考：

事件	決定模式	關鍵欄位
UserPromptSubmit、PostToolUse、PostToolUseFailure、Stop、SubagentStop、ConfigChange	頂層 `decision`	`decision: "block"`、`reason`

事件	決定模式	關鍵欄位
TeammateIdle、TaskCompleted	退出代碼或 `continue: false`	退出代碼 2 使用 stderr 反饋阻止操作。JSON `{"continue": false, "stopReason": "..."}` 也會完全停止隊友，匹配 `Stop` hook 行為
PreToolUse	`hookSpecificOutput`	`permissionDecision`（allow/deny/ask）、`permissionDecisionReason`
PermissionRequest	`hookSpecificOutput`	`decision.behavior`（allow/deny）
WorktreeCreate	stdout 路徑	Hook 列印建立的 worktree 的絕對路徑。非零退出失敗建立
WorktreeRemove、Notification、SessionEnd、PreCompact、InstructionsLoaded	無	無決定控制。用於副作用，如記錄或清理

以下是每種模式的實際範例：

頂層決定

由 `UserPromptSubmit`、`PostToolUse`、`PostToolUseFailure`、`Stop`、`SubagentStop` 和 `ConfigChange` 使用。唯一的值是 `"block"`。要允許操作進行，請從 JSON 中省略 `decision`，或以 0 退出而不帶任何 JSON：

```
{
  "decision": "block",
  "reason": "Test suite must pass before proceeding"
}
```

PreToolUse

使用 `hookSpecificOutput` 進行更豐富的控制：允許、拒絕或升級給使用者。您還可以在執行前修改工具輸入或為 Claude 注入額外上下文。有關完整的選項集，請參閱 PreToolUse 決定控制。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Database writes are not allowed"
  }
}
```

PermissionRequest

使用 `hookSpecificOutput` 代表使用者允許或拒絕權限請求。允許時，您還可以修改工具的輸入或應用權限規則，以便使用者不會再次被提示。有關完整的選項集，請參閱 PermissionRequest 決定控制。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

有關擴展範例，包括 Bash 命令驗證、提示篩選和自動批准指令碼，請參閱指南中的 您可以自動化的內容 和 Bash 命令驗證器參考實現。

Hook 事件

每個事件對應於 Claude Code 生命週期中 hooks 可以執行的一個點。下面的部分按生命週期順序排列：從工作階段設定通過代理迴圈到工作階段結束。每個部分描述事件何時觸發、它支援的匹配器、它接收的 JSON 輸入，以及如何通過輸出控制行為。

SessionStart

在 Claude Code 啟動新工作階段或恢復現有工作階段時執行。用於載入開發上下文，例如現有問題或程式碼庫的最近變更，或設定環境變數。對於不需要指令碼的靜態上下文，請改用 CLAUDE.md。

SessionStart 在每個工作階段執行，因此請保持這些 hooks 快速。僅支援 `type: "command"` hooks。

匹配器值對應於工作階段的啟動方式：

匹配器	何時觸發
`startup`	新工作階段
`resume`	`--resume`、`--continue` 或 `/resume`
`clear`	`/clear`
`compact`	自動或手動壓縮

SessionStart 輸入

除了 通用輸入欄位 外，SessionStart hooks 還接收 `source`、`model` 和可選的 `agent_type`。`source` 欄位指示工作階段如何啟動：新工作階段為 `"startup"`，恢復的工作階段為 `"resume"`，`/clear` 後為 `"clear"`，或壓縮後為 `"compact"`。`model` 欄位包含模型識別碼。如果您使用 `claude --agent <name>` 啟動 Claude Code，`agent_type` 欄位包含代理名稱。

```
{
  "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 決定控制

您的 hook 指令碼列印到 stdout 的任何文字都被新增為 Claude 的上下文。除了所有 hooks 可用的 JSON 輸出欄位 外，您還可以返回這些事件特定欄位：

欄位	描述
`additionalContext`	新增到 Claude 上下文的字串。多個 hooks 的值被連接

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "My additional context here"
  }
}
```

持久化環境變數

SessionStart hooks 可以存取 `CLAUDE_ENV_FILE` 環境變數，該變數提供一個檔案路徑，您可以在其中為後續 Bash 命令持久化環境變數。

要設定個別環境變數，請將 `export` 陳述式寫入 `CLAUDE_ENV_FILE`。使用追加（`>>`）來保留由其他 hooks 設定的變數：

```
#!/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
```

要捕獲設定命令中的所有環境變更，請比較之前和之後的匯出變數：

```
#!/bin/bash

ENV_BEFORE=$(export -p | sort)

## 執行修改環境的設定命令
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
```

寫入此檔案的任何變數都將在工作階段期間 Claude Code 執行的所有後續 Bash 命令中可用。

> **Note:**
>
> `CLAUDE_ENV_FILE` 可用於 SessionStart hooks。其他 hook 類型無法存取此變數。

InstructionsLoaded

當 `CLAUDE.md` 或 `.claude/rules/*.md` 檔案被載入到上下文中時觸發。此事件在工作階段啟動時為急切載入的檔案觸發，稍後當檔案被延遲載入時再次觸發，例如當 Claude 存取包含嵌套 `CLAUDE.md` 的子目錄時，或當具有 `paths:` frontmatter 的條件規則匹配時。該 hook 不支援阻止或決定控制。它以非同步方式執行以用於可觀測性目的。

InstructionsLoaded 不支援匹配器，在每次載入時觸發。

InstructionsLoaded 輸入

除了 通用輸入欄位 外，InstructionsLoaded hooks 還接收這些欄位：

欄位	描述
`file_path`	已載入的指令檔案的絕對路徑
`memory_type`	檔案的範圍：`"User"`、`"Project"`、`"Local"` 或 `"Managed"`

欄位	描述
`load_reason`	檔案被載入的原因：`"session_start"`、`"nested_traversal"`、`"path_glob_match"` 或 `"include"`
`globs`	檔案 `paths:` frontmatter 中的路徑 glob 模式（如果有）。僅針對 `path_glob_match` 載入出現
`trigger_file_path`	觸發此載入的檔案的路徑，用於延遲載入
`parent_file_path`	包含此檔案的父指令檔案的路徑，用於 `include` 載入

```
{
  "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 決定控制

InstructionsLoaded hooks 沒有決定控制。它們無法阻止或修改指令載入。使用此事件進行審計記錄、合規性追蹤或可觀測性。

UserPromptSubmit

在使用者提交提示時執行，在 Claude 處理之前。這允許您根據提示/對話新增額外上下文、驗證提示或阻止某些類型的提示。

UserPromptSubmit 輸入

除了 通用輸入欄位 外，UserPromptSubmit hooks 還接收包含使用者提交的文字的 `prompt` 欄位。

```
{
  "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 決定控制

`UserPromptSubmit` hooks 可以控制使用者提示是否被處理並新增上下文。所有 JSON 輸出欄位 都可用。

有兩種方法可以在退出代碼 0 時向對話新增上下文：

- **純文字 stdout**：寫入 stdout 的任何非 JSON 文字都被新增為上下文
- **帶 `additionalContext` 的 JSON**：使用下面的 JSON 格式以獲得更多控制。`additionalContext` 欄位被新增為上下文

純 stdout 在成績單中顯示為 hook 輸出。`additionalContext` 欄位被更謹慎地新增。

要阻止提示，請返回一個 JSON 物件，其 `decision` 設定為 `"block"`：

欄位	描述
`decision`	`"block"` 防止提示被處理並從上下文中清除它。省略以允許提示進行
`reason`	當 `decision` 為 `"block"` 時向使用者顯示。不新增到上下文
`additionalContext`	新增到 Claude 上下文的字串

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "My additional context here"
  }
}
```

> **Note:**
>
> JSON 格式對於簡單用例不是必需的。要新增上下文，您可以使用退出代碼 0 將純文字列印到 stdout。當您需要阻止提示或想要更結構化的控制時，使用 JSON。

PreToolUse

在 Claude 建立工具參數後和處理工具呼叫之前執行。匹配工具名稱：`Bash`、`Edit`、`Write`、`Read`、`Glob`、`Grep`、`Agent`、`WebFetch`、`WebSearch` 和任何 MCP 工具名稱。

使用 PreToolUse 決定控制 來允許、拒絕或要求使用工具的權限。

PreToolUse 輸入

除了 通用輸入欄位 外，PreToolUse hooks 還接收 `tool_name`、`tool_input` 和 `tool_use_id`。`tool_input` 欄位取決於工具：

Bash

執行 shell 命令。

欄位	類型	範例	描述
`command`	字串	`"npm test"`	要執行的 shell 命令
`description`	字串	`"Run test suite"`	命令執行內容的可選描述
`timeout`	數字	`120000`	可選逾時（毫秒）
`run_in_background`	布林值	`false`	是否在背景執行命令

Write

建立或覆寫檔案。

欄位	類型	範例	描述
`file_path`	字串	`"/path/to/file.txt"`	要寫入的檔案的絕對路徑
`content`	字串	`"file content"`	要寫入檔案的內容

Edit

替換現有檔案中的字串。

欄位	類型	範例	描述
`file_path`	字串	`"/path/to/file.txt"`	要編輯的檔案的絕對路徑
`old_string`	字串	`"original text"`	要查詢和替換的文字
`new_string`	字串	`"replacement text"`	替換文字
`replace_all`	布林值	`false`	是否替換所有出現

Read

讀取檔案內容。

欄位	類型	範例	描述
`file_path`	字串	`"/path/to/file.txt"`	要讀取的檔案的絕對路徑
`offset`	數字	`10`	可選的開始讀取的行號
`limit`	數字	`50`	可選的要讀取的行數

Glob

尋找與 glob 模式匹配的檔案。

欄位	類型	範例	描述
`pattern`	字串	`"**/*.ts"`	要匹配檔案的 glob 模式

欄位	類型	範例	描述
`path`	字串	`"/path/to/dir"`	可選的搜尋目錄。預設為目前工作目錄

Grep

使用正規表達式搜尋檔案內容。

欄位	類型	範例	描述
`pattern`	字串	`"TODO.*fix"`	要搜尋的正規表達式模式
`path`	字串	`"/path/to/dir"`	可選的要搜尋的檔案或目錄
`glob`	字串	`"*.ts"`	可選的 glob 模式以篩選檔案
`output_mode`	字串	`"content"`	`"content"`、`"files_with_matches"` 或 `"count"`。預設為 `"files_with_matches"`
`-i`	布林值	`true`	不區分大小寫的搜尋
`multiline`	布林值	`false`	啟用多行匹配

WebFetch

擷取和處理網路內容。

欄位	類型	範例	描述
`url`	字串	`"https://example.com/api"`	要擷取內容的 URL
`prompt`	字串	`"Extract the API endpoints"`	在擷取的內容上執行的提示

WebSearch

搜尋網路。

欄位	類型	範例	描述
`query`	字串	`"react hooks best practices"`	搜尋查詢
`allowed_domains`	陣列	`["docs.example.com"]`	可選：僅包含來自這些網域的結果
`blocked_domains`	陣列	`["spam.example.com"]`	可選：排除來自這些網域的結果

Agent

生成一個 subagent。

欄位	類型	範例	描述
`prompt`	字串	`"Find all API endpoints"`	代理要執行的任務
`description`	字串	`"Find API endpoints"`	任務的簡短描述
`subagent_type`	字串	`"Explore"`	要使用的專門代理類型
`model`	字串	`"sonnet"`	可選的模型別名以覆寫預設值

PreToolUse 決定控制

`PreToolUse` hooks 可以控制工具呼叫是否進行。與使用頂層 `decision` 欄位的其他 hooks 不同，PreToolUse 在 `hookSpecificOutput` 物件內返回其決定。這提供了更豐富的控制：三個結果（允許、拒絕或詢問）加上在執行前修改工具輸入的能力。

欄位	描述
`permissionDecision`	`"allow"` 繞過權限系統，`"deny"` 防止工具呼叫，`"ask"` 提示使用者確認
`permissionDecisionReason`	對於 `"allow"` 和 `"ask"`，向使用者顯示但不向 Claude 顯示。對於 `"deny"`，向 Claude 顯示
`updatedInput`	在執行前修改工具的輸入參數。與 `"allow"` 結合以自動批准，或與 `"ask"` 結合以向使用者顯示修改後的輸入

欄位	描述
`additionalContext`	在工具執行前新增到 Claude 上下文的字串

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow",
    "permissionDecisionReason": "My reason here",
    "updatedInput": {
      "field_to_modify": "new value"
    },
    "additionalContext": "Current environment: production. Proceed with caution."
  }
}
```

> **Note:**
>
> PreToolUse 之前使用頂層 `decision` 和 `reason` 欄位，但這些對此事件已棄用。改用 `hookSpecificOutput.permissionDecision` 和 `hookSpecificOutput.permissionDecisionReason`。棄用的值 `"approve"` 和 `"block"` 分別對應 `"allow"` 和 `"deny"`。PostToolUse 和 Stop 等其他事件繼續使用頂層 `decision` 和 `reason` 作為其目前格式。

PermissionRequest

在向使用者顯示權限對話框時執行。使用 PermissionRequest 決定控制 代表使用者允許或拒絕。

匹配工具名稱，與 PreToolUse 相同的值。

PermissionRequest 輸入

PermissionRequest hooks 接收 `tool_name` 和 `tool_input` 欄位，如 PreToolUse hooks，但沒有 `tool_use_id`。可選的 `permission_suggestions` 陣列包含使用者通常在權限對話框中看到的「總是允許」選項。區別在於 hook 何時觸發：PermissionRequest hooks 在權限對話框即將向使用者顯示時執行，而 PreToolUse hooks 在工具執行前執行，無論權限狀態如何。

```
{
  "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 決定控制

`PermissionRequest` hooks 可以允許或拒絕權限請求。除了所有 hooks 可用的 JSON 輸出欄位 外，您的 hook 指令碼可以返回一個 `decision` 物件，其中包含這些事件特定欄位：

欄位	描述
`behavior`	`"allow"` 授予權限，`"deny"` 拒絕它
`updatedInput`	僅適用於 `"allow"`：在執行前修改工具的輸入參數
`updatedPermissions`	僅適用於 `"allow"`：應用權限規則更新，等同於使用者選擇「總是允許」選項
`message`	僅適用於 `"deny"`：告訴 Claude 為什麼權限被拒絕
`interrupt`	僅適用於 `"deny"`：如果為 `true`，停止 Claude

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

PostToolUse

在工具成功完成後立即執行。

匹配工具名稱，與 PreToolUse 相同的值。

PostToolUse 輸入

`PostToolUse` hooks 在工具已經成功執行後觸發。輸入包括 `tool_input`（發送給工具的參數）和 `tool_response`（它返回的結果）。兩者的確切架構取決於工具。

```
{
  "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 決定控制

`PostToolUse` hooks 可以在工具執行後向 Claude 提供反饋。除了所有 hooks 可用的 JSON 輸出欄位 外，您的 hook 指令碼可以返回這些事件特定欄位：

欄位	描述
`decision`	`"block"` 提示 Claude 使用 `reason` 。省略以允許操作進行
`reason`	當 `decision` 為 `"block"` 時向 Claude 顯示的解釋
`additionalContext`	Claude 要考慮的額外上下文
`updatedMCPToolOutput`	僅適用於 MCP 工具：用提供的值替換工具的輸出

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "additionalContext": "Additional information for Claude"
  }
}
```

PostToolUseFailure

當工具執行失敗時執行。此事件針對拋出錯誤或返回失敗結果的工具呼叫觸發。使用此項來記錄失敗、發送警報或向 Claude 提供更正反饋。

匹配工具名稱，與 PreToolUse 相同的值。

PostToolUseFailure 輸入

PostToolUseFailure hooks 接收與 PostToolUse 相同的 `tool_name` 和 `tool_input` 欄位，以及作為頂層欄位的錯誤資訊：

```
{
  "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
}
```

欄位	描述
`error`	描述出錯的字串
`is_interrupt`	可選的布林值，指示失敗是否由使用者中斷引起

PostToolUseFailure 決定控制

`PostToolUseFailure` hooks 可以在工具失敗後向 Claude 提供上下文。除了所有 hooks 可用的 JSON 輸出欄位 外，您的 hook 指令碼可以返回這些事件特定欄位：

欄位	描述
`additionalContext`	Claude 要與錯誤一起考慮的額外上下文

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUseFailure",
    "additionalContext": "Additional information about the failure for Claude"
  }
}
```

Notification

在 Claude Code 發送通知時執行。匹配通知類型：`permission_prompt`、`idle_prompt`、`auth_success`、`elicitation_dialog`。省略匹配器以針對所有通知類型執行 hooks。

使用單獨的匹配器根據通知類型執行不同的處理程式。此配置在 Claude 需要權限批准時觸發權限特定的警報指令碼，在 Claude 閒置時觸發不同的通知：

```
{
  "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 輸入

除了 通用輸入欄位 外，Notification hooks 還接收包含通知文字的 `message`、可選的 `title` 和 `notification_type` 指示哪個類型觸發。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "Notification",
  "message": "Claude needs your permission to use Bash",
  "title": "Permission needed",
  "notification_type": "permission_prompt"
}
```

Notification hooks 無法阻止或修改通知。除了所有 hooks 可用的 JSON 輸出欄位 外，您可以返回 `additionalContext` 以向對話新增上下文：

欄位	描述
`additionalContext`	新增到 Claude 上下文的字串

SubagentStart

當通過 Agent 工具生成 Claude Code subagent 時執行。支援匹配器以按代理類型名稱篩選（內建代理，如 `Bash`、`Explore`、`Plan` 或來自 `.claude/agents/` 的自訂代理名稱）。

SubagentStart 輸入

除了 通用輸入欄位 外，SubagentStart hooks 還接收 `agent_id`（subagent 的唯一識別碼）和 `agent_type`（代理名稱，內建代理，如 `"Bash"`、`"Explore"`、`"Plan"` 或自訂代理名稱）。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SubagentStart",
  "agent_id": "agent-abc123",
  "agent_type": "Explore"
}
```

SubagentStart hooks 無法阻止 subagent 建立，但它們可以將上下文注入到 subagent 中。除了所有 hooks 可用的 JSON 輸出欄位 外，您可以返回：

欄位	描述
`additionalContext`	新增到 subagent 上下文的字串

```
{
  "hookSpecificOutput": {
    "hookEventName": "SubagentStart",
    "additionalContext": "Follow security guidelines for this task"
  }
}
```

SubagentStop

在 Claude Code subagent 完成回應時執行。匹配代理類型，與 SubagentStart 相同的值。

SubagentStop 輸入

除了 通用輸入欄位 外，SubagentStop hooks 還接收 `stop_hook_active`、`agent_id`、`agent_type`、`agent_transcript_path` 和 `last_assistant_message`。`agent_type` 欄位是用於匹配器篩選的值。`transcript_path` 是主工作階段的成績單，而 `agent_transcript_path` 是 subagent 自己的成績單，存儲在嵌套的 `subagents/` 資料夾中。`last_assistant_message` 欄位包含 subagent 最終回應的文字內容，因此 hooks 可以存取它而無需解析成績單檔案。

```
{
  "session_id": "abc123",
  "transcript_path": "~/.claude/projects/.../abc123.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SubagentStop",
  "stop_hook_active": false,
  "agent_id": "def456",
  "agent_type": "Explore",
  "agent_transcript_path": "~/.claude/projects/.../abc123/subagents/agent-
def456.jsonl",
  "last_assistant_message": "Analysis complete. Found 3 potential issues..."
}
```

SubagentStop hooks 使用與 Stop hooks 相同的決定控制格式。

Stop

在主 Claude Code 代理完成回應時執行。如果停止是由於使用者中斷而發生的，則不執行。

Stop 輸入

除了 通用輸入欄位 外，Stop hooks 還接收 `stop_hook_active` 和 `last_assistant_message`。`stop_hook_active` 欄位在 Claude Code 已經作為 stop hook 的結果繼續時為 `true`。檢查此值或處理成績單以防止 Claude Code 無限執行。`last_assistant_message` 欄位包含 Claude 最終回應的文字內容，因此 hooks 可以存取它而無需解析成績單檔案。

```
{
  "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 決定控制

`Stop` 和 `SubagentStop` hooks 可以控制 Claude 是否繼續。除了所有 hooks 可用的 JSON 輸出欄位 外，您的 hook 指令碼可以返回這些事件特定欄位：

欄位	描述
`decision`	`"block"` 防止 Claude 停止。省略以允許 Claude 停止
`reason`	當 `decision` 為 `"block"` 時必需。告訴 Claude 為什麼它應該繼續

```
{
  "decision": "block",
  "reason": "Must be provided when Claude is blocked from stopping"
}
```

TeammateIdle

在 agent team 隊友在完成其輪次後即將閒置時執行。使用此項來在隊友停止工作之前強制執行品質閘門，例如要求通過 lint 檢查或驗證輸出檔案存在。

當 `TeammateIdle` hook 以代碼 2 退出時，隊友會收到 stderr 訊息作為反饋，並繼續工作而不是閒置。要完全停止隊友而不是重新執行它，請返回 JSON，其中 `{"continue": false, "stopReason": "..."}`。TeammateIdle hooks 不支援匹配器，在每次出現時觸發。

TeammateIdle 輸入

除了 通用輸入欄位 外，TeammateIdle hooks 還接收 `teammate_name` 和 `team_name`。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "TeammateIdle",
  "teammate_name": "researcher",
  "team_name": "my-project"
}
```

欄位	描述
`teammate_name`	即將閒置的隊友的名稱
`team_name`	團隊的名稱

TeammateIdle 決定控制

TeammateIdle hooks 支援兩種方式來控制隊友行為：

- **退出代碼 2**：隊友會收到 stderr 訊息作為反饋，並繼續工作而不是閒置。
- **JSON** `{"continue": false, "stopReason": "..."}`：完全停止隊友，匹配 `Stop` hook 行為。`stopReason` 向使用者顯示。

此範例檢查建置成品是否存在，然後允許隊友閒置：

```
#!/bin/bash

if [ ! -f "./dist/output.js" ]; then
  echo "Build artifact missing. Run the build before stopping." >&2
  exit 2
fi

exit 0
```

TaskCompleted

當任務被標記為已完成時執行。這在兩種情況下觸發：當任何代理通過 TaskUpdate 工具明確標記任務為已完成時，或當 agent team 隊友完成其輪次且有進行中的任務時。使用此項來強制執行完成條件，例如通過測試或 lint 檢查，然後任務才能關閉。

當 `TaskCompleted` hook 以代碼 2 退出時，任務不被標記為已完成，stderr 訊息作為反饋反饋給模型。要完全停止隊友而不是重新執行它，請返回 JSON，其中 `{"continue": false, "stopReason": "..."}`。TaskCompleted hooks 不支援匹配器，在每次出現時觸發。

TaskCompleted 輸入

除了 通用輸入欄位 外，TaskCompleted hooks 還接收 `task_id`、`task_subject` 和可選的 `task_description`、`teammate_name` 和 `team_name`。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "TaskCompleted",
  "task_id": "task-001",
  "task_subject": "Implement user authentication",
  "task_description": "Add login and signup endpoints",
  "teammate_name": "implementer",
  "team_name": "my-project"
}
```

欄位	描述
`task_id`	被完成的任務的識別碼
`task_subject`	任務的標題
`task_description`	任務的詳細描述。可能不存在
`teammate_name`	完成任務的隊友的名稱。可能不存在
`team_name`	團隊的名稱。可能不存在

TaskCompleted 決定控制

TaskCompleted hooks 支援兩種方式來控制任務完成：

- **退出代碼 2**：任務不被標記為已完成，stderr 訊息作為反饋反饋給模型。
- **JSON `{"continue": false, "stopReason": "..."}`**：完全停止隊友，匹配 `Stop` hook 行為。`stopReason` 向使用者顯示。

此範例執行測試並在測試失敗時阻止任務完成：

```
#!/bin/bash
INPUT=$(cat)
TASK_SUBJECT=$(echo "$INPUT" | jq -r '.task_subject')

## 執行測試套件
if ! npm test 2>&1; then
  echo "Tests not passing. Fix failing tests before completing: $TASK_SUBJECT" >&2
  exit 2
fi

exit 0
```

ConfigChange

當配置檔案在工作階段期間變更時執行。使用此項來審計設定變更、強制執行安全原則或阻止對配置檔案的未授權修改。

ConfigChange hooks 針對設定檔、受管理的原則設定和 skill 檔案的變更觸發。輸入中的 `source` 欄位告訴您哪種類型的配置變更，可選的 `file_path` 欄位提供變更檔案的路徑。

匹配器篩選配置來源：

匹配器	何時觸發
`user_settings`	`~/.claude/settings.json` 變更
`project_settings`	`.claude/settings.json` 變更
`local_settings`	`.claude/settings.local.json` 變更
`policy_settings`	受管理的原則設定變更
`skills`	`.claude/skills/` 中的 skill 檔案變更

此範例記錄所有配置變更以進行安全審計：

```
{
  "hooks": {
    "ConfigChange": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/audit-config-
change.sh"
          }
        ]
      }
    ]
  }
}
```

ConfigChange 輸入

除了 通用輸入欄位 外，ConfigChange hooks 還接收 `source` 和可選的 `file_path`。`source` 欄位指示哪種配置類型變更，`file_path` 提供被修改的特定檔案的路徑。

```
{
  "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 決定控制

ConfigChange hooks 可以阻止配置變更生效。使用退出代碼 2 或 JSON `decision` 來防止變更。被阻止時，新設定不會應用於執行中的工作階段。

欄位	描述
`decision`	`"block"` 防止配置變更被應用。省略以允許變更
`reason`	當 `decision` 為 `"block"` 時向使用者顯示的解釋

```
{
  "decision": "block",
  "reason": "Configuration changes to project settings require admin approval"
}
```

`policy_settings` 變更無法被阻止。Hooks 仍然針對 `policy_settings` 來源觸發，因此您可以使用它們進行審計記錄，但任何阻止決定都會被忽略。這確保企業管理的設定始終生效。

WorktreeCreate

當您執行 `claude --worktree` 或 subagent 使用 `isolation: "worktree"` 時，Claude Code 使用 `git worktree` 建立隔離的工作副本。如果您配置 WorktreeCreate hook，它會替換預設的 git 行為，讓您使用不同的版本控制系統，如 SVN、Perforce 或 Mercurial。

Hook 必須在 stdout 上列印建立的 worktree 目錄的絕對路徑。Claude Code 使用此路徑作為隔離工作階段的工作目錄。

此範例建立 SVN 工作副本並列印路徑供 Claude Code 使用。將儲存庫 URL 替換為您自己的：

```
{
  "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\"'"
          }
        ]
      }
    ]
  }
}
```

Hook 從 stdin 上的 JSON 輸入讀取 worktree `name`，將新副本簽出到新目錄，並列印目錄路徑。最後一行的 `echo` 是 Claude Code 讀取的 worktree 路徑。將任何其他輸出重定向到 stderr，以免干擾路徑。

WorktreeCreate 輸入

除了 通用輸入欄位 外，WorktreeCreate hooks 還接收 `name` 欄位。這是新 worktree 的 slug 識別碼，由使用者指定或自動生成（例如 `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 輸出

Hook 必須在 stdout 上列印建立的 worktree 目錄的絕對路徑。如果 hook 失敗或不產生輸出，worktree 建立會失敗並出現錯誤。

WorktreeCreate hooks 不使用標準的允許/阻止決定模型。相反，hook 的成功或失敗決定結果。僅支援 `type: "command"` hooks。

WorktreeRemove

WorktreeCreate 的清理對應項。此 hook 在 worktree 被移除時觸發，要麼當您退出 `--worktree` 工作階段並選擇移除它時，要麼當具有 `isolation: "worktree"` 的 subagent 完成時。對於基於 git 的 worktrees，Claude 使用 `git worktree remove` 自動處理清理。如果您為非 git 版本控制系統配置了 WorktreeCreate hook，請將其與 WorktreeRemove hook 配對以處理清理。沒有它，worktree 目錄會留在磁碟上。

Claude Code 將 WorktreeCreate 在 stdout 上列印的路徑作為 `worktree_path` 在 hook 輸入中傳遞。此範例讀取該路徑並移除目錄：

```
{
  "hooks": {
    "WorktreeRemove": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'jq -r .worktree_path | xargs rm -rf'"
          }
        ]
      }
    ]
  }
}
```

WorktreeRemove 輸入

除了 通用輸入欄位 外，WorktreeRemove hooks 還接收 `worktree_path` 欄位，這是被移除的 worktree 的絕對路徑。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "WorktreeRemove",
  "worktree_path": "/Users/.../my-project/.claude/worktrees/feature-auth"
}
```

WorktreeRemove hooks 沒有決定控制。它們無法阻止 worktree 移除，但可以執行清理任務，如移除版本控制狀態或存檔變更。Hook 失敗僅在偵錯模式中記錄。僅支援 `type: "command"` hooks。

PreCompact

在 Claude Code 即將執行壓縮操作之前執行。

匹配器值指示壓縮是手動還是自動觸發：

匹配器	何時觸發
`manual`	`/compact`
`auto`	當上下文視窗滿時自動壓縮

PreCompact 輸入

除了 通用輸入欄位 外，PreCompact hooks 還接收 `trigger` 和 `custom_instructions`。對於 `manual`，`custom_instructions` 包含使用者傳遞到 `/compact` 的內容。對於 `auto`，`custom_instructions` 為空。

```
{
  "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

在 Claude Code 工作階段結束時執行。用於清理任務、記錄工作階段統計資訊或儲存工作階段狀態。支援匹配器以按退出原因篩選。

hook 輸入中的 `reason` 欄位指示工作階段為何結束：

原因	描述
`clear`	使用 `/clear` 命令清除工作階段
`logout`	使用者登出
`prompt_input_exit`	使用者在提示輸入可見時退出
`bypass_permissions_disabled`	繞過權限模式被停用
`other`	其他退出原因

SessionEnd 輸入

除了 通用輸入欄位 外，SessionEnd hooks 還接收 `reason` 欄位，指示工作階段為何結束。有關所有值，請參閱上面的 原因表。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../
00893aaf-19fa-41d2-8238-13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SessionEnd",
  "reason": "other"
}
```

SessionEnd hooks 沒有決定控制。它們無法阻止工作階段終止，但可以執行清理任務。

基於提示的 Hooks

除了命令和 HTTP hooks 外，Claude Code 還支援基於提示的 hooks（`type: "prompt"`），使用 LLM 評估是否允許或阻止操作，以及代理 hooks（`type: "agent"`），生成具有工具存取的代理驗證器。並非所有事件都支援每種 hook 類型。

支援所有四種 hook 類型（`command`、`http`、`prompt` 和 `agent`）的事件：

- `PermissionRequest`
- `PostToolUse`
- `PostToolUseFailure`
- `PreToolUse`
- `Stop`
- `SubagentStop`
- `TaskCompleted`
- `UserPromptSubmit`

僅支援 `type: "command"` hooks 的事件：

- `ConfigChange`
- `InstructionsLoaded`
- `Notification`
- `PreCompact`
- `SessionEnd`
- `SessionStart`
- `SubagentStart`

- `TeammateIdle`
- `WorktreeCreate`
- `WorktreeRemove`

基於提示的 Hooks 如何工作

基於提示的 hooks 不執行 Bash 命令，而是：

1. 將 hook 輸入和您的提示發送到 Claude 模型，預設為 Haiku
2. LLM 使用包含決定的結構化 JSON 回應
3. Claude Code 自動處理決定

提示 hook 配置

將 `type` 設定為 `"prompt"` 並提供 `prompt` 字串而不是 `command`。使用 `$ARGUMENTS` 佔位符將 hook 的 JSON 輸入資料注入到您的提示文字中。Claude Code 將組合的提示和輸入發送到快速 Claude 模型，該模型返回 JSON 決定。

此 `Stop` hook 要求 LLM 評估在允許 Claude 完成之前是否所有任務都已完成：

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if Claude should stop: $ARGUMENTS. Check if all
tasks are complete."
          }
        ]
      }
    ]
  }
}
```

欄位	必需	描述
`type`	是	必須為 `"prompt"`

欄位	必需	描述
`prompt`	是	要發送到 LLM 的提示文字。使用 `$ARGUMENTS` 作為 hook 輸入 JSON 的佔位符。如果 `$ARGUMENTS` 不存在，輸入 JSON 會附加到提示
`model`	否	用於評估的模型。預設為快速模型
`timeout`	否	逾時（秒）。預設值：30

回應架構

LLM 必須使用以下 JSON 回應：

```
{
  "ok": true | false,
  "reason": "Explanation for the decision"
}
```

欄位	描述
`ok`	`true` 允許操作，`false` 防止它
`reason`	當 `ok` 為 `false` 時必需。向 Claude 顯示的解釋

範例：多條件 Stop hook

此 `Stop` hook 使用詳細提示檢查三個條件，然後允許 Claude 停止。如果 `"ok"` 為 `false`，Claude 繼續工作，提供的原因作為其下一個指令。`SubagentStop` hooks 使用相同的格式來評估 subagent 是否應該停止：

```
{
  "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
          }
        ]
      }
    ]
  }
}
```

基於代理的 Hooks

基於代理的 hooks（`type: "agent"`）類似於基於提示的 hooks，但具有多輪工具存取。代理 hook 不是單一 LLM 呼叫，而是生成一個可以讀取檔案、搜尋程式碼和檢查程式碼庫以驗證條件的 subagent。代理 hooks 支援與基於提示的 hooks 相同的事件。

基於代理的 Hooks 如何工作

當代理 hook 觸發時：

1. Claude Code 生成一個 subagent，使用您的提示和 hook 的 JSON 輸入
2. Subagent 可以使用 Read、Grep 和 Glob 等工具進行調查
3. 在最多 50 輪後，subagent 返回結構化的 `{ "ok": true/false }` 決定
4. Claude Code 以與提示 hook 相同的方式處理決定

代理 hooks 在驗證需要檢查實際檔案或測試輸出時很有用，而不僅僅是評估 hook 輸入資料。

代理 hook 配置

將 `type` 設定為 `"agent"` 並提供 `prompt` 字串。配置欄位與 提示 hooks 相同，但逾時更長：

欄位	必需	描述
`type`	是	必須為 `"agent"`
`prompt`	是	描述要驗證的內容的提示。使用 `$ARGUMENTS` 作為 hook 輸入 JSON 的佔位符
`model`	否	要使用的模型。預設為快速模型
`timeout`	否	逾時（秒）。預設值：60

回應架構與提示 hooks 相同：`{ "ok": true }` 允許或 `{ "ok": false, "reason": "..." }` 阻止。

此 `Stop` hook 驗證所有單元測試通過，然後允許 Claude 完成：

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

在背景執行 Hooks

預設情況下，hooks 會阻止 Claude 的執行，直到它們完成。對於長時間執行的任務，如部署、測試套件或外部 API 呼叫，設定 `"async": true` 以在背景執行 hook，同時 Claude 繼續工作。非同步 hooks 無法阻止或控制 Claude 的行為：回應欄位，如 `decision`、`permissionDecision` 和 `continue` 沒有效果，因為它們會控制的操作已經完成。

配置非同步 Hook

將 `"async": true` 新增到命令 hook 的配置以在背景執行它而不阻止 Claude。此欄位僅在 `type: "command"` hooks 上可用。

此 hook 在每個 `Write` 工具呼叫後執行測試指令碼。Claude 立即繼續工作，同時 `run-tests.sh` 執行最多 120 秒。當指令碼完成時，其輸出在下一個對話輪次上傳遞：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/run-tests.sh",
            "async": true,
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

`timeout` 欄位設定背景程序的最大時間（秒）。如果未指定，非同步 hooks 使用與同步 hooks 相同的 10 分鐘預設值。

非同步 Hooks 如何執行

當非同步 hook 觸發時，Claude Code 啟動 hook 程序並立即繼續，而不等待它完成。Hook 在 stdin 上接收與同步 hook 相同的 JSON 輸入。

背景程序退出後，如果 hook 產生了帶有 `systemMessage` 或 `additionalContext` 欄位的 JSON 回應，該內容會在下一個對話輪次上作為上下文傳遞給 Claude。

範例：檔案變更後執行測試

此 hook 在 Claude 寫入檔案時在背景啟動測試套件，然後在測試完成時將結果報告回 Claude。將此指令碼儲存到專案中的 `.claude/hooks/run-tests-async.sh` 並使用 `chmod +x` 使其可執行：

```
#!/bin/bash
## run-tests-async.sh

## 從 stdin 讀取 hook 輸入
INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

## 僅針對原始檔案執行測試
if [[ "$FILE_PATH" =~ *.ts && "$FILE_PATH" =~ *.js ]]; then
  exit 0
fi

## 執行測試並通過 systemMessage 報告結果
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
```

然後將此配置新增到專案根目錄中的 `.claude/settings.json`。`async: true` 標誌讓 Claude 在測試執行時繼續工作：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/run-tests-async.sh",
            "async": true,
            "timeout": 300
          }
        ]
      }
    ]
  }
}
```

限制

非同步 hooks 與同步 hooks 相比有幾個限制：

- 僅 `type: "command"` hooks 支援 `async`。基於提示的 hooks 無法非同步執行。
- 非同步 hooks 無法阻止工具呼叫或返回決定。當 hook 完成時，觸發操作已經進行。
- Hook 輸出在下一個對話輪次上傳遞。如果工作階段閒置，回應會等待直到下一個使用者互動。
- 每次執行都會建立一個單獨的背景程序。同一非同步 hook 的多次觸發之間沒有去重。

安全考慮

免責聲明

命令 hooks 以您的系統使用者的完整權限執行。

> **Warning:**
>
> 命令 hooks 以您的完整使用者權限執行 shell 命令。它們可以修改、刪除或存取您的使用者帳戶可以存取的任何檔案。在將任何 hook 命令新增到您的配置之前，請審查並測試它們。

安全最佳實踐

編寫 hooks 時，請記住這些實踐：

- **驗證和清理輸入**：永遠不要盲目信任輸入資料
- **始終引用 shell 變數**：使用 `"$VAR"` 而不是 `$VAR`
- **阻止路徑遍歷**：檢查檔案路徑中的 `..`
- **使用絕對路徑**：為指令碼指定完整路徑，使用 `"$CLAUDE_PROJECT_DIR"` 作為專案根目錄
- **跳過敏感檔案**：避免 `.env`、`.git/`、金鑰等

偵錯 Hooks

執行 `claude --debug` 以查看 hook 執行詳細資訊，包括哪些 hooks 匹配、它們的退出代碼和輸出。使用 `Ctrl+O` 切換詳細模式以在成績單中查看 hook 進度。

```
[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>
```

有關故障排除常見問題，如 hooks 未觸發、無限 Stop hook 迴圈或配置錯誤，請參閱指南中的 限制和故障排除。

使用 hooks 自動化工作流程

> 當 Claude Code 編輯檔案、完成任務或需要輸入時自動執行 shell 命令。格式化程式碼、發送通知、驗證命令並強制執行專案規則。

Hooks 是使用者定義的 shell 命令，在 Claude Code 生命週期中的特定點執行。它們提供對 Claude Code 行為的確定性控制，確保某些操作始終發生，而不是依賴 LLM 選擇執行它們。使用 hooks 來強制執行專案規則、自動化重複任務，並將 Claude Code 與您現有的工具整合。

對於需要判斷而不是確定性規則的決策，您也可以使用基於提示的 hooks 或基於代理的 hooks，它們使用 Claude 模型來評估條件。

有關擴展 Claude Code 的其他方式，請參閱skills以提供 Claude 額外的指令和可執行命令、subagents以在隔離的上下文中執行任務，以及plugins以打包要在專案間共享的擴展。

> **Tip:**
>
> 本指南涵蓋常見用例和如何開始。有關完整的事件架構、JSON 輸入/輸出格式和非同步 hooks 和 MCP 工具 hooks 等進階功能，請參閱 Hooks 參考。

設定您的第一個 hook

在 Claude Code 中透過 `/hooks` 互動式選單建立 hook 的最快方式。本逐步解說建立一個桌面通知 hook，因此每當 Claude 等待您的輸入而不是監視終端時，您都會收到警報。

Step 1: 開啟 hooks 選單

在 Claude Code CLI 中輸入 `/hooks`。您將看到所有可用 hook 事件的列表，以及禁用所有 hooks 的選項。每個事件對應於 Claude 生命週期中的一個點，您可以在該點執行自訂程式碼。選擇 `Notification` 以建立當 Claude 需要您注意時觸發的 hook。

Step 2: 配置匹配器

選單顯示匹配器列表，這些匹配器篩選 hook 何時觸發。將匹配器設定為 `*` 以在所有通知類型上觸發。您稍後可以透過將匹配器變更為特定值（如 `permission_prompt` 或 `idle_prompt`）來縮小範圍。

Step 3: 新增您的命令

選擇 `+ Add new hook…`。選單會提示您輸入事件觸發時要執行的 shell 命令。Hooks 執行您提供的任何 shell 命令，因此您可以使用您平台的內建通知工具。複製您的作業系統的命令：

macOS

使用 `osascript` 透過 AppleScript 觸發原生 macOS 通知：

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

Linux

使用 `notify-send`，它在大多數帶有通知守護程式的 Linux 桌面上預先安裝：

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

Windows (PowerShell)

使用 PowerShell 透過 .NET 的 Windows Forms 顯示原生訊息框：

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

Step 4: 選擇儲存位置

選單詢問在何處儲存 hook 配置。選擇 `User settings` 將其儲存在 `~/.claude/settings.json` 中，這會將 hook 應用於您的所有專案。您也可以選擇 `Project settings` 將其限制在目前專案。有關所有可用範圍，請參閱配置 hook 位置。

Step 5: 測試 hook

按 `Esc` 返回 CLI。要求 Claude 執行需要權限的操作，然後切換離開終端。您應該會收到桌面通知。

您可以自動化的內容

Hooks 讓您在 Claude Code 生命週期中的關鍵點執行程式碼：編輯後格式化檔案、在執行前阻止命令、當 Claude 需要輸入時發送通知、在工作階段開始時注入上下文等。有關 hook 事件的完整列表，請參閱 Hooks 參考。

每個範例都包含一個現成可用的配置區塊，您可以將其新增到設定檔。最常見的模式：

- 當 Claude 需要輸入時收到通知
- 編輯後自動格式化程式碼
- 阻止編輯受保護的檔案
- 壓縮後重新注入上下文
- 審計配置變更

當 Claude 需要輸入時收到通知

每當 Claude 完成工作並需要您的輸入時收到桌面通知，以便您可以切換到其他任務而無需檢查終端。

此 hook 使用 `Notification` 事件，當 Claude 等待輸入或權限時觸發。下面的每個標籤使用平台的原生通知命令。將此新增到 `~/.claude/settings.json`，或使用上面的互動式逐步解說透過 `/hooks` 配置它：

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')\""
          }
        ]
      }
    ]
  }
}
```

編輯後自動格式化程式碼

在 Claude 編輯的每個檔案上自動執行 Prettier，以便格式化保持一致而無需手動干預。

此 hook 使用 `PostToolUse` 事件搭配 `Edit|Write` 匹配器，因此它只在檔案編輯工具之後執行。該命令使用 `jq` 提取編輯的檔案路徑並將其傳遞給 Prettier。將此新增到您的專案根目錄中的 `.claude/settings.json`：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --
write"
          }
        ]
      }
    ]
  }
}
```

> **Note:**
>
> 本頁上的 Bash 範例使用 `jq` 進行 JSON 解析。使用 `brew install jq`（macOS）、`apt-get install jq`（Debian/Ubuntu）安裝它，或參閱 `jq` 下載。

阻止編輯受保護的檔案

防止 Claude 修改敏感檔案，如 `.env`、`package-lock.json` 或 `.git/` 中的任何內容。Claude 會收到解釋為什麼編輯被阻止的反饋，因此它可以調整其方法。

此範例使用 hook 呼叫的單獨指令檔。該指令檢查目標檔案路徑是否與受保護的模式列表相符，並以代碼 2 退出以阻止編輯。

Step 1: 建立 hook 指令

將此儲存到 `.claude/hooks/protect-files.sh`：

```
#!/bin/bash
## protect-files.sh

INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

PROTECTED_PATTERNS=(".env" "package-lock.json" ".git/")

for pattern in "${PROTECTED_PATTERNS[@]}"; do
  if [[ "$FILE_PATH" == *"$pattern"* ]]; then
    echo "Blocked: $FILE_PATH matches protected pattern '$pattern'" >&2
    exit 2
  fi
done

exit 0
```

Step 2: 使指令可執行（macOS/Linux）

Hook 指令必須可執行，Claude Code 才能執行它們：

```
chmod +x .claude/hooks/protect-files.sh
```

Step 3: 註冊 hook

將 `PreToolUse` hook 新增到 `.claude/settings.json`，在任何 `Edit` 或 `Write` 工具呼叫之前執行指令：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/protect-files.sh"
          }
        ]
      }
    ]
  }
}
```

壓縮後重新注入上下文

當 Claude 的上下文視窗填滿時，壓縮會總結對話以釋放空間。這可能會遺失重要細節。使用帶有 `compact` 匹配器的 `SessionStart` hook 在每次壓縮後重新注入關鍵上下文。

您的命令寫入 stdout 的任何文字都會新增到 Claude 的上下文中。此範例提醒 Claude 專案慣例和最近的工作。將此新增到您的專案根目錄中的 `.claude/settings.json`：

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Reminder: use Bun, not npm. Run bun test before
committing. Current sprint: auth refactor.'"
          }
        ]
      }
    ]
  }
}
```

您可以將 `echo` 替換為任何產生動態輸出的命令，如 `git log --oneline -5` 以顯示最近的提交。有關在每個工作階段開始時注入上下文，請考慮改用 CLAUDE.md。有關環境變數，請參閱參考中的 `CLAUDE_ENV_FILE`。

審計配置變更

追蹤工作階段期間設定或 skills 檔案何時變更。`ConfigChange` 事件在外部程序或編輯器修改配置檔案時觸發，因此您可以記錄變更以進行合規性檢查或阻止未授權的修改。

此範例將每個變更附加到審計日誌。將此新增到 `~/.claude/settings.json`：

```
{
  "hooks": {
    "ConfigChange": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "jq -c '{timestamp: now | todate, source: .source,
file: .file_path}' >> ~/claude-config-audit.log"
          }
        ]
      }
    ]
  }
}
```

匹配器按配置類型篩選：`user_settings`、`project_settings`、`local_settings`、`policy_settings` 或 `skills`。要阻止變更生效，以代碼 2 退出或傳回 `{"decision": "block"}`。有關完整的輸入架構，請參閱 ConfigChange 參考。

Hooks 如何工作

Hook 事件在 Claude Code 中的特定生命週期點觸發。當事件觸發時，所有匹配的 hooks 並行執行，相同的 hook 命令會自動去重。下表顯示每個事件及其觸發時間：

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

Event	When it fires
`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 `.claude/rules/*.md` 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 `--worktree` or `isolation: "worktree"`. 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

每個 hook 都有一個 `type` 來決定它如何執行。大多數 hooks 使用 `"type": "command"`，它執行 shell 命令。還有三種其他類型可用：

- `"type": "http"`：POST 事件資料到 URL。請參閱 HTTP hooks。
- `"type": "prompt"`：單輪 LLM 評估。請參閱基於提示的 hooks。
- `"type": "agent"`：具有工具存取的多輪驗證。請參閱基於代理的 hooks。

讀取輸入並傳回輸出

Hooks 透過 stdin、stdout、stderr 和退出代碼與 Claude Code 通訊。當事件觸發時，Claude Code 將事件特定的資料作為 JSON 傳遞到您的指令的 stdin。您的指令讀取該資料、執行其工作，並透過退出代碼告訴 Claude Code 接下來要做什麼。

Hook 輸入

每個事件都包含常見欄位，如 `session_id` 和 `cwd`，但每個事件類型都新增不同的資料。例如，當 Claude 執行 Bash 命令時，`PreToolUse` hook 在 stdin 上接收類似以下內容：

```
{
  "session_id": "abc123",           // 此工作階段的唯一 ID
  "cwd": "/Users/sarah/myproject", // 事件觸發時的工作目錄
  "hook_event_name": "PreToolUse", // 哪個事件觸發了此 hook
  "tool_name": "Bash",              // Claude 即將使用的工具
  "tool_input": {                   // Claude 傳遞給工具的引數
    "command": "npm test"           // 對於 Bash，這是 shell 命令
  }
}
```

您的指令可以解析該 JSON 並對任何這些欄位採取行動。`UserPromptSubmit` hooks 改為取得 `prompt` 文字，`SessionStart` hooks 取得 `source`（startup、resume、clear、compact），等等。有關共享欄位，請參閱參考中的常見輸入欄位，以及每個事件的部分以了解事件特定的架構。

Hook 輸出

您的指令透過寫入 stdout 或 stderr 並以特定代碼退出來告訴 Claude Code 接下來要做什麼。例如，想要阻止命令的 `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 變成 Claude 的反饋
  exit 2 # exit 2 = 阻止操作
fi

exit 0  # exit 0 = 讓它繼續
```

退出代碼決定接下來會發生什麼：

- **Exit 0**：操作繼續。對於 `UserPromptSubmit` 和 `SessionStart` hooks，您寫入 stdout 的任何內容都會新增到 Claude 的上下文中。
- **Exit 2**：操作被阻止。寫入原因到 stderr，Claude 會將其作為反饋接收，以便它可以調整。
- **任何其他退出代碼**：操作繼續。Stderr 被記錄但不顯示給 Claude。使用 `Ctrl+O` 切換詳細模式以在文字記錄中查看這些訊息。

結構化 JSON 輸出

退出代碼給您兩個選項：允許或阻止。為了獲得更多控制，退出 0 並改為將 JSON 物件列印到 stdout。

> **Note:**
>
> 使用 exit 2 以 stderr 訊息阻止，或使用 exit 0 搭配 JSON 進行結構化控制。不要混合它們：Claude Code 在您退出 2 時忽略 JSON。

例如，`PreToolUse` hook 可以拒絕工具呼叫並告訴 Claude 為什麼，或將其升級給使用者以獲得批准：

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Use rg instead of grep for better performance"
  }
}
```

Claude Code 讀取 `permissionDecision` 並取消工具呼叫，然後將 `permissionDecisionReason` 反饋給 Claude。這三個選項特定於 `PreToolUse`：

- `"allow"`：繼續而不顯示權限提示
- `"deny"`：取消工具呼叫並將原因傳送給 Claude
- `"ask"`：照常向使用者顯示權限提示

其他事件使用不同的決策模式。例如，`PostToolUse` 和 `Stop` hooks 使用頂級 `decision: "block"` 欄位，而 `PermissionRequest` 使用 `hookSpecificOutput.decision.behavior`。有關按事件的完整分解，請參閱參考中的摘要表。

對於 `UserPromptSubmit` hooks，改用 `additionalContext` 將文字注入到 Claude 的上下文中。基於提示的 hooks（`type: "prompt"`）以不同方式處理輸出：請參閱基於提示的 hooks。

使用匹配器篩選 hooks

沒有匹配器，hook 會在其事件的每次出現時觸發。匹配器讓您縮小範圍。例如，如果您只想在檔案編輯後執行格式化程式（而不是在每次工具呼叫後），請將匹配器新增到您的 `PostToolUse` hook：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "prettier --write ..." }
        ]
      }
    ]
  }
}
```

`"Edit|Write"` 匹配器是與工具名稱相符的正規表達式模式。hook 只在 Claude 使用 `Edit` 或 `Write` 工具時觸發，而不是在它使用 `Bash`、`Read` 或任何其他工具時。

每個事件類型都在特定欄位上進行匹配。匹配器支援精確字串和正規表達式模式：

事件	匹配器篩選的內容	範例匹配器值
`PreToolUse`、`PostToolUse`、`PostToolUseFailure`、`PermissionRequest`	工具名稱	`Bash`、`Edit|Write`、`mcp__.*`
`SessionStart`	工作階段如何開始	`startup`、`resume`、`clear`、`compact`
`SessionEnd`	工作階段為什麼結束	`clear`、`logout`、`prompt_input_exit`、`bypass_permissions_disabled`、`other`
`Notification`	通知類型	`permission_prompt`、`idle_prompt`、`auth_success`、`elicitation_dialog`
`SubagentStart`	代理類型	`Bash`、`Explore`、`Plan` 或自訂代理名稱
`PreCompact`	什麼觸發了壓縮	`manual`、`auto`

事件	匹配器篩選的內容	範例匹配器值
`SubagentStop`	代理類型	與 `SubagentStart` 相同的值
`ConfigChange`	配置來源	`user_settings`、`project_settings`、`local_settings`、`policy_settings`、`skills`
`UserPromptSubmit`、`Stop`、`TeammateIdle`、`TaskCompleted`、`WorktreeCreate`、`WorktreeRemove`	不支援匹配器	始終在每次出現時觸發

下面是在不同事件類型上顯示匹配器的幾個更多範例：

記錄每個 Bash 命令

僅匹配 `Bash` 工具呼叫並將每個命令記錄到檔案。`PostToolUse` 事件在命令完成後觸發，因此 `tool_input.command` 包含執行的內容。hook 在 stdin 上接收事件資料作為 JSON，`jq -r '.tool_input.command'` 僅提取命令字串，`>>` 將其附加到日誌檔案：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.command' >> ~/.claude/command-log.txt"
          }
        ]
      }
    ]
  }
}
```

匹配 MCP 工具

MCP 工具使用與內建工具不同的命名慣例：`mcp__<server>__<tool>`，其中 `<server>` 是 MCP 伺服器名稱，`<tool>` 是它提供的工具。例如，`mcp__github__search_repositories` 或 `mcp__filesystem__read_file`。使用正規表達式匹配器來針對來自特定伺服器的所有工具，或使用 `mcp__.*__write.*` 之類的模式跨伺服器進行匹配。有關完整的範例列表，請參閱參考中的匹配 MCP 工具。

下面的命令使用 `jq` 從 hook 的 JSON 輸入中提取工具名稱，並將其寫入 stderr，其中它在詳細模式（`Ctrl+O`）中顯示：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__github__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"GitHub tool called: $(jq -r '.tool_name')\" >&2"
          }
        ]
      }
    ]
  }
}
```

在工作階段結束時清理

`SessionEnd` 事件支援工作階段結束原因的匹配器。此 hook 只在 `clear` 時觸發（當您執行 `/clear` 時），而不是在正常退出時：

```
{
  "hooks": {
    "SessionEnd": [
      {
        "matcher": "clear",
        "hooks": [
          {
            "type": "command",
            "command": "rm -f /tmp/claude-scratch-*.txt"
          }
        ]
      }
    ]
  }
}
```

有關完整的匹配器語法，請參閱 Hooks 參考。

配置 hook 位置

您新增 hook 的位置決定了其範圍：

位置	範圍	可共享
`~/.claude/settings.json`	您的所有專案	否，本機到您的機器
`.claude/settings.json`	單個專案	是，可以提交到儲存庫
`.claude/settings.local.json`	單個專案	否，gitignored
受管理的原則設定	組織範圍	是，由管理員控制
Plugin `hooks/hooks.json`	啟用外掛時	是，與外掛捆綁
Skill 或代理 frontmatter	當 skill 或代理處於活動狀態時	是，在元件檔案中定義

您也可以在 Claude Code 中使用 `/hooks` 選單以互動方式新增、刪除和檢視 hooks。要一次禁用所有 hooks，請使用 `/hooks` 選單底部的切換或在設定檔中設定 `"disableAllHooks": true`。

透過 `/hooks` 選單新增的 Hooks 會立即生效。如果您在 Claude Code 執行時直接編輯設定檔，變更在您在 `/hooks` 選單中檢查它們或重新啟動工作階段之前不會生效。

基於提示的 hooks

對於需要判斷而不是確定性規則的決策，使用 `type: "prompt"` hooks。Claude Code 不執行 shell 命令，而是將您的提示和 hook 的輸入資料傳送到 Claude 模型（預設為 Haiku）以做出決策。如果您需要更多功能，可以使用 `model` 欄位指定不同的模型。

模型的唯一工作是傳回 yes/no 決策作為 JSON：

- `"ok": true`：操作繼續
- `"ok": false`：操作被阻止。模型的 `"reason"` 被反饋給 Claude，以便它可以調整。

此範例使用 `Stop` hook 詢問模型是否所有請求的任務都已完成。如果模型傳回 `"ok": false`，Claude 會繼續工作並使用 `reason` 作為其下一個指令：

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Check if all tasks are complete. If not, respond with
{\"ok\": false, \"reason\": \"what remains to be done\"}."
          }
        ]
      }
    ]
  }
}
```

有關完整的配置選項，請參閱參考中的基於提示的 hooks。

基於代理的 hooks

當驗證需要檢查檔案或執行命令時，使用 `type: "agent"` hooks。與只進行單個 LLM 呼叫的提示 hooks 不同，代理 hooks 會生成一個 subagent，它可以讀取檔案、搜尋程式碼和使用其他工具在傳回決策之前驗證條件。

代理 hooks 使用與提示 hooks 相同的 `"ok"` / `"reason"` 回應格式，但預設超時時間更長（60 秒）且最多 50 個工具使用輪次。

此範例驗證在允許 Claude 停止之前測試通過：

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

當 hook 輸入資料本身足以做出決策時使用提示 hooks。當您需要根據程式碼庫的實際狀態驗證某些內容時使用代理 hooks。

有關完整的配置選項，請參閱參考中的基於代理的 hooks。

HTTP hooks

使用 `type: "http"` hooks 將事件資料 POST 到 HTTP 端點，而不是執行 shell 命令。端點接收命令 hook 在 stdin 上接收的相同 JSON，並使用相同的 JSON 格式透過 HTTP 回應主體傳回結果。

當您希望 Web 伺服器、雲端函數或外部服務處理 hook 邏輯時，HTTP hooks 很有用：例如，一個共享的審計服務，在整個團隊中記錄工具使用事件。

此範例將每個工具使用 POST 到本機記錄服務：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/tool-use",
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}
```

端點應使用與命令 hooks 相同的輸出格式傳回 JSON 回應主體。要阻止工具呼叫，傳回 2xx 回應搭配適當的 `hookSpecificOutput` 欄位。HTTP 狀態代碼本身無法阻止操作。

標頭值支援使用 `$VAR_NAME` 或 `${VAR_NAME}` 語法的環境變數插值。只有在 `allowedEnvVars` 陣列中列出的變數才會被解析；所有其他 `$VAR` 參考保持為空。

> **Note:**
>
> HTTP hooks 必須透過直接編輯您的設定 JSON 來配置。`/hooks` 互動式選單只支援新增命令 hooks。

有關完整的配置選項和回應處理，請參閱參考中的 HTTP hooks。

限制和故障排除

限制

- 命令 hooks 只透過 stdout、stderr 和退出代碼通訊。它們無法直接觸發命令或工具呼叫。HTTP hooks 改為透過回應主體通訊。

- Hook 超時預設為 10 分鐘，可透過 `timeout` 欄位（以秒為單位）按 hook 配置。
- `PostToolUse` hooks 無法撤銷操作，因為工具已經執行。
- `PermissionRequest` hooks 在非互動模式（`-p`）中不觸發。對於自動化權限決策，使用 `PreToolUse` hooks。
- `Stop` hooks 在 Claude 完成回應時觸發，而不僅在任務完成時。它們在使用者中斷時不觸發。

Hook 未觸發

hook 已配置但從不執行。

- 執行 `/hooks` 並確認 hook 出現在正確的事件下
- 檢查匹配器模式是否與工具名稱完全相符（匹配器區分大小寫）
- 驗證您是否觸發了正確的事件類型（例如，`PreToolUse` 在工具執行前觸發，`PostToolUse` 在之後觸發）
- 如果在非互動模式（`-p`）中使用 `PermissionRequest` hooks，改用 `PreToolUse`

Hook 輸出中的錯誤

您在文字記錄中看到類似「PreToolUse hook error: ...」的訊息。

- 您的指令意外以非零代碼退出。透過管道傳輸範例 JSON 來手動測試它：

```
echo '{"tool_name":"Bash","tool_input":{"command":"ls"}}' | ./my-hook.sh
echo $?  # 檢查退出代碼
```

- 如果您看到「command not found」，使用絕對路徑或 `$CLAUDE_PROJECT_DIR` 來參考指令
- 如果您看到「jq: command not found」，安裝 `jq` 或使用 Python/Node.js 進行 JSON 解析
- 如果指令根本沒有執行，使其可執行：`chmod +x ./my-hook.sh`

`/hooks` 顯示未配置任何 hooks

您編輯了設定檔但 hooks 未出現在選單中。

- 重新啟動您的工作階段或開啟 `/hooks` 以重新載入。透過 `/hooks` 選單新增的 Hooks 會立即生效，但手動檔案編輯需要重新載入。
- 驗證您的 JSON 有效（不允許尾隨逗號和註解）
- 確認設定檔在正確的位置：`.claude/settings.json` 用於專案 hooks，`~/.claude/settings.json` 用於全域 hooks

Stop hook 永遠執行

Claude 在無限迴圈中繼續工作而不是停止。

您的 Stop hook 指令需要檢查它是否已經觸發了延續。從 JSON 輸入解析 `stop_hook_active` 欄位，如果為 `true` 則提前退出：

```
#!/bin/bash
INPUT=$(cat)
if [ "$(echo "$INPUT" | jq -r '.stop_hook_active')" = "true" ]; then
  exit 0  # 允許 Claude 停止
fi
## ... 您的 hook 邏輯的其餘部分
```

JSON 驗證失敗

Claude Code 顯示 JSON 解析錯誤，即使您的 hook 指令輸出有效的 JSON。

當 Claude Code 執行 hook 時，它會生成一個 shell，該 shell 會來源您的設定檔（`~/.zshrc` 或 `~/.bashrc`）。如果您的設定檔包含無條件的 `echo` 陳述式，該輸出會被前置到您的 hook 的 JSON：

```
Shell ready on arm64
{"decision": "block", "reason": "Not allowed"}
```

Claude Code 嘗試將其解析為 JSON 並失敗。要修復此問題，在您的 shell 設定檔中包裝 echo 陳述式，使其只在互動式 shell 中執行：

```
## 在 ~/.zshrc 或 ~/.bashrc 中
if [[ $- == *i* ]]; then
  echo "Shell ready"
fi
```

`$-` 變數包含 shell 旗標，`i` 表示互動式。Hooks 在非互動式 shell 中執行，因此 echo 被跳過。

除錯技術

使用 `Ctrl+O` 切換詳細模式以在文字記錄中查看 hook 輸出，或執行 `claude --debug` 以獲得完整的執行詳細資訊，包括哪些 hooks 相符及其退出代碼。

深入瞭解

- Hooks 參考：完整的事件架構、JSON 輸出格式、非同步 hooks 和 MCP 工具 hooks
- 安全考量：在共享或生產環境中部署 hooks 之前進行檢查
- Bash 命令驗證器範例：完整的參考實現

透過 MCP 將 Claude Code 連接到工具

> 了解如何使用 Model Context Protocol 將 Claude Code 連接到您的工具。

Claude Code 可以透過 Model Context Protocol (MCP) 連接到數百個外部工具和資料來源，這是一個開源標準，用於 AI 工具整合。MCP servers 讓 Claude Code 能夠存取您的工具、資料庫和 API。

使用 MCP 可以做什麼

連接 MCP servers 後，您可以要求 Claude Code：

- **從問題追蹤器實現功能**：“新增 JIRA 問題 ENG-4521 中描述的功能，並在 GitHub 上建立 PR。”
- **分析監控資料**：“檢查 Sentry 和 Statsig，以檢查 ENG-4521 中描述的功能使用情況。”
- **查詢資料庫**：“根據我們的 PostgreSQL 資料庫，找到 10 個使用功能 ENG-4521 的隨機使用者的電子郵件。”
- **整合設計**：“根據在 Slack 中發佈的新 Figma 設計更新我們的標準電子郵件範本”
- **自動化工作流程**：“建立 Gmail 草稿，邀請這 10 個使用者參加關於新功能的回饋會議。”

熱門 MCP servers

以下是一些您可以連接到 Claude Code 的常用 MCP servers：

> **Warning:**
>
> 使用第三方 MCP servers 需自行承擔風險 - Anthropic 尚未驗證 所有這些 servers 的正確性或安全性。 請確保您信任要安裝的 MCP servers。 使用可能會取得不受信任內容的 MCP servers 時要特別小心， 因為這些可能會使您面臨提示注入風險。

Server	Description	Command
Notion	Connect your Notion workspace to search, update, and power workflows across tools	`claude mcp add --transport http notion https://mcp.notion.com/mcp`

Server	Description	Command
Canva	Search, create, autofill, and export Canva designs	`claude mcp add --transport http canva https://mcp.canva.com/mcp`
Figma	Generate diagrams and better code from Figma context	`claude mcp add --transport http figma-remote-mcp https://mcp.figma.com/mcp`
Atlassian	Access Jira & Confluence from Claude	`claude mcp add --transport http atlassian https://mcp.atlassian.com/v1/mcp`
Linear	Manage issues, projects & team workflows in Linear	`claude mcp add --transport http linear https://mcp.linear.app/mcp`
monday.com	Manage projects, boards, and workflows in monday.com	`claude mcp add --transport http monday https://mcp.monday.com/mcp`
Intercom	Access to Intercom data for better customer insights	`claude mcp add --transport http intercom https://mcp.intercom.com/mcp`
Vercel	Analyze, debug, and manage projects and deployments	`claude mcp add --transport http vercel https://mcp.vercel.com`
Granola	The AI notepad for meetings	`claude mcp add --transport http granola https://mcp.granola.ai/mcp`

Server	Description	Command
Asana	Connect to Asana to coordinate tasks, projects, and goals	`claude mcp add --transport streamable-http asana https://mcp.asana.com/v2/mcp`
Miro	Access and create new content on Miro boards	`claude mcp add --transport http miro https://mcp.miro.com/`
Sentry	Search, query, and debug errors intelligently	`claude mcp add --transport http sentry https://mcp.sentry.dev/mcp`
Supabase	Manage databases, authentication, and storage	`claude mcp add --transport http supabase https://mcp.supabase.com/mcp`
Hugging Face	Access the Hugging Face Hub and thousands of Gradio Apps	`claude mcp add --transport http hugging-face https://huggingface.co/mcp`
Context7	Up-to-date docs for LLMs and AI code editors	`claude mcp add --transport http context7 https://mcp.context7.com/mcp`
Stripe	Payment processing and financial infrastructure tools	`claude mcp add --transport http stripe https://mcp.stripe.com`
Microsoft Learn	Search trusted Microsoft docs to power your development	`claude mcp add --transport http microsoft-learn https://learn.microsoft.com/api/mcp`

Server	Description	Command
Clay	Find prospects. Research accounts. Personalize outreach	`claude mcp add --transport http clay https://api.clay.com/v3/mcp`
Webflow	Manage Webflow CMS, pages, assets and sites	`claude mcp add --transport http webflow https://mcp.webflow.com/mcp`
Cloudflare	Build applications with compute, storage, and AI	`claude mcp add --transport http cloudflare https://bindings.mcp.cloudflare.com/mcp`
Ramp	Search, access, and analyze your Ramp financial data	`claude mcp add --transport http ramp https://ramp-mcp-remote.ramp.com/mcp`
ZoomInfo	Enrich contacts & accounts with GTM intelligence	`claude mcp add --transport http zoominfo https://mcp.zoominfo.com/mcp`
Netlify	Create, deploy, manage, and secure websites on Netlify	`claude mcp add --transport http netlify https://netlify-mcp.netlify.app/mcp`
Make	Run Make scenarios and manage your Make account	`claude mcp add --transport http make https://mcp.make.com`
GoDaddy	Search domains and check availability	`claude mcp add --transport http godaddy https://api.godaddy.com/v1/domains/mcp`

Server	Description	Command
Google Cloud BigQuery	BigQuery: Advanced analytical insights for agents	`claude mcp add --transport http bigquery https://bigquery.googleapis.com/mcp`
PayPal	Access PayPal payments platform	`claude mcp add --transport http paypal https://mcp.paypal.com/mcp`
PostHog	Query, analyze, and manage your PostHog insights	`claude mcp add --transport http posthog https://mcp.posthog.com/mcp`
Similarweb	Real time web, mobile app, and market data	`claude mcp add --transport http similarweb https://mcp.similarweb.com`
Crypto.com	Real time prices, orders, charts, and more for crypto	`claude mcp add --transport http crypto.com https://mcp.crypto.com/market-data/mcp`
Attio	Search, manage, and update your Attio CRM from Claude	`claude mcp add --transport http attio https://mcp.attio.com/mcp`
Trivago	Find your ideal hotel at the best price	`claude mcp add --transport http trivago https://mcp.trivago.com/mcp`
Jam	Record screen and collect automatic context for issues	`claude mcp add --transport http jam https://mcp.jam.dev/mcp`

Server	Description	Command
Consensus	Explore scientific research	`claude mcp add --transport http consensus https://mcp.consensus.app/mcp`
Clockwise	Advanced scheduling and time management for work	`claude mcp add --transport http clockwise https://mcp.getclockwise.com/mcp`
Square	Search and manage transaction, merchant, and payment data	`claude mcp add --transport sse square https://mcp.squareup.com/sse`
Egnyte	Securely access and analyze Egnyte content	`claude mcp add --transport http egnyte https://mcp-server.egnyte.com/mcp`
Pylon	Search and manage Pylon support issues	`claude mcp add --transport http pylon https://mcp.usepylon.com/`
Honeycomb	Query and explore observability data and SLOs	`claude mcp add --transport http honeycomb https://mcp.honeycomb.io/mcp`

> **Note:**
>
> **需要特定的整合？** 在 GitHub 上找到數百個更多 MCP servers，或使用 MCP SDK 建立您自己的。

安裝 MCP servers

MCP servers 可以根據您的需求以三種不同的方式進行配置：

選項 1：新增遠端 HTTP server

HTTP servers 是連接到遠端 MCP servers 的推薦選項。這是雲端服務最廣泛支援的傳輸方式。

```
## 基本語法
claude mcp add --transport http <name> <url>

## 實際範例：連接到 Notion
claude mcp add --transport http notion https://mcp.notion.com/mcp

## 使用 Bearer token 的範例
claude mcp add --transport http secure-api https://api.example.com/mcp \
  --header "Authorization: Bearer your-token"
```

選項 2：新增遠端 SSE server

> **Warning:**
>
> SSE (Server-Sent Events) 傳輸已棄用。請改用 HTTP servers（如果可用）。

```
## 基本語法
claude mcp add --transport sse <name> <url>

## 實際範例：連接到 Asana
claude mcp add --transport sse asana https://mcp.asana.com/sse

## 使用驗證標頭的範例
claude mcp add --transport sse private-api https://api.company.com/sse \
  --header "X-API-Key: your-key-here"
```

選項 3：新增本機 stdio server

Stdio servers 在您的機器上作為本機程序執行。它們非常適合需要直接系統存取或自訂指令碼的工具。

```
## 基本語法
claude mcp add [options] <name> -- <command> [args...]

## 實際範例：新增 Airtable server
claude mcp add --transport stdio --env AIRTABLE_API_KEY=YOUR_KEY airtable \
  -- npx -y airtable-mcp-server
```

Note:

重要：選項順序

所有選項（`--transport`、`--env`、`--scope`、`--header`）必須在 server 名稱**之前**。然後 `--` (雙破折號) 將 server 名稱與傳遞給 MCP server 的命令和引數分開。

例如：

- `claude mcp add --transport stdio myserver -- npx server` → 執行 `npx server`
- `claude mcp add --transport stdio --env KEY=value myserver -- python server.py --port 8080` → 執行 `python server.py --port 8080`，環境中有 `KEY=value`

這可以防止 Claude 的旗標與 server 旗標之間的衝突。

管理您的 servers

配置後，您可以使用這些命令管理您的 MCP servers：

```
## 列出所有已配置的 servers
claude mcp list

## 取得特定 server 的詳細資訊
claude mcp get github

## 移除 server
claude mcp remove github

## (在 Claude Code 中) 檢查 server 狀態
/mcp
```

動態工具更新

Claude Code 支援 MCP `list_changed` 通知，允許 MCP servers 動態更新其可用工具、提示和資源，而無需您斷開連接並重新連接。當 MCP server 傳送 `list_changed` 通知時，Claude Code 會自動重新整理該 server 的可用功能。

> **Tip:**
>
> 提示：
>
> - 使用 `--scope` 旗標指定配置的儲存位置：
> - `local` (預設)：僅在目前專案中對您可用 (在較舊版本中稱為 `project`)
> - `project`：透過 `.mcp.json` 檔案與專案中的所有人共享
> - `user`：在所有專案中對您可用 (在較舊版本中稱為 `global`)
> - 使用 `--env` 旗標設定環境變數 (例如，`--env KEY=value`)
> - 使用 MCP_TIMEOUT 環境變數配置 MCP server 啟動逾時 (例如，`MCP_TIMEOUT=10000 claude` 設定 10 秒逾時)
> - 當 MCP 工具輸出超過 10,000 個 tokens 時，Claude Code 會顯示警告。若要增加此限制，請設定 `MAX_MCP_OUTPUT_TOKENS` 環境變數 (例如，`MAX_MCP_OUTPUT_TOKENS=50000`)
> - 使用 `/mcp` 向需要 OAuth 2.0 驗證的遠端 servers 進行驗證

> **Warning:**
>
> **Windows 使用者**：在原生 Windows (不是 WSL) 上，使用 `npx` 的本機 MCP servers 需要 `cmd /c` 包裝器以確保正確執行。
>
> ```
> ## 這會建立 command="cmd"，Windows 可以執行
> claude mcp add --transport stdio my-server -- cmd /c npx -y @some/package
> ```
>
> 沒有 `cmd /c` 包裝器，您會遇到「Connection closed」錯誤，因為 Windows 無法直接執行 `npx`。(請參閱上面的注意事項，了解 `--` 參數的說明。)

Plugin 提供的 MCP servers

Plugins 可以捆綁 MCP servers，在啟用 plugin 時自動提供工具和整合。Plugin MCP servers 的工作方式與使用者配置的 servers 相同。

Plugin MCP servers 的工作方式：

- Plugins 在 plugin 根目錄的 `.mcp.json` 中或在 `plugin.json` 中內聯定義 MCP servers
- 啟用 plugin 時，其 MCP servers 會自動啟動
- Plugin MCP 工具與手動配置的 MCP 工具一起出現

- Plugin servers 透過 plugin 安裝進行管理 (不是 `/mcp` 命令)

Plugin MCP 配置範例：

在 plugin 根目錄的 `.mcp.json` 中：

```
{
  "database-tools": {
    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
    "env": {
      "DB_URL": "${DB_URL}"
    }
  }
}
```

或在 `plugin.json` 中內聯：

```
{
  "name": "my-plugin",
  "mcpServers": {
    "plugin-api": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/api-server",
      "args": ["--port", "8080"]
    }
  }
}
```

Plugin MCP 功能：

- **自動生命週期**：Servers 在 plugin 啟用時啟動，但您必須重新啟動 Claude Code 以應用 MCP server 變更 (啟用或停用)
- **環境變數**：使用 `${CLAUDE_PLUGIN_ROOT}` 表示 plugin 相對路徑
- **使用者環境存取**：存取與手動配置的 servers 相同的環境變數
- **多種傳輸類型**：支援 stdio、SSE 和 HTTP 傳輸 (傳輸支援可能因 server 而異)

檢視 plugin MCP servers：

```
## 在 Claude Code 中，查看所有 MCP servers，包括 plugin 的
/mcp
```

Plugin servers 在列表中出現，並有指示器顯示它們來自 plugins。

Plugin MCP servers 的優點：

- **捆綁分發**：工具和 servers 一起打包
- **自動設定**：無需手動 MCP 配置
- **團隊一致性**：安裝 plugin 時，每個人都會獲得相同的工具

請參閱 plugin 元件參考，了解有關使用 plugins 捆綁 MCP servers 的詳細資訊。

MCP 安裝範圍

MCP servers 可以在三個不同的範圍級別進行配置，每個級別都用於管理 server 可存取性和共享的不同目的。了解這些範圍可幫助您確定為特定需求配置 servers 的最佳方式。

本機範圍

本機範圍的 servers 代表預設配置級別，儲存在您專案路徑下的 `~/.claude.json` 中。這些 servers 對您保持私密，只有在專案目錄中工作時才可存取。此範圍非常適合個人開發 servers、實驗配置或包含不應共享的敏感認證的 servers。

> **Note:**
>
> MCP servers 的「本機範圍」術語與一般本機設定不同。MCP 本機範圍的 servers 儲存在 `~/.claude.json` (您的主目錄) 中，而一般本機設定使用 `.claude/settings.local.json` (在專案目錄中)。請參閱 Settings 了解設定檔案位置的詳細資訊。

```
## 新增本機範圍的 server（預設）
claude mcp add --transport http stripe https://mcp.stripe.com

## 明確指定本機範圍
claude mcp add --transport http stripe --scope local https://mcp.stripe.com
```

專案範圍

專案範圍的 servers 透過在專案根目錄中儲存配置在 `.mcp.json` 檔案中來啟用團隊協作。此檔案設計為簽入版本控制，確保所有團隊成員都能存取相同的 MCP 工具和服務。新增專案範圍的 server 時，Claude Code 會自動建立或更新此檔案，使用適當的配置結構。

```
## 新增專案範圍的 server
claude mcp add --transport http paypal --scope project https://mcp.paypal.com/mcp
```

產生的 `.mcp.json` 檔案遵循標準化格式：

```
{
  "mcpServers": {
    "shared-server": {
      "command": "/path/to/server",
      "args": [],
      "env": {}
    }
  }
}
```

出於安全考慮，Claude Code 在使用來自 `.mcp.json` 檔案的專案範圍 servers 之前會提示批准。如果您需要重設這些批准選擇，請使用 `claude mcp reset-project-choices` 命令。

使用者範圍

使用者範圍的 servers 儲存在 `~/.claude.json` 中，並提供跨專案可存取性，使其在您機器上的所有專案中可用，同時對您的使用者帳戶保持私密。此範圍非常適合個人公用程式 servers、開發工具或您在不同專案中經常使用的服務。

```
## 新增使用者 server
claude mcp add --transport http hubspot --scope user https://mcp.hubspot.com/
anthropic
```

選擇正確的範圍

根據以下條件選擇您的範圍：

- **本機範圍**：個人 servers、實驗配置或特定於一個專案的敏感認證
- **專案範圍**：團隊共享的 servers、專案特定的工具或協作所需的服務
- **使用者範圍**：跨多個專案所需的個人公用程式、開發工具或經常使用的服務

> **Note:**

MCP servers 儲存在哪裡？

- **使用者和本機範圍**：`~/.claude.json`（在 `mcpServers` 欄位或專案路徑下）
- **專案範圍**：專案根目錄中的 `.mcp.json`（簽入原始碼控制）
- **受管理**：系統目錄中的 `managed-mcp.json`（請參閱 受管理 MCP 配置）

範圍階層和優先順序

MCP server 配置遵循清晰的優先順序階層。當相同名稱的 servers 存在於多個範圍時，系統透過優先考慮本機範圍的 servers 來解決衝突，其次是專案範圍的 servers，最後是使用者範圍的 servers。此設計確保個人配置可以在需要時覆蓋共享配置。

`.mcp.json` 中的環境變數擴展

Claude Code 支援 `.mcp.json` 檔案中的環境變數擴展，允許團隊共享配置，同時保持機器特定路徑和 API 金鑰等敏感值的靈活性。

支援的語法：

- `${VAR}` - 擴展為環境變數 `VAR` 的值
- `${VAR:-default}` - 如果設定了 `VAR`，則擴展為 `VAR`，否則使用 `default`

擴展位置： 環境變數可以在以下位置擴展：

- `command` - server 可執行檔路徑
- `args` - 命令列引數
- `env` - 傳遞給 server 的環境變數
- `url` - 對於 HTTP server 類型
- `headers` - 對於 HTTP server 驗證

使用變數擴展的範例：

```
{
  "mcpServers": {
    "api-server": {
      "type": "http",
      "url": "${API_BASE_URL:-https://api.example.com}/mcp",
      "headers": {
        "Authorization": "Bearer ${API_KEY}"
      }
    }
  }
}
```

如果未設定必需的環境變數且沒有預設值，Claude Code 將無法解析配置。

實用範例

{/* ### 範例：使用 Playwright 自動化瀏覽器測試

```
claude mcp add --transport stdio playwright -- npx -y @playwright/mcp@latest
```

然後編寫並執行瀏覽器測試：

```
Test if the login flow works with test@example.com
```

```
Take a screenshot of the checkout page on mobile
```

```
  Verify that the search feature returns results
  ``` */}

範例：使用 Sentry 監控錯誤

```bash  theme={null}
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

使用您的 Sentry 帳戶進行驗證：

```
/mcp
```

然後除錯生產問題：

```
過去 24 小時內最常見的錯誤是什麼？
```

```
顯示錯誤 ID abc123 的堆疊追蹤
```

```
哪個部署引入了這些新錯誤？
```

範例：連接到 GitHub 進行程式碼審查

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

如果需要，透過為 GitHub 選擇「Authenticate」進行驗證：

```
/mcp
```

然後使用 GitHub：

```
審查 PR #456 並建議改進
```

```
為我們剛發現的錯誤建立新問題
```

```
顯示所有指派給我的開放 PRs
```

範例：查詢您的 PostgreSQL 資料庫

```
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \
  --dsn "postgresql://readonly:pass@prod.db.com:5432/analytics"
```

然後自然地查詢您的資料庫：

```
本月我們的總收入是多少？
```

```
顯示 orders 表的架構
```

```
找到 90 天內未進行購買的客戶
```

使用遠端 MCP servers 進行驗證

許多雲端 MCP servers 需要驗證。Claude Code 支援 OAuth 2.0 以進行安全連接。

Step 1: 新增需要驗證的 server

例如：

```
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Step 2: 在 Claude Code 中使用 /mcp 命令

在 Claude Code 中，使用命令：

```
/mcp
```

然後按照瀏覽器中的步驟登入。

> **Tip:**
>
> 提示：
>
> - 驗證 tokens 安全儲存並自動重新整理
> - 使用 `/mcp` 功能表中的「Clear authentication」撤銷存取權
> - 如果瀏覽器未自動開啟，請複製提供的 URL 並手動開啟
> - 如果瀏覽器重新導向在驗證後失敗並出現連接錯誤，請將瀏覽器位址列中的完整回呼 URL 貼到 Claude Code 中出現的 URL 提示中
> - OAuth 驗證適用於 HTTP servers

使用固定的 OAuth 回呼連接埠

某些 MCP servers 需要預先註冊的特定重新導向 URI。根據預設，Claude Code 為 OAuth 回呼選擇隨機可用連接埠。使用 `--callback-port` 固定連接埠，使其符合 `http://localhost:PORT/callback` 形式的預先註冊重新導向 URI。

您可以單獨使用 `--callback-port` (使用動態用戶端註冊) 或與 `--client-id` 一起使用 (使用預先配置的認證)。

```
## 使用動態用戶端註冊的固定回呼連接埠
claude mcp add --transport http \
  --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

使用預先配置的 OAuth 認證

某些 MCP servers 不支援自動 OAuth 設定。如果您看到類似「Incompatible auth server: does not support dynamic client registration」的錯誤，server 需要預先配置的認證。首先透過 server 的開發人員入口網站註冊 OAuth 應用程式，然後在新增 server 時提供認證。

Step 1: 使用 server 註冊 OAuth 應用程式

透過 server 的開發人員入口網站建立應用程式，並記下您的用戶端 ID 和用戶端密碼。

許多 servers 也需要重新導向 URI。如果是這樣，請選擇一個連接埠並以 `http://localhost:PORT/callback` 的格式註冊重新導向 URI。在下一步中使用該相同連接埠搭配 `--callback-port`。

Step 2: 使用您的認證新增 server

選擇以下方法之一。用於 `--callback-port` 的連接埠可以是任何可用的連接埠。它只需要符合您在上一步中註冊的重新導向 URI。

claude mcp add

使用 `--client-id` 傳遞您應用程式的用戶端 ID。`--client-secret` 旗標會提示輸入帶有遮罩輸入的密碼：

```
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

在 JSON 配置中包含 `oauth` 物件，並將 `--client-secret` 作為單獨的旗標傳遞：

```
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 (僅回呼連接埠)

使用 `--callback-port` 而不使用用戶端 ID 來固定連接埠，同時使用動態用戶端註冊：

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"callbackPort":8080}}'
```

CI / env var

透過環境變數設定密碼以跳過互動式提示：

```
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: 在 Claude Code 中進行驗證

在 Claude Code 中執行 `/mcp` 並按照瀏覽器登入流程。

> **Tip:**
>
> 提示：
>
> - 用戶端密碼安全地儲存在您的系統鑰匙圈 (macOS) 或認證檔案中，而不是在您的配置中
> - 如果 server 使用沒有密碼的公開 OAuth 用戶端，請僅使用 `--client-id` 而不使用 `--client-secret`
> - `--callback-port` 可以與或不與 `--client-id` 一起使用
> - 這些旗標僅適用於 HTTP 和 SSE 傳輸。它們對 stdio servers 沒有影響
> - 使用 `claude mcp get <name>` 驗證為 server 配置了 OAuth 認證

覆蓋 OAuth 中繼資料探索

如果您的 MCP server 在標準 OAuth 中繼資料端點 (`/.well-known/oauth-authorization-server`) 上返回錯誤，但公開了工作的 OIDC 端點，您可以告訴 Claude Code 直接從您指定的 URL 取得 OAuth 中繼資料，繞過標準探索鏈。

在 `.mcp.json` 中 server 配置的 `oauth` 物件中設定 `authServerMetadataUrl` ：

```
{
  "mcpServers": {
    "my-server": {
      "type": "http",
      "url": "https://mcp.example.com/mcp",
      "oauth": {
        "authServerMetadataUrl": "https://auth.example.com/.well-known/openid-
configuration"
      }
    }
  }
}
```

URL 必須使用 `https://` 。此選項需要 Claude Code v2.1.64 或更新版本。

從 JSON 配置新增 MCP servers

如果您有 MCP server 的 JSON 配置，您可以直接新增它：

Step 1: 從 JSON 新增 MCP server

```
## 基本語法
claude mcp add-json <name> '<json>'

## 範例：使用 JSON 配置新增 HTTP server
claude mcp add-json weather-api '{"type":"http","url":"https://api.weather.com/
mcp","headers":{"Authorization":"Bearer token"}}'

## 範例：使用 JSON 配置新增 stdio server
claude mcp add-json local-weather '{"type":"stdio","command":"/path/to/weather-
cli","args":["--api-key","abc123"],"env":{"CACHE_DIR":"/tmp"}}'

## 範例：使用預先配置的 OAuth 認證新增 HTTP server
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: 驗證 server 已新增

```
claude mcp get weather-api
```

> **Tip:**
>
> 提示：
>
> - 確保 JSON 在您的 shell 中正確逸出
> - JSON 必須符合 MCP server 配置架構
> - 您可以使用 `--scope user` 將 server 新增到您的使用者配置，而不是專案特定的配置

從 Claude Desktop 匯入 MCP servers

如果您已在 Claude Desktop 中配置了 MCP servers，您可以匯入它們：

Step 1: 從 Claude Desktop 匯入 servers

```
## 基本語法
claude mcp add-from-claude-desktop
```

Step 2: 選擇要匯入的 servers

執行命令後，您會看到一個互動式對話框，允許您選擇要匯入的 servers。

Step 3: 驗證 servers 已匯入

```
claude mcp list
```

> **Tip:**
>
> 提示：
>
> - 此功能僅適用於 macOS 和 Windows Subsystem for Linux (WSL)
> - 它從這些平台上的標準位置讀取 Claude Desktop 配置檔案
> - 使用 `--scope user` 旗標將 servers 新增到您的使用者配置
> - 匯入的 servers 將具有與 Claude Desktop 中相同的名稱
> - 如果已存在相同名稱的 servers，它們將獲得數字尾碼 (例如， `server_1`)

使用來自 Claude.ai 的 MCP servers

如果您已使用 Claude.ai 帳戶登入 Claude Code，您在 Claude.ai 中新增的 MCP servers 會自動在 Claude Code 中可用：

Step 1: 在 Claude.ai 中配置 MCP servers

在 claude.ai/settings/connectors 新增 servers。在 Team 和 Enterprise 計畫上，只有管理員可以新增 servers。

Step 2: 驗證 MCP server

在 Claude.ai 中完成任何必需的驗證步驟。

Step 3: 在 Claude Code 中檢視和管理 servers

在 Claude Code 中，使用命令：

```
/mcp
```

Claude.ai servers 在列表中出現，並有指示器顯示它們來自 Claude.ai。

若要在 Claude Code 中停用 claude.ai MCP servers，請將 `ENABLE_CLAUDEAI_MCP_SERVERS` 環境變數設定為 `false` ：

```
ENABLE_CLAUDEAI_MCP_SERVERS=false claude
```

使用 Claude Code 作為 MCP server

您可以使用 Claude Code 本身作為其他應用程式可以連接到的 MCP server：

```
## 啟動 Claude 作為 stdio MCP server
claude mcp serve
```

您可以透過將此配置新增到 claude_desktop_config.json 在 Claude Desktop 中使用它：

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Warning:

配置可執行檔路徑：`command` 欄位必須參考 Claude Code 可執行檔。如果 `claude` 命令不在您的系統 PATH 中，您需要指定可執行檔的完整路徑。

若要找到完整路徑：

```
which claude
```

然後在您的配置中使用完整路徑：

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "/full/path/to/claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

沒有正確的可執行檔路徑，您會遇到類似 `spawn claude ENOENT` 的錯誤。

Tip:

提示：

- server 提供對 Claude 工具 (如 View、Edit、LS 等) 的存取
- 在 Claude Desktop 中，嘗試要求 Claude 讀取目錄中的檔案、進行編輯等。
- 請注意，此 MCP server 僅將 Claude Code 的工具公開給您的 MCP 用戶端，因此您自己的用戶端負責為個別工具呼叫實現使用者確認。

MCP 輸出限制和警告

當 MCP 工具產生大型輸出時，Claude Code 可幫助管理 token 使用情況，以防止淹沒您的對話內容：

- **輸出警告閾值**：當任何 MCP 工具輸出超過 10,000 個 tokens 時，Claude Code 會顯示警告
- **可配置限制**：您可以使用 `MAX_MCP_OUTPUT_TOKENS` 環境變數調整最大允許的 MCP 輸出 tokens
- **預設限制**：預設最大值為 25,000 個 tokens

若要增加產生大型輸出的工具的限制：

```
## 為 MCP 工具輸出設定更高的限制
export MAX_MCP_OUTPUT_TOKENS=50000
claude
```

這在使用以下 MCP servers 時特別有用：

- 查詢大型資料集或資料庫
- 產生詳細報告或文件
- 處理廣泛的日誌檔案或除錯資訊

> **Warning:**
> 如果您經常遇到特定 MCP servers 的輸出警告，請考慮增加限制或配置 server 以分頁或篩選其回應。

使用 MCP 資源

MCP servers 可以公開資源，您可以使用 @ 提及來參考，類似於您參考檔案的方式。

參考 MCP 資源

Step 1: 列出可用資源

在您的提示中輸入 `@` 以查看所有連接的 MCP servers 中的可用資源。資源與檔案一起出現在自動完成功能表中。

Step 2: 參考特定資源

使用格式 `@server:protocol://resource/path` 來參考資源：

```
Can you analyze @github:issue://123 and suggest a fix?
```

```
Please review the API documentation at @docs:file://api/authentication
```

Step 3: 多個資源參考

您可以在單個提示中參考多個資源：

```
Compare @postgres:schema://users with @docs:file://database/user-model
```

> **Tip:**
> 提示：
>
> - 資源在參考時會自動取得並作為附件包含
> - 資源路徑在 @ 提及自動完成中可進行模糊搜尋

- Claude Code 在 servers 支援時自動提供列出和讀取 MCP 資源的工具
- 資源可以包含 MCP server 提供的任何類型的內容 (文字、JSON、結構化資料等)

使用 MCP Tool Search 進行擴展

當您配置了許多 MCP servers 時，工具定義可能會消耗您內容視窗的很大一部分。MCP Tool Search 透過動態按需載入工具而不是預先載入所有工具來解決此問題。

工作原理

當您的 MCP 工具描述會消耗超過 10% 的內容視窗時，Claude Code 會自動啟用 Tool Search。您可以 調整此閾值 或完全停用 tool search。觸發時：

1. MCP 工具被延遲而不是預先載入到內容中
2. Claude 使用搜尋工具在需要時探索相關的 MCP 工具
3. 只有 Claude 實際需要的工具才會載入到內容中
4. MCP 工具從您的角度繼續完全正常工作

對於 MCP server 作者

如果您正在建立 MCP server，啟用 Tool Search 時 server 指示欄位會變得更有用。Server 指示可幫助 Claude 了解何時搜尋您的工具，類似於 skills 的工作方式。

新增清晰、描述性的 server 指示，說明：

- 您的工具處理的任務類別
- Claude 應何時搜尋您的工具
- 您的 server 提供的關鍵功能

配置 tool search

Tool search 預設以自動模式執行，這意味著它僅在您的 MCP 工具定義超過內容閾值時啟動。如果您的工具很少，它們會正常載入，無需 tool search。此功能需要支援 `tool_reference` 區塊的模型：Sonnet 4 及更新版本，或 Opus 4 及更新版本。Haiku 模型不支援 tool search。

使用 `ENABLE_TOOL_SEARCH` 環境變數控制 tool search 行為：

值	行為
`auto`	當 MCP 工具超過 10% 的內容時啟動 (預設)
`auto:<N>`	在自訂閾值啟動，其中 `<N>` 是百分比 (例如，`auto:5` 表示 5%)

值	行為
`true`	始終啟用
`false`	停用，所有 MCP 工具預先載入

```
## 使用自訂 5% 閾值
ENABLE_TOOL_SEARCH=auto:5 claude

## 完全停用 tool search
ENABLE_TOOL_SEARCH=false claude
```

或在您的 settings.json `env` 欄位 中設定值。

您也可以使用 `disallowedTools` 設定特別停用 MCPSearch 工具：

```
{
  "permissions": {
    "deny": ["MCPSearch"]
  }
}
```

使用 MCP 提示作為命令

MCP servers 可以公開提示，這些提示在 Claude Code 中變成可用的命令。

執行 MCP 提示

Step 1: 探索可用提示

輸入 `/` 以查看所有可用命令，包括來自 MCP servers 的命令。MCP 提示以 `/mcp__servername__promptname` 的格式出現。

Step 2: 執行沒有引數的提示

```
/mcp__github__list_prs
```

Step 3: 執行帶有引數的提示

許多提示接受引數。在命令後以空格分隔的方式傳遞它們：

```
/mcp__github__pr_review 456
```

```
/mcp__jira__create_issue "Bug in login flow" high
```

> **Tip:**
>
> 提示：
>
> - MCP 提示從連接的 servers 動態探索
> - 引數根據提示的定義參數進行解析
> - 提示結果直接注入到對話中
> - Server 和提示名稱已標準化 (空格變成底線)

受管理的 MCP 配置

對於需要對 MCP servers 進行集中控制的組織，Claude Code 支援兩個配置選項：

1. **使用 `managed-mcp.json` 的獨佔控制**：部署一組固定的 MCP servers，使用者無法修改或擴展
2. **使用允許清單/拒絕清單的基於原則的控制**：允許使用者新增自己的 servers，但限制允許的 servers

這些選項允許 IT 管理員：

- **控制 MCP servers 員工可以存取的內容**：在整個組織中部署一組標準化的已批准 MCP servers
- **防止未授權的 MCP servers**：限制使用者新增未批准的 MCP servers
- **完全停用 MCP**：如果需要，完全移除 MCP 功能

選項 1：使用 managed-mcp.json 的獨佔控制

當您部署 `managed-mcp.json` 檔案時，它對所有 MCP servers 進行**獨佔控制**。使用者無法新增、修改或使用此檔案中定義的 MCP servers 以外的任何 MCP servers。這是希望完全控制的組織的最簡單方法。

系統管理員將配置檔案部署到系統範圍的目錄：

- macOS：`/Library/Application Support/ClaudeCode/managed-mcp.json`
- Linux 和 WSL：`/etc/claude-code/managed-mcp.json`
- Windows：`C:\Program Files\ClaudeCode\managed-mcp.json`

> **Note:**
>
> 這些是系統範圍的路徑 (不是像 `~/Library/...` 這樣的使用者主目錄)，需要管理員權限。它們設計為由 IT 管理員部署。

`managed-mcp.json` 檔案使用與標準 `.mcp.json` 檔案相同的格式：

```
{
  "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"
      }
    }
  }
}
```

選項 2：使用允許清單和拒絕清單的基於原則的控制

管理員可以允許使用者配置自己的 MCP servers，同時對允許的 servers 強制執行限制，而不是進行獨佔控制。此方法在 受管理設定檔案 中使用 `allowedMcpServers` 和 `deniedMcpServers`。

> **Note:**
>
> **在選項之間選擇**：當您想要部署一組固定的 servers 而不進行使用者自訂時，使用選項 1 (`managed-mcp.json`)。當您想要允許使用者在原則約束內新增自己的 servers 時，使用選項 2 (允許清單/拒絕清單)。

限制選項

允許清單或拒絕清單中的每個項目可以透過三種方式限制 servers：

1. **按 server 名稱**（`serverName`）：符合 server 的已配置名稱
2. **按命令**（`serverCommand`）：符合用於啟動 stdio servers 的確切命令和引數
3. **按 URL 模式**（`serverUrl`）：符合遠端 server URLs，支援萬用字元

重要：每個項目必須恰好有 `serverName`、`serverCommand` 或 `serverUrl` 之一。

配置範例

```
{
  "allowedMcpServers": [
    // 按 server 名稱允許
    { "serverName": "github" },
    { "serverName": "sentry" },

    // 按確切命令允許（對於 stdio servers）
    { "serverCommand": ["npx", "-y", "@modelcontextprotocol/server-filesystem"] },
    { "serverCommand": ["python", "/usr/local/bin/approved-server.py"] },

    // 按 URL 模式允許（對於遠端 servers）
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ],
  "deniedMcpServers": [
    // 按 server 名稱阻止
    { "serverName": "dangerous-server" },

    // 按確切命令阻止（對於 stdio servers）
    { "serverCommand": ["npx", "-y", "unapproved-package"] },

    // 按 URL 模式阻止（對於遠端 servers）
    { "serverUrl": "https://*.untrusted.com/*" }
  ]
}
```

基於命令的限制如何工作

確切符合：

- 命令陣列必須**確切**符合 - 命令和所有引數的順序正確
- 範例：`["npx", "-y", "server"]` 將**不**符合 `["npx", "server"]` 或 `["npx", "-y", "server", "--flag"]`

Stdio server 行為：

- 當允許清單包含**任何** `serverCommand` 項目時，stdio servers **必須**符合其中一個命令
- Stdio servers 在存在命令限制時無法單獨按名稱通過
- 這確保管理員可以強制執行允許執行的命令

非 stdio server 行為：

- 遠端 servers (HTTP、SSE、WebSocket) 在允許清單中存在 `serverUrl` 項目時使用基於 URL 的符合
- 如果不存在 URL 項目，遠端 servers 會回退到基於名稱的符合
- 命令限制不適用於遠端 servers

基於 URL 的限制如何工作

URL 模式使用 `*` 支援萬用字元以符合任何字元序列。這對於允許整個網域或子網域很有用。

萬用字元範例：

- `https://mcp.company.com/*` - 允許特定網域上的所有路徑
- `https://*.example.com/*` - 允許 example.com 的任何子網域
- `http://localhost:*/*` - 允許 localhost 上的任何連接埠

遠端 server 行為：

- 當允許清單包含**任何** `serverUrl` 項目時，遠端 servers **必須**符合其中一個 URL 模式
- 遠端 servers 在存在 URL 限制時無法單獨按名稱通過
- 這確保管理員可以強制執行允許的遠端端點

```
{
  "allowedMcpServers": [
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ]
}
```

結果：

- `https://mcp.company.com/api` 上的 HTTP server：✅ 允許 (符合 URL 模式)
- `https://api.internal.corp/mcp` 上的 HTTP server：✅ 允許 (符合萬用字元子網域)
- `https://external.com/mcp` 上的 HTTP server：❌ 阻止 (不符合任何 URL 模式)
- 任何命令的 Stdio server：❌ 阻止 (沒有名稱或命令項目可符合)

```
{
  "allowedMcpServers": [
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

結果：

- 使用 `["npx", "-y", "approved-package"]` 的 Stdio server：✅ 允許 (符合命令)
- 使用 `["node", "server.js"]` 的 Stdio server：❌ 阻止 (不符合命令)
- 名為「my-api」的 HTTP server：❌ 阻止 (沒有名稱項目可符合)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

結果：

- 名為「local-tool」、使用 `["npx", "-y", "approved-package"]` 的 Stdio server：✅ 允許 (符合命令)

- 名為「local-tool」、使用 `["node", "server.js"]` 的 Stdio server：❌阻止 (存在命令項目但不符合)
- 名為「github」、使用 `["node", "server.js"]` 的 Stdio server：❌阻止 (存在命令限制時 stdio servers 必須符合命令)
- 名為「github」的 HTTP server：✅ 允許 (符合名稱)
- 名為「other-api」的 HTTP server：❌阻止 (名稱不符合)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverName": "internal-tool" }
  ]
}
```

結果：

- 名為「github」、任何命令的 Stdio server：✅ 允許 (沒有命令限制)
- 名為「internal-tool」、任何命令的 Stdio server：✅ 允許 (沒有命令限制)
- 名為「github」的 HTTP server：✅ 允許 (符合名稱)
- 任何名為「other」的 server：❌阻止 (名稱不符合)

允許清單行為 (`allowedMcpServers`)

- `undefined` (預設)：無限制 - 使用者可以配置任何 MCP server
- 空陣列 `[]`：完全鎖定 - 使用者無法配置任何 MCP servers
- 項目清單：使用者只能配置符合名稱、命令或 URL 模式的 servers

拒絕清單行為 (`deniedMcpServers`)

- `undefined` (預設)：沒有 servers 被阻止
- 空陣列 `[]`：沒有 servers 被阻止
- 項目清單：指定的 servers 在所有範圍中被明確阻止

重要注意事項

- **選項 1 和選項 2 可以結合**：如果 `managed-mcp.json` 存在，它具有獨佔控制，使用者無法新增 servers。允許清單/拒絕清單仍然適用於受管理的 servers 本身。
- **拒絕清單具有絕對優先順序**：如果 server 符合拒絕清單項目 (按名稱、命令或 URL)，即使它在允許清單上也會被阻止

- 基於名稱、基於命令和基於 URL 的限制一起工作：如果 server 符合**任何**名稱項目、命令項目或 URL 模式，它就會通過 (除非被拒絕清單阻止)

> **Note:**
>
> **使用 `managed-mcp.json` 時**：使用者無法透過 `claude mcp add` 或配置檔案新增 MCP servers。`allowedMcpServers` 和 `deniedMcpServers` 設定仍然適用於篩選實際載入的受管理 servers。

使用 skills 擴展 Claude

> 建立、管理和分享 skills 以在 Claude Code 中擴展 Claude 的功能。包括自訂命令和捆綁的 skills。

Skills 擴展了 Claude 能做的事情。建立一個 `SKILL.md` 檔案，其中包含說明，Claude 就會將其新增到其工具包中。Claude 在相關時使用 skills，或者您可以使用 `/skill-name` 直接呼叫一個。

> **Note:**
>
> 對於內建命令（如 `/help` 和 `/compact`），請參閱互動模式。
>
> **自訂命令已合併到 skills 中。** `.claude/commands/deploy.md` 中的檔案和 `.claude/skills/deploy/SKILL.md` 中的 skill 都會建立 `/deploy` 並以相同方式運作。您現有的 `.claude/commands/` 檔案繼續運作。Skills 新增了可選功能：支援檔案的目錄、控制您或 Claude 是否呼叫它們的 frontmatter，以及 Claude 在相關時自動載入它們的能力。

Claude Code skills 遵循 Agent Skills 開放標準，該標準適用於多個 AI 工具。Claude Code 使用額外功能擴展了該標準，例如呼叫控制、subagent 執行和動態上下文注入。

捆綁的 skills

捆綁的 skills 隨 Claude Code 一起提供，在每個工作階段中都可用。與內建命令不同，內建命令直接執行固定邏輯，捆綁的 skills 是基於提示的：它們為 Claude 提供詳細的劇本，並讓它使用其工具來協調工作。這意味著捆綁的 skills 可以生成平行代理、讀取檔案並適應您的程式碼庫。

您以與任何其他 skill 相同的方式呼叫捆綁的 skills：輸入 `/` 後跟 skill 名稱。

- **`/simplify`**：檢查您最近更改的檔案以查找程式碼重用、品質和效率問題，然後修復它們。在實現功能或錯誤修復後執行它以清理您的工作。它並行生成三個審查代理（程式碼重用、程式碼品質、效率），彙總其發現並應用修復。傳遞可選文字以專注於特定問題：`/simplify focus on memory efficiency`。
- **`/batch <instruction>`**：在程式碼庫中並行協調大規模變更。提供變更的描述，`/batch` 研究程式碼庫，將工作分解為 5 到 30 個獨立單位，並提出計畫供您批准。批准後，它為每個單位生成一個背景代理，每個都在隔離的 git worktree 中。每個代理實現其單位、執行測試並開啟拉取請求。需要 git 存放庫。範例：`/batch migrate src/ from Solid to React`。

- `/debug [description]`：透過讀取工作階段偵錯日誌來排查您目前的 Claude Code 工作階段。可選地描述問題以專注分析。
- `/loop [interval] <prompt>`：在工作階段保持開啟時以間隔重複執行提示。Claude 解析間隔、排程循環 cron 任務並確認節奏。適用於輪詢部署、監督 PR 或定期重新執行另一個 skill。範例：`/loop 5m check if the deploy finished`。請參閱按排程執行提示。
- `/claude-api`：為您的專案語言（Python、TypeScript、Java、Go、Ruby、C#、PHP 或 cURL）載入 Claude API 參考資料，以及 Python 和 TypeScript 的 Agent SDK 參考。涵蓋工具使用、串流、批次、結構化輸出和常見陷阱。當您的程式碼匯入 `anthropic`、`@anthropic-ai/sdk` 或 `claude_agent_sdk` 時也會自動啟動。

開始使用

建立您的第一個 skill

此範例建立一個 skill，教導 Claude 使用視覺圖表和類比來解釋程式碼。由於它使用預設 frontmatter，Claude 可以在您詢問某事如何運作時自動載入它，或者您可以使用 `/explain-code` 直接呼叫它。

Step 1: 建立 skill 目錄

在您的個人 skills 資料夾中為 skill 建立一個目錄。個人 skills 在您的所有專案中都可用。

```
mkdir -p ~/.claude/skills/explain-code
```

Step 2: 編寫 SKILL.md

每個 skill 都需要一個 `SKILL.md` 檔案，包含兩部分：YAML frontmatter（在 `---` 標記之間），告訴 Claude 何時使用該 skill，以及包含 Claude 在呼叫該 skill 時遵循的說明的 markdown 內容。`name` 欄位變成 `/slash-command`，`description` 幫助 Claude 決定何時自動載入它。

建立 `~/.claude/skills/explain-code/SKILL.md`：

```
---
name: explain-code
description: Explains code with visual diagrams and analogies. Use when explaining
how code works, teaching about a codebase, or when the user asks "how does this
work?"
---

When explaining code, always include:

1. **Start with an analogy**: Compare the code to something from everyday life
2. **Draw a diagram**: Use ASCII art to show the flow, structure, or relationships
3. **Walk through the code**: Explain step-by-step what happens
4. **Highlight a gotcha**: What's a common mistake or misconception?

Keep explanations conversational. For complex concepts, use multiple analogies.
```

Step 3: 測試 skill

您可以透過兩種方式測試它：

讓 Claude 自動呼叫它，詢問與描述相符的內容：

```
How does this code work?
```

或直接使用 skill 名稱呼叫它：

```
/explain-code src/auth/login.ts
```

無論哪種方式，Claude 都應該在其解釋中包含類比和 ASCII 圖表。

Skills 的位置

您儲存 skill 的位置決定了誰可以使用它：

位置	路徑	適用於
企業	請參閱受管設定	您組織中的所有使用者
個人	`~/.claude/skills/<skill-name>/SKILL.md`	您的所有專案

位置	路徑	適用於
專案	`.claude/skills/<skill-name>/SKILL.md`	僅此專案
外掛程式	`<plugin>/skills/<skill-name>/SKILL.md`	啟用外掛程式的位置

當 skills 在各個級別共享相同名稱時，優先級較高的位置獲勝：企業 > 個人 > 專案。外掛程式 skills 使用 `plugin-name:skill-name` 命名空間，因此它們不能與其他級別衝突。如果您在 `.claude/commands/` 中有檔案，它們的運作方式相同，但如果 skill 和命令共享相同名稱，skill 優先。

從巢狀目錄自動發現

當您在子目錄中使用檔案時，Claude Code 會自動從巢狀 `.claude/skills/` 目錄中發現 skills。例如，如果您正在編輯 `packages/frontend/` 中的檔案，Claude Code 也會在 `packages/frontend/.claude/skills/` 中尋找 skills。這支援 monorepo 設定，其中套件有自己的 skills。

每個 skill 都是一個以 `SKILL.md` 作為進入點的目錄：

```
my-skill/
├── SKILL.md           # 主要說明（必需）
├── template.md        # Claude 要填入的範本
├── examples/
│   └── sample.md      # 顯示預期格式的範例輸出
└── scripts/
    └── validate.sh    # Claude 可以執行的指令碼
```

`SKILL.md` 包含主要說明並且是必需的。其他檔案是可選的，讓您建立更強大的 skills：Claude 要填入的範本、顯示預期格式的範例輸出、Claude 可以執行的指令碼或詳細的參考文件。從您的 `SKILL.md` 參考這些檔案，以便 Claude 知道它們包含什麼以及何時載入它們。有關更多詳細資訊，請參閱新增支援檔案。

> **Note:**
>
> `.claude/commands/` 中的檔案仍然有效並支援相同的 frontmatter。建議使用 Skills，因為它們支援額外功能，如支援檔案。

來自其他目錄的 skills

在透過 `--add-dir` 新增的目錄中的 `.claude/skills/` 中定義的 skills 會自動載入並由即時變更偵測拾取，因此您可以在工作階段期間編輯它們而無需重新啟動。

> **Note:**
>
> 來自 `--add-dir` 目錄的 CLAUDE.md 檔案預設不會載入。若要載入它們，請設定 `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1`。請參閱從其他目錄載入。

設定 skills

Skills 透過 `SKILL.md` 頂部的 YAML frontmatter 和隨後的 markdown 內容進行設定。

Skills 內容的類型

Skill 檔案可以包含任何說明，但思考您想如何呼叫它們有助於指導要包含的內容：

參考內容新增 Claude 應用於您目前工作的知識。慣例、模式、風格指南、領域知識。此內容內聯執行，以便 Claude 可以將其與您的對話上下文一起使用。

```
---
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
```

任務內容為 Claude 提供特定操作的逐步說明，如部署、提交或程式碼生成。這些通常是您想使用 `/skill-name` 直接呼叫的操作，而不是讓 Claude 決定何時執行它們。新增 `disable-model-invocation: true` 以防止 Claude 自動觸發它。

```
---
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
```

您的 `SKILL.md` 可以包含任何內容，但思考您想如何呼叫該 skill（由您、由 Claude 或兩者）以及您想在哪裡執行它（內聯或在 subagent 中）有助於指導要包含的內容。對於複雜的 skills，您也可以新增支援檔案以保持主要 skill 專注。

Frontmatter 參考

除了 markdown 內容外，您可以使用 `SKILL.md` 檔案頂部 `---` 標記之間的 YAML frontmatter 欄位來設定 skill 行為：

```
---
name: my-skill
description: What this skill does
disable-model-invocation: true
allowed-tools: Read, Grep
---

Your skill instructions here...
```

所有欄位都是可選的。只建議使用 `description`，以便 Claude 知道何時使用該 skill。

欄位	必需	描述
`name`	否	Skill 的顯示名稱。如果省略，使用目錄名稱。僅小寫字母、數字和連字號（最多 64 個字元）。

欄位	必需	描述
`description`	建議	Skill 的功能以及何時使用它。Claude 使用此來決定何時應用該 skill。如果省略，使用 markdown 內容的第一段。
`argument-hint`	否	自動完成期間顯示的提示，指示預期的引數。範例：`[issue-number]` 或 `[filename] [format]`。
`disable-model-invocation`	否	設定為 `true` 以防止 Claude 自動載入此 skill。用於您想使用 `/name` 手動觸發的工作流程。預設值：`false`。
`user-invocable`	否	設定為 `false` 以從 `/` 功能表中隱藏。用於使用者不應直接呼叫的背景知識。預設值：`true`。
`allowed-tools`	否	當此 skill 處於活動狀態時，Claude 可以使用而無需詢問許可的工具。
`model`	否	當此 skill 處於活動狀態時要使用的模型。
`context`	否	設定為 `fork` 以在分叉的 subagent 上下文中執行。
`agent`	否	當設定 `context: fork` 時要使用的 subagent 類型。
`hooks`	否	限定於此 skill 生命週期的 hooks。有關設定格式，請參閱 Skills 和代理中的 Hooks。

可用的字串替換

Skills 支援 skill 內容中動態值的字串替換：

變數	描述
`$ARGUMENTS`	呼叫 skill 時傳遞的所有引數。如果內容中不存在 `$ARGUMENTS`，引數會附加為 `ARGUMENTS: <value>`。
`$ARGUMENTS[N]`	按 0 為基礎的索引存取特定引數，例如 `$ARGUMENTS[0]` 表示第一個引數。
`$N`	`$ARGUMENTS[N]` 的簡寫，例如 `$0` 表示第一個引數或 `$1` 表示第二個引數。
`${CLAUDE_SESSION_ID}`	目前的工作階段 ID。適用於記錄、建立工作階段特定檔案或將 skill 輸出與工作階段相關聯。
`${CLAUDE_SKILL_DIR}`	包含 skill 的 `SKILL.md` 檔案的目錄。對於外掛程式 skills，這是外掛程式中 skill 的子目錄，而不是外掛程式根目錄。在 bash 注入命令中使用此項以參考與 skill 捆綁的指令碼或檔案，無論目前的工作目錄如何。

使用替換的範例：

```
---
name: session-logger
description: Log activity for this session
---

Log the following to logs/${CLAUDE_SESSION_ID}.log:

$ARGUMENTS
```

新增支援檔案

Skills 可以在其目錄中包含多個檔案。這使 `SKILL.md` 專注於基本要素，同時讓 Claude 僅在需要時存取詳細的參考資料。大型參考文件、API 規格或範例集合不需要在每次 skill 執行時載入上下文。

```
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)
```

從 `SKILL.md` 參考支援檔案，以便 Claude 知道每個檔案包含什麼以及何時載入它：

```
### Additional resources

- For complete API details, see [reference.md](reference.md)
- For usage examples, see [examples.md](examples.md)
```

> **Tip:**
>
> 將 `SKILL.md` 保持在 500 行以下。將詳細的參考資料移至單獨的檔案。

控制誰呼叫 skill

預設情況下，您和 Claude 都可以呼叫任何 skill。您可以輸入 `/skill-name` 直接呼叫它，Claude 可以在與您的對話相關時自動載入它。兩個 frontmatter 欄位讓您限制此行為：

- `disable-model-invocation: true`：只有您可以呼叫該 skill。用於具有副作用或您想控制時機的工作流程，如 `/commit`、`/deploy` 或 `/send-slack-message`。您不希望 Claude 因為您的程式碼看起來準備好就決定部署。
- `user-invocable: false`：只有 Claude 可以呼叫該 skill。用於不可作為命令操作的背景知識。`legacy-system-context` skill 解釋舊系統如何運作。Claude 在相關時應該知道這一點，但 `/legacy-system-context` 對使用者來說不是有意義的操作。

此範例建立一個只有您可以觸發的部署 skill。`disable-model-invocation: true` 欄位防止 Claude 自動執行它：

```
---
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
```

以下是兩個欄位如何影響呼叫和上下文載入：

Frontmatter	您可以呼叫	Claude 可以呼叫	何時載入上下文
（預設）	是	是	描述始終在上下文中，呼叫時載入完整 skill
`disable-model-invocation: true`	是	否	描述不在上下文中，您呼叫時載入完整 skill
`user-invocable: false`	否	是	描述始終在上下文中，呼叫時載入完整 skill

> **Note:**
>
> 在常規工作階段中，skill 描述會載入上下文，以便 Claude 知道可用的內容，但完整 skill 內容僅在呼叫時載入。預載入 skills 的 Subagents 的運作方式不同：完整 skill 內容在啟動時注入。

限制工具存取

使用 `allowed-tools` 欄位來限制當 skill 處於活動狀態時 Claude 可以使用的工具。此 skill 建立一個唯讀模式，其中 Claude 可以探索檔案但不能修改它們：

```
---
name: safe-reader
description: Read files without making changes
allowed-tools: Read, Grep, Glob
---
```

將引數傳遞給 skills

您和 Claude 都可以在呼叫 skill 時傳遞引數。引數可透過 `$ARGUMENTS` 預留位置取得。

此 skill 按編號修復 GitHub 問題。`$ARGUMENTS` 預留位置被替換為 skill 名稱後面的任何內容：

```
---
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
```

當您執行 `/fix-issue 123` 時，Claude 會收到’按照我們的編碼標準修復 GitHub 問題 123…’

如果您使用引數呼叫 skill 但 skill 不包含 `$ARGUMENTS`，Claude Code 會將 `ARGUMENTS: <your input>` 附加到 skill 內容的末尾，以便 Claude 仍然看到您輸入的內容。

若要按位置存取個別引數，請使用 `$ARGUMENTS[N]` 或較短的 `$N`：

```
---
name: migrate-component
description: Migrate a component from one framework to another
---

Migrate the $ARGUMENTS[0] component from $ARGUMENTS[1] to $ARGUMENTS[2].
Preserve all existing behavior and tests.
```

執行 `/migrate-component SearchBar React Vue` 會將 `$ARGUMENTS[0]` 替換為 `SearchBar`、`$ARGUMENTS[1]` 替換為 `React`、`$ARGUMENTS[2]` 替換為 `Vue`。使用 `$N` 簡寫的相同 skill：

```
---
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.
```

進階模式

注入動態上下文

`!`command`` 語法在將 skill 內容傳送給 Claude 之前執行 shell 命令。命令輸出替換預留位置，因此 Claude 接收實際資料，而不是命令本身。

此 skill 透過使用 GitHub CLI 擷取即時 PR 資料來總結拉取請求。`!`gh pr diff`` 和其他命令首先執行，其輸出被插入到提示中：

```
---
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...
```

當此 skill 執行時：

1. 每個 `!` command`` 立即執行（在 Claude 看到任何內容之前）
2. 輸出替換 skill 內容中的預留位置
3. Claude 接收具有實際 PR 資料的完全呈現的提示

這是預處理，不是 Claude 執行的內容。Claude 只看到最終結果。

> **Tip:**
> 若要在 skill 中啟用擴展思考，請在您的 skill 內容中的任何位置包含單詞’ultrathink’。

在 subagent 中執行 skills

當您想要 skill 在隔離中執行時，將 `context: fork` 新增到您的 frontmatter。skill 內容變成驅動 subagent 的提示。它將無法存取您的對話歷史記錄。

> **Warning:**
> `context: fork` 僅對具有明確說明的 skills 有意義。如果您的 skill 包含’使用這些 API 慣例’之類的指南而沒有任務，subagent 會收到指南但沒有可操作的提示，並返回而沒有有意義的輸出。

Skills 和 subagents 以兩個方向協同工作：

方法	系統提示	任務	也載入
具有 `context: fork` 的 Skill	來自代理類型（`Explore`、`Plan` 等）	SKILL.md 內容	CLAUDE.md
具有 `skills` 欄位的 Subagent	Subagent 的 markdown 主體	Claude 的委派訊息	預載入的 skills + CLAUDE.md

使用 `context: fork`，您在 skill 中編寫任務並選擇代理類型來執行它。對於反向（定義使用 skills 作為參考資料的自訂 subagent），請參閱 Subagents。

範例：使用 Explore 代理的研究 skill

此 skill 在分叉的 Explore 代理中執行研究。skill 內容變成任務，代理提供針對程式碼庫探索最佳化的唯讀工具：

```
---
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
```

當此 skill 執行時：

1. 建立新的隔離上下文
2. Subagent 接收 skill 內容作為其提示（‘徹底研究 $ARGUMENTS…’）
3. `agent` 欄位決定執行環境（模型、工具和許可）
4. 結果被總結並返回到您的主要對話

`agent` 欄位指定使用哪個 subagent 設定。選項包括內建代理（`Explore`、`Plan`、`general-purpose`）或 `.claude/agents/` 中的任何自訂 subagent。如果省略，使用 `general-purpose`。

限制 Claude 的 skill 存取

預設情況下，Claude 可以呼叫任何未設定 `disable-model-invocation: true` 的 skill。定義 `allowed-tools` 的 skills 在 skill 處於活動狀態時授予 Claude 對這些工具的存取權限，無需逐次批准。您的許可設定仍然管理所有其他工具的基線批准行為。內建命令（如 `/compact` 和 `/init`）無法透過 Skill 工具取得。

控制 Claude 可以呼叫哪些 skills 的三種方式：

透過在 `/permissions` 中拒絕 Skill 工具來禁用所有 skills：

```
## Add to deny rules:
Skill
```

使用許可規則允許或拒絕特定 skills：

```
## Allow only specific skills
Skill(commit)
Skill(review-pr *)

## Deny specific skills
Skill(deploy *)
```

許可語法：`Skill(name)` 用於精確匹配，`Skill(name *)` 用於帶有任何引數的前綴匹配。

透過將 `disable-model-invocation: true` 新增到其 frontmatter 來隱藏個別 skills。這會從 Claude 的上下文中完全移除該 skill。

> **Note:**
>
> `user-invocable` 欄位僅控制功能表可見性，不控制 Skill 工具存取。使用 `disable-model-invocation: true` 來阻止程式化呼叫。

分享 skills

Skills 可以根據您的受眾以不同的範圍分發：

- **專案 skills**：將 `.claude/skills/` 提交到版本控制
- **外掛程式**：在您的外掛程式中建立 `skills/` 目錄
- **受管**：透過受管設定部署組織範圍

生成視覺輸出

Skills 可以捆綁並執行任何語言的指令碼，為 Claude 提供超越單個提示可能的功能。一個強大的模式是生成視覺輸出：在您的瀏覽器中開啟的互動式 HTML 檔案，用於探索資料、偵錯或建立報告。

此範例建立一個程式碼庫探索器：一個互動式樹狀檢視，您可以在其中展開和摺疊目錄、一目瞭然地查看檔案大小，並按顏色識別檔案類型。

建立 Skill 目錄：

```
mkdir -p ~/.claude/skills/codebase-visualizer/scripts
```

建立 `~/.claude/skills/codebase-visualizer/SKILL.md`。描述告訴 Claude 何時啟動此 Skill，說明告訴 Claude 執行捆綁的指令碼：

````
---
name: codebase-visualizer
description: Generate an interactive collapsible tree visualization of your
codebase. Use when exploring a new repo, understanding project structure, or
identifying large files.
allowed-tools: Bash(python *)
---

## Codebase Visualizer

Generate an interactive HTML tree view that shows your project's file structure
with collapsible directories.

### Usage

Run the visualization script from your project root:

```bash
python ~/.claude/skills/codebase-visualizer/scripts/visualize.py .
```text

This creates `codebase-map.html` in the current directory and opens it in your
default browser.

### What the visualization shows

- **Collapsible directories**: Click folders to expand/collapse
- **File sizes**: Displayed next to each file
- **Colors**: Different colors for different file types
- **Directory totals**: Shows aggregate size of each folder
````

建立 `~/.claude/skills/codebase-visualizer/scripts/visualize.py`。此指令碼掃描目錄樹並生成一個自包含的 HTML 檔案，包含：

- 一個**摘要側邊欄**，顯示檔案計數、目錄計數、總大小和檔案類型數量
- 一個**長條圖**，按檔案類型（按大小排名前 8）分解程式碼庫
- 一個**可摺疊樹**，您可以在其中展開和摺疊目錄，具有顏色編碼的檔案類型指示器

該指令碼需要 Python，但僅使用內建庫，因此無需安裝套件：

```
#!/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>📊 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>📁 {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()}')
```

若要測試，在任何專案中開啟 Claude Code 並詢問「視覺化此程式碼庫。」Claude 執行指令碼、生成 `codebase-map.html` 並在您的瀏覽器中開啟它。

此模式適用於任何視覺輸出：依賴關係圖、測試覆蓋率報告、API 文件或資料庫架構視覺化。捆綁的指令碼完成繁重工作，而 Claude 處理協調。

疑難排解

Skill 未觸發

如果 Claude 在預期時不使用您的 skill：

1. 檢查描述是否包含使用者會自然說出的關鍵字
2. 驗證 skill 是否出現在「有哪些 skills 可用？」中
3. 嘗試重新表述您的請求以更密切地匹配描述
4. 如果 skill 是使用者可呼叫的，請使用 `/skill-name` 直接呼叫它

Skill 觸發過於頻繁

如果 Claude 在您不想要時使用您的 skill：

1. 使描述更具體
2. 如果您只想手動呼叫，請新增 `disable-model-invocation: true`

Claude 看不到我的所有 skills

Skill 描述會載入上下文，以便 Claude 知道可用的內容。如果您有許多 skills，它們可能會超過字元預算。預算在上下文視窗的 2% 處動態縮放，回退為 16,000 個字元。執行 `/context` 以檢查有關排除的 skills 的警告。

若要覆蓋限制，請設定 `SLASH_COMMAND_TOOL_CHAR_BUDGET` 環境變數。

相關資源

- **Subagents**：將任務委派給專門的代理
- **外掛程式**：使用其他擴展功能打包和分發 skills
- **Hooks**：自動化工具事件周圍的工作流程
- **記憶**：管理 CLAUDE.md 檔案以獲得持久上下文
- **互動模式**：內建命令和快捷方式
- **許可**：控制工具和 skill 存取

建立 plugins

> 建立自訂 plugins 以使用 skills、agents、hooks 和 MCP servers 擴展 Claude Code。

Plugins 讓您使用可在專案和團隊中共享的自訂功能來擴展 Claude Code。本指南涵蓋使用 skills、agents、hooks 和 MCP servers 建立您自己的 plugins。

想要安裝現有的 plugins？請參閱探索和安裝 plugins。如需完整的技術規格，請參閱 Plugins 參考。

何時使用 plugins 與獨立配置

Claude Code 支援兩種方式來新增自訂 skills、agents 和 hooks：

方法	Skill 名稱	最適合
獨立（`.claude/` 目錄）	`/hello`	個人工作流程、專案特定的自訂、快速實驗
Plugins（包含 `.claude-plugin/plugin.json` 的目錄）	`/plugin-name:hello`	與隊友共享、分發到社群、版本化發佈、跨專案重複使用

在以下情況下使用獨立配置：

- 您正在為單一專案自訂 Claude Code
- 配置是個人的，不需要共享
- 您在將 skills 或 hooks 打包之前進行實驗
- 您想要簡短的 skill 名稱，例如 `/hello` 或 `/deploy`

在以下情況下使用 plugins：

- 您想與您的團隊或社群共享功能
- 您需要在多個專案中使用相同的 skills/agents
- 您想要版本控制和輕鬆更新您的擴展
- 您正在透過市場進行分發
- 您可以接受命名空間化的 skills，例如 `/my-plugin:hello`（命名空間可防止 plugins 之間的衝突）

> **Tip:**
>
> 在 `.claude/` 中從獨立配置開始進行快速迭代，然後在準備好共享時轉換為 plugin。

快速入門

本快速入門將引導您建立具有自訂 skill 的 plugin。您將建立一個清單（定義您的 plugin 的配置檔案）、新增一個 skill，並使用 `--plugin-dir` 旗標在本地進行測試。

先決條件

- Claude Code 已安裝並驗證
- Claude Code 版本 1.0.33 或更新版本（執行 `claude --version` 以檢查）

> **Note:**
>
> 如果您沒有看到 `/plugin` 命令，請將 Claude Code 更新到最新版本。如需升級說明，請參閱 Troubleshooting。

建立您的第一個 plugin

Step 1: 建立 plugin 目錄

每個 plugin 都位於其自己的目錄中，包含一個清單和您的 skills、agents 或 hooks。現在建立一個：

```
mkdir my-first-plugin
```

Step 2: 建立 plugin 清單

位於 `.claude-plugin/plugin.json` 的清單檔案定義您的 plugin 的身份：其名稱、描述和版本。Claude Code 使用此中繼資料在 plugin 管理器中顯示您的 plugin。

在您的 plugin 資料夾內建立 `.claude-plugin` 目錄：

```
mkdir my-first-plugin/.claude-plugin
```

然後使用此內容建立 `my-first-plugin/.claude-plugin/plugin.json`：

```
{
"name": "my-first-plugin",
"description": "A greeting plugin to learn the basics",
"version": "1.0.0",
"author": {
"name": "Your Name"
}
}
```

欄位	用途
`name`	唯一識別碼和 skill 命名空間。Skills 以此為前綴（例如 `/my-first-plugin:hello`）。
`description`	在瀏覽或安裝 plugins 時在 plugin 管理器中顯示。
`version`	使用語義版本控制追蹤發佈。
`author`	選用。有助於歸屬。

如需 `homepage`、`repository` 和 `license` 等其他欄位，請參閱完整清單架構。

Step 3: 新增 skill

Skills 位於 `skills/` 目錄中。每個 skill 是一個包含 `SKILL.md` 檔案的資料夾。資料夾名稱成為 skill 名稱，以 plugin 的命名空間為前綴（在名為 `my-first-plugin` 的 plugin 中的 `hello/` 建立 `/my-first-plugin:hello`）。

在您的 plugin 資料夾中建立一個 skill 目錄：

```
mkdir -p my-first-plugin/skills/hello
```

然後使用此內容建立 `my-first-plugin/skills/hello/SKILL.md`：

```
---
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: 測試您的 plugin

使用 `--plugin-dir` 旗標執行 Claude Code 以載入您的 plugin：

```
claude --plugin-dir ./my-first-plugin
```

Claude Code 啟動後，嘗試您的新 skill：

```
/my-first-plugin:hello
```

您將看到 Claude 以問候語回應。執行 `/help` 以查看您的 skill 列在 plugin 命名空間下。

> **Note:**
>
> **為什麼要命名空間？** Plugin skills 始終被命名空間化（例如 `/greet:hello`），以防止多個 plugins 具有相同名稱的 skills 時發生衝突。
>
> 若要變更命名空間前綴，請更新 `plugin.json` 中的 `name` 欄位。

Step 5: 新增 skill 引數

透過接受使用者輸入使您的 skill 動態化。`$ARGUMENTS` 佔位符會擷取使用者在 skill 名稱後提供的任何文字。

更新您的 `SKILL.md` 檔案：

```
---
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.
```

執行 `/reload-plugins` 以取得變更，然後嘗試使用您的名稱執行 skill：

```
/my-first-plugin:hello Alex
```

Claude 將按名稱向您問候。如需有關將引數傳遞給 skills 的更多資訊，請參閱 Skills。

您已成功建立並測試了具有這些關鍵元件的 plugin：

- **Plugin 清單**（`.claude-plugin/plugin.json`）：描述您的 plugin 的中繼資料
- **Skills 目錄**（`skills/`）：包含您的自訂 skills
- **Skill 引數**（`$ARGUMENTS`）：擷取使用者輸入以實現動態行為

> **Tip:**
>
> `--plugin-dir` 旗標對於開發和測試很有用。當您準備好與他人共享您的 plugin 時，請參閱建立和分發 plugin 市場。

Plugin 結構概述

您已建立了具有 skill 的 plugin，但 plugins 可以包含更多內容：自訂 agents、hooks、MCP servers 和 LSP servers。

> **Warning:**
>
> **常見錯誤**：不要將 `commands/`、`agents/`、`skills/` 或 `hooks/` 放在 `.claude-plugin/` 目錄內。只有 `plugin.json` 應該在 `.claude-plugin/` 內。所有其他目錄必須位於 plugin 根目錄級別。

目錄	位置	用途
`.claude-plugin/`	Plugin 根目錄	包含 `plugin.json` 清單（如果元件使用預設位置，則為選用）
`commands/`	Plugin 根目錄	作為 Markdown 檔案的 Skills
`agents/`	Plugin 根目錄	自訂 agent 定義
`skills/`	Plugin 根目錄	具有 `SKILL.md` 檔案的 Agent Skills
`hooks/`	Plugin 根目錄	`hooks.json` 中的事件處理程式
`.mcp.json`	Plugin 根目錄	MCP server 配置
`.lsp.json`	Plugin 根目錄	用於程式碼智慧的 LSP server 配置
`settings.json`	Plugin 根目錄	啟用 plugin 時應用的預設設定

> **Note:**
>
> **後續步驟**：準備好新增更多功能？跳至開發更複雜的 plugins 以新增 agents、hooks、MCP servers 和 LSP servers。如需所有 plugin 元件的完整技術規格，請參閱 Plugins 參考。

開發更複雜的 plugins

一旦您熟悉了基本 plugins，您就可以建立更複雜的擴展。

將 Skills 新增到您的 plugin

Plugins 可以包含 Agent Skills 以擴展 Claude 的功能。Skills 是模型調用的：Claude 根據任務上下文自動使用它們。

在您的 plugin 根目錄中新增 `skills/` 目錄，其中包含包含 `SKILL.md` 檔案的 Skill 資料夾：

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json
└── skills/
    └── code-review/
        └── SKILL.md
```

每個 `SKILL.md` 需要包含 `name` 和 `description` 欄位的前置事項，後面跟著說明：

```
---
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
```

安裝 plugin 後，執行 `/reload-plugins` 以載入 Skills。如需完整的 Skill 編寫指南，包括漸進式揭露和工具限制，請參閱 Agent Skills。

將 LSP servers 新增到您的 plugin

> **Tip:**
>
> 對於 TypeScript、Python 和 Rust 等常見語言，請從官方市場安裝預先建立的 LSP plugins。只有在您需要支援尚未涵蓋的語言時，才建立自訂 LSP plugins。

LSP（語言伺服器協議）plugins 為 Claude 提供即時程式碼智慧。如果您需要支援沒有官方 LSP plugin 的語言，您可以透過將 `.lsp.json` 檔案新增到您的 plugin 來建立自己的：

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

安裝您的 plugin 的使用者必須在其機器上安裝語言伺服器二進位檔。

如需完整的 LSP 配置選項，請參閱 LSP servers。

使用您的 plugin 提供預設設定

Plugins 可以在 plugin 根目錄中包含 `settings.json` 檔案，以在啟用 plugin 時應用預設配置。目前，只支援 `agent` 金鑰。

設定 `agent` 會啟動 plugin 的其中一個自訂 agents 作為主執行緒，應用其系統提示、工具限制和模型。這讓 plugin 可以在啟用時透過預設方式變更 Claude Code 的行為。

```
{
  "agent": "security-reviewer"
}
```

此範例啟動在 plugin 的 `agents/` 目錄中定義的 `security-reviewer` agent。來自 `settings.json` 的設定優先於在 `plugin.json` 中宣告的 `settings`。未知的金鑰會被無聲地忽略。

組織複雜的 plugins

對於具有許多元件的 plugins，請按功能組織您的目錄結構。如需完整的目錄配置和組織模式，請參閱 Plugin 目錄結構。

在本地測試您的 plugins

使用 `--plugin-dir` 旗標在開發期間測試 plugins。這會直接載入您的 plugin，無需安裝。

```
claude --plugin-dir ./my-plugin
```

當您對 plugin 進行變更時，執行 `/reload-plugins` 以取得更新，無需重新啟動。對 LSP server 配置的變更仍需要完整重新啟動。測試您的 plugin 元件：

- 使用 `/plugin-name:skill-name` 嘗試您的 skills
- 檢查 agents 是否出現在 `/agents` 中
- 驗證 hooks 是否按預期工作

> **Tip:**
>
> 您可以透過多次指定旗標來一次載入多個 plugins：
>
> ```
> claude --plugin-dir ./plugin-one --plugin-dir ./plugin-two
> ```

偵錯 plugin 問題

如果您的 plugin 未按預期工作：

1. **檢查結構**：確保您的目錄位於 plugin 根目錄，而不是在 `.claude-plugin/` 內
2. **個別測試元件**：分別檢查每個命令、agent 和 hook
3. **使用驗證和偵錯工具**：如需 CLI 命令和故障排除技術，請參閱偵錯和開發工具

共享您的 plugins

當您的 plugin 準備好共享時：

1. **新增文件**：包含 `README.md`，其中包含安裝和使用說明
2. **版本化您的 plugin**：在您的 `plugin.json` 中使用語義版本控制
3. **建立或使用市場**：透過 plugin 市場 進行分發以進行安裝
4. **與他人測試**：在更廣泛的分發之前讓團隊成員測試 plugin

一旦您的 plugin 在市場中，其他人可以使用探索和安裝 plugins 中的說明進行安裝。

將您的 plugin 提交到官方市場

若要將 plugin 提交到官方 Anthropic 市場，請使用其中一個應用內提交表單：

- **Claude.ai**：claude.ai/settings/plugins/submit
- **Console**：platform.claude.com/plugins/submit

> **Note:**
>
> 如需完整的技術規格、偵錯技術和分發策略，請參閱 Plugins 參考。

將現有配置轉換為 plugins

如果您已經在 `.claude/` 目錄中有 skills 或 hooks，您可以將它們轉換為 plugin，以便更輕鬆地共享和分發。

遷移步驟

Step 1: 建立 plugin 結構

建立新的 plugin 目錄：

```
mkdir -p my-plugin/.claude-plugin
```

在 `my-plugin/.claude-plugin/plugin.json` 建立清單檔案：

```
{
  "name": "my-plugin",
  "description": "Migrated from standalone configuration",
  "version": "1.0.0"
}
```

Step 2: 複製您現有的檔案

將您現有的配置複製到 plugin 目錄：

```
## 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: 遷移 hooks

如果您在設定中有 hooks，請建立一個 hooks 目錄：

```
mkdir my-plugin/hooks
```

使用您的 hooks 配置建立 `my-plugin/hooks/hooks.json`。從您的 `.claude/settings.json` 或 `settings.local.json` 複製 `hooks` 物件，因為格式相同。命令在 stdin 上接收 hook 輸入作為 JSON，因此使用 `jq` 來提取檔案路徑：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [{ "type": "command", "command":
"jq -r '.tool_input.file_path' | xargs npm run lint:fix" }]
      }
    ]
  }
}
```

Step 4: 測試您遷移的 plugin

載入您的 plugin 以驗證一切正常：

```
claude --plugin-dir ./my-plugin
```

測試每個元件：執行您的命令、檢查 agents 是否出現在 `/agents` 中，並驗證 hooks 是否正確觸發。

遷移時的變更

獨立（`.claude/`）	Plugin
僅在一個專案中可用	可以透過市場共享
`.claude/commands/` 中的檔案	`plugin-name/commands/` 中的檔案
`settings.json` 中的 Hooks	`hooks/hooks.json` 中的 Hooks
必須手動複製以共享	使用 `/plugin install` 安裝

> **Note:**
>
> 遷移後，您可以從 `.claude/` 中移除原始檔案以避免重複。載入時，plugin 版本將優先。

後續步驟

現在您已了解 Claude Code 的 plugin 系統，以下是針對不同目標的建議路徑：

對於 plugin 使用者

- 探索和安裝 plugins：瀏覽市場並安裝 plugins
- 配置團隊市場：為您的團隊設定儲存庫級別的 plugins

對於 plugin 開發人員

- 建立和分發市場：打包和共享您的 plugins
- Plugins 參考：完整的技術規格
- 深入探討特定的 plugin 元件：
  - Skills：skill 開發詳情
  - Subagents：agent 配置和功能
  - Hooks：事件處理和自動化
  - MCP：外部工具整合

Plugins 參考

> Claude Code 外掛系統的完整技術參考，包括架構、CLI 命令和元件規格。

Tip:

想要安裝外掛？請參閱 探索和安裝外掛。如需建立外掛，請參閱 Plugins。如需發佈外掛，請參閱 Plugin marketplaces。

本參考提供 Claude Code 外掛系統的完整技術規格，包括元件架構、CLI 命令和開發工具。

plugin 是一個自包含的目錄，包含擴展 Claude Code 功能的元件。Plugin 元件包括 skills、agents、hooks、MCP servers 和 LSP servers。

Plugin 元件參考

Skills

Plugins 將 skills 新增至 Claude Code，建立可由您或 Claude 叫用的 `/name` 快捷方式。

位置：plugin 根目錄中的 `skills/` 或 `commands/` 目錄

檔案格式：Skills 是包含 `SKILL.md` 的目錄；commands 是簡單的 markdown 檔案

Skill 結構：

```
skills/
├── pdf-processor/
│   ├── SKILL.md
│   ├── reference.md (optional)
│   └── scripts/ (optional)
└── code-reviewer/
    └── SKILL.md
```

整合行為：

- 安裝 plugin 時會自動探索 skills 和 commands
- Claude 可以根據任務上下文自動叫用它們

- Skills 可以在 SKILL.md 旁邊包含支援檔案

如需完整詳細資訊，請參閱 Skills。

Agents

Plugins 可以提供專門的 subagents，用於 Claude 在適當時自動叫用的特定任務。

位置：plugin 根目錄中的 `agents/` 目錄

檔案格式：描述 agent 功能的 Markdown 檔案

Agent 結構：

```
---
name: agent-name
description: What this agent specializes in and when Claude should invoke it
---

Detailed system prompt for the agent describing its role, expertise, and behavior.
```

整合點：

- Agents 出現在 `/agents` 介面中
- Claude 可以根據任務上下文自動叫用 agents
- Users 可以手動叫用 agents
- Plugin agents 與內建 Claude agents 一起運作

如需完整詳細資訊，請參閱 Subagents。

Hooks

Plugins 可以提供事件處理程式，自動回應 Claude Code 事件。

位置：plugin 根目錄中的 `hooks/hooks.json`，或在 plugin.json 中內聯

格式：具有事件匹配器和動作的 JSON 設定

Hook 設定：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format-code.sh"
          }
        ]
      }
    ]
  }
}
```

可用事件：

- `PreToolUse`：Claude 使用任何工具之前
- `PostToolUse`：Claude 成功使用任何工具之後
- `PostToolUseFailure`：Claude 工具執行失敗之後
- `PermissionRequest`：顯示權限對話框時
- `UserPromptSubmit`：使用者提交提示時
- `Notification`：Claude Code 傳送通知時
- `Stop`：Claude 嘗試停止時
- `SubagentStart`：subagent 啟動時
- `SubagentStop`：subagent 嘗試停止時
- `SessionStart`：在工作階段開始時
- `SessionEnd`：在工作階段結束時
- `TeammateIdle`：agent 團隊隊友即將閒置時
- `TaskCompleted`：任務被標記為已完成時
- `PreCompact`：對話歷史記錄被壓縮之前

Hook 類型：

- `command`：執行 shell 命令或指令碼
- `prompt`：使用 LLM 評估提示（使用 `$ARGUMENTS` 佔位符表示上下文）

- `agent`：執行具有工具的 agentic 驗證器以進行複雜驗證任務

MCP servers

Plugins 可以捆綁 Model Context Protocol (MCP) servers，將 Claude Code 與外部工具和服務連接。

位置：plugin 根目錄中的 `.mcp.json`，或在 plugin.json 中內聯

格式：標準 MCP server 設定

MCP server 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}"
    }
  }
}
```

整合行為：

- 啟用 plugin 時，Plugin MCP servers 會自動啟動
- Servers 在 Claude 的工具組中顯示為標準 MCP 工具
- Server 功能與 Claude 的現有工具無縫整合
- Plugin servers 可以獨立於使用者 MCP servers 進行設定

LSP servers

> **Tip:**

> 想要使用 LSP plugins？從官方 marketplace 安裝它們：在 `/plugin` Discover 標籤中搜尋「lsp」。本節記錄如何為官方 marketplace 未涵蓋的語言建立 LSP plugins。

Plugins 可以提供 Language Server Protocol (LSP) servers，在處理程式碼庫時為 Claude 提供即時程式碼智慧。

LSP 整合提供：

- **即時診斷**：Claude 在每次編輯後立即看到錯誤和警告
- **程式碼導航**：前往定義、尋找參考和懸停資訊
- **語言感知**：程式碼符號的類型資訊和文件

位置：plugin 根目錄中的 `.lsp.json`，或在 `plugin.json` 中內聯

格式：將語言伺服器名稱對應到其設定的 JSON 設定

`.lsp.json` 檔案格式：

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

在 `plugin.json` 中內聯：

```
{
  "name": "my-plugin",
  "lspServers": {
    "go": {
      "command": "gopls",
      "args": ["serve"],
      "extensionToLanguage": {
        ".go": "go"
      }
    }
  }
}
```

必需欄位：

欄位	描述
`command`	要執行的 LSP 二進位檔（必須在 PATH 中）
`extensionToLanguage`	將檔案副檔名對應到語言識別碼

選用欄位：

欄位	描述
`args`	LSP server 的命令列引數
`transport`	通訊傳輸：`stdio`（預設）或 `socket`
`env`	啟動 server 時要設定的環境變數
`initializationOptions`	在初始化期間傳遞給 server 的選項
`settings`	透過 `workspace/didChangeConfiguration` 傳遞的設定
`workspaceFolder`	server 的工作區資料夾路徑
`startupTimeout`	等待 server 啟動的最長時間（毫秒）
`shutdownTimeout`	等待正常關閉的最長時間（毫秒）
`restartOnCrash`	如果 server 當機，是否自動重新啟動

欄位	描述
`maxRestarts`	放棄前的最大重新啟動嘗試次數

> **Warning:**
>
> **您必須單獨安裝語言伺服器二進位檔。** LSP plugins 設定 Claude Code 如何連接到語言伺服器，但它們不包括伺服器本身。如果您在 `/plugin` Errors 標籤中看到 `Executable not found in $PATH`，請為您的語言安裝所需的二進位檔。

可用的 LSP plugins：

Plugin	語言伺服器	安裝命令
`pyright-lsp`	Pyright (Python)	`pip install pyright` 或 `npm install -g pyright`
`typescript-lsp`	TypeScript Language Server	`npm install -g typescript-language-server typescript`
`rust-lsp`	rust-analyzer	參閱 rust-analyzer 安裝

先安裝語言伺服器，然後從 marketplace 安裝 plugin。

Plugin 安裝範圍

安裝 plugin 時，您選擇一個**範圍**，決定 plugin 的可用位置和誰可以使用它：

範圍	設定檔	使用案例
`user`	`~/.claude/settings.json`	在所有專案中可用的個人 plugins（預設）
`project`	`.claude/settings.json`	透過版本控制共享的團隊 plugins
`local`	`.claude/settings.local.json`	專案特定的 plugins，gitignored
`managed`	Managed settings	受管理的 plugins（唯讀，僅更新）

Plugins 使用與其他 Claude Code 設定相同的範圍系統。如需安裝說明和範圍旗標，請參閱 安裝 plugins。如需範圍的完整說明，請參閱 Configuration scopes。

Plugin manifest 架構

`.claude-plugin/plugin.json` 檔案定義您的 plugin 的中繼資料和設定。本節記錄所有支援的欄位和選項。

manifest 是選用的。如果省略，Claude Code 會自動探索 預設位置 中的元件，並從目錄名稱衍生 plugin 名稱。當您需要提供中繼資料或自訂元件路徑時，請使用 manifest。

完整架構

```
{
  "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"
}
```

必需欄位

如果您包含 manifest，`name` 是唯一必需的欄位。

欄位	類型	描述	範例
`name`	string	唯一識別碼（kebab-case，無空格）	`"deployment-tools"`

此名稱用於命名空間元件。例如，在 UI 中，名稱為 `plugin-dev` 的 plugin 的 agent `agent-creator` 將顯示為 `plugin-dev:agent-creator`。

中繼資料欄位

欄位	類型	描述	範例
`version`	string	語義版本。如果也在 marketplace 項目中設定，`plugin.json` 優先。您只需在一個位置設定它。	`"2.1.0"`
`description`	string	plugin 用途的簡短說明	`"Deployment automation tools"`
`author`	object	作者資訊	`{"name": "Dev Team", "email": "dev@company.com"}`
`homepage`	string	文件 URL	`"https://docs.example.com"`
`repository`	string	原始程式碼 URL	`"https://github.com/user/plugin"`
`license`	string	授權識別碼	`"MIT"`、`"Apache-2.0"`
`keywords`	array	探索標籤	`["deployment", "ci-cd"]`

元件路徑欄位

欄位	類型	描述	範例
`commands`	string|array	其他命令檔案/目錄	`"./custom/cmd.md"` 或 `["./cmd1.md"]`
`agents`	string|array	其他 agent 檔案	`"./custom/agents/reviewer.md"`
`skills`	string|array	其他 skill 目錄	`"./custom/skills/"`
`hooks`	string|array|object	Hook 設定路徑或內聯設定	`"./my-extra-hooks.json"`
`mcpServers`	string|array|object	MCP 設定路徑或內聯設定	`"./my-extra-mcp-config.json"`
`outputStyles`	string|array	其他輸出樣式檔案/目錄	`"./styles/"`
`lspServers`	string|array|object	Language Server Protocol 設定，用於程式碼智慧（前往定義、尋找參考等）	`"./.lsp.json"`

路徑行為規則

重要：自訂路徑補充預設目錄 - 它們不會取代預設目錄。

- 如果 `commands/` 存在，除了自訂命令路徑外，還會載入它
- 所有路徑必須相對於 plugin 根目錄，並以 `./` 開頭
- 來自自訂路徑的命令使用相同的命名和命名空間規則
- 可以將多個路徑指定為陣列以提高靈活性

路徑範例：

```
{
  "commands": [
    "./specialized/deploy.md",
    "./utilities/batch-process.md"
  ],
  "agents": [
    "./custom-agents/reviewer.md",
    "./custom-agents/tester.md"
  ]
}
```

環境變數

`${CLAUDE_PLUGIN_ROOT}`：包含 plugin 目錄的絕對路徑。在 hooks、MCP servers 和指令碼中使用此變數，以確保無論安裝位置如何都能使用正確的路徑。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/process.sh"
          }
        ]
      }
    ]
  }
}
```

Plugin 快取和檔案解析

Plugins 可以透過以下兩種方式之一指定：

- 透過 `claude --plugin-dir`，在工作階段期間。
- 透過 marketplace，為未來的工作階段安裝。

出於安全和驗證目的，Claude Code 將 marketplace plugins 複製到使用者的本機 **plugin 快取**（`~/.claude/plugins/cache`），而不是就地使用它們。在開發參考外部檔案的 plugins 時，理解此行為很重要。

路徑遍歷限制

已安裝的 plugins 無法參考其目錄外的檔案。遍歷 plugin 根目錄外的路徑（例如 `../shared-utils`）在安裝後將無法運作，因為這些外部檔案不會複製到快取中。

使用外部依賴項

如果您的 plugin 需要存取其目錄外的檔案，您可以在 plugin 目錄中建立指向外部檔案的符號連結。在複製過程中會遵守符號連結：

```
## Inside your plugin directory
ln -s /path/to/shared-utils ./shared-utils
```

符號連結的內容將被複製到 plugin 快取中。這在維持快取系統安全優勢的同時提供了靈活性。

Plugin 目錄結構

標準 plugin 配置

完整的 plugin 遵循此結構：

```
enterprise-plugin/
├── .claude-plugin/           # Metadata directory (optional)
│   └── plugin.json             # plugin manifest
├── commands/                 # Default command location
│   ├── status.md
│   └── logs.md
├── agents/                   # Default agent location
│   ├── security-reviewer.md
│   ├── performance-tester.md
│   └── compliance-checker.md
├── skills/                   # Agent Skills
│   ├── code-reviewer/
│   │   └── SKILL.md
│   └── pdf-processor/
│       ├── SKILL.md
│       └── scripts/
├── hooks/                    # Hook configurations
│   ├── hooks.json           # Main hook config
│   └── security-hooks.json  # Additional hooks
├── settings.json            # Default settings for the plugin
├── .mcp.json                # MCP server definitions
├── .lsp.json                # LSP server configurations
├── scripts/                 # Hook and utility scripts
│   ├── security-scan.sh
│   ├── format-code.py
│   └── deploy.js
├── LICENSE                  # License file
└── CHANGELOG.md             # Version history
```

Warning:

`.claude-plugin/` 目錄包含 `plugin.json` 檔案。所有其他目錄（commands/、agents/、skills/、hooks/）必須位於 plugin 根目錄，而不是在 `.claude-plugin/` 內。

檔案位置參考

元件	預設位置	用途
Manifest	`.claude-plugin/plugin.json`	Plugin 中繼資料和設定（選用）
Commands	`commands/`	Skill Markdown 檔案（舊版；新 skills 使用 `skills/`）
Agents	`agents/`	Subagent Markdown 檔案
Skills	`skills/`	具有 `<name>/SKILL.md` 結構的 Skills
Hooks	`hooks/hooks.json`	Hook 設定
MCP servers	`.mcp.json`	MCP server 定義
LSP servers	`.lsp.json`	語言伺服器設定
Settings	`settings.json`	啟用 plugin 時套用的預設設定。目前僅支援 `agent` 設定

CLI 命令參考

Claude Code 提供 CLI 命令用於非互動式 plugin 管理，適用於指令碼和自動化。

plugin install

從可用的 marketplaces 安裝 plugin。

```
claude plugin install <plugin> [options]
```

引數：

- `<plugin>`：Plugin 名稱或 `plugin-name@marketplace-name` 用於特定 marketplace

選項：

選項	描述	預設
`-s, --scope <scope>`	安裝範圍：`user`、`project` 或 `local`	`user`
`-h, --help`	顯示命令說明	

範圍決定已安裝的 plugin 新增到哪個設定檔。例如，–scope project 寫入 `.claude/settings.json` 中的 `enabledPlugins`，使 plugin 對克隆專案存放庫的每個人都可用。

範例：

```
## Install to user scope (default)
claude plugin install formatter@my-marketplace

## Install to project scope (shared with team)
claude plugin install formatter@my-marketplace --scope project

## Install to local scope (gitignored)
claude plugin install formatter@my-marketplace --scope local
```

plugin uninstall

移除已安裝的 plugin。

```
claude plugin uninstall <plugin> [options]
```

引數：

- `<plugin>`：Plugin 名稱或 `plugin-name@marketplace-name`

選項：

選項	描述	預設
`-s, --scope <scope>`	從範圍卸載：`user`、`project` 或 `local`	`user`
`-h, --help`	顯示命令說明	

別名： `remove`、`rm`

plugin enable

啟用已停用的 plugin。

```
claude plugin enable <plugin> [options]
```

引數：

- `<plugin>`：Plugin 名稱或 `plugin-name@marketplace-name`

選項：

選項	描述	預設
`-s, --scope <scope>`	要啟用的範圍：`user`、`project` 或 `local`	`user`
`-h, --help`	顯示命令說明	

plugin disable

停用 plugin 而不卸載它。

```
claude plugin disable <plugin> [options]
```

引數：

- `<plugin>`：Plugin 名稱或 `plugin-name@marketplace-name`

選項：

選項	描述	預設
`-s, --scope <scope>`	要停用的範圍：`user`、`project` 或 `local`	`user`
`-h, --help`	顯示命令說明	

plugin update

將 plugin 更新到最新版本。

```
claude plugin update <plugin> [options]
```

引數：

- `<plugin>`：Plugin 名稱或 `plugin-name@marketplace-name`

選項：

選項	描述	預設
`-s, --scope <scope>`	要更新的範圍：`user`、`project`、`local` 或 `managed`	`user`
`-h, --help`	顯示命令說明	

偵錯和開發工具

偵錯命令

使用 `claude --debug`（或 TUI 中的 `/debug`）查看 plugin 載入詳細資訊：

這會顯示：

- 正在載入哪些 plugins
- plugin manifests 中的任何錯誤
- 命令、agent 和 hook 註冊
- MCP server 初始化

常見問題

問題	原因	解決方案
Plugin 未載入	無效的 `plugin.json`	使用 `claude plugin validate` 或 `/plugin validate` 驗證 JSON 語法
命令未出現	目錄結構錯誤	確保 `commands/` 在根目錄，而不是在 `.claude-plugin/` 中
Hooks 未觸發	指令碼不可執行	執行 `chmod +x script.sh`
MCP server 失敗	缺少 `${CLAUDE_PLUGIN_ROOT}`	對所有 plugin 路徑使用變數

問題	原因	解決方案
路徑錯誤	使用了絕對路徑	所有路徑必須是相對的，並以 `./` 開頭
LSP `Executable not found in $PATH`	未安裝語言伺服器	安裝二進位檔（例如 `npm install -g typescript-language-server typescript`）

範例錯誤訊息

Manifest 驗證錯誤：

- `Invalid JSON syntax: Unexpected token } in JSON at position 142`：檢查是否缺少逗號、多餘逗號或未引用的字串
- `Plugin has an invalid manifest file at .claude-plugin/plugin.json. Validation errors: name: Required`：缺少必需欄位
- `Plugin has a corrupt manifest file at .claude-plugin/plugin.json. JSON parse error: ...`：JSON 語法錯誤

Plugin 載入錯誤：

- `Warning: No commands found in plugin my-plugin custom directory: ./cmds. Expected .md files or SKILL.md in subdirectories.`：命令路徑存在但不包含有效的命令檔案
- `Plugin directory not found at path: ./plugins/my-plugin. Check that the marketplace entry has the correct path.`：marketplace.json 中的 `source` 路徑指向不存在的目錄
- `Plugin my-plugin has conflicting manifests: both plugin.json and marketplace entry specify components.`：移除重複的元件定義或移除 marketplace 項目中的 `strict: false`

Hook 疑難排解

Hook 指令碼未執行：

1. 檢查指令碼是否可執行：`chmod +x ./scripts/your-script.sh`
2. 驗證 shebang 行：第一行應為 `#!/bin/bash` 或 `#!/usr/bin/env bash`
3. 檢查路徑是否使用 `${CLAUDE_PLUGIN_ROOT}`：`"command": "${CLAUDE_PLUGIN_ROOT}/scripts/your-script.sh"`
4. 手動測試指令碼：`./scripts/your-script.sh`

Hook 未在預期事件上觸發：

1. 驗證事件名稱是否正確（區分大小寫）：`PostToolUse`，而不是 `postToolUse`
2. 檢查匹配器模式是否與您的工具相符：`"matcher": "Write|Edit"` 用於檔案操作
3. 確認 hook 類型有效：`command`、`prompt` 或 `agent`

MCP server 疑難排解

Server 未啟動：

1. 檢查命令是否存在且可執行
2. 驗證所有路徑是否使用 `${CLAUDE_PLUGIN_ROOT}` 變數
3. 檢查 MCP server 日誌：`claude --debug` 顯示初始化錯誤
4. 在 Claude Code 外手動測試 server

Server 工具未出現：

1. 確保 server 在 `.mcp.json` 或 `plugin.json` 中正確設定
2. 驗證 server 是否正確實現 MCP 協定
3. 檢查偵錯輸出中的連接逾時

目錄結構錯誤

症狀：Plugin 載入但元件（命令、agents、hooks）遺失。

正確結構：元件必須位於 plugin 根目錄，而不是在 `.claude-plugin/` 內。只有 `plugin.json` 屬於 `.claude-plugin/`。

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json      ← Only manifest here
├── commands/            ← At root level
├── agents/              ← At root level
└── hooks/               ← At root level
```

如果您的元件在 `.claude-plugin/` 內，請將它們移到 plugin 根目錄。

偵錯檢查清單：

1. 執行 `claude --debug` 並查找「loading plugin」訊息
2. 檢查每個元件目錄是否列在偵錯輸出中
3. 驗證檔案權限允許讀取 plugin 檔案

發佈和版本控制參考

版本管理

遵循 plugin 發行的語義版本控制：

```
{
  "name": "my-plugin",
  "version": "2.1.0"
}
```

版本格式：`MAJOR.MINOR.PATCH`

- **MAJOR**：破壞性變更（不相容的 API 變更）
- **MINOR**：新功能（向後相容的新增）
- **PATCH**：錯誤修正（向後相容的修正）

最佳實踐：

- 從 `1.0.0` 開始進行第一個穩定版本
- 在發佈變更前更新 `plugin.json` 中的版本
- 在 `CHANGELOG.md` 檔案中記錄變更
- 使用預發行版本（如 `2.0.0-beta.1`）進行測試

> **Warning:**
>
> Claude Code 使用版本來決定是否更新您的 plugin。如果您變更 plugin 的程式碼但未在 `plugin.json` 中提升版本，您的 plugin 的現有使用者將因為快取而看不到您的變更。
>
> 如果您的 plugin 在 marketplace 目錄中，您可以改為透過 `marketplace.json` 管理版本，並從 `plugin.json` 中省略 `version` 欄位。

另請參閱

- Plugins - 教學和實際使用
- Plugin marketplaces - 建立和管理 marketplaces
- Skills - Skill 開發詳細資訊
- Subagents - Agent 設定和功能
- Hooks - 事件處理和自動化

- MCP - 外部工具整合
- Settings - Plugins 的設定選項

建立並分發 plugin marketplace

> 建立並託管 plugin marketplace，以在團隊和社群中分發 Claude Code 擴充功能。

plugin marketplace 是一個目錄，可讓您將 plugin 分發給他人。Marketplace 提供集中式發現、版本追蹤、自動更新，以及對多種來源類型（git 儲存庫、本機路徑等）的支援。本指南將向您展示如何建立自己的 marketplace，以與您的團隊或社群分享 plugin。

想要從現有 marketplace 安裝 plugin？請參閱探索並安裝預先建立的 plugin。

概述

建立並分發 marketplace 涉及：

1. **建立 plugin**：使用命令、agent、hook、MCP server 或 LSP server 建立一個或多個 plugin。本指南假設您已經有要分發的 plugin；有關如何建立 plugin 的詳細資訊，請參閱建立 plugin。
2. **建立 marketplace 檔案**：定義 `marketplace.json`，列出您的 plugin 及其位置（請參閱建立 marketplace 檔案）。
3. **託管 marketplace**：推送到 GitHub、GitLab 或其他 git 主機（請參閱託管並分發 marketplace）。
4. **與使用者分享**：使用者使用 `/plugin marketplace add` 新增您的 marketplace 並安裝個別 plugin（請參閱探索並安裝 plugin）。

一旦您的 marketplace 上線，您可以透過推送變更到您的儲存庫來更新它。使用者使用 `/plugin marketplace update` 重新整理其本機副本。

逐步解說：建立本機 marketplace

此範例建立一個包含一個 plugin 的 marketplace：用於程式碼審查的 `/quality-review` skill。您將建立目錄結構、新增 skill、建立 plugin manifest 和 marketplace 目錄，然後安裝並測試它。

Step 1: 建立目錄結構

```
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: 建立 skill

建立 `SKILL.md` 檔案，定義 `/quality-review` skill 的功能。

```
---
description: 檢查程式碼中的錯誤、安全性和效能問題
disable-model-invocation: true
---

檢查我選擇的程式碼或最近的變更，查找：
- 潛在的錯誤或邊界情況
- 安全性問題
- 效能問題
- 可讀性改進

簡潔且可行動。
```

Step 3: 建立 plugin manifest

建立 `plugin.json` 檔案，描述 plugin。manifest 位於 `.claude-plugin/` 目錄中。

```
{
  "name": "quality-review-plugin",
  "description": "新增 /quality-review skill 以進行快速程式碼審查",
  "version": "1.0.0"
}
```

Step 4: 建立 marketplace 檔案

建立列出您的 plugin 的 marketplace 目錄。

```
{
  "name": "my-plugins",
  "owner": {
    "name": "Your Name"
  },
  "plugins": [
    {
      "name": "quality-review-plugin",
      "source": "./plugins/quality-review-plugin",
      "description": "新增 /quality-review skill 以進行快速程式碼審查"
    }
  ]
}
```

Step 5: 新增並安裝

新增 marketplace 並安裝 plugin。

```
/plugin marketplace add ./my-marketplace
/plugin install quality-review-plugin@my-plugins
```

Step 6: 試試看

在編輯器中選擇一些程式碼並執行您的新命令。

```
/review
```

若要深入瞭解 plugin 可以執行的操作，包括 hook、agent、MCP server 和 LSP server，請參閱 Plugin。

> **Note:**
>
> **plugin 如何安裝**：當使用者安裝 plugin 時，Claude Code 會將 plugin 目錄複製到快取位置。這表示 plugin 無法使用 `../shared-utils` 之類的路徑參考其目錄外的檔案，因為這些檔案不會被複製。
>
> 如果您需要在 plugin 之間共享檔案，請使用符號連結（在複製期間會被追蹤）。有關詳細資訊，請參閱 Plugin 快取和檔案解析。

建立 marketplace 檔案

在您的儲存庫根目錄中建立 `.claude-plugin/marketplace.json`。此檔案定義您的 marketplace 名稱、擁有者資訊以及包含其來源的 plugin 清單。

每個 plugin 項目至少需要 `name` 和 `source`（從何處取得）。有關所有可用欄位，請參閱下面的完整架構。

```
{
  "name": "company-tools",
  "owner": {
    "name": "DevTools Team",
    "email": "devtools@example.com"
  },
  "plugins": [
    {
      "name": "code-formatter",
      "source": "./plugins/formatter",
      "description": "在保存時自動格式化程式碼",
      "version": "2.1.0",
      "author": {
        "name": "DevTools Team"
      }
    },
    {
      "name": "deployment-tools",
      "source": {
        "source": "github",
        "repo": "company/deploy-plugin"
      },
      "description": "部署自動化工具"
    }
  ]
}
```

Marketplace 架構

必需欄位

欄位	類型	描述	範例
`name`	string	Marketplace 識別碼（kebab-case，無空格）。這是公開的：使用者在安裝 plugin 時會看到它（例如，`/plugin install my-tool@your-marketplace`）。	`"acme-tools"`
`owner`	object	Marketplace 維護者資訊（請參閱下面的欄位）	
`plugins`	array	可用 plugin 的清單	請參閱下面

> **Note:**
>
> **保留名稱**：以下 marketplace 名稱保留供 Anthropic 官方使用，第三方 marketplace 無法使用：`claude-code-marketplace`、`claude-code-plugins`、`claude-plugins-official`、`anthropic-marketplace`、`anthropic-plugins`、`agent-skills`、`life-sciences`。模仿官方 marketplace 的名稱（如 `official-claude-plugins` 或 `anthropic-tools-v2`）也被阻止。

擁有者欄位

欄位	類型	必需	描述
`name`	string	是	維護者或團隊的名稱
`email`	string	否	維護者的聯絡電子郵件

選用中繼資料

欄位	類型	描述
`metadata.description`	string	簡短的 marketplace 描述

欄位	類型	描述
`metadata.version`	string	Marketplace 版本
`metadata.pluginRoot`	string	前置於相對 plugin 來源路徑的基本目錄（例如，`"./plugins"` 可讓您寫入 `"source": "formatter"` 而不是 `"source": "./plugins/formatter"`）

Plugin 項目

`plugins` 陣列中的每個 plugin 項目描述一個 plugin 及其位置。您可以包含 plugin manifest 架構中的任何欄位（如 `description`、`version`、`author`、`commands`、`hooks` 等），加上這些 marketplace 特定欄位：`source`、`category`、`tags` 和 `strict`。

必需欄位

欄位	類型	描述
`name`	string	Plugin 識別碼（kebab-case，無空格）。這是公開的：使用者在安裝時會看到它（例如，`/plugin install my-plugin@marketplace`）。
`source`	string|object	從何處取得 plugin（請參閱下面的 Plugin 來源）

選用 plugin 欄位

標準中繼資料欄位：

欄位	類型	描述
`description`	string	簡短的 plugin 描述
`version`	string	Plugin 版本

欄位	類型	描述
`author`	object	Plugin 作者資訊（`name` 必需，`email` 選用）
`homepage`	string	Plugin 首頁或文件 URL
`repository`	string	原始碼儲存庫 URL
`license`	string	SPDX 授權識別碼（例如，MIT、Apache-2.0）
`keywords`	array	用於 plugin 發現和分類的標籤
`category`	string	Plugin 類別以供組織
`tags`	array	用於可搜尋性的標籤
`strict`	boolean	控制 `plugin.json` 是否為元件定義的權威（預設值：true）。請參閱下面的 Strict 模式。

元件配置欄位：

欄位	類型	描述
`commands`	string|array	命令檔案或目錄的自訂路徑
`agents`	string|array	agent 檔案的自訂路徑
`hooks`	string|object	自訂 hook 配置或 hook 檔案的路徑
`mcpServers`	string|object	MCP server 配置或 MCP 配置的路徑
`lspServers`	string|object	LSP server 配置或 LSP 配置的路徑

Plugin 來源

Plugin 來源告訴 Claude Code 在您的 marketplace 中列出的每個個別 plugin 從何處取得。這些在 `marketplace.json` 中每個 plugin 項目的 `source` 欄位中設定。

一旦 plugin 被複製或複製到本機，它就會被複製到本機版本化 plugin 快取中，位於 `~/.claude/plugins/cache`。

來源	類型	欄位	備註
相對路徑	`string`（例如 `"./my-plugin"`）	—	marketplace 儲存庫內的本機目錄。必須以 `./` 開頭
`github`	object	`repo`、`ref?`、`sha?`	
`url`	object	`url`（必須以 .git 結尾）、`ref?`、`sha?`	Git URL 來源
`git-subdir`	object	`url`、`path`、`ref?`、`sha?`	git 儲存庫內的子目錄。稀疏複製以最小化大型 monorepo 的頻寬
`npm`	object	`package`、`version?`、`registry?`	透過 `npm install` 安裝
`pip`	object	`package`、`version?`、`registry?`	透過 pip 安裝

Note:

Marketplace 來源與 plugin 來源：這些是控制不同事物的不同概念。

- **Marketplace 來源** — 從何處取得 `marketplace.json` 目錄本身。在使用者執行 `/plugin marketplace add` 或在 `extraKnownMarketplaces` 設定中設定。支援 `ref`（分支/標籤）但不支援 `sha`。
- **Plugin 來源** — 從何處取得 marketplace 中列出的個別 plugin。在 `marketplace.json` 內每個 plugin 項目的 `source` 欄位中設定。支援 `ref`（分支/標籤）和 `sha`（確切提交）。

例如，託管在 `acme-corp/plugin-catalog`（marketplace 來源）的 marketplace 可以列出從 `acme-corp/code-formatter`（plugin 來源）取得的 plugin。marketplace 來源和 plugin 來源指向不同的儲存庫，並獨立固定。

相對路徑

對於同一儲存庫中的 plugin：

```
{
  "name": "my-plugin",
  "source": "./plugins/my-plugin"
}
```

Note:

相對路徑僅在使用者透過 Git（GitHub、GitLab 或 git URL）新增您的 marketplace 時有效。如果使用者透過直接 URL 新增您的 marketplace 到 `marketplace.json` 檔案，相對路徑將無法正確解析。對於基於 URL 的分發，請改用 GitHub、npm 或 git URL 來源。有關詳細資訊，請參閱疑難排解。

GitHub 儲存庫

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo"
  }
}
```

您可以固定到特定分支、標籤或提交：

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

欄位	類型	描述
`repo`	string	必需。`owner/repo` 格式的 GitHub 儲存庫
`ref`	string	選用。Git 分支或標籤（預設為儲存庫預設分支）
`sha`	string	選用。完整的 40 字元 git 提交 SHA 以固定到確切版本

Git 儲存庫

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git"
  }
}
```

您可以固定到特定分支、標籤或提交：

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git",
    "ref": "main",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

欄位	類型	描述
`url`	string	必需。完整的 git 儲存庫 URL（必須以 `.git` 結尾）

欄位	類型	描述
`ref`	string	選用。Git 分支或標籤（預設為儲存庫預設分支）
`sha`	string	選用。完整的 40 字元 git 提交 SHA 以固定到確切版本

Git 子目錄

使用 `git-subdir` 指向位於 git 儲存庫子目錄內的 plugin。Claude Code 使用稀疏、部分複製來僅取得子目錄，最小化大型 monorepo 的頻寬。

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin"
  }
}
```

您可以固定到特定分支、標籤或提交：

```
{
  "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` 欄位也接受 GitHub 簡寫（`owner/repo`）或 SSH URL（`git@github.com:owner/repo.git`）。

欄位	類型	描述
`url`	string	必需。Git 儲存庫 URL、GitHub `owner/repo` 簡寫或 SSH URL
`path`	string	必需。儲存庫內包含 plugin 的子目錄路徑（例如，`"tools/claude-plugin"`）
`ref`	string	選用。Git 分支或標籤（預設為儲存庫預設分支）
`sha`	string	選用。完整的 40 字元 git 提交 SHA 以固定到確切版本

npm 套件

作為 npm 套件分發的 plugin 使用 `npm install` 安裝。這適用於公開 npm 登錄表或您的團隊託管的任何私人登錄表上的任何套件。

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin"
  }
}
```

若要固定到特定版本，請新增 `version` 欄位：

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "2.1.0"
  }
}
```

若要從私人或內部登錄表安裝，請新增 `registry` 欄位：

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "^2.0.0",
    "registry": "https://npm.example.com"
  }
}
```

欄位	類型	描述
`package`	string	必需。套件名稱或範圍套件（例如，`@org/plugin`）
`version`	string	選用。版本或版本範圍（例如，`2.1.0`、`^2.0.0`、`~1.5.0`）
`registry`	string	選用。自訂 npm 登錄表 URL。預設為系統 npm 登錄表（通常為 npmjs.org）

進階 plugin 項目

此範例顯示使用許多選用欄位的 plugin 項目，包括命令、agent、hook 和 MCP server 的自訂路徑：

```
{
  "name": "enterprise-tools",
  "source": {
    "source": "github",
    "repo": "company/enterprise-plugin"
  },
  "description": "企業工作流程自動化工具",
  "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
}
```

需要注意的關鍵事項：

- **`commands` 和 `agents`**：您可以指定多個目錄或個別檔案。路徑相對於 plugin 根目錄。
- **`${CLAUDE_PLUGIN_ROOT}`**：在 hook 和 MCP server 配置中使用此變數來參考 plugin 安裝目錄內的檔案。這是必要的，因為 plugin 在安裝時被複製到快取位置。
- **`strict: false`**：由於此設定為 false，plugin 不需要自己的 `plugin.json`。marketplace 項目定義所有內容。請參閱下面的 Strict 模式。

Strict 模式

`strict` 欄位控制 `plugin.json` 是否為元件定義（命令、agent、hook、skill、MCP server、輸出樣式）的權威。

值	行為
`true`（預設）	`plugin.json` 是權威。marketplace 項目可以用額外的元件補充它，兩個來源都會合併。
`false`	marketplace 項目是完整定義。如果 plugin 也有宣告元件的 `plugin.json`，那就是衝突，plugin 無法載入。

何時使用每種模式：

- **`strict: true`**：plugin 有自己的 `plugin.json` 並管理自己的元件。marketplace 項目可以在頂部新增額外的命令或 hook。這是預設值，適用於大多數 plugin。
- **`strict: false`**：marketplace 運營商想要完全控制。plugin 儲存庫提供原始檔案，marketplace 項目定義這些檔案中的哪些被公開為命令、agent、hook 等。當 marketplace 以不同於 plugin 作者預期的方式重組或策劃 plugin 的元件時很有用。

託管並分發 marketplace

在 GitHub 上託管（推薦）

GitHub 提供最簡單的分發方法：

1. **建立儲存庫**：為您的 marketplace 設定新儲存庫

2. **新增 marketplace 檔案**：使用您的 plugin 定義建立 `.claude-plugin/marketplace.json`
3. **與團隊分享**：使用者使用 `/plugin marketplace add owner/repo` 新增您的 marketplace

優點：內建版本控制、問題追蹤和團隊協作功能。

在其他 git 服務上託管

任何 git 託管服務都可以使用，例如 GitLab、Bitbucket 和自託管伺服器。使用者使用完整儲存庫 URL 新增：

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

私人儲存庫

Claude Code 支援從私人儲存庫安裝 plugin。對於手動安裝和更新，Claude Code 使用您現有的 git 認證助手。如果 `git clone` 在您的終端中適用於私人儲存庫，它在 Claude Code 中也適用。常見的認證助手包括用於 GitHub 的 `gh auth login`、macOS Keychain 和 `git-credential-store`。

背景自動更新在啟動時執行，不使用認證助手，因為互動式提示會阻止 Claude Code 啟動。若要為私人 marketplace 啟用自動更新，請在您的環境中設定適當的驗證令牌：

提供者	環境變數	備註
GitHub	`GITHUB_TOKEN` 或 `GH_TOKEN`	個人存取令牌或 GitHub App 令牌
GitLab	`GITLAB_TOKEN` 或 `GL_TOKEN`	個人存取令牌或專案令牌
Bitbucket	`BITBUCKET_TOKEN`	應用程式密碼或儲存庫存取令牌

在您的 shell 配置中設定令牌（例如，`.bashrc`、`.zshrc`）或在執行 Claude Code 時傳遞它：

```
export GITHUB_TOKEN=ghp_xxxxxxxxxxxxxxxxxxxx
```

> **Note:**

> 對於 CI/CD 環境，將令牌配置為秘密環境變數。GitHub Actions 自動為同一組織中的儲存庫提供 `GITHUB_TOKEN`。

在分發前在本機測試

在分享前在本機測試您的 marketplace：

```
/plugin marketplace add ./my-local-marketplace
/plugin install test-plugin@my-local-marketplace
```

有關完整的新增命令範圍（GitHub、Git URL、本機路徑、遠端 URL），請參閱新增 marketplace。

為您的團隊要求 marketplace

您可以配置您的儲存庫，以便當團隊成員信任專案資料夾時，他們會自動被提示安裝您的 marketplace。將您的 marketplace 新增到 `.claude/settings.json`：

```
{
  "extraKnownMarketplaces": {
    "company-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

您也可以指定預設應啟用哪些 plugin：

```
{
  "enabledPlugins": {
    "code-formatter@company-tools": true,
    "deployment-tools@company-tools": true
  }
}
```

有關完整的配置選項，請參閱 Plugin 設定。

受管 marketplace 限制

對於需要對 plugin 來源進行嚴格控制的組織，管理員可以使用受管設定中的 `strictKnownMarketplaces` 設定限制使用者允許新增的 plugin marketplace。

當在受管設定中配置 `strictKnownMarketplaces` 時，限制行為取決於值：

值	行為
未定義（預設）	無限制。使用者可以新增任何 marketplace
空陣列 `[]`	完全鎖定。使用者無法新增任何新 marketplace
來源清單	使用者只能新增與允許清單完全相符的 marketplace

常見配置

停用所有 marketplace 新增：

```
{
  "strictKnownMarketplaces": []
}
```

僅允許特定 marketplace：

```
{
  "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"
    }
  ]
}
```

使用主機上的正規表達式模式匹配允許來自內部 git 伺服器的所有 marketplace：

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

使用路徑上的正規表達式模式匹配允許來自特定目錄的檔案系統型 marketplace：

```
{
  "strictKnownMarketplaces": [
    {
      "source": "pathPattern",
      "pathPattern": "^/opt/approved/"
    }
  ]
}
```

使用 `".*"` 作為 `pathPattern` 以允許任何檔案系統路徑，同時仍使用 `hostPattern` 控制網路來源。

限制如何運作

限制在 plugin 安裝過程的早期進行驗證，在任何網路請求或檔案系統操作之前。這可防止未授權的 marketplace 存取嘗試。

允許清單對大多數來源類型使用精確匹配。若要允許 marketplace，所有指定的欄位必須完全相符：

- 對於 GitHub 來源：`repo` 是必需的，如果在允許清單中指定，`ref` 或 `path` 也必須相符
- 對於 URL 來源：完整 URL 必須完全相符
- 對於 `hostPattern` 來源：marketplace 主機與正規表達式模式相符
- 對於 `pathPattern` 來源：marketplace 的檔案系統路徑與正規表達式模式相符

因為 `strictKnownMarketplaces` 在受管設定中設定，個別使用者和專案配置無法覆蓋這些限制。

有關完整的配置詳細資訊，包括所有支援的來源類型和與 `extraKnownMarketplaces` 的比較，請參閱 strictKnownMarketplaces 參考。

版本解析和發行通道

Plugin 版本決定快取路徑和更新偵測。您可以在 plugin manifest（`plugin.json`）或 marketplace 項目（`marketplace.json`）中指定版本。

> **Warning:**

盡可能避免在兩個地方設定版本。plugin manifest 總是無聲地獲勝，這可能導致 marketplace 版本被忽略。對於相對路徑 plugin，在 marketplace 項目中設定版本。對於所有其他 plugin 來源，在 plugin manifest 中設定它。

設定發行通道

若要為您的 plugin 支援「穩定」和「最新」發行通道，您可以設定兩個指向同一儲存庫的不同 ref 或 SHA 的 marketplace。然後，您可以透過受管設定將兩個 marketplace 指派給不同的使用者群組。

Warning:

plugin 的 `plugin.json` 必須在每個固定的 ref 或提交處宣告不同的 `version`。如果兩個 ref 或提交具有相同的 manifest 版本，Claude Code 會將它們視為相同並跳過更新。

範例

```
{
  "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"
      }
    }
  ]
}
```

將通道指派給使用者群組

透過受管設定將每個 marketplace 指派給適當的使用者群組。例如，穩定群組接收：

```
{
  "extraKnownMarketplaces": {
    "stable-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/stable-tools"
      }
    }
  }
}
```

早期存取群組改為接收 `latest-tools`：

```
{
  "extraKnownMarketplaces": {
    "latest-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/latest-tools"
      }
    }
  }
}
```

驗證和測試

在分享前測試您的 marketplace。

驗證您的 marketplace JSON 語法：

```
claude plugin validate .
```

或從 Claude Code 內：

```
/plugin validate .
```

新增 marketplace 進行測試：

```
/plugin marketplace add ./path/to/marketplace
```

安裝測試 plugin 以驗證一切正常運作：

```
/plugin install test-plugin@marketplace-name
```

有關完整的 plugin 測試工作流程，請參閱在本機測試您的 plugin。有關技術疑難排解，請參閱 Plugin 參考。

疑難排解

Marketplace 未載入

症狀：無法新增 marketplace 或看不到其中的 plugin

解決方案：

- 驗證 marketplace URL 可存取
- 檢查 `.claude-plugin/marketplace.json` 是否存在於指定路徑
- 使用 `claude plugin validate` 或 `/plugin validate` 確保 JSON 語法有效
- 對於私人儲存庫，確認您有存取權限

Marketplace 驗證錯誤

從您的 marketplace 目錄執行 `claude plugin validate .` 或 `/plugin validate .` 以檢查問題。常見錯誤：

錯誤	原因	解決方案
`File not found: .claude-plugin/marketplace.json`	缺少 manifest	使用必需欄位建立 `.claude-plugin/marketplace.json`
`Invalid JSON syntax: Unexpected token...`	JSON 語法錯誤	檢查缺少的逗號、多餘的逗號或未引用的字串
`Duplicate plugin name "x" found in marketplace`	兩個 plugin 共享相同名稱	為每個 plugin 指定唯一的 `name` 值
`plugins[0].source: Path traversal not allowed`	來源路徑包含 `..`	使用相對於 marketplace 根目錄的路徑，不含 `..`

警告（非阻止性）：

- `Marketplace has no plugins defined`：將至少一個 plugin 新增到 `plugins` 陣列
- `No marketplace description provided`：新增 `metadata.description` 以幫助使用者瞭解您的 marketplace

Plugin 安裝失敗

症狀：Marketplace 出現但 plugin 安裝失敗

解決方案：

- 驗證 plugin 來源 URL 可存取
- 檢查 plugin 目錄是否包含必需的檔案
- 對於 GitHub 來源，確保儲存庫是公開的或您有存取權
- 透過手動複製/下載測試 plugin 來源

私人儲存庫驗證失敗

症狀：從私人儲存庫安裝 plugin 時出現驗證錯誤

解決方案：

對於手動安裝和更新：

- 驗證您已使用您的 git 提供者進行驗證（例如，為 GitHub 執行 `gh auth status`）
- 檢查您的認證助手是否正確配置：`git config --global credential.helper`
- 嘗試手動複製儲存庫以驗證您的認證有效

對於背景自動更新：

- 在您的環境中設定適當的令牌：`echo $GITHUB_TOKEN`
- 檢查令牌是否具有必需的權限（對儲存庫的讀取存取權）
- 對於 GitHub，確保令牌對私人儲存庫具有 `repo` 範圍
- 對於 GitLab，確保令牌至少具有 `read_repository` 範圍
- 驗證令牌未過期

Git 操作逾時

症狀：Plugin 安裝或 marketplace 更新失敗，出現逾時錯誤，例如「Git clone timed out after 120s」或「Git pull timed out after 120s」。

原因：Claude Code 對所有 git 操作（包括複製 plugin 儲存庫和拉取 marketplace 更新）使用 120 秒逾時。大型儲存庫或緩慢的網路連線可能超過此限制。

解決方案：使用 `CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS` 環境變數增加逾時。值以毫秒為單位：

```
export CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS=300000  # 5 分鐘
```

相對路徑 plugin 在基於 URL 的 marketplace 中失敗

症狀：透過 URL（例如 `https://example.com/marketplace.json`）新增 marketplace，但具有相對路徑來源（如 `"./plugins/my-plugin"`）的 plugin 無法安裝，出現「path not found」錯誤。

原因：基於 URL 的 marketplace 僅下載 `marketplace.json` 檔案本身。它們不從伺服器下載 plugin 檔案。marketplace 項目中的相對路徑參考未下載的遠端伺服器上的檔案。

解決方案：

- **使用外部來源**：將 plugin 項目變更為使用 GitHub、npm 或 git URL 來源，而不是相對路徑：

```
{ "name": "my-plugin", "source": { "source": "github", "repo": "owner/repo" } }
```

- **使用基於 Git 的 marketplace**：在 Git 儲存庫中託管您的 marketplace 並使用 git URL 新增它。基於 Git 的 marketplace 複製整個儲存庫，使相對路徑正常運作。

安裝後找不到檔案

症狀：Plugin 安裝但對檔案的參考失敗，特別是 plugin 目錄外的檔案

原因：Plugin 被複製到快取目錄而不是就地使用。參考 plugin 目錄外檔案的路徑（例如 `../shared-utils`）無法運作，因為這些檔案不會被複製。

解決方案：有關解決方案（包括符號連結和目錄重組），請參閱 Plugin 快取和檔案解析。

有關其他偵錯工具和常見問題，請參閱偵錯和開發工具。

另請參閱

- 探索並安裝預先建立的 plugin - 從現有 marketplace 安裝 plugin
- Plugin - 建立您自己的 plugin
- Plugin 參考 - 完整的技術規格和架構
- Plugin 設定 - Plugin 配置選項
- strictKnownMarketplaces 參考 - 受管 marketplace 限制

透過市場探索和安裝預建外掛程式

> 從市場探索和安裝外掛程式，以使用新命令、代理和功能擴展 Claude Code。

外掛程式透過技能、代理、hooks 和 MCP servers 擴展 Claude Code。外掛程式市場是幫助您探索和安裝這些擴展的目錄，無需自己構建它們。

想要建立和分發您自己的市場？請參閱建立和分發外掛程式市場。

市場如何運作

市場是他人建立和共享的外掛程式目錄。使用市場是一個兩步流程：

Step 1: 新增市場

這會向 Claude Code 註冊目錄，以便您可以瀏覽可用內容。尚未安裝任何外掛程式。

Step 2: 安裝個別外掛程式

瀏覽目錄並安裝您想要的外掛程式。

將其視為新增應用程式商店：新增商店可讓您存取瀏覽其集合，但您仍然可以選擇個別下載哪些應用程式。

官方 Anthropic 市場

官方 Anthropic 市場 (`claude-plugins-official`) 在您啟動 Claude Code 時自動可用。執行 `/plugin` 並前往 **Discover** 標籤以瀏覽可用內容。

若要從官方市場安裝外掛程式：

```
/plugin install plugin-name@claude-plugins-official
```

> **Note:**
>
> 官方市場由 Anthropic 維護。若要將外掛程式提交到官方市場，請使用其中一個應用內提交表單：
>
> - **Claude.ai**: claude.ai/settings/plugins/submit
> - **Console**: platform.claude.com/plugins/submit
>
> 若要獨立分發外掛程式，請建立您自己的市場並與使用者共享。

官方市場包括多個外掛程式類別：

程式碼智能

程式碼智能外掛程式啟用 Claude Code 的內建 LSP 工具，使 Claude 能夠跳轉到定義、尋找參考資料，並在編輯後立即查看類型錯誤。這些外掛程式配置語言伺服器協議連接，這是為 VS Code 程式碼智能提供動力的相同技術。

這些外掛程式需要在您的系統上安裝語言伺服器二進位檔。如果您已經安裝了語言伺服器，當您開啟專案時，Claude 可能會提示您安裝相應的外掛程式。

語言	外掛程式	所需的二進位檔
C/C++	`clangd-lsp`	`clangd`
C#	`csharp-lsp`	`csharp-ls`
Go	`gopls-lsp`	`gopls`
Java	`jdtls-lsp`	`jdtls`
Kotlin	`kotlin-lsp`	`kotlin-language-server`
Lua	`lua-lsp`	`lua-language-server`
PHP	`php-lsp`	`intelephense`
Python	`pyright-lsp`	`pyright-langserver`
Rust	`rust-analyzer-lsp`	`rust-analyzer`
Swift	`swift-lsp`	`sourcekit-lsp`
TypeScript	`typescript-lsp`	`typescript-language-server`

您也可以為其他語言建立您自己的 LSP 外掛程式。

> **Note:**
>
> 如果在安裝外掛程式後在 `/plugin` Errors 標籤中看到 `Executable not found in $PATH`，請從上表安裝所需的二進位檔。

Claude 從程式碼智能外掛程式獲得的功能

安裝程式碼智能外掛程式並且其語言伺服器二進位檔可用後，Claude 獲得兩項功能：

- **自動診斷**：在 Claude 進行每次檔案編輯後，語言伺服器分析變更並自動報告錯誤和警告。Claude 看到類型錯誤、遺漏的匯入和語法問題，無需執行編譯器或 linter。如果 Claude 引入錯誤，它會注意到並在同一輪中修復問題。這不需要超出安裝外掛程式的任何配置。當「找到診斷」指示器出現時，您可以按 **Ctrl+O** 來內聯查看診斷。
- **程式碼導航**：Claude 可以使用語言伺服器跳轉到定義、尋找參考資料、懸停時取得類型資訊、列出符號、尋找實現和追蹤呼叫層次結構。這些操作為 Claude 提供比基於 grep 的搜尋更精確的導航，儘管可用性可能因語言和環境而異。

如果您遇到問題，請參閱程式碼智能故障排除。

外部整合

這些外掛程式捆綁預先配置的 MCP servers，以便您可以連接 Claude 到外部服務，無需手動設定：

- **原始碼控制**：`github`、`gitlab`
- **專案管理**：`atlassian`（Jira/Confluence）、`asana`、`linear`、`notion`
- **設計**：`figma`
- **基礎設施**：`vercel`、`firebase`、`supabase`
- **通訊**：`slack`
- **監控**：`sentry`

開發工作流程

為常見開發任務新增命令和代理的外掛程式：

- **commit-commands**：Git 提交工作流程，包括提交、推送和 PR 建立
- **pr-review-toolkit**：用於審查拉取請求的專門代理
- **agent-sdk-dev**：使用 Claude Agent SDK 構建的工具
- **plugin-dev**：建立您自己的外掛程式的工具組

輸出樣式

自訂 Claude 的回應方式：

- **explanatory-output-style**：關於實現選擇的教育見解
- **learning-output-style**：用於技能建立的互動式學習模式

試試看：新增演示市場

Anthropic 也維護一個演示外掛程式市場（`claude-code-plugins`），其中包含展示外掛程式系統可能性的範例外掛程式。與官方市場不同，您需要手動新增此市場。

Step 1: 新增市場

在 Claude Code 中，為 `anthropics/claude-code` 市場執行 `plugin marketplace add` 命令：

```
/plugin marketplace add anthropics/claude-code
```

這會下載市場目錄並使其外掛程式可供您使用。

Step 2: 瀏覽可用外掛程式

執行 `/plugin` 以開啟外掛程式管理器。這會開啟一個標籤式介面，其中有四個標籤，您可以使用 **Tab** 鍵（或 **Shift+Tab** 向後）循環瀏覽：

- **Discover**：從所有市場瀏覽可用外掛程式
- **Installed**：檢視和管理已安裝的外掛程式
- **Marketplaces**：新增、移除或更新已新增的市場
- **Errors**：檢視任何外掛程式載入錯誤

前往 **Discover** 標籤以查看您剛新增的市場中的外掛程式。

Step 3: 安裝外掛程式

選擇外掛程式以檢視其詳細資訊，然後選擇安裝範圍：

- **User scope**：在所有專案中為自己安裝
- **Project scope**：為此儲存庫上的所有協作者安裝
- **Local scope**：僅在此儲存庫中為自己安裝

例如，選擇 **commit-commands**（新增 git 工作流程命令的外掛程式）並將其安裝到您的使用者範圍。

您也可以直接從命令列安裝：

```
/plugin install commit-commands@anthropics-claude-code
```

請參閱配置範圍以深入瞭解範圍。

Step 4: 使用您的新外掛程式

安裝後，外掛程式的命令立即可用。外掛程式命令由外掛程式名稱命名空間，因此 **commit-commands** 提供 `/commit-commands:commit` 之類的命令。

透過對檔案進行變更並執行以下命令來試試看：

```
/commit-commands:commit
```

這會暫存您的變更、產生提交訊息並建立提交。

每個外掛程式的工作方式不同。檢查 **Discover** 標籤中的外掛程式描述或其首頁，以瞭解它提供的命令和功能。

本指南的其餘部分涵蓋了您可以新增市場、安裝外掛程式和管理配置的所有方式。

新增市場

使用 `/plugin marketplace add` 命令從不同來源新增市場。

> **Tip:**
>
> **快捷方式**：您可以使用 `/plugin market` 代替 `/plugin marketplace`，以及 `rm` 代替 `remove`。

- **GitHub 儲存庫**：`owner/repo` 格式（例如，`anthropics/claude-code`）
- **Git URL**：任何 git 儲存庫 URL（GitLab、Bitbucket、自託管）
- **本機路徑**：目錄或 `marketplace.json` 檔案的直接路徑
- **遠端 URL**：託管 `marketplace.json` 檔案的直接 URL

從 GitHub 新增

使用 `owner/repo` 格式新增包含 `.claude-plugin/marketplace.json` 檔案的 GitHub 儲存庫，其中 `owner` 是 GitHub 使用者名稱或組織，`repo` 是儲存庫名稱。

例如，`anthropics/claude-code` 指的是由 `anthropics` 擁有的 `claude-code` 儲存庫：

```
/plugin marketplace add anthropics/claude-code
```

從其他 Git 主機新增

透過提供完整 URL 新增任何 git 儲存庫。這適用於任何 Git 主機，包括 GitLab、Bitbucket 和自託管伺服器：

使用 HTTPS：

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

使用 SSH：

```
/plugin marketplace add git@gitlab.com:company/plugins.git
```

若要新增特定分支或標籤，請在 `#` 後面附加 ref：

```
/plugin marketplace add https://gitlab.com/company/plugins.git#v1.0.0
```

從本機路徑新增

新增包含 `.claude-plugin/marketplace.json` 檔案的本機目錄：

```
/plugin marketplace add ./my-marketplace
```

您也可以新增 `marketplace.json` 檔案的直接路徑：

```
/plugin marketplace add ./path/to/marketplace.json
```

從遠端 URL 新增

透過 URL 新增遠端 `marketplace.json` 檔案：

```
/plugin marketplace add https://example.com/marketplace.json
```

> **Note:**
>
> 與基於 Git 的市場相比，基於 URL 的市場有一些限制。如果在安裝外掛程式時遇到「找不到路徑」錯誤，請參閱故障排除。

安裝外掛程式

新增市場後，您可以直接安裝外掛程式（預設安裝到使用者範圍）：

```
/plugin install plugin-name@marketplace-name
```

若要選擇不同的安裝範圍，請使用互動式 UI：執行 `/plugin`，前往 **Discover** 標籤，然後在外掛程式上按 **Enter**。您將看到以下選項：

- **User scope**（預設）：在所有專案中為自己安裝
- **Project scope**：為此儲存庫上的所有協作者安裝（新增到 `.claude/settings.json`）
- **Local scope**：僅在此儲存庫中為自己安裝（不與協作者共享）

您也可能看到具有 **managed** 範圍的外掛程式，這些是由管理員透過受管設定安裝的，無法修改。

執行 `/plugin` 並前往 **Installed** 標籤以查看按範圍分組的外掛程式。

> **Warning:**
>
> 在安裝外掛程式之前，請確保您信任它。Anthropic 不控制外掛程式中包含的 MCP servers、檔案或其他軟體，也無法驗證它們是否按預期工作。檢查每個外掛程式的首頁以獲取更多資訊。

管理已安裝的外掛程式

執行 `/plugin` 並前往 **Installed** 標籤以檢視、啟用、停用或解除安裝外掛程式。輸入以按外掛程式名稱或描述篩選清單。

您也可以使用直接命令管理外掛程式。

停用外掛程式而不解除安裝：

```
/plugin disable plugin-name@marketplace-name
```

重新啟用已停用的外掛程式：

```
/plugin enable plugin-name@marketplace-name
```

完全移除外掛程式：

```
/plugin uninstall plugin-name@marketplace-name
```

`--scope` 選項可讓您使用 CLI 命令針對特定範圍：

```
claude plugin install formatter@your-org --scope project
claude plugin uninstall formatter@your-org --scope project
```

在不重新啟動的情況下套用外掛程式變更

當您在工作階段期間安裝、啟用或停用外掛程式時，某些變更（如新命令和 hooks）會立即生效。其他變更（包括 LSP 伺服器更新）需要重新啟動。

若要在不重新啟動的情況下啟動所有待處理的外掛程式變更，請執行：

```
/reload-plugins
```

Claude Code 重新載入所有活動外掛程式並報告已載入的內容。如果新增或更新了任何 LSP 伺服器，它會讓您知道這些需要重新啟動才能生效。

管理市場

您可以透過互動式 `/plugin` 介面或使用 CLI 命令管理市場。

使用互動式介面

執行 `/plugin` 並前往 **Marketplaces** 標籤以：

- 檢視所有已新增的市場及其來源和狀態
- 新增新市場
- 更新市場清單以取得最新外掛程式
- 移除您不再需要的市場

使用 CLI 命令

您也可以使用直接命令管理市場。

列出所有已配置的市場：

```
/plugin marketplace list
```

從市場重新整理外掛程式清單：

```
/plugin marketplace update marketplace-name
```

移除市場：

```
/plugin marketplace remove marketplace-name
```

> **Warning:**
> 移除市場將解除安裝您從中安裝的任何外掛程式。

配置自動更新

Claude Code 可以在啟動時自動更新市場及其已安裝的外掛程式。為市場啟用自動更新後，Claude Code 會重新整理市場資料並將已安裝的外掛程式更新到其最新版本。如果任何外掛程式已更新，您將看到提示您執行 `/reload-plugins` 的通知。

透過 UI 為個別市場切換自動更新：

1. 執行 `/plugin` 以開啟外掛程式管理器
2. 選擇 **Marketplaces**
3. 從清單中選擇市場
4. 選擇 **Enable auto-update** 或 **Disable auto-update**

官方 Anthropic 市場預設啟用自動更新。第三方和本機開發市場預設停用自動更新。

若要完全停用 Claude Code 和所有外掛程式的所有自動更新，請設定 `DISABLE_AUTOUPDATER` 環境變數。有關詳細資訊，請參閱自動更新。

若要在停用 Claude Code 自動更新的同時保持外掛程式自動更新啟用，請設定 `FORCE_AUTOUPDATE_PLUGINS=true` 以及 `DISABLE_AUTOUPDATER`：

```
export DISABLE_AUTOUPDATER=true
export FORCE_AUTOUPDATE_PLUGINS=true
```

當您想要手動管理 Claude Code 更新但仍然接收自動外掛程式更新時，這很有用。

配置團隊市場

團隊管理員可以透過將市場配置新增到 `.claude/settings.json` 來為專案設定自動市場安裝。當團隊成員信任儲存庫資料夾時，Claude Code 會提示他們安裝這些市場和外掛程式。

將 `extraKnownMarketplaces` 新增到您的專案的 `.claude/settings.json`：

```
{
  "extraKnownMarketplaces": {
    "my-team-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

如需完整配置選項（包括 `extraKnownMarketplaces` 和 `enabledPlugins`），請參閱外掛程式設定。

安全性

外掛程式和市場是高度受信任的元件，可以使用您的使用者權限在您的機器上執行任意程式碼。僅從您信任的來源安裝外掛程式和新增市場。組織可以使用受管市場限制限制使用者可以新增的市場。

故障排除

/plugin 命令無法識別

如果您看到「未知命令」或 `/plugin` 命令未出現：

1. **檢查您的版本**：執行 `claude --version`。外掛程式需要版本 1.0.33 或更高版本。
2. **更新 Claude Code**：
   - **Homebrew**：`brew upgrade claude-code`
   - **npm**：`npm update -g @anthropic-ai/claude-code`
   - **原生安裝程式**：從設定重新執行安裝命令
3. **重新啟動 Claude Code**：更新後，重新啟動您的終端機並再次執行 `claude`。

常見問題

- **市場未載入**：驗證 URL 是否可存取以及 `.claude-plugin/marketplace.json` 是否存在於路徑中
- **外掛程式安裝失敗**：檢查外掛程式來源 URL 是否可存取以及儲存庫是否為公開（或您有存取權）

- **安裝後找不到檔案**：外掛程式被複製到快取中，因此參考外掛程式目錄外檔案的路徑將無法運作
- **外掛程式技能未出現**：使用 `rm -rf ~/.claude/plugins/cache` 清除快取，重新啟動 Claude Code，然後重新安裝外掛程式。

如需詳細的故障排除和解決方案，請參閱市場指南中的故障排除。如需偵錯工具，請參閱偵錯和開發工具。

程式碼智能問題

- **語言伺服器未啟動**：驗證二進位檔已安裝且在您的 `$PATH` 中可用。檢查 `/plugin` Errors 標籤以獲取詳細資訊。
- **高記憶體使用量**：`rust-analyzer` 和 `pyright` 等語言伺服器在大型專案上可能會消耗大量記憶體。如果您遇到記憶體問題，請使用 `/plugin disable <plugin-name>` 停用外掛程式，並改為依賴 Claude 的內建搜尋工具。
- **monorepos 中的誤報診斷**：如果工作區配置不正確，語言伺服器可能會報告內部套件的未解決匯入錯誤。這些不會影響 Claude 編輯程式碼的能力。

後續步驟

- **構建您自己的外掛程式**：請參閱外掛程式以建立技能、代理和 hooks
- **建立市場**：請參閱建立外掛程式市場以將外掛程式分發給您的團隊或社群
- **技術參考**：請參閱外掛程式參考以取得完整規格

Part 6: Advanced & Automation

建立自訂 subagents

> 在 Claude Code 中建立並使用專門的 AI subagents，用於特定任務的工作流程和改進的上下文管理。

Subagents 是專門的 AI 助手，用於處理特定類型的任務。每個 subagent 在自己的 context window 中執行，具有自訂系統提示、特定工具存取和獨立權限。當 Claude 遇到與 subagent 描述相符的任務時，它會委派給該 subagent，該 subagent 獨立工作並返回結果。

> **Note:**
>
> 如果您需要多個代理並行工作並相互通訊，請改為參閱 agent teams。Subagents 在單一工作階段內工作；agent teams 跨越多個獨立工作階段進行協調。

Subagents 可幫助您：

- **保留上下文**，將探索和實現保持在主要對話之外
- **強制執行約束**，限制 subagent 可以使用的工具
- **跨專案重複使用配置**，使用使用者層級的 subagents
- **專門化行為**，針對特定領域使用專注的系統提示
- **控制成本**，將任務路由到更快、更便宜的模型（如 Haiku）

Claude 使用每個 subagent 的描述來決定何時委派任務。建立 subagent 時，請寫清楚的描述，以便 Claude 知道何時使用它。

Claude Code 包含多個內建 subagents，例如 **Explore**、**Plan** 和 **general-purpose**。您也可以建立自訂 subagents 來處理特定任務。本頁涵蓋 內建 subagents、如何建立您自己的、完整配置選項、使用 subagents 的模式 和 範例 subagents。

內建 subagents

Claude Code 包含內建 subagents，Claude 會在適當時自動使用。每個都繼承父對話的權限，並有額外的工具限制。

Explore

一個快速、唯讀的代理，針對搜尋和分析程式碼庫進行最佳化。

- **模型**：Haiku（快速、低延遲）
- **工具**：唯讀工具（拒絕存取 Write 和 Edit 工具）
- **目的**：檔案發現、程式碼搜尋、程式碼庫探索

當 Claude 需要搜尋或理解程式碼庫而不進行更改時，它會委派給 Explore。這可以將探索結果保持在主要對話上下文之外。

叫用 Explore 時，Claude 會指定詳盡程度：**quick** 用於目標查詢、**medium** 用於平衡探索，或 **very thorough** 用於全面分析。

Plan

在 plan mode 期間用於在呈現計畫之前收集上下文的研究代理。

- **模型**：繼承自主對話
- **工具**：唯讀工具（拒絕存取 Write 和 Edit 工具）
- **目的**：用於規劃的程式碼庫研究

當您處於 plan mode 且 Claude 需要理解您的程式碼庫時，它會將研究委派給 Plan subagent。這可以防止無限嵌套（subagents 無法產生其他 subagents），同時仍然收集必要的上下文。

General-purpose

一個能夠處理複雜、多步驟任務的代理，需要探索和行動。

- **模型**：繼承自主對話
- **工具**：所有工具
- **目的**：複雜研究、多步驟操作、程式碼修改

當任務需要探索和修改、複雜推理來解釋結果，或多個相依步驟時，Claude 會委派給 general-purpose。

Other

Claude Code 包含用於特定任務的其他輔助代理。這些通常會自動叫用，因此您不需要直接使用它們。

代理	模型	Claude 何時使用它
Bash	繼承	在單獨的上下文中執行終端命令

代理	模型	Claude 何時使用它
statusline-setup	Sonnet	當您執行 `/statusline` 來配置您的狀態行時
Claude Code Guide	Haiku	當您詢問有關 Claude Code 功能的問題時

除了這些內建 subagents 之外，您可以建立自己的 subagents，具有自訂提示、工具限制、權限模式、hooks 和 skills。以下各節展示如何開始和自訂 subagents。

快速入門：建立您的第一個 subagent

Subagents 在具有 YAML frontmatter 的 Markdown 檔案中定義。您可以 手動建立它們 或使用 `/agents` 命令。

本逐步解說指導您使用 `/agent` 命令建立使用者層級的 subagent。該 subagent 審查程式碼並為程式碼庫建議改進。

Step 1: 開啟 subagents 介面

在 Claude Code 中，執行：

```
/agents
```

Step 2: 建立新的使用者層級代理

選擇 **Create new agent**，然後選擇 **User-level**。這會將 subagent 儲存到 `~/.claude/agents/`，以便在所有專案中使用。

Step 3: 使用 Claude 產生

選擇 **Generate with Claude**。出現提示時，描述 subagent：

```
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 產生系統提示和配置。按 `e` 在編輯器中開啟它，如果您想自訂它。

Step 4: 選擇工具

對於唯讀審查者，取消選擇除 **Read-only tools** 之外的所有內容。如果您保持所有工具選中，subagent 會繼承主對話可用的所有工具。

Step 5: 選擇模型

選擇 subagent 使用的模型。對於此範例代理，選擇 **Sonnet**，它在分析程式碼模式的能力和速度之間取得平衡。

Step 6: 選擇顏色

為 subagent 選擇背景顏色。這可幫助您識別在 UI 中執行的 subagent。

Step 7: 儲存並試用

儲存 subagent。它立即可用（無需重新啟動）。試用它：

```
Use the code-improver agent to suggest improvements in this project
```

Claude 委派給您的新 subagent，它掃描程式碼庫並返回改進建議。

您現在有一個 subagent，可以在機器上的任何專案中使用，以分析程式碼庫並建議改進。

您也可以手動建立 subagents 作為 Markdown 檔案、透過 CLI 旗標定義它們，或透過 plugins 分發它們。以下各節涵蓋所有配置選項。

配置 subagents

使用 /agents 命令

`/agents` 命令提供用於管理 subagents 的互動式介面。執行 `/agents` 以：

- 檢視所有可用的 subagents（內建、使用者、專案和 plugin）
- 使用引導式設定或 Claude 產生建立新的 subagents
- 編輯現有 subagent 配置和工具存取
- 刪除自訂 subagents
- 查看重複項存在時哪些 subagents 處於活動狀態

這是建立和管理 subagents 的建議方式。對於手動建立或自動化，您也可以直接新增 subagent 檔案。

若要從命令行列出所有已配置的 subagents 而不啟動互動式工作階段，請執行 `claude agents`。這會按來源分組顯示代理，並指示哪些被更高優先級定義覆蓋。

選擇 subagent 範圍

Subagents 是具有 YAML frontmatter 的 Markdown 檔案。根據範圍將它們儲存在不同位置。當多個 subagents 共享相同名稱時，更高優先級位置獲勝。

位置	範圍	優先級	如何建立
`--agents` CLI 旗標	目前工作階段	1（最高）	啟動 Claude Code 時傳遞 JSON
`.claude/agents/`	目前專案	2	互動式或手動
`~/.claude/agents/`	您的所有專案	3	互動式或手動
Plugin 的 `agents/` 目錄	啟用 plugin 的位置	4（最低）	使用 plugins 安裝

專案 subagents（`.claude/agents/`）非常適合特定於程式碼庫的 subagents。將它們簽入版本控制，以便您的團隊可以協作使用和改進它們。

使用者 subagents（`~/.claude/agents/`）是在所有專案中可用的個人 subagents。

CLI 定義的 subagents 在啟動 Claude Code 時作為 JSON 傳遞。它們僅存在於該工作階段，不會儲存到磁碟，使其適用於快速測試或自動化指令碼：

```
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` 旗標接受 JSON，具有與檔案型 subagents 相同的 frontmatter 欄位：`description`、`prompt`、`tools`、`disallowedTools`、`model`、`permissionMode`、`mcpServers`、`hooks`、`maxTurns`、`skills` 和 `memory`。使用 `prompt` 作為系統提示，等同於檔案型 subagents 中的 markdown 主體。請參閱 CLI 參考 以取得完整的 JSON 格式。

Plugin subagents 來自您已安裝的 plugins。它們與您的自訂 subagents 一起出現在 `/agents` 中。請參閱 plugin 元件參考 以取得建立 plugin subagents 的詳細資訊。

編寫 subagent 檔案

Subagent 檔案使用 YAML frontmatter 進行配置，後面跟著 Markdown 中的系統提示：

> **Note:**
>
> Subagents 在工作階段開始時載入。如果您透過手動新增檔案來建立 subagent，請重新啟動您的工作階段或使用 `/agents` 立即載入它。

```
---
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.
```

Frontmatter 定義 subagent 的中繼資料和配置。主體成為指導 subagent 行為的系統提示。Subagents 僅接收此系統提示（加上基本環境詳細資訊，如工作目錄），而不是完整的 Claude Code 系統提示。

支援的 frontmatter 欄位

以下欄位可用於 YAML frontmatter。只有 `name` 和 `description` 是必需的。

欄位	必需	描述
`name`	是	使用小寫字母和連字號的唯一識別碼
`description`	是	Claude 何時應委派給此 subagent
`tools`	否	subagent 可以使用的工具。如果省略，繼承所有工具
`disallowedTools`	否	要拒絕的工具，從繼承或指定的清單中移除

欄位	必需	描述
`model`	否	要使用的 模型：`sonnet`、`opus`、`haiku` 或 `inherit`。預設為 `inherit`
`permissionMode`	否	權限模式：`default`、`acceptEdits`、`dontAsk`、`bypassPermissions` 或 `plan`
`maxTurns`	否	subagent 停止前的最大代理轉數
`skills`	否	在啟動時載入到 subagent 上下文中的 Skills。注入完整的 skill 內容，而不僅僅是可供叫用。Subagents 不繼承父對話中的 skills
`mcpServers`	否	此 subagent 可用的 MCP servers。每個項目要麼是參考已配置伺服器的伺服器名稱（例如 `"slack"`），要麼是內聯定義，其中伺服器名稱為鍵，完整的 MCP 伺服器配置 為值
`hooks`	否	限定於此 subagent 的 生命週期 hooks
`memory`	否	持久記憶體範圍：`user`、`project` 或 `local`。啟用跨工作階段學習
`background`	否	設定為 `true` 以始終將此 subagent 作為 背景任務 執行。預設：`false`

欄位	必需	描述
`isolation`	否	設定為 `worktree` 以在臨時 git worktree 中執行 subagent，為其提供儲存庫的隔離副本。如果 subagent 不進行任何更改，worktree 會自動清理

選擇模型

`model` 欄位控制 subagent 使用的 AI 模型：

- **模型別名**：使用可用的別名之一：`sonnet`、`opus` 或 `haiku`
- **inherit**：使用與主對話相同的模型
- **省略**：如果未指定，預設為 `inherit`（使用與主對話相同的模型）

控制 subagent 功能

您可以透過工具存取、權限模式和條件規則來控制 subagents 可以執行的操作。

可用工具

Subagents 可以使用 Claude Code 的任何內部工具。預設情況下，subagents 繼承主對話的所有工具，包括 MCP 工具。

若要限制工具，請使用 `tools` 欄位（允許清單）或 `disallowedTools` 欄位（拒絕清單）：

```
---
name: safe-researcher
description: Research agent with restricted capabilities
tools: Read, Grep, Glob, Bash
disallowedTools: Write, Edit
---
```

限制可以產生的 subagents

當代理以 `claude --agent` 作為主執行緒執行時，它可以使用 Agent 工具產生 subagents。若要限制它可以產生的 subagent 類型，請在 `tools` 欄位中使用 `Agent(agent_type)` 語法。

> **Note:**

> 在版本 2.1.63 中，Task 工具已重新命名為 Agent。設定和代理定義中的現有 `Task(...)` 參考仍可作為別名使用。

```
---
name: coordinator
description: Coordinates work across specialized agents
tools: Agent(worker, researcher), Read, Bash
---
```

這是一個允許清單：只有 `worker` 和 `researcher` subagents 可以產生。如果代理嘗試產生任何其他類型，請求會失敗，代理在其提示中只看到允許的類型。若要在允許所有其他類型的同時阻止特定代理，請改用 `permissions.deny`。

若要允許產生任何 subagent 而不受限制，請使用不帶括號的 `Agent`：

```
tools: Agent, Read, Bash
```

如果 `Agent` 完全從 `tools` 清單中省略，代理無法產生任何 subagents。此限制僅適用於以 `claude --agent` 作為主執行緒執行的代理。Subagents 無法產生其他 subagents，因此 `Agent(agent_type)` 在 subagent 定義中無效。

權限模式

`permissionMode` 欄位控制 subagent 如何處理權限提示。Subagents 繼承主對話的權限上下文，但可以覆蓋模式。

模式	行為
`default`	標準權限檢查，帶提示
`acceptEdits`	自動接受檔案編輯
`dontAsk`	自動拒絕權限提示（明確允許的工具仍然有效）
`bypassPermissions`	跳過所有權限檢查
`plan`	Plan mode（唯讀探索）

> **Warning:**

> 謹慎使用 `bypassPermissions`。它跳過所有權限檢查，允許 subagent 執行任何操作而無需批准。

如果父級使用 `bypassPermissions`，這優先級最高，無法覆蓋。

將 skills 預載入 subagents

使用 `skills` 欄位在啟動時將 skill 內容注入到 subagent 的上下文中。這為 subagent 提供領域知識，而無需在執行期間發現和載入 skills。

```
---
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.
```

每個 skill 的完整內容被注入到 subagent 的上下文中，而不僅僅是可供叫用。Subagents 不繼承父對話中的 skills；您必須明確列出它們。

> **Note:**
>
> 這與 在 subagent 中執行 skill 相反。使用 subagent 中的 `skills`，subagent 控制系統提示並載入 skill 內容。使用 skill 中的 `context: fork`，skill 內容被注入到您指定的代理中。兩者都使用相同的基礎系統。

啟用持久記憶體

`memory` 欄位為 subagent 提供一個跨對話存活的持久目錄。Subagent 使用此目錄隨著時間推移建立知識，例如程式碼庫模式、除錯見解和架構決策。

```
---
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.
```

根據記憶體應該應用的廣泛程度選擇範圍：

範圍	位置	使用時機
`user`	`~/.claude/agent-memory/<name-of-agent>/`	subagent 應該記住跨所有專案的學習
`project`	`.claude/agent-memory/<name-of-agent>/`	subagent 的知識是特定於專案的並可透過版本控制共享
`local`	`.claude/agent-memory-local/<name-of-agent>/`	subagent 的知識是特定於專案的但不應簽入版本控制

啟用記憶體時：

- subagent 的系統提示包括讀取和寫入記憶體目錄的說明。
- subagent 的系統提示還包括記憶體目錄中 `MEMORY.md` 的前 200 行，以及如果 `MEMORY.md` 超過 200 行則策劃 `MEMORY.md` 的說明。
- Read、Write 和 Edit 工具會自動啟用，以便 subagent 可以管理其記憶體檔案。

持久記憶體提示

- `user` 是建議的預設範圍。當 subagent 的知識僅與特定程式碼庫相關時，使用 `project` 或 `local` 。
- 要求 subagent 在開始工作前查詢其記憶體：“Review this PR, and check your memory for patterns you’ve seen before.”
- 要求 subagent 在完成任務後更新其記憶體：“Now that you’re done, save what you learned to your memory.” 隨著時間推移，這會建立一個知識庫，使 subagent 更有效。
- 直接在 subagent 的 markdown 檔案中包含記憶體說明，以便它主動維護自己的知識庫：

```
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.
```

使用 hooks 的條件規則

為了更動態地控制工具使用，請使用 `PreToolUse` hooks 在執行前驗證操作。當您需要允許工具的某些操作同時阻止其他操作時，這很有用。

此範例建立一個只允許唯讀資料庫查詢的 subagent。`PreToolUse` hook 在每個 Bash 命令執行前執行 `command` 中指定的指令碼：

```
---
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 透過 stdin 將 hook 輸入作為 JSON 傳遞 給 hook 命令。驗證指令碼讀取此 JSON，提取 Bash 命令，並 以代碼 2 退出 以阻止寫入操作：

```
#!/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
```

請參閱 Hook 輸入 以取得完整的輸入架構，以及 退出代碼 以了解退出代碼如何影響行為。

禁用特定 subagents

您可以透過將 subagents 新增到 設定 中的 `deny` 陣列來防止 Claude 使用特定 subagents。使用格式 `Agent(subagent-name)`，其中 `subagent-name` 與 subagent 的 name 欄位相符。

```
{
  "permissions": {
    "deny": ["Agent(Explore)", "Agent(my-custom-agent)"]
  }
}
```

這適用於內建和自訂 subagents。您也可以使用 `--disallowedTools` CLI 旗標：

```
claude --disallowedTools "Agent(Explore)"
```

請參閱 權限文件 以取得有關權限規則的更多詳細資訊。

為 subagents 定義 hooks

Subagents 可以定義在 subagent 生命週期期間執行的 hooks。有兩種方式來配置 hooks：

1. **在 subagent 的 frontmatter 中**：定義僅在該 subagent 活動時執行的 hooks
2. **在 `settings.json` 中**：定義在 subagents 啟動或停止時在主工作階段中執行的 hooks

Subagent frontmatter 中的 Hooks

直接在 subagent 的 markdown 檔案中定義 hooks。這些 hooks 僅在該特定 subagent 活動時執行，並在完成時清理。

支援所有 hook 事件。subagents 最常見的事件是：

事件	匹配器輸入	何時觸發
`PreToolUse`	工具名稱	在 subagent 使用工具之前
`PostToolUse`	工具名稱	在 subagent 使用工具之後
`Stop`	（無）	當 subagent 完成時（在執行時轉換為 `SubagentStop`）

此範例使用 `PreToolUse` hook 驗證 Bash 命令，並在檔案編輯後使用 `PostToolUse` 執行 linter：

```
---
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"
---
```

Frontmatter 中的 `Stop` hooks 會自動轉換為 `SubagentStop` 事件。

用於 subagent 事件的專案層級 hooks

在 `settings.json` 中配置 hooks，以回應主工作階段中的 subagent 生命週期事件。

事件	匹配器輸入	何時觸發
`SubagentStart`	代理類型名稱	當 subagent 開始執行時
`SubagentStop`	代理類型名稱	當 subagent 完成時

兩個事件都支援匹配器以按名稱針對特定代理類型。此範例僅在 `db-agent` subagent 啟動時執行設定指令碼，並在任何 subagent 停止時執行清理指令碼：

```
{
  "hooks": {
    "SubagentStart": [
      {
        "matcher": "db-agent",
        "hooks": [
          { "type": "command", "command": "./scripts/setup-db-connection.sh" }
        ]
      }
    ],
    "SubagentStop": [
      {
        "hooks": [
          { "type": "command", "command": "./scripts/cleanup-db-connection.sh" }
        ]
      }
    ]
  }
}
```

請參閱 Hooks 以取得完整的 hook 配置格式。

使用 subagents

理解自動委派

Claude 根據您請求中的任務描述、subagent 配置中的 `description` 欄位和目前上下文自動委派任務。為了鼓勵主動委派，在 subagent 的 description 欄位中包含”use proactively”之類的短語。

您也可以明確要求特定 subagent：

```
Use the test-runner subagent to fix failing tests
Have the code-reviewer subagent look at my recent changes
```

在前景或背景中執行 subagents

Subagents 可以在前景（阻止）或背景（並行）中執行：

- **前景 subagents** 阻止主對話直到完成。權限提示和澄清問題（如 `AskUserQuestion`）會傳遞給您。
- **背景 subagents** 在您繼續工作時並行執行。啟動前，Claude Code 會提示輸入 subagent 需要的任何工具權限，確保它具有必要的批准。執行後，subagent 繼承這些權限並自動拒絕未預先批准的任何內容。如果背景 subagent 需要提出澄清問題，該工具呼叫會失敗，但 subagent 會繼續。

如果背景 subagent 因缺少權限而失敗，您可以 恢復它 在前景中以使用互動式提示重試。

Claude 根據任務決定是否在前景或背景中執行 subagents。您也可以：

- 要求 Claude “run this in the background”
- 按 **Ctrl+B** 將執行中的任務放在背景中

若要禁用所有背景任務功能，請將 `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` 環境變數設定為 `1`。請參閱 環境變數。

常見模式

隔離高容量操作

subagents 最有效的用途之一是隔離產生大量輸出的操作。執行測試、獲取文件或處理日誌檔案可能會消耗大量上下文。透過將這些委派給 subagent，詳細輸出保留在 subagent 的上下文中，而只有相關摘要返回到主對話。

```
Use a subagent to run the test suite and report only the failing tests with their
error messages
```

執行並行研究

對於獨立調查，產生多個 subagents 以同時工作：

```
Research the authentication, database, and API modules in parallel using separate
subagents
```

每個 subagent 獨立探索其區域，然後 Claude 綜合發現。當研究路徑彼此不相依時，這效果最佳。

> **Warning:**
>
> 當 subagents 完成時，其結果返回到主對話。執行許多 subagents，每個都返回詳細結果，可能會消耗大量上下文。

對於需要持續並行性或超過上下文視窗的任務，agent teams 為每個工作者提供自己的獨立上下文。

鏈接 subagents

對於多步驟工作流程，要求 Claude 按順序使用 subagents。每個 subagent 完成其任務並將結果返回給 Claude，然後 Claude 將相關上下文傳遞給下一個 subagent。

```
Use the code-reviewer subagent to find performance issues, then use the optimizer
subagent to fix them
```

在 subagents 和主對話之間選擇

在以下情況下使用**主對話**：

- 任務需要頻繁的來回或反覆改進
- 多個階段共享重要上下文（規劃 → 實現 → 測試）
- 您進行快速、有針對性的更改
- 延遲很重要。Subagents 從頭開始，可能需要時間收集上下文

在以下情況下使用 **subagents**：

- 任務產生您不需要在主上下文中的詳細輸出
- 您想強制執行特定的工具限制或權限
- 工作是自包含的，可以返回摘要

當您想要可重複使用的提示或在主對話上下文中執行的工作流程而不是隔離的 subagent 上下文時，請改為考慮 Skills。

對於關於對話中已有內容的快速問題，請使用 `/btw` 而不是 subagent。它看到您的完整上下文但沒有工具存取，答案被丟棄而不是新增到歷史記錄。

> **Note:**
>
> Subagents 無法產生其他 subagents。如果您的工作流程需要嵌套委派，請使用 Skills 或從主對話 鏈接 subagents。

管理 subagent 上下文

恢復 subagents

每個 subagent 叫用都會建立一個具有新鮮上下文的新實例。若要繼續現有 subagent 的工作而不是重新開始，請要求 Claude 恢復它。

恢復的 subagents 保留其完整的對話歷史記錄，包括所有先前的工具呼叫、結果和推理。Subagent 從停止的地方精確繼續，而不是從頭開始。

當 subagent 完成時，Claude 接收其代理 ID。若要恢復 subagent，請要求 Claude 繼續先前的工作：

```
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]
```

您也可以要求 Claude 提供代理 ID（如果您想明確參考它），或在 `~/.claude/projects/{project}/{sessionId}/subagents/` 的文字檔案中找到 ID。每個文字檔案儲存為 `agent-{agentId}.jsonl`。

Subagent 文字檔案獨立於主對話持續存在：

- **主對話壓縮**：當主對話壓縮時，subagent 文字檔案不受影響。它們儲存在單獨的檔案中。
- **工作階段持續性**：Subagent 文字檔案在其工作階段內持續存在。您可以透過恢復相同工作階段在重新啟動 Claude Code 後 恢復 subagent。
- **自動清理**：文字檔案根據 `cleanupPeriodDays` 設定（預設：30 天）進行清理。

自動壓縮

Subagents 支援使用與主對話相同的邏輯進行自動壓縮。預設情況下，自動壓縮在大約 95% 容量時觸發。若要更早觸發壓縮，請將 `CLAUDE_AUTOCOMPACT_PCT_OVERRIDE` 設定為較低的百分比（例如 `50`）。請參閱 環境變數 以取得詳細資訊。

壓縮事件記錄在 subagent 文字檔案中：

```
{
  "type": "system",
  "subtype": "compact_boundary",
  "compactMetadata": {
    "trigger": "auto",
    "preTokens": 167189
  }
}
```

`preTokens` 值顯示壓縮發生前使用了多少個 tokens。

範例 subagents

這些範例展示了建立 subagents 的有效模式。將它們用作起點，或使用 Claude 產生自訂版本。

> **Tip:**
>
> **最佳實踐：**
>
> - **設計專注的 subagents：** 每個 subagent 應該在一個特定任務上表現出色
> - **編寫詳細的描述：** Claude 使用描述來決定何時委派
> - **限制工具存取：** 僅授予必要的權限以確保安全和專注
> - **簽入版本控制：** 與您的團隊共享專案 subagents

程式碼審查者

一個唯讀 subagent，審查程式碼而不修改它。此範例展示如何設計一個具有有限工具存取（無 Edit 或 Write）和詳細提示的專注 subagent，該提示明確指定要查找的內容以及如何格式化輸出。

```
---
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.
```

除錯器

一個可以分析和修復問題的 subagent。與程式碼審查者不同，這個包括 Edit，因為修復錯誤需要修改程式碼。提示提供了從診斷到驗證的清晰工作流程。

```
---
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.
```

資料科學家

用於資料分析工作的特定領域 subagent。此範例展示如何為典型編碼任務之外的專門工作流程建立 subagents。它明確設定 `model: sonnet` 以進行更有能力的分析。

```
---
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.
```

資料庫查詢驗證器

一個允許 Bash 存取但驗證命令以僅允許唯讀 SQL 查詢的 subagent。此範例展示如何使用 `PreToolUse` hooks 進行條件驗證，當您需要比 `tools` 欄位更精細的控制時。

```
---
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 透過 stdin 將 hook 輸入作為 JSON 傳遞 給 hook 命令。驗證指令碼讀取此 JSON，提取正在執行的命令，並根據 SQL 寫入操作清單檢查它。如果檢測到寫入操作，指令碼 以代碼 2 退出 以阻止執行，並透過 stderr 向 Claude 返回錯誤訊息。

在專案中的任何位置建立驗證指令碼。路徑必須與 hook 配置中的 `command` 欄位相符：

```
#!/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
```

使指令碼可執行：

```
chmod +x ./scripts/validate-readonly-query.sh
```

Hook 透過 stdin 接收 JSON，Bash 命令在 `tool_input.command` 中。退出代碼 2 阻止操作並將錯誤訊息反饋給 Claude。請參閱 Hooks 以取得有關退出代碼的詳細資訊，以及 Hook 輸入 以取得完整的輸入架構。

後續步驟

現在您理解了 subagents，請探索這些相關功能：

- 使用 plugins 分發 subagents 以跨團隊或專案共享 subagents
- 以程式方式執行 Claude Code 使用 Agent SDK 進行 CI/CD 和自動化
- 使用 MCP servers 為 subagents 提供外部工具和資料的存取

協調 Claude Code 工作階段團隊

> 協調多個 Claude Code 實例作為團隊一起工作，具有共享任務、代理間訊息傳遞和集中管理。

Warning:

Agent teams 是實驗性功能，預設為停用。透過在 settings.json 或環境中新增 `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` 來啟用。Agent teams 在工作階段恢復、任務協調和關閉行為方面有已知限制。

Agent teams 讓您協調多個 Claude Code 實例一起工作。一個工作階段充當團隊主管，協調工作、分配任務並綜合結果。隊友獨立工作，各自在自己的 context window 中，並直接相互溝通。

與 subagents 不同，subagents 在單個工作階段內運行，只能向主代理報告，您也可以直接與個別隊友互動，無需透過主管。

Note:

Agent teams 需要 Claude Code v2.1.32 或更新版本。使用 `claude --version` 檢查您的版本。

本頁涵蓋：

- 何時使用 agent teams，包括最佳使用案例以及與 subagents 的比較
- 啟動團隊
- 控制隊友，包括顯示模式、任務分配和委派
- 並行工作的最佳實踐

何時使用 agent teams

Agent teams 最適合用於並行探索能增加真實價值的任務。請參閱使用案例範例以了解完整情景。最強的使用案例是：

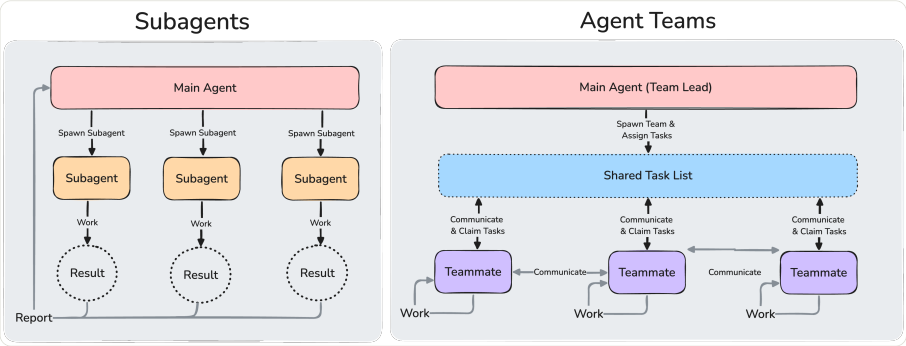
- **研究和審查**：多個隊友可以同時調查問題的不同方面，然後分享並質疑彼此的發現
- **新模組或功能**：隊友可以各自擁有一個獨立部分，不會相互干擾
- **使用競爭假設進行除錯**：隊友並行測試不同的理論，更快地收斂到答案

- **跨層協調**：跨越前端、後端和測試的變更，各由不同的隊友負責

Agent teams 增加了協調開銷，並使用的 tokens 遠多於單個工作階段。當隊友可以獨立運作時，它們效果最佳。對於順序任務、相同檔案編輯或具有許多依賴關係的工作，單個工作階段或 subagents 更有效。

與 subagents 比較

Agent teams 和 subagents 都讓您並行化工作，但它們的運作方式不同。根據您的工作人員是否需要相互溝通來選擇：



比較 subagent 和 agent team 架構的圖表。Subagents 由主代理生成、執行工作並報告結果。Agent teams 透過共享任務列表進行協調，隊友彼此直接溝通。

	Subagents	Agent teams
Context	自己的 context window；結果返回給呼叫者	自己的 context window；完全獨立
溝通	只向主代理報告結果	隊友直接相互訊息傳遞
協調	主代理管理所有工作	具有自我協調的共享任務列表
最適合	只有結果重要的專注任務	需要討論和協作的複雜工作
Token 成本	較低：結果摘要返回到主 context	較高：每個隊友是一個獨立的 Claude 實例

當您需要快速、專注的工作人員報告結果時，使用 subagents。當隊友需要分享發現、相互質疑並自行協調時，使用 agent teams。

啟用 agent teams

Agent teams 預設為停用。透過將 `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` 環境變數設定為 `1`，在您的 shell 環境或透過 settings.json 來啟用：

```
{
  "env": {
    "CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS": "1"
  }
}
```

啟動您的第一個 agent team

啟用 agent teams 後，告訴 Claude 建立一個 agent team 並用自然語言描述您想要的任務和團隊結構。Claude 建立團隊、生成隊友並根據您的提示協調工作。

此範例效果很好，因為三個角色是獨立的，可以在不相互等待的情況下探索問題：

```
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.
```

從那裡，Claude 建立一個具有共享任務列表的團隊，為每個觀點生成隊友，讓他們探索問題，綜合發現，並嘗試在完成時清理團隊。

主管的終端列出所有隊友及其正在進行的工作。使用 Shift+Down 循環瀏覽隊友並直接向他們傳送訊息。在最後一個隊友之後，Shift+Down 會回到主管。

如果您希望每個隊友都在自己的分割窗格中，請參閱選擇顯示模式。

控制您的 agent team

用自然語言告訴主管您想要什麼。它根據您的指示處理團隊協調、任務分配和委派。

選擇顯示模式

Agent teams 支援兩種顯示模式：

- **In-process**：所有隊友在您的主終端內運行。使用 Shift+Down 循環瀏覽隊友並輸入以直接向他們傳送訊息。在任何終端中工作，無需額外設定。
- **Split panes**：每個隊友都有自己的窗格。您可以同時看到所有人的輸出並點擊窗格直接互動。需要 tmux 或 iTerm2。

> **Note:**
>
> `tmux` 在某些作業系統上有已知限制，傳統上在 macOS 上效果最佳。在 iTerm2 中使用 `tmux -CC` 是進入 `tmux` 的建議入口點。

預設值是 `"auto"`，如果您已在 tmux 工作階段內運行，則使用分割窗格，否則使用 in-process。`"tmux"` 設定啟用分割窗格模式，並根據您的終端自動偵測是否使用 tmux 或 iTerm2。若要覆蓋，請在 settings.json 中設定 `teammateMode`：

```
{
  "teammateMode": "in-process"
}
```

若要為單個工作階段強制 in-process 模式，請將其作為旗標傳遞：

```
claude --teammate-mode in-process
```

分割窗格模式需要 tmux 或 iTerm2 搭配 `it2` CLI。若要手動安裝：

- **tmux**：透過您系統的套件管理器安裝。請參閱 tmux wiki 以了解平台特定的指示。
- **iTerm2**：安裝 `it2` CLI，然後在 **iTerm2 → Settings → General → Magic → Enable Python API** 中啟用 Python API。

指定隊友和模型

Claude 根據您的任務決定要生成的隊友數量，或者您可以指定您想要的確切內容：

```
Create a team with 4 teammates to refactor these modules in parallel.
Use Sonnet for each teammate.
```

要求隊友的計畫批准

對於複雜或有風險的任務，您可以要求隊友在實施前進行計畫。隊友在唯讀計畫模式下工作，直到主管批准其方法：

```
Spawn an architect teammate to refactor the authentication module.
Require plan approval before they make any changes.
```

當隊友完成計畫時，它會向主管發送計畫批准請求。主管審查計畫並批准或拒絕並提供反饋。如果被拒絕，隊友保持在計畫模式，根據反饋進行修訂並重新提交。一旦批准，隊友退出計畫模式並開始實施。

主管自主做出批准決定。若要影響主管的判斷，在您的提示中提供標準，例如「只批准包含測試覆蓋的計畫」或「拒絕修改資料庫架構的計畫」。

直接與隊友交談

每個隊友都是一個完整、獨立的 Claude Code 工作階段。您可以直接向任何隊友傳送訊息，以提供額外指示、提出後續問題或重新定向其方法。

- **In-process 模式**：使用 Shift+Down 循環瀏覽隊友，然後輸入以向他們傳送訊息。按 Enter 查看隊友的工作階段，然後按 Escape 中斷其目前回合。按 Ctrl+T 切換任務列表。
- **Split-pane 模式**：點擊隊友的窗格以直接與其工作階段互動。每個隊友都有自己終端的完整檢視。

分配和認領任務

共享任務列表協調整個團隊的工作。主管建立任務，隊友完成它們。任務有三種狀態：待處理、進行中和已完成。任務也可以依賴其他任務：具有未解決依賴關係的待處理任務在這些依賴關係完成之前無法被認領。

主管可以明確分配任務，或隊友可以自行認領：

- **主管分配**：告訴主管將哪個任務分配給哪個隊友
- **自行認領**：完成任務後，隊友自行選擇下一個未分配、未阻止的任務

任務認領使用檔案鎖定來防止多個隊友同時嘗試認領同一任務時的競爭條件。

關閉隊友

若要優雅地結束隊友的工作階段：

```
Ask the researcher teammate to shut down
```

主管發送關閉請求。隊友可以批准並優雅地退出，或拒絕並提供解釋。

清理團隊

完成後，要求主管清理：

```
Clean up the team
```

這會移除共享的團隊資源。當主管運行清理時，它會檢查活躍的隊友，如果仍有任何隊友在運行，則失敗，因此請先關閉他們。

> **Warning:**
>
> 始終使用主管進行清理。隊友不應運行清理，因為他們的團隊 context 可能無法正確解析，可能會使資源處於不一致的狀態。

使用 hooks 強制執行品質閘門

使用 hooks 在隊友完成工作或任務完成時強制執行規則：

- `TeammateIdle`：當隊友即將閒置時運行。以代碼 2 退出以發送反饋並保持隊友工作。
- `TaskCompleted`：當任務被標記為完成時運行。以代碼 2 退出以防止完成並發送反饋。

Agent teams 如何工作

本節涵蓋 agent teams 背後的架構和機制。如果您想開始使用它們，請參閱上面的控制您的 agent team。

Claude 如何啟動 agent teams

Agent teams 有兩種啟動方式：

- **您請求一個團隊**：給 Claude 一個受益於並行工作的任務，並明確要求一個 agent team。Claude 根據您的指示建立一個。
- **Claude 提議一個團隊**：如果 Claude 確定您的任務將受益於並行工作，它可能會建議建立一個團隊。您在它繼續之前確認。

在這兩種情況下，您都保持控制。Claude 不會在沒有您批准的情況下建立團隊。

架構

Agent team 由以下部分組成：

元件	角色
Team lead	建立團隊、生成隊友並協調工作的主要 Claude Code 工作階段
Teammates	各自處理分配任務的獨立 Claude Code 實例
Task list	隊友認領和完成的共享工作項目列表

元件	角色
Mailbox	代理之間通訊的訊息系統

請參閱選擇顯示模式以了解顯示配置選項。隊友訊息自動到達主管。

系統自動管理任務依賴關係。當隊友完成其他任務依賴的任務時，被阻止的任務會自動解除阻止。

團隊和任務存儲在本地：

- **Team config**：`~/.claude/teams/{team-name}/config.json`
- **Task list**：`~/.claude/tasks/{team-name}/`

團隊配置包含一個 `members` 陣列，其中包含每個隊友的名稱、代理 ID 和代理類型。隊友可以讀取此檔案以發現其他團隊成員。

權限

隊友開始時具有主管的權限設定。如果主管使用 `--dangerously-skip-permissions` 運行，所有隊友也會這樣做。生成後，您可以更改個別隊友模式，但在生成時無法設定每個隊友的模式。

Context 和通訊

每個隊友都有自己的 context window。生成時，隊友載入與常規工作階段相同的專案 context：CLAUDE.md、MCP servers 和 skills。它還接收來自主管的生成提示。主管的對話歷史不會延續。

隊友如何分享資訊：

- **自動訊息傳遞**：當隊友發送訊息時，它們會自動傳遞給收件人。主管不需要輪詢更新。
- **閒置通知**：當隊友完成並停止時，他們會自動通知主管。
- **共享任務列表**：所有代理都可以看到任務狀態並認領可用工作。

隊友訊息傳遞：

- **message**：向一個特定隊友發送訊息
- **broadcast**：同時發送給所有隊友。謹慎使用，因為成本隨團隊規模而增加。

Token 使用

Agent teams 使用的 tokens 遠多於單個工作階段。每個隊友都有自己的 context window，token 使用量隨活躍隊友數量而增加。對於研究、審查和新功能工作，額外的 tokens 通常是值得的。對於日常任務，單個工作階段更具成本效益。請參閱 agent team token 成本以了解使用指南。

使用案例範例

這些範例展示了 agent teams 如何處理並行探索增加價值的任務。

運行並行程式碼審查

單個審查者傾向於一次專注於一種類型的問題。將審查標準分成獨立領域意味著安全性、效能和測試覆蓋都同時獲得徹底的關注。提示為每個隊友分配一個不同的視角，以便他們不重疊：

```
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.
```

每個審查者從相同的 PR 工作，但應用不同的篩選器。主管在他們完成後綜合所有三個的發現。

使用競爭假設進行調查

當根本原因不清楚時，單個代理傾向於找到一個看似合理的解釋並停止尋找。提示透過使隊友明確對抗來對抗這一點：每個隊友的工作不僅是調查自己的理論，還要質疑其他隊友的理論。

```
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.
```

辯論結構是這裡的關鍵機制。順序調查受到錨定的影響：一旦探索了一個理論，後續調查就會偏向於它。

有多個獨立調查人員積極嘗試相互反駁，倖存的理論更有可能是實際的根本原因。

最佳實踐

給隊友足夠的 context

隊友自動載入專案 context，包括 CLAUDE.md、MCP servers 和 skills，但他們不繼承主管的對話歷史。請參閱Context 和通訊以了解詳情。在生成提示中包含任務特定的詳情：

```
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."
```

選擇適當的團隊規模

隊友數量沒有硬性限制，但實際限制適用：

- **Token 成本線性增加**：每個隊友都有自己的 context window 並獨立消耗 tokens。請參閱 agent team token 成本以了解詳情。
- **協調開銷增加**：更多隊友意味著更多通訊、任務協調和潛在衝突
- **收益遞減**：超過一定點後，額外的隊友不會按比例加快工作

對於大多數工作流程，從 3-5 個隊友開始。這平衡了並行工作與可管理的協調。本指南中的範例使用 3-5 個隊友，因為該範圍在不同任務類型中效果很好。

每個隊友有 5-6 個任務可以保持每個人的生產力，而不會過度的上下文切換。如果您有 15 個獨立任務，3 個隊友是一個很好的起點。

只有當工作真正受益於隊友同時工作時才擴展。三個專注的隊友通常優於五個分散的隊友。

適當調整任務大小

- **太小**：協調開銷超過收益
- **太大**：隊友工作時間過長而沒有檢查點，增加浪費努力的風險
- **恰到好處**：自包含的單位，產生清晰的可交付成果，例如函數、測試檔案或審查

> **Tip:**
>
> 主管將工作分解為任務並自動分配給隊友。如果它沒有建立足夠的任務，要求它將工作分成更小的部分。每個隊友有 5-6 個任務可以保持每個人的生產力，並讓主管在有人卡住時重新分配工作。

等待隊友完成

有時主管開始自己實施任務，而不是等待隊友。如果您注意到這一點：

```
Wait for your teammates to complete their tasks before proceeding
```

從研究和審查開始

如果您是 agent teams 的新手，請從具有清晰邊界且不需要編寫程式碼的任務開始：審查 PR、研究庫或調查錯誤。這些任務展示了並行探索的價值，而不會帶來並行實施所帶來的協調挑戰。

避免檔案衝突

兩個隊友編輯同一檔案會導致覆蓋。分解工作，使每個隊友擁有不同的檔案集。

監控和引導

檢查隊友的進度，重新定向不起作用的方法，並在發現時綜合發現。讓團隊無人值守運行太長時間會增加浪費努力的風險。

故障排除

隊友未出現

如果在您要求 Claude 建立團隊後隊友未出現：

- 在 in-process 模式中，隊友可能已在運行但不可見。按 Shift+Down 循環瀏覽活躍隊友。
- 檢查您給 Claude 的任務是否足夠複雜以保證團隊。Claude 根據任務決定是否生成隊友。
- 如果您明確要求分割窗格，請確保 tmux 已安裝並在您的 PATH 中可用：

```
which tmux
```

- 對於 iTerm2，驗證 `it2` CLI 已安裝且 Python API 在 iTerm2 偏好設定中啟用。

過多權限提示

隊友權限請求冒泡到主管，這可能會造成摩擦。在生成隊友之前在 permission settings 中預批准常見操作以減少中斷。

隊友在錯誤時停止

隊友可能在遇到錯誤後停止，而不是恢復。使用 in-process 模式中的 Shift+Down 或分割模式中的點擊窗格檢查其輸出，然後：

- 直接給他們額外的指示
- 生成替換隊友以繼續工作

主管在工作完成前關閉

主管可能在所有任務實際完成之前決定團隊已完成。如果發生這種情況，告訴它繼續。您也可以告訴主管在繼續之前等待隊友完成，如果它開始做工作而不是委派。

孤立的 tmux 工作階段

如果 tmux 工作階段在團隊結束後仍然存在，它可能未被完全清理。列出工作階段並殺死由團隊建立的工作階段：

```
tmux ls
tmux kill-session -t <session-name>
```

限制

Agent teams 是實驗性的。要注意的目前限制：

- **In-process 隊友沒有工作階段恢復**：`/resume` 和 `/rewind` 不會恢復 in-process 隊友。恢復工作階段後，主管可能會嘗試向不再存在的隊友傳送訊息。如果發生這種情況，告訴主管生成新隊友。
- **任務狀態可能滯後**：隊友有時無法將任務標記為已完成，這會阻止依賴任務。如果任務似乎卡住，請檢查工作是否實際完成並手動更新任務狀態或告訴主管推動隊友。
- **關閉可能很慢**：隊友在關閉前完成其目前請求或工具呼叫，這可能需要時間。
- **每個工作階段一個團隊**：主管一次只能管理一個團隊。在啟動新團隊之前清理目前團隊。
- **沒有嵌套團隊**：隊友無法生成自己的團隊或隊友。只有主管可以管理團隊。
- **主管是固定的**：建立團隊的工作階段在其生命週期內是主管。您無法將隊友提升為主管或轉移領導權。
- **權限在生成時設定**：所有隊友開始時具有主管的權限模式。您可以在生成後更改個別隊友模式，但在生成時無法設定每個隊友的模式。
- **分割窗格需要 tmux 或 iTerm2**：預設 in-process 模式在任何終端中工作。VS Code 的整合終端、Windows Terminal 或 Ghostty 不支援分割窗格模式。

> **Tip:**
>
> `CLAUDE.md` **正常工作**：隊友從其工作目錄讀取 `CLAUDE.md` 檔案。使用此為所有隊友提供專案特定的指導。

後續步驟

探索並行工作和委派的相關方法：

- **輕量級委派**：subagents 在您的工作階段內為研究或驗證生成幫助代理，更適合不需要代理間協調的任務
- **手動並行工作階段**：Git worktrees 讓您自己運行多個 Claude Code 工作階段，無需自動化團隊協調
- **比較方法**：請參閱 subagent vs agent team 比較以了解並排細分

以程式方式執行 Claude Code

> 使用 Agent SDK 從 CLI、Python 或 TypeScript 以程式方式執行 Claude Code。

Agent SDK 提供與 Claude Code 相同的工具、agent 迴圈和上下文管理。它可作為 CLI 用於指令碼和 CI/CD，或作為 Python 和 TypeScript 套件供完整的程式控制。

> **Note:**
>
> CLI 之前稱為「無頭模式」。`-p` 旗標和所有 CLI 選項的工作方式相同。

若要從 CLI 以程式方式執行 Claude Code，請傳遞 `-p` 和您的提示以及任何 CLI 選項：

```
claude -p "Find and fix the bug in auth.py" --allowedTools "Read,Edit,Bash"
```

本頁涵蓋透過 CLI（`claude -p`）使用 Agent SDK。如需具有結構化輸出、工具核准回呼和原生訊息物件的 Python 和 TypeScript SDK 套件，請參閱完整 Agent SDK 文件。

基本用法

將 `-p`（或 `--print`）旗標新增至任何 `claude` 命令以非互動方式執行它。所有 CLI 選項都適用於 `-p`，包括：

- `--continue` 用於繼續對話
- `--allowedTools` 用於自動核准工具
- `--output-format` 用於結構化輸出

此範例詢問 Claude 關於您的程式碼庫的問題並列印回應：

```
claude -p "What does the auth module do?"
```

範例

這些範例突出顯示常見的 CLI 模式。

取得結構化輸出

使用 `--output-format` 控制回應的傳回方式：

- `text`（預設）：純文字輸出
- `json`：包含結果、工作階段 ID 和中繼資料的結構化 JSON
- `stream-json`：用於即時串流的換行分隔 JSON

此範例以 JSON 格式傳回專案摘要及工作階段中繼資料，文字結果在 `result` 欄位中：

```
claude -p "Summarize this project" --output-format json
```

若要取得符合特定結構描述的輸出，請使用 `--output-format json` 搭配 `--json-schema` 和 JSON Schema 定義。回應包含關於請求的中繼資料（工作階段 ID、使用情況等），結構化輸出在 `structured_output` 欄位中。

此範例從 auth.py 提取函式名稱並將其傳回為字串陣列：

```
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:

使用 jq 之類的工具來解析回應並提取特定欄位：

```
## Extract the text result
claude -p "Summarize this project" --output-format json | jq -r '.result'

## Extract structured output
claude -p "Extract function names from auth.py" \
  --output-format json \
  --json-schema '{"type":"object","properties":{"functions":
{"type":"array","items":{"type":"string"}}},"required":["functions"]}' \
  | jq '.structured_output'
```

串流回應

使用 `--output-format stream-json` 搭配 `--verbose` 和 `--include-partial-messages` 以在產生權杖時接收它們。每一行都是代表事件的 JSON 物件：

```
claude -p "Explain recursion" --output-format stream-json --verbose --include-
partial-messages
```

下列範例使用 jq 篩選文字差異並僅顯示串流文字。`-r` 旗標輸出原始字串（無引號），`-j` 不使用換行符號進行聯結，以便權杖連續串流：

```
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'
```

如需具有回呼和訊息物件的程式化串流，請參閱 Agent SDK 文件中的 即時串流回應。

自動核准工具

使用 `--allowedTools` 讓 Claude 使用某些工具而無需提示。此範例執行測試套件並修復失敗，允許 Claude 執行 Bash 命令和讀取/編輯檔案而無需請求許可：

```
claude -p "Run the test suite and fix any failures" \
  --allowedTools "Bash,Read,Edit"
```

建立提交

此範例檢查暫存的變更並建立具有適當訊息的提交：

```
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` 旗標使用 權限規則語法。尾部的 `*` 啟用前綴匹配，因此 `Bash(git diff *)` 允許任何以 `git diff` 開頭的命令。空格在 `*` 之前很重要：沒有它，`Bash(git diff*)` 也會符合 `git diff-index`。

> **Note:**
>
> 使用者叫用的 skills 如 `/commit` 和 內建命令 僅在互動模式中可用。在 `-p` 模式中，改為描述您想要完成的任務。

自訂系統提示

使用 `--append-system-prompt` 新增指示同時保持 Claude Code 的預設行為。此範例將 PR 差異管道傳送至 Claude 並指示它檢查安全漏洞：

```
gh pr diff "$1" | claude -p \
  --append-system-prompt "You are a security engineer. Review for
vulnerabilities." \
  --output-format json
```

請參閱 系統提示旗標 以取得更多選項，包括 `--system-prompt` 以完全取代預設提示。

繼續對話

使用 `--continue` 繼續最近的對話，或使用 `--resume` 搭配工作階段 ID 以繼續特定對話。此範例執行檢查，然後傳送後續提示：

```
## First request
claude -p "Review this codebase for performance issues"

## Continue the most recent conversation
claude -p "Now focus on the database queries" --continue
claude -p "Generate a summary of all issues found" --continue
```

如果您執行多個對話，請擷取工作階段 ID 以繼續特定對話：

```
session_id=$(claude -p "Start a review" --output-format json | jq -r
'.session_id')
claude -p "Continue that review" --resume "$session_id"
```

後續步驟

- Agent SDK 快速入門：使用 Python 或 TypeScript 建立您的第一個 agent
- CLI 參考：所有 CLI 旗標和選項
- GitHub Actions：在 GitHub 工作流程中使用 Agent SDK
- GitLab CI/CD：在 GitLab 管道中使用 Agent SDK

使用遠端控制從任何裝置繼續本機工作階段

> 使用遠端控制從您的手機、平板電腦或任何瀏覽器繼續本機 Claude Code 工作階段。適用於 claude.ai/code 和 Claude 行動應用程式。

Note:

遠端控制適用於所有方案。Team 和 Enterprise 管理員必須先在管理員設定中啟用 Claude Code。

遠端控制將 claude.ai/code 或 Claude 應用程式（iOS 和 Android）連接到在您機器上執行的 Claude Code 工作階段。在您的辦公桌開始一項任務，然後從沙發上的手機或另一台電腦上的瀏覽器繼續進行。

當您在機器上啟動遠端控制工作階段時，Claude 會在整個過程中在本機執行，因此沒有任何內容會移至雲端。使用遠端控制，您可以：

- **遠端使用您的完整本機環境**：您的檔案系統、MCP servers、工具和專案設定都保持可用
- **同時在兩個介面上工作**：對話在所有連接的裝置上保持同步，因此您可以從終端機、瀏覽器和手機交替傳送訊息
- **度過中斷**：如果您的筆記型電腦進入睡眠狀態或網路中斷，當您的機器重新上線時，工作階段會自動重新連接

與在雲端基礎設施上執行的 Claude Code on the web 不同，遠端控制工作階段直接在您的機器上執行並與您的本機檔案系統互動。Web 和行動介面只是該本機工作階段的一個視窗。

Note:

遠端控制需要 Claude Code v2.1.51 或更新版本。使用 `claude --version` 檢查您的版本。

本頁涵蓋設定、如何啟動和連接到工作階段，以及遠端控制與 Claude Code on the web 的比較。

需求

在使用遠端控制之前，請確認您的環境符合以下條件：

- **訂閱**：適用於 Pro、Max、Team 和 Enterprise 方案。Team 和 Enterprise 管理員必須先在管理員設定中啟用 Claude Code。不支援 API 金鑰。
- **驗證**：執行 `claude` 並使用 `/login` 透過 claude.ai 登入（如果您還沒有登入）。
- **工作區信任**：在您的專案目錄中至少執行一次 `claude` 以接受工作區信任對話。

啟動遠端控制工作階段

您可以直接在遠端控制中啟動新工作階段，或連接已在執行的工作階段。

新工作階段

導覽至您的專案目錄並執行：

```
claude remote-control
```

該程序在您的終端機中保持執行，等待遠端連接。它顯示一個工作階段 URL，您可以使用該 URL 從另一個裝置連接，您可以按空格鍵顯示 QR 碼以從您的手機快速存取。當遠端工作階段處於活動狀態時，終端機會顯示連接狀態和工具活動。

此命令支援以下旗標：

- `--name "My Project"`：設定在 claude.ai/code 的工作階段清單中可見的自訂工作階段標題。您也可以將名稱作為位置引數傳遞：`claude remote-control "My Project"`
- `--verbose`：顯示詳細的連接和工作階段日誌
- `--sandbox` / `--no-sandbox`：啟用或停用工作階段期間檔案系統和網路隔離的沙箱。預設情況下沙箱已關閉。

從現有工作階段

如果您已經在 Claude Code 工作階段中並想遠端繼續，請使用 `/remote-control`（或 `/rc`）命令：

```
/remote-control
```

傳遞名稱作為引數以設定自訂工作階段標題：

```
/remote-control My Project
```

這會啟動一個遠端控制工作階段，該工作階段會延續您目前的對話歷史記錄，並顯示一個工作階段 URL 和 QR 碼，您可以使用它從另一個裝置連接。`--verbose`、`--sandbox` 和 `--no-sandbox` 旗標不適用於此命令。

從另一個裝置連接

遠端控制工作階段啟動後，您有幾種方式可以從另一個裝置連接：

- **開啟工作階段 URL** 在任何瀏覽器中直接前往 claude.ai/code 上的工作階段。`claude remote-control` 和 `/remote-control` 都會在終端機中顯示此 URL。
- **掃描 QR 碼** 顯示在工作階段 URL 旁邊，直接在 Claude 應用程式中開啟它。使用 `claude remote-control` 時，按空格鍵切換 QR 碼顯示。
- **開啟 claude.ai/code 或 Claude 應用程式**，並在工作階段清單中按名稱找到工作階段。遠端控制工作階段在線上時會顯示帶有綠色狀態點的電腦圖示。

遠端工作階段從 `--name` 引數（或傳遞給 `/remote-control` 的名稱）、您的最後一條訊息、您的 `/rename` 值或「遠端控制工作階段」（如果沒有對話歷史記錄）取得其名稱。如果環境已經有一個活動工作階段，系統會詢問您是否要繼續它或啟動一個新工作階段。

如果您還沒有 Claude 應用程式，請在 Claude Code 內使用 `/mobile` 命令顯示 iOS 或 Android 的下載 QR 碼。

為所有工作階段啟用遠端控制

預設情況下，遠端控制只在您明確執行 `claude remote-control` 或 `/remote-control` 時啟動。若要為每個工作階段自動啟用它，請在 Claude Code 內執行 `/config` 並將**為所有工作階段啟用遠端控制**設定為 `true`。將其設定回 `false` 以停用。

每個 Claude Code 執行個體一次支援一個遠端工作階段。如果您執行多個執行個體，每個執行個體都會獲得自己的環境和工作階段。

連接和安全性

您的本機 Claude Code 工作階段僅發出出站 HTTPS 請求，永遠不會在您的機器上開啟入站連接埠。當您啟動遠端控制時，它會向 Anthropic API 註冊並輪詢工作。當您從另一個裝置連接時，伺服器會透過串流連接在 Web 或行動用戶端與您的本機工作階段之間路由訊息。

所有流量都透過 TLS 上的 Anthropic API 傳輸，與任何 Claude Code 工作階段相同的傳輸安全性。連接使用多個短期認證，每個認證的範圍限定為單一目的並獨立過期。

遠端控制與 Claude Code on the web

遠端控制和 Claude Code on the web 都使用 claude.ai/code 介面。關鍵區別在於工作階段執行的位置：遠端控制在您的機器上執行，因此您的本機 MCP servers、工具和專案設定保持可用。Claude Code on the web 在 Anthropic 管理的雲端基礎設施中執行。

當您在本機工作中途並想從另一個裝置繼續時，請使用遠端控制。當您想在沒有任何本機設定的情況下啟動任務、處理您沒有複製的儲存庫或並行執行多個任務時，請使用 Claude Code on the web。

限制

- **一次一個遠端工作階段**：每個 Claude Code 工作階段支援一個遠端連接。
- **終端機必須保持開啟**：遠端控制作為本機程序執行。如果您關閉終端機或停止 `claude` 程序，工作階段就會結束。再次執行 `claude remote-control` 以啟動新工作階段。
- **延長網路中斷**：如果您的機器處於喚醒狀態但無法在大約 10 分鐘以上的時間內連接到網路，工作階段會逾時並且程序會退出。再次執行 `claude remote-control` 以啟動新工作階段。

相關資源

- Claude Code on the web：在 Anthropic 管理的雲端環境中執行工作階段，而不是在您的機器上
- 驗證：設定 `/login` 並管理 claude.ai 的認證
- CLI 參考：完整的旗標和命令清單，包括 `claude remote-control`
- 安全性：遠端控制工作階段如何適應 Claude Code 安全性模型
- 資料使用：在本機和遠端工作階段期間透過 Anthropic API 流動的資料

按排程執行提示

> 使用 /loop 和 cron 排程工具在 Claude Code 工作階段內重複執行提示、輪詢狀態或設定一次性提醒。

> **Note:**
> 排程任務需要 Claude Code v2.1.72 或更新版本。使用 `claude --version` 檢查您的版本。

排程任務讓 Claude 按間隔自動重新執行提示。使用它們來輪詢部署、監督 PR、檢查長時間執行的建置，或在工作階段稍後提醒自己執行某些操作。

任務的範圍限於工作階段：它們存在於目前的 Claude Code 程序中，當您退出時就會消失。如需在重新啟動後仍能持續且無需活躍終端工作階段即可執行的持久排程，請參閱 Desktop 排程任務 或 GitHub Actions。

使用 /loop 排程重複提示

`/loop` bundled skill 是排程重複提示的最快方式。傳遞選用的間隔和提示，Claude 會設定在背景執行的 cron 工作，同時工作階段保持開啟。

```
/loop 5m check if the deployment finished and tell me what happened
```

Claude 解析間隔、將其轉換為 cron 表達式、排程工作，並確認節奏和工作 ID。

間隔語法

間隔是選用的。您可以在開頭使用、在結尾使用，或完全省略。

形式	範例	解析的間隔
前置令牌	`/loop 30m check the build`	每 30 分鐘
尾部 `every` 子句	`/loop check the build every 2 hours`	每 2 小時
無間隔	`/loop check the build`	預設為每 10 分鐘

支援的單位為 `s`（秒）、`m`（分鐘）、`h`（小時）和 `d`（天）。秒數會四捨五入到最近的分鐘，因為 cron 的粒度為一分鐘。不能均勻分割其單位的間隔（例如 `7m` 或 `90m`）會四捨五入到最近的整潔間隔，Claude 會告訴您它選擇了什麼。

迴圈執行另一個命令

排程提示本身可以是命令或 skill 呼叫。這對於重新執行您已經打包的工作流程很有用。

```
/loop 20m /review-pr 1234
```

每次工作執行時，Claude 會執行 `/review-pr 1234`，就像您輸入了它一樣。

設定一次性提醒

對於一次性提醒，請用自然語言描述您想要的內容，而不是使用 `/loop`。Claude 排程一個執行後自動刪除的單次執行任務。

```
remind me at 3pm to push the release branch
```

```
in 45 minutes, check whether the integration tests passed
```

Claude 使用 cron 表達式將執行時間固定到特定的分鐘和小時，並確認何時執行。

管理排程任務

用自然語言要求 Claude 列出或取消任務，或直接參考基礎工具。

```
what scheduled tasks do I have?
```

```
cancel the deploy check job
```

在幕後，Claude 使用這些工具：

工具	用途
`CronCreate`	排程新任務。接受 5 欄位 cron 表達式、要執行的提示，以及是否重複或執行一次。

工具	用途
`CronList`	列出所有排程任務及其 ID、排程和提示。
`CronDelete`	按 ID 取消任務。

每個排程任務都有一個 8 字元的 ID，您可以傳遞給 `CronDelete`。一個工作階段最多可以同時保存 50 個排程任務。

排程任務如何執行

排程器每秒檢查一次到期的任務，並以低優先級將其加入佇列。排程提示在您的回合之間執行，而不是在 Claude 正在回應時執行。如果 Claude 在任務到期時忙碌，提示會等到目前回合結束。

所有時間都以您的本地時區解釋。cron 表達式（例如 `0 9 * * *`）表示您執行 Claude Code 的任何地方的上午 9 點，而不是 UTC。

抖動

為了避免每個工作階段在同一牆上時刻點擊 API，排程器會為執行時間添加一個小的確定性偏移：

- 重複任務的執行時間最多晚於其週期的 10%，上限為 15 分鐘。每小時的工作可能在 `:00` 到 `:06` 之間的任何時間執行。
- 為小時頂部或底部排程的一次性任務最多提前 90 秒執行。

偏移是從任務 ID 衍生的，因此相同的任務始終獲得相同的偏移。如果精確時序很重要，請選擇不是 `:00` 或 `:30` 的分鐘，例如 `3 9 * * *` 而不是 `0 9 * * *`，一次性抖動將不適用。

三天過期

重複任務在建立後 3 天自動過期。任務最後執行一次，然後刪除自己。這限制了被遺忘的迴圈可以執行多長時間。如果您需要重複任務持續更長時間，請在過期前取消並重新建立它，或使用 Desktop 排程任務 進行持久排程。

Cron 表達式參考

`CronCreate` 接受標準 5 欄位 cron 表達式：`minute hour day-of-month month day-of-week`。所有欄位都支援萬用字元（`*`）、單一值（`5`）、步驟（`*/15`）、範圍（`1-5`）和逗號分隔清單（`1,15,30`）。

範例	含義
`*/5 * * * *`	每 5 分鐘
`0 * * * *`	每小時整點
`7 * * * *`	每小時的第 7 分鐘
`0 9 * * *`	每天上午 9 點（本地時間）
`0 9 * * 1-5`	工作日上午 9 點（本地時間）
`30 14 15 3 *`	3 月 15 日下午 2:30（本地時間）

星期幾使用 `0` 或 `7` 表示星期日，`6` 表示星期六。不支援擴展語法，例如 `L`、`W`、`?` 和名稱別名，例如 `MON` 或 `JAN`。

當月份日期和星期幾都受到限制時，如果任一欄位匹配，日期就匹配。這遵循標準 vixie-cron 語義。

停用排程任務

在您的環境中設定 `CLAUDE_CODE_DISABLE_CRON=1` 以完全停用排程器。cron 工具和 `/loop` 變得不可用，任何已排程的任務都停止執行。請參閱 環境變數 以取得完整的停用旗標清單。

限制

工作階段範圍的排程有固有的限制：

- 任務只在 Claude Code 執行且閒置時執行。關閉終端或讓工作階段退出會取消所有內容。
- 沒有錯過執行的追趕。如果任務的排程時間在 Claude 忙於長時間執行的請求時經過，它會在 Claude 變為閒置時執行一次，而不是每個錯過的間隔執行一次。
- 重新啟動時沒有持久性。重新啟動 Claude Code 會清除所有工作階段範圍的任務。

對於需要無人值守執行的 cron 驅動自動化，請使用具有 `schedule` 觸發器的 GitHub Actions 工作流程，或如果您想要圖形化設定流程，請使用 Desktop 排程任務。

Code Review

> 設定自動化 PR 審查，使用多代理分析您的完整程式碼庫來捕捉邏輯錯誤、安全漏洞和迴歸

Note:

Code Review 處於研究預覽階段，適用於 Teams 和 Enterprise 訂閱。不適用於啟用 Zero Data Retention 的組織。

Code Review 分析您的 GitHub pull request，並在發現問題的程式碼行上發佈內聯評論。一群專門的代理在您完整程式碼庫的背景下檢查程式碼變更，尋找邏輯錯誤、安全漏洞、破損的邊界情況和細微的迴歸。

發現結果按嚴重程度標記，不會批准或阻止您的 PR，因此現有的審查工作流程保持不變。您可以通過在您的儲存庫中新增 `CLAUDE.md` 或 `REVIEW.md` 檔案來調整 Claude 標記的內容。

若要在您自己的 CI 基礎設施中執行 Claude 而不是此託管服務，請參閱 GitHub Actions 或 GitLab CI/CD。

本頁涵蓋：

- 審查如何運作
- 設定
- 自訂審查，使用 `CLAUDE.md` 和 `REVIEW.md`
- 定價

審查如何運作

一旦管理員為您的組織啟用 Code Review，當 pull request 開啟或更新時，審查會自動執行。多個代理在 Anthropic 基礎設施上並行分析差異和周圍程式碼。每個代理尋找不同類別的問題，然後驗證步驟根據實際程式碼行為檢查候選項以過濾掉誤報。結果被去重複、按嚴重程度排名，並作為內聯評論發佈在發現問題的特定行上。如果未發現問題，Claude 會在 PR 上發佈簡短的確認評論。

審查成本隨著 PR 大小和複雜性而擴展，平均在 20 分鐘內完成。管理員可以通過分析儀表板監控審查活動和支出。

嚴重程度級別

每個發現都標記有嚴重程度級別：

標記	嚴重程度	含義
🔴	Normal	應在合併前修復的錯誤
🟡	Nit	輕微問題，值得修復但不阻止
🟣	Pre-existing	程式碼庫中存在但未由此 PR 引入的錯誤

發現包括可摺疊的擴展推理部分，您可以展開以了解 Claude 為什麼標記該問題以及它如何驗證問題。

Code Review 檢查的內容

預設情況下，Code Review 專注於正確性：會破壞生產的錯誤，而不是格式設定偏好或缺少的測試覆蓋。您可以通過新增指導檔案到您的儲存庫來擴展它檢查的內容。

設定 Code Review

管理員為組織啟用 Code Review 一次，並選擇要包含的儲存庫。

Step 1: 開啟 Claude Code 管理員設定

前往 claude.ai/admin-settings/claude-code 並找到 Code Review 部分。您需要對您的 Claude 組織具有管理員存取權限，以及在您的 GitHub 組織中安裝 GitHub Apps 的權限。

Step 2: 開始設定

點擊**設定**。這開始 GitHub App 安裝流程。

Step 3: 安裝 Claude GitHub App

按照提示將 Claude GitHub App 安裝到您的 GitHub 組織。該應用程式要求這些儲存庫權限：

- **Contents**：讀取和寫入
- **Issues**：讀取和寫入
- **Pull requests**：讀取和寫入

Code Review 使用對內容的讀取存取權限和對 pull request 的寫入存取權限。更廣泛的權限集也支援 GitHub Actions（如果您稍後啟用）。

Step 4: 選擇儲存庫

選擇要為 Code Review 啟用的儲存庫。如果您看不到儲存庫，請確保您在安裝期間給予 Claude GitHub App 存取權限。您可以稍後新增更多儲存庫。

Step 5: 設定每個儲存庫的審查觸發器

設定完成後，Code Review 部分在表格中顯示您的儲存庫。對於每個儲存庫，使用下拉式選單選擇審查執行的時間：

- **僅在 PR 建立後**：當 PR 開啟或標記為準備審查時，審查執行一次
- **在每次推送到 PR 分支後**：審查在每次推送時執行，在 PR 演變時捕捉新問題，並在您修復標記的問題時自動解決執行緒

在每次推送時審查會執行更多審查並花費更多。從 PR 建立開始，然後對於您想要持續覆蓋和自動執行緒清理的儲存庫切換到推送時。

儲存庫表格還根據最近的活動顯示每個儲存庫的平均審查成本。使用行操作選單按儲存庫開啟或關閉 Code Review，或完全移除儲存庫。

若要驗證設定，請開啟測試 PR。名為 **Claude Code Review** 的檢查執行會在幾分鐘內出現。如果沒有，請確認儲存庫列在您的管理員設定中，並且 Claude GitHub App 有權存取它。

自訂審查

Code Review 從您的儲存庫讀取兩個檔案來指導它標記的內容。兩者都是在預設正確性檢查之上的附加項：

- `CLAUDE.md`：Claude Code 用於所有任務（不僅僅是審查）的共享專案指示。當指導也適用於互動式 Claude Code 工作階段時使用它。
- `REVIEW.md`：僅審查指導，在程式碼審查期間專門讀取。對於嚴格關於在審查期間標記或跳過什麼的規則，以及會使您的一般 `CLAUDE.md` 混亂的規則，使用它。

CLAUDE.md

Code Review 讀取您儲存庫的 `CLAUDE.md` 檔案，並將新引入的違規視為 nit 級別的發現。這是雙向工作的：如果您的 PR 以使 `CLAUDE.md` 陳述過時的方式更改程式碼，Claude 會標記文件需要更新。

Claude 在您目錄層次結構的每個級別讀取 `CLAUDE.md` 檔案，因此子目錄的 `CLAUDE.md` 中的規則僅適用於該路徑下的檔案。有關 `CLAUDE.md` 如何運作的更多資訊，請參閱記憶體文件。

對於您不想應用於一般 Claude Code 工作階段的審查特定指導，請改用 `REVIEW.md`。

REVIEW.md

將 `REVIEW.md` 檔案新增到您的儲存庫根目錄以獲取審查特定規則。使用它來編碼：

- 公司或團隊風格指南：“優先使用早期返回而不是嵌套條件”
- 語言或框架特定的約定，不由 linter 涵蓋
- Claude 應始終標記的事項：“任何新 API 路由必須有整合測試”
- Claude 應跳過的事項：“不要評論 `/gen/` 下生成程式碼中的格式設定”

範例 `REVIEW.md`：

```
## 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 在儲存庫根目錄自動發現 `REVIEW.md`。無需配置。

檢視使用情況

前往 claude.ai/analytics/code-review 以查看整個組織的 Code Review 活動。儀表板顯示：

部分	它顯示什麼
PRs reviewed	在選定時間範圍內審查的 pull request 的每日計數
Cost weekly	Code Review 的每週支出
Feedback	因開發人員解決問題而自動解決的審查評論計數
Repository breakdown	每個儲存庫的審查 PR 計數和已解決評論

管理員設定中的儲存庫表格也顯示每個儲存庫的平均審查成本。

定價

Code Review 根據令牌使用情況計費。審查平均 $15-25，隨著 PR 大小、程式碼庫複雜性和需要驗證的問題數量而擴展。Code Review 使用通過額外使用單獨計費，不計入您計劃的包含使用量。

您選擇的審查觸發器會影響總成本：

- **僅在 PR 建立後**：每個 PR 執行一次
- **在每次推送時**：在每次提交時執行，將成本乘以推送次數

無論您的組織是否為其他 Claude Code 功能使用 AWS Bedrock 或 Google Vertex AI，成本都會出現在您的 Anthropic 帳單上。若要為 Code Review 設定每月支出上限，請前往 claude.ai/admin-settings/usage 並為 Claude Code Review 服務配置限制。

通過分析中的每週成本圖表或管理員設定中的每個儲存庫平均成本欄監控支出。

相關資源

Code Review 設計用於與 Claude Code 的其餘部分一起工作。如果您想在開啟 PR 之前在本地執行審查、需要自託管設定，或想深入了解 `CLAUDE.md` 如何在工具中塑造 Claude 的行為，這些頁面是很好的下一步：

- Plugins：瀏覽外掛程式市場，包括用於在推送前在本地執行按需審查的 `code-review` 外掛程式
- GitHub Actions：在您自己的 GitHub Actions 工作流程中執行 Claude，以實現超越程式碼審查的自訂自動化
- GitLab CI/CD：GitLab 管道的自託管 Claude 整合
- Memory：`CLAUDE.md` 檔案如何在 Claude Code 中工作
- Analytics：追蹤超越程式碼審查的 Claude Code 使用情況

Part 7: CI/CD & Integrations

Claude Code GitHub Actions

> 了解如何將 Claude Code 整合到您的開發工作流程中，使用 Claude Code GitHub Actions

Claude Code GitHub Actions 為您的 GitHub 工作流程帶來 AI 驅動的自動化。只需在任何 PR 或議題中提及 `@claude`，Claude 就可以分析您的程式碼、建立 pull request、實現功能和修復錯誤 - 同時遵循您專案的標準。如需在每個 PR 上自動發佈評論而無需觸發，請參閱 GitHub Code Review。

> **Note:**
>
> Claude Code GitHub Actions 建立在 Claude Agent SDK 之上，該 SDK 可實現 Claude Code 與您的應用程式的程式化整合。您可以使用 SDK 來建立超越 GitHub Actions 的自訂自動化工作流程。

> **Info:**
>
> **Claude Opus 4.6 現已推出。** Claude Code GitHub Actions 預設使用 Sonnet。若要使用 Opus 4.6，請設定 model 參數以使用 `claude-opus-4-6`。

為什麼使用 Claude Code GitHub Actions？

- **即時 PR 建立**：描述您需要的內容，Claude 會建立包含所有必要變更的完整 PR
- **自動化程式碼實現**：使用單一命令將議題轉換為可運作的程式碼
- **遵循您的標準**：Claude 尊重您的 `CLAUDE.md` 指南和現有程式碼模式
- **簡單設定**：使用我們的安裝程式和 API 金鑰在幾分鐘內開始使用
- **預設安全**：您的程式碼保留在 Github 的執行器上

Claude 可以做什麼？

Claude Code 提供了一個強大的 GitHub Action，改變了您使用程式碼的方式：

Claude Code Action

此 GitHub Action 允許您在 GitHub Actions 工作流程中執行 Claude Code。您可以使用此功能在 Claude Code 之上建立任何自訂工作流程。

檢視儲存庫 →

設定

快速設定

設定此 action 的最簡單方法是透過終端機中的 Claude Code。只需開啟 claude 並執行 `/install-github-app`。

此命令將引導您完成 GitHub app 和所需密鑰的設定。

> **Note:**
>
> - 您必須是儲存庫管理員才能安裝 GitHub app 並新增密鑰
> - GitHub app 將要求對內容、議題和 Pull request 的讀取和寫入權限
> - 此快速入門方法僅適用於直接 Claude API 使用者。如果您使用 AWS Bedrock 或 Google Vertex AI，請參閱 使用 AWS Bedrock 和 Google Vertex AI 部分。

手動設定

如果 `/install-github-app` 命令失敗或您偏好手動設定，請遵循以下手動設定說明：

1. **安裝 Claude GitHub app** 到您的儲存庫：https://github.com/apps/claude
   Claude GitHub app 需要以下儲存庫權限：
   - **Contents**：讀取和寫入（修改儲存庫檔案）
   - **Issues**：讀取和寫入（回應議題）
   - **Pull requests**：讀取和寫入（建立 PR 和推送變更）

   如需有關安全性和權限的更多詳細資訊，請參閱 安全性文件。
2. **新增 ANTHROPIC_API_KEY** 到您的儲存庫密鑰（了解如何在 GitHub Actions 中使用密鑰）
3. **複製工作流程檔案** 從 examples/claude.yml 到您的儲存庫的 `.github/workflows/`

> **Tip:**
>
> 完成快速入門或手動設定後，透過在議題或 PR 評論中標記 `@claude` 來測試 action。

從 Beta 升級

> **Warning:**
>
> Claude Code GitHub Actions v1.0 引入了重大變更，需要更新您的工作流程檔案才能從 beta 版本升級到 v1.0。

如果您目前使用 Claude Code GitHub Actions 的 beta 版本，我們建議您更新工作流程以使用 GA 版本。新版本簡化了設定，同時新增了強大的新功能，如自動模式偵測。

基本變更

所有 beta 使用者必須對其工作流程檔案進行這些變更才能升級：

1. **更新 action 版本**：將 `@beta` 變更為 `@v1`
2. **移除模式設定**：刪除 `mode: "tag"` 或 `mode: "agent"`（現在自動偵測）
3. **更新提示輸入**：將 `direct_prompt` 替換為 `prompt`
4. **移動 CLI 選項**：將 `max_turns`、`model`、`custom_instructions` 等轉換為 `claude_args`

重大變更參考

舊 Beta 輸入	新 v1.0 輸入
`mode`	（已移除 - 自動偵測）
`direct_prompt`	`prompt`
`override_prompt`	`prompt` 搭配 GitHub 變數
`custom_instructions`	`claude_args: --append-system-prompt`
`max_turns`	`claude_args: --max-turns`
`model`	`claude_args: --model`
`allowed_tools`	`claude_args: --allowedTools`
`disallowed_tools`	`claude_args: --disallowedTools`
`claude_env`	`settings` JSON 格式

前後範例

Beta 版本：

```yaml
- 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 版本 (v1.0)：

```yaml
- 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:**
>
> 該 action 現在會根據您的設定自動偵測是否在互動模式（回應 `@claude` 提及）或自動化模式（立即使用提示執行）中執行。

範例使用案例

Claude Code GitHub Actions 可以幫助您完成各種任務。examples 目錄包含適用於不同情境的現成工作流程。

基本工作流程

```yaml
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
```

使用 skills

```yaml
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"
```

使用提示的自訂自動化

```yaml
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"
```

常見使用案例

在議題或 PR 評論中：

```
@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 將自動分析上下文並做出適當的回應。

最佳實踐

CLAUDE.md 設定

在您的儲存庫根目錄建立 `CLAUDE.md` 檔案，以定義程式碼風格指南、審查標準、專案特定規則和偏好的模式。此檔案指導 Claude 對您的專案標準的理解。

安全考量

> **Warning:**
>
> 永遠不要直接將 API 金鑰提交到您的儲存庫。

如需包括權限、身份驗證和最佳實踐的全面安全指導，請參閱 Claude Code Action 安全性文件。

始終使用 GitHub Secrets 來存放 API 金鑰：

- 將您的 API 金鑰新增為名為 `ANTHROPIC_API_KEY` 的儲存庫密鑰
- 在工作流程中參考它：`anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}`
- 將 action 權限限制為僅必要的權限
- 在合併前審查 Claude 的建議

始終使用 GitHub Secrets（例如 `${{ secrets.ANTHROPIC_API_KEY }}`）而不是直接在工作流程檔案中硬編碼 API 金鑰。

最佳化效能

使用議題範本提供上下文，保持您的 `CLAUDE.md` 簡潔且專注，並為您的工作流程設定適當的逾時。

CI 成本

使用 Claude Code GitHub Actions 時，請注意相關成本：

GitHub Actions 成本：

- Claude Code 在 GitHub 託管的執行器上執行，這會消耗您的 GitHub Actions 分鐘數
- 請參閱 GitHub 的計費文件以了解詳細的定價和分鐘限制

API 成本：

- 每次 Claude 互動都會根據提示和回應的長度消耗 API 令牌
- 令牌使用量因任務複雜性和程式碼庫大小而異
- 請參閱 Claude 的定價頁面以了解目前的令牌費率

成本最佳化提示：

- 使用特定的 `@claude` 命令來減少不必要的 API 呼叫
- 在 `claude_args` 中設定適當的 `--max-turns` 以防止過度迭代
- 設定工作流程級別的逾時以避免失控的工作
- 考慮使用 GitHub 的並行控制來限制平行執行

設定範例

Claude Code Action v1 使用統一參數簡化了設定：

```
- 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
```

主要功能：

- **統一提示介面** - 對所有指令使用 `prompt`
- **Skills** - 直接從提示中呼叫已安裝的 skills
- **CLI 傳遞** - 透過 `claude_args` 的任何 Claude Code CLI 引數
- **靈活的觸發器** - 適用於任何 GitHub 事件

訪問 examples 目錄以取得完整的工作流程檔案。

> **Tip:**
>
> 當回應議題或 PR 評論時，Claude 會自動回應 @claude 提及。對於其他事件，使用 `prompt` 參數來提供指令。

使用 AWS Bedrock 和 Google Vertex AI

對於企業環境，您可以使用 Claude Code GitHub Actions 搭配您自己的雲端基礎設施。此方法讓您可以控制資料駐留和計費，同時保持相同的功能。

先決條件

在使用雲端提供者設定 Claude Code GitHub Actions 之前，您需要：

對於 Google Cloud Vertex AI：

1. 啟用了 Vertex AI 的 Google Cloud 專案
2. 為 GitHub Actions 設定的工作負載身份聯盟
3. 具有所需權限的服務帳戶
4. GitHub App（建議）或使用預設 GITHUB_TOKEN

對於 AWS Bedrock：

1. 啟用了 Amazon Bedrock 的 AWS 帳戶
2. 在 AWS 中設定的 GitHub OIDC 身份提供者
3. 具有 Bedrock 權限的 IAM 角色

4. GitHub App（建議）或使用預設 GITHUB_TOKEN

Step 1: 建立自訂 GitHub App（建議用於第三方提供者）

為了在使用 Vertex AI 或 Bedrock 等第三方提供者時獲得最佳控制和安全性，我們建議建立您自己的 GitHub App：

1. 前往 https://github.com/settings/apps/new
2. 填寫基本資訊：

- **GitHub App 名稱**：選擇唯一的名稱（例如’YourOrg Claude Assistant’）
- **首頁 URL**：您的組織網站或儲存庫 URL

1. 設定 app 設定：

- **Webhooks**：取消勾選’Active’（此整合不需要）

1. 設定所需的權限：

- **儲存庫權限**：
- Contents：讀取和寫入
- Issues：讀取和寫入
- Pull requests：讀取和寫入

1. 點擊’Create GitHub App’
2. 建立後，點擊’Generate a private key’並儲存下載的 `.pem` 檔案
3. 從 app 設定頁面記下您的 App ID
4. 將 app 安裝到您的儲存庫：

- 從您的 app 設定頁面，點擊左側邊欄中的’Install App’
- 選擇您的帳戶或組織
- 選擇’Only select repositories’並選擇特定儲存庫
- 點擊’Install’

1. 將私鑰新增為儲存庫密鑰：

- 前往您的儲存庫的 Settings → Secrets and variables → Actions
- 建立名為 `APP_PRIVATE_KEY` 的新密鑰，內容為 `.pem` 檔案的內容

1. 將 App ID 新增為密鑰：

- 建立名為 `APP_ID` 的新密鑰，內容為您的 GitHub App 的 ID

> **Note:**

> 此 app 將與 actions/create-github-app-token action 一起使用，以在您的工作流程中產生身份驗證令牌。

Claude API 的替代方案或如果您不想設定自己的 Github app：使用官方 Anthropic app：

1. 從以下位置安裝：https://github.com/apps/claude
2. 無需額外的身份驗證設定

Step 2: 設定雲端提供者身份驗證

選擇您的雲端提供者並設定安全的身份驗證：

設定 AWS 以允許 GitHub Actions 安全地進行身份驗證，而無需儲存認證。

> **安全性注意**：使用儲存庫特定的設定並僅授予最少所需的權限。

必需的設定：

1. **啟用 Amazon Bedrock**：

- 請求在 Amazon Bedrock 中存取 Claude 模型
- 對於跨區域模型，請在所有必需的區域中請求存取

1. **設定 GitHub OIDC 身份提供者**：

- 提供者 URL：`https://token.actions.githubusercontent.com`
- 受眾：`sts.amazonaws.com`

1. **為 GitHub Actions 建立 IAM 角色**：

- 受信任的實體類型：Web 身份
- 身份提供者：`token.actions.githubusercontent.com`
- 權限：`AmazonBedrockFullAccess` 政策
- 為您的特定儲存庫設定信任政策

必需的值：

設定後，您將需要：

- **AWS_ROLE_TO_ASSUME**：您建立的 IAM 角色的 ARN

> **Tip:**
>
> OIDC 比使用靜態 AWS 存取金鑰更安全，因為認證是臨時的並自動輪換。

請參閱 AWS 文件以取得詳細的 OIDC 設定說明。

設定 Google Cloud 以允許 GitHub Actions 安全地進行身份驗證，而無需儲存認證。

> **安全性注意**：使用儲存庫特定的設定並僅授予最少所需的權限。

必需的設定：

1. **在您的 Google Cloud 專案中啟用 API**：

- IAM Credentials API
- Security Token Service (STS) API
- Vertex AI API

1. **建立工作負載身份聯盟資源**：

- 建立工作負載身份池
- 新增 GitHub OIDC 提供者，具有：
- 簽發者：`https://token.actions.githubusercontent.com`
- 儲存庫和擁有者的屬性對應
- **安全性建議**：使用儲存庫特定的屬性條件

1. **建立服務帳戶**：

- 僅授予 `Vertex AI User` 角色
- **安全性建議**：為每個儲存庫建立專用服務帳戶

1. **設定 IAM 繫結**：

- 允許工作負載身份池模擬服務帳戶
- **安全性建議**：使用儲存庫特定的主體集

必需的值：

設定後，您將需要：

- **GCP_WORKLOAD_IDENTITY_PROVIDER**：完整的提供者資源名稱
- **GCP_SERVICE_ACCOUNT**：服務帳戶電子郵件地址

> **Tip:**
>
> 工作負載身份聯盟消除了對可下載服務帳戶金鑰的需求，提高了安全性。

如需詳細的設定說明，請參閱 Google Cloud 工作負載身份聯盟文件。

Step 3: 新增必需的密鑰

將以下密鑰新增到您的儲存庫（Settings → Secrets and variables → Actions）：

對於 Claude API（直接）：

1. **對於 API 身份驗證：**

- `ANTHROPIC_API_KEY`：您的 Claude API 金鑰，來自 console.anthropic.com

1. **對於 GitHub App（如果使用您自己的 app）：**

- `APP_ID`：您的 GitHub App 的 ID
- `APP_PRIVATE_KEY`：私鑰 (.pem) 內容

對於 Google Cloud Vertex AI

1. **對於 GCP 身份驗證：**

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

1. **對於 GitHub App（如果使用您自己的 app）：**

- `APP_ID`：您的 GitHub App 的 ID
- `APP_PRIVATE_KEY`：私鑰 (.pem) 內容

對於 AWS Bedrock

1. **對於 AWS 身份驗證：**

- `AWS_ROLE_TO_ASSUME`

1. **對於 GitHub App（如果使用您自己的 app）：**

- `APP_ID`：您的 GitHub App 的 ID
- `APP_PRIVATE_KEY`：私鑰 (.pem) 內容

Step 4: 建立工作流程檔案

建立與您的雲端提供者整合的 GitHub Actions 工作流程檔案。以下範例顯示了 AWS Bedrock 和 Google Vertex AI 的完整設定：

先決條件：

- 啟用了 AWS Bedrock 存取且具有 Claude 模型權限
- GitHub 在 AWS 中設定為 OIDC 身份提供者
- 具有 Bedrock 權限且信任 GitHub Actions 的 IAM 角色

必需的 GitHub 密鑰：

密鑰名稱	描述
`AWS_ROLE_TO_ASSUME`	Bedrock 存取的 IAM 角色的 ARN
`APP_ID`	您的 GitHub App ID（來自 app 設定）
`APP_PRIVATE_KEY`	您為 GitHub App 產生的私鑰

```yaml
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 的模型 ID 格式包括區域前綴（例如 `us.anthropic.claude-sonnet-4-6`）。

先決條件：

- 在您的 GCP 專案中啟用了 Vertex AI API
- 為 GitHub 設定的工作負載身份聯盟
- 具有 Vertex AI 權限的服務帳戶

必需的 GitHub 密鑰：

密鑰名稱	描述
`GCP_WORKLOAD_IDENTITY_PROVIDER`	工作負載身份提供者資源名稱
`GCP_SERVICE_ACCOUNT`	具有 Vertex AI 存取權限的服務帳戶電子郵件
`APP_ID`	您的 GitHub App ID（來自 app 設定）
`APP_PRIVATE_KEY`	您為 GitHub App 產生的私鑰

```yaml
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:**
>
> 專案 ID 會自動從 Google Cloud 身份驗證步驟中擷取，因此您無需硬編碼它。

故障排除

Claude 不回應 @claude 命令

驗證 GitHub App 是否正確安裝，檢查工作流程是否已啟用，確保 API 金鑰已在儲存庫密鑰中設定，並確認評論包含 `@claude`（不是 `/claude`）。

CI 不在 Claude 的提交上執行

確保您使用的是 GitHub App 或自訂 app（不是 Actions 使用者），檢查工作流程觸發器是否包括必要的事件，並驗證 app 權限是否包括 CI 觸發器。

身份驗證錯誤

確認 API 金鑰有效且具有足夠的權限。對於 Bedrock/Vertex，檢查認證設定並確保密鑰在工作流程中命名正確。

進階設定

Action 參數

Claude Code Action v1 使用簡化的設定：

參數	描述	必需
`prompt`	Claude 的指令（純文字或 skill 名稱）	否*
`claude_args`	傳遞給 Claude Code 的 CLI 引數	否
`anthropic_api_key`	Claude API 金鑰	是**
`github_token`	用於 API 存取的 GitHub 令牌	否
`trigger_phrase`	自訂觸發短語（預設：「@claude」）	否
`use_bedrock`	使用 AWS Bedrock 而不是 Claude API	否
`use_vertex`	使用 Google Vertex AI 而不是 Claude API	否

*提示是可選的 - 當在議題/PR 評論中省略時，Claude 回應觸發短語
**對於直接 Claude API 是必需的，對於 Bedrock/Vertex 不是必需的

傳遞 CLI 引數

`claude_args` 參數接受任何 Claude Code CLI 引數：

```
claude_args: "--max-turns 5 --model claude-sonnet-4-6 --mcp-config /path/to/
config.json"
```

常見引數：

- `--max-turns`：最大對話輪數（預設：10）
- `--model`：要使用的模型（例如 `claude-sonnet-4-6`）
- `--mcp-config`：MCP 設定的路徑
- `--allowed-tools`：允許的工具的逗號分隔清單
- `--debug`：啟用偵錯輸出

替代整合方法

雖然 `/install-github-app` 命令是推薦的方法，但您也可以：

- **自訂 GitHub App**：對於需要品牌使用者名稱或自訂身份驗證流程的組織。建立您自己的 GitHub App，具有所需的權限（contents、issues、pull requests），並使用 actions/create-github-app-token action 在您的工作流程中產生令牌。
- **手動 GitHub Actions**：直接工作流程設定以獲得最大靈活性
- **MCP 設定**：Model Context Protocol 伺服器的動態載入

請參閱 Claude Code Action 文件以取得有關身份驗證、安全性和進階設定的詳細指南。

自訂 Claude 的行為

您可以透過兩種方式自訂 Claude 的行為：

1. **CLAUDE.md**：在您的儲存庫根目錄中的 `CLAUDE.md` 檔案中定義編碼標準、審查標準和專案特定規則。Claude 在建立 PR 和回應請求時將遵循這些指南。請查看我們的 Memory 文件以取得更多詳細資訊。
2. **自訂提示**：在工作流程檔案中使用 `prompt` 參數來提供工作流程特定的指令。這允許您為不同的工作流程或任務自訂 Claude 的行為。

Claude 在建立 PR 和回應請求時將遵循這些指南。

Claude Code GitLab CI/CD

> 了解如何將 Claude Code 整合到您的開發工作流程中，使用 GitLab CI/CD

Info:

Claude Code for GitLab CI/CD 目前處於測試版。隨著我們改進體驗，功能和功能可能會演變。

此整合由 GitLab 維護。如需支援，請參閱以下 GitLab issue。

Note:

此整合建立在 Claude Code CLI and Agent SDK 之上，可在您的 CI/CD 工作和自訂自動化工作流程中以程式設計方式使用 Claude。

為什麼要在 GitLab 中使用 Claude Code？

- **即時 MR 建立**：描述您需要的內容，Claude 會提出完整的 MR，包括變更和說明
- **自動化實現**：使用單一命令或提及將問題轉變為可運作的程式碼
- **專案感知**：Claude 遵循您的 `CLAUDE.md` 指南和現有程式碼模式
- **簡單設定**：在 `.gitlab-ci.yml` 中新增一個工作和一個遮罩 CI/CD 變數
- **企業就緒**：選擇 Claude API、AWS Bedrock 或 Google Vertex AI 以滿足資料駐留和採購需求
- **預設安全**：在您的 GitLab runners 中執行，具有您的分支保護和核准

運作方式

Claude Code 使用 GitLab CI/CD 在隔離的工作中執行 AI 任務，並透過 MR 將結果提交回去：

1. **事件驅動的編排**：GitLab 監聽您選擇的觸發器（例如，在問題、MR 或審查執行緒中提及 `@claude` 的評論）。該工作從執行緒和儲存庫收集上下文，從該輸入建立提示，並執行 Claude Code。
2. **提供者抽象化**：使用適合您環境的提供者：
   - Claude API (SaaS)
   - AWS Bedrock (基於 IAM 的存取、跨區域選項)

    - Google Vertex AI (GCP 原生、Workload Identity Federation)
3. **沙箱執行**：每次互動都在具有嚴格網路和檔案系統規則的容器中執行。Claude Code 強制執行工作區範圍的權限以限制寫入。每項變更都透過 MR 流動，以便審查者看到差異並且核准仍然適用。

選擇區域端點以減少延遲並滿足資料主權要求，同時使用現有的雲端協議。

Claude 可以做什麼？

Claude Code 啟用強大的 CI/CD 工作流程，改變您與程式碼的互動方式：

- 從問題描述或評論建立和更新 MR
- 分析效能回歸並提出最佳化建議
- 直接在分支中實現功能，然後開啟 MR
- 修復由測試或評論識別的錯誤和回歸
- 回應後續評論以反覆進行請求的變更

設定

快速設定

最快的入門方式是在您的 `.gitlab-ci.yml` 中新增最小工作，並將您的 API 金鑰設定為遮罩變數。

1. **新增遮罩 CI/CD 變數**
    - 前往 **Settings** → **CI/CD** → **Variables**
    - 新增 `ANTHROPIC_API_KEY`（遮罩，根據需要保護）
2. **在 `.gitlab-ci.yml` 中新增 Claude 工作**

```yaml
stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  # 調整規則以符合您想要觸發工作的方式：
  # - 手動執行
  # - 合併請求事件
  # - 當評論包含 '@claude' 時的 web/API 觸發
  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:
    # 選用：如果您的設定提供，啟動 GitLab MCP server
    - /bin/gitlab-mcp-server || true
    # 透過 web/API 觸發器使用 AI_FLOW_* 變數時，使用上下文負載
    - 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
```

新增工作和您的 `ANTHROPIC_API_KEY` 變數後，透過 **CI/CD** → **Pipelines** 手動執行工作進行測試，或從 MR 觸發它，讓 Claude 在分支中提出更新並在需要時開啟 MR。

> **Note:**
>
> 若要改為在 AWS Bedrock 或 Google Vertex AI 上執行而不是 Claude API，請參閱下方的 Using with AWS Bedrock & Google Vertex AI 部分，了解驗證和環境設定。

手動設定（建議用於生產）

如果您偏好更受控的設定或需要企業提供者：

1. **設定提供者存取**：
   - **Claude API**：建立並將 `ANTHROPIC_API_KEY` 儲存為遮罩 CI/CD 變數
   - **AWS Bedrock**：**Configure GitLab → AWS OIDC** 並為 Bedrock 建立 IAM 角色
   - **Google Vertex AI**：**Configure Workload Identity Federation for GitLab → GCP**
2. **為 GitLab API 操作新增專案認證**：
   - 預設使用 `CI_JOB_TOKEN`，或建立具有 `api` 範圍的專案存取令牌
   - 如果使用 PAT，將其儲存為 `GITLAB_ACCESS_TOKEN`（遮罩）
3. **在 `.gitlab-ci.yml` 中新增 Claude 工作**（請參閱下方的範例）
4. **（選用）啟用提及驅動的觸發器**：
   - 為「Comments (notes)」新增專案 webhook 到您的事件監聽器（如果您使用的話）
   - 當評論包含 `@claude` 時，讓監聽器使用 `AI_FLOW_INPUT` 和 `AI_FLOW_CONTEXT` 等變數呼叫管道觸發 API

範例使用案例

將問題轉變為 MR

在問題評論中：

```
@claude implement this feature based on the issue description
```

Claude 分析問題和程式碼庫，在分支中寫入變更，並開啟 MR 供審查。

獲得實現幫助

在 MR 討論中：

```
@claude suggest a concrete approach to cache the results of this API call
```

Claude 提出變更，新增具有適當快取的程式碼，並更新 MR。

快速修復錯誤

在問題或 MR 評論中：

```
@claude fix the TypeError in the user dashboard component
```

Claude 定位錯誤，實現修復，並更新分支或開啟新的 MR。

使用 AWS Bedrock 和 Google Vertex AI

對於企業環境，您可以在您的雲端基礎設施上完全執行 Claude Code，具有相同的開發人員體驗。

AWS Bedrock

先決條件

在使用 AWS Bedrock 設定 Claude Code 之前，您需要：

1. 具有 Amazon Bedrock 存取權限的 AWS 帳戶，可存取所需的 Claude 模型
2. 在 AWS IAM 中設定為 OIDC 身份提供者的 GitLab
3. 具有 Bedrock 權限和信任政策的 IAM 角色，限制於您的 GitLab 專案/refs
4. 用於角色假設的 GitLab CI/CD 變數：

- `AWS_ROLE_TO_ASSUME`（角色 ARN）
- `AWS_REGION`（Bedrock 區域）

設定說明

設定 AWS 以允許 GitLab CI 工作透過 OIDC 假設 IAM 角色（無靜態金鑰）。

必需的設定：

1. 啟用 Amazon Bedrock 並請求存取您的目標 Claude 模型
2. 為 GitLab 建立 IAM OIDC 提供者（如果尚未存在）
3. 建立由 GitLab OIDC 提供者信任的 IAM 角色，限制於您的專案和受保護的 refs
4. 為 Bedrock invoke API 附加最小權限

要儲存在 CI/CD 變數中的必需值：

- `AWS_ROLE_TO_ASSUME`
- `AWS_REGION`

在 Settings → CI/CD → Variables 中新增變數：

```
## 對於 AWS Bedrock：
- AWS_ROLE_TO_ASSUME
- AWS_REGION
```

使用上面的 AWS Bedrock 工作範例在執行時交換 GitLab 工作令牌以取得臨時 AWS 認證。

Google Vertex AI

先決條件

在使用 Google Vertex AI 設定 Claude Code 之前，您需要：

1. 具有以下內容的 Google Cloud 專案：

- 啟用 Vertex AI API
- 設定 Workload Identity Federation 以信任 GitLab OIDC

1. 僅具有所需 Vertex AI 角色的專用服務帳戶
2. 用於 WIF 的 GitLab CI/CD 變數：

- `GCP_WORKLOAD_IDENTITY_PROVIDER`（完整資源名稱）
- `GCP_SERVICE_ACCOUNT`（服務帳戶電子郵件）

設定說明

設定 Google Cloud 以允許 GitLab CI 工作透過 Workload Identity Federation 模擬服務帳戶。

必需的設定：

1. 啟用 IAM Credentials API、STS API 和 Vertex AI API
2. 為 GitLab OIDC 建立 Workload Identity Pool 和提供者
3. 建立具有 Vertex AI 角色的專用服務帳戶
4. 授予 WIF 主體權限以模擬服務帳戶

要儲存在 CI/CD 變數中的必需值：

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

在 Settings → CI/CD → Variables 中新增變數：

```
## 對於 Google Vertex AI：
- GCP_WORKLOAD_IDENTITY_PROVIDER
- GCP_SERVICE_ACCOUNT
- CLOUD_ML_REGION（例如，us-east5）
```

使用上面的 Google Vertex AI 工作範例在不儲存金鑰的情況下進行驗證。

設定範例

以下是您可以調整到您的管道的現成程式碼片段。

基本 .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 將使用 CI/CD 變數中的 ANTHROPIC_API_KEY
```

AWS Bedrock 工作範例 (OIDC)

先決條件：

- 啟用 Amazon Bedrock 並存取您選擇的 Claude 模型
- 在 AWS 中設定 GitLab OIDC，具有信任您的 GitLab 專案和 refs 的角色
- 具有 Bedrock 權限的 IAM 角色（建議最小權限）

必需的 CI/CD 變數：

- `AWS_ROLE_TO_ASSUME`：Bedrock 存取的 IAM 角色的 ARN
- `AWS_REGION`：Bedrock 區域（例如，`us-west-2`）

```
claude-bedrock:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apk add --no-cache bash curl jq git python3 py3-pip
    - pip install --no-cache-dir awscli
    - curl -fsSL https://claude.ai/install.sh | bash
    # 交換 GitLab OIDC 令牌以取得 AWS 認證
    - export AWS_WEB_IDENTITY_TOKEN_FILE="${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 的模型 ID 包括區域特定的前綴（例如，`us.anthropic.claude-sonnet-4-6`）。如果您的工作流程支援，透過您的工作設定或提示傳遞所需的模型。

Google Vertex AI 工作範例 (Workload Identity Federation)

先決條件：

- 在您的 GCP 專案中啟用 Vertex AI API
- 設定 Workload Identity Federation 以信任 GitLab OIDC
- 具有 Vertex AI 權限的服務帳戶

必需的 CI/CD 變數：

- `GCP_WORKLOAD_IDENTITY_PROVIDER`：完整提供者資源名稱
- `GCP_SERVICE_ACCOUNT`：服務帳戶電子郵件
- `CLOUD_ML_REGION`：Vertex 區域（例如，`us-east5`）

```
claude-vertex:
  stage: ai
  image: gcr.io/google.com/cloudsdktool/google-cloud-cli:slim
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apt-get update && apt-get install -y git && apt-get clean
    - curl -fsSL https://claude.ai/install.sh | bash
    # 透過 WIF 驗證到 Google Cloud（無下載的金鑰）
    - >
      gcloud auth login --cred-file=<(cat <<EOF
      {
        "type": "external_account",
        "audience": "${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:**
>
> 使用 Workload Identity Federation，您不需要儲存服務帳戶金鑰。使用儲存庫特定的信任條件和最小權限服務帳戶。

最佳實踐

CLAUDE.md 設定

在儲存庫根目錄建立 `CLAUDE.md` 檔案以定義編碼標準、審查標準和專案特定規則。Claude 在執行期間讀取此檔案並在提出變更時遵循您的慣例。

安全考量

永遠不要將 API 金鑰或雲端認證提交到您的儲存庫。始終使用 GitLab CI/CD 變數：

- 將 `ANTHROPIC_API_KEY` 新增為遮罩變數（並根據需要保護它）
- 盡可能使用提供者特定的 OIDC（無長期金鑰）
- 限制工作權限和網路出口
- 像審查任何其他貢獻者一樣審查 Claude 的 MR

最佳化效能

- 保持 `CLAUDE.md` 專注和簡潔
- 提供清晰的問題/MR 描述以減少反覆
- 設定合理的工作逾時以避免失控執行
- 在可能的情況下在 runners 中快取 npm 和套件安裝

CI 成本

使用 Claude Code 與 GitLab CI/CD 時，請注意相關成本：

- **GitLab Runner 時間**：
  - Claude 在您的 GitLab runners 上執行並消耗計算分鐘數
  - 有關詳細資訊，請參閱您的 GitLab 計畫的 runner 計費
- **API 成本**：
  - 每次 Claude 互動根據提示和回應大小消耗令牌
  - 令牌使用量因任務複雜性和程式碼庫大小而異
  - 有關詳細資訊，請參閱 Anthropic 定價

- **成本最佳化提示**：
  - 使用特定的 `@claude` 命令以減少不必要的轉換
  - 設定適當的 `max_turns` 和工作逾時值
  - 限制並行以控制平行執行

安全和治理

- 每個工作都在具有受限網路存取的隔離容器中執行
- Claude 的變更透過 MR 流動，以便審查者看到每個差異
- 分支保護和核准規則適用於 AI 生成的程式碼
- Claude Code 使用工作區範圍的權限以限制寫入
- 成本保持在您的控制下，因為您帶來自己的提供者認證

疑難排解

Claude 不回應 @claude 命令

- 驗證您的管道正在被觸發（手動、MR 事件或透過 note 事件監聽器/webhook）
- 確保 CI/CD 變數（`ANTHROPIC_API_KEY` 或雲端提供者設定）存在且未遮罩
- 檢查評論是否包含 `@claude`（不是 `/claude`）以及您的提及觸發器是否已設定

工作無法寫入評論或開啟 MR

- 確保 `CI_JOB_TOKEN` 對專案具有足夠的權限，或使用具有 `api` 範圍的專案存取令牌
- 檢查 `mcp__gitlab` 工具是否在 `--allowedTools` 中啟用
- 確認工作在 MR 的上下文中執行或透過 `AI_FLOW_*` 變數有足夠的上下文

驗證錯誤

- **對於 Claude API**：確認 `ANTHROPIC_API_KEY` 有效且未過期
- **對於 Bedrock/Vertex**：驗證 OIDC/WIF 設定、角色模擬和祕密名稱；確認區域和模型可用性

進階設定

常見參數和變數

Claude Code 支援這些常用輸入：

- `prompt` / `prompt_file`：內聯提供說明（`-p`）或透過檔案
- `max_turns`：限制來回反覆的次數

- `timeout_minutes`：限制總執行時間
- `ANTHROPIC_API_KEY`：Claude API 所需（不用於 Bedrock/Vertex）
- 提供者特定環境：`AWS_REGION`、Vertex 的專案/區域變數

> **Note:**
>
> 確切的旗標和參數可能因 `@anthropic-ai/claude-code` 的版本而異。在您的工作中執行 `claude --help` 以查看支援的選項。

自訂 Claude 的行為

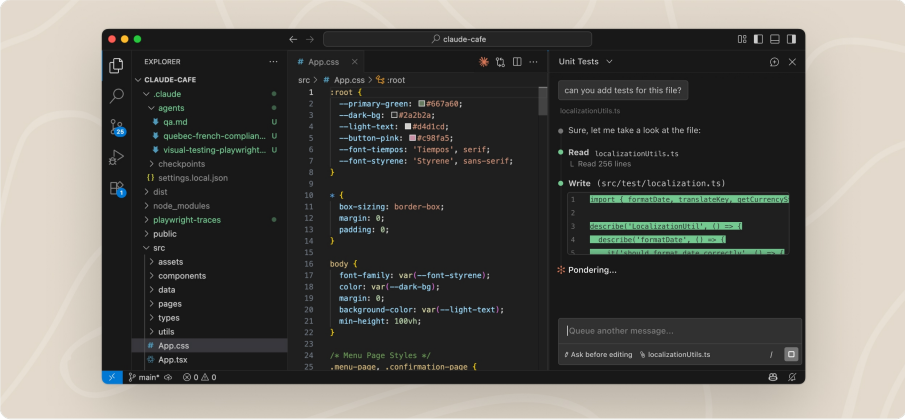
您可以透過兩種主要方式指導 Claude：

1. **CLAUDE.md**：定義編碼標準、安全要求和專案慣例。Claude 在執行期間讀取此檔案並遵循您的規則。
2. **自訂提示**：透過工作中的 `prompt` / `prompt_file` 傳遞任務特定的說明。為不同的工作使用不同的提示（例如，審查、實現、重構）。

Part 8: IDE & Platform Integration

在 VS Code 中使用 Claude Code

安裝並配置 VS Code 的 Claude Code 擴充功能。透過內聯差異、@-提及、計畫審查和快捷鍵獲得 AI 編碼協助。



VS Code 編輯器，右側開啟 Claude Code 擴充功能面板，顯示與 Claude 的對話

VS Code 擴充功能為 Claude Code 提供了原生圖形介面，直接整合到您的 IDE 中。這是在 VS Code 中使用 Claude Code 的推薦方式。

透過擴充功能，您可以在接受 Claude 的計畫之前進行審查和編輯，在進行編輯時自動接受，使用 @-提及來引用具有特定行範圍的檔案，存取對話歷史記錄，以及在單獨的標籤或視窗中開啟多個對話。

先決條件

安裝前，請確保您擁有：

- VS Code 1.98.0 或更高版本
- Anthropic 帳戶（首次開啟擴充功能時您將登入）。如果您使用第三方提供者（如 Amazon Bedrock 或 Google Vertex AI），請改為參閱使用第三方提供者。

Tip:

擴充功能包含 CLI（命令列介面），您可以從 VS Code 的整合終端存取它以獲得進階功能。有關詳細資訊，請參閱 VS Code 擴充功能與 Claude Code CLI。

安裝擴充功能

點擊您的 IDE 的連結以直接安裝：

- 為 VS Code 安裝
- 為 Cursor 安裝

或在 VS Code 中，按 `Cmd+Shift+X`（Mac）或 `Ctrl+Shift+X`（Windows/Linux）開啟擴充功能檢視，搜尋「Claude Code」，然後點擊**安裝**。

> **Note:**
>
> 如果安裝後擴充功能未出現，請重新啟動 VS Code 或從命令面板執行「Developer: Reload Window」。

開始使用

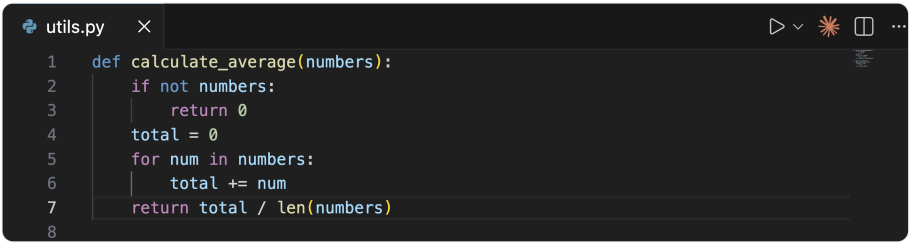
安裝後，您可以透過 VS Code 介面開始使用 Claude Code：

Step 1: 開啟 Claude Code 面板

在整個 VS Code 中，Spark 圖示表示 Claude Code：

Spark icon

開啟 Claude 的最快方式是點擊**編輯器工具列**（編輯器右上角）中的 Spark 圖示。當您開啟檔案時，該圖示才會出現。



VS Code 編輯器顯示編輯器工具列中的 Spark 圖示

開啟 Claude Code 的其他方式：

- **活動列**：點擊左側邊欄中的 Spark 圖示以開啟工作階段清單。點擊任何工作階段以將其作為完整編輯器標籤開啟，或開始新的工作階段。此圖示在活動列中始終可見。
- **命令面板**：`Cmd+Shift+P`（Mac）或 `Ctrl+Shift+P`（Windows/Linux），輸入「Claude Code」，然後選擇一個選項，如「在新標籤中開啟」
- **狀態列**：點擊視窗右下角的 **Claude Code**。即使沒有開啟檔案，這也有效。

首次開啟面板時，會出現**學習 Claude Code** 檢查清單。透過點擊**顯示給我**來完成每一項，或使用 X 關閉它。若要稍後重新開啟它，請在 VS Code 設定中的「擴充功能」→「Claude Code」下取消勾選**隱藏入門**。

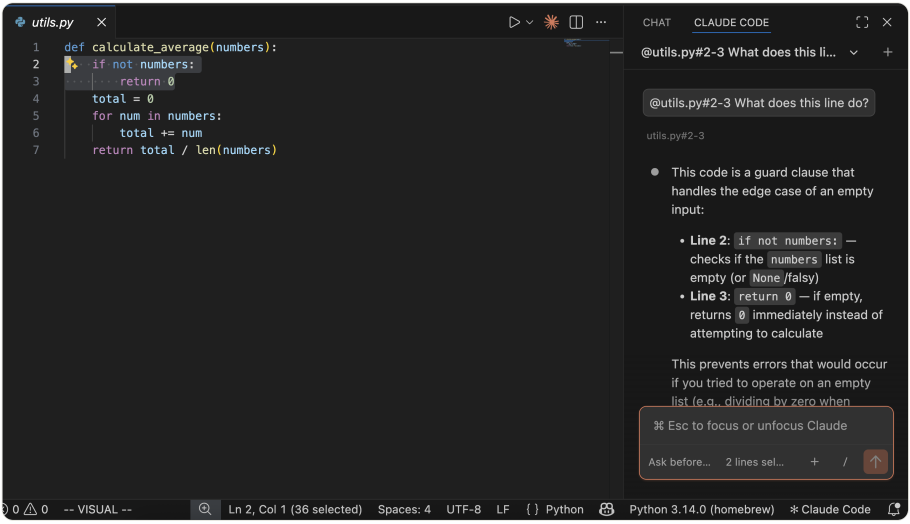
您可以拖動 Claude 面板以在 VS Code 中重新定位它。有關詳細資訊，請參閱自訂您的工作流程。

Step 2: 發送提示

要求 Claude 幫助您的程式碼或檔案，無論是解釋某些內容的工作原理、除錯問題還是進行更改。

> **Tip:**
>
> Claude 會自動看到您選擇的文字。按 `Option+K`（Mac）/ `Alt+K`（Windows/Linux）也可以在您的提示中插入 @-提及參考（如 `@file.ts#5-10`）。

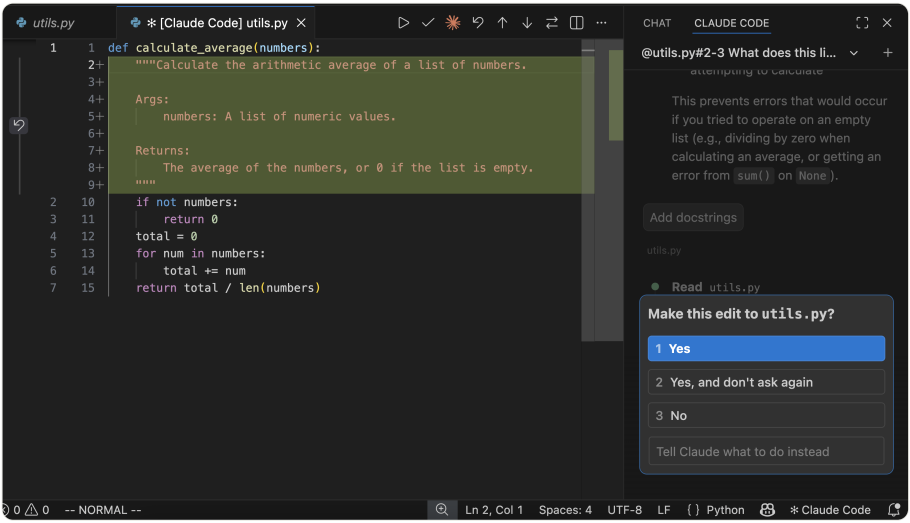
以下是詢問檔案中特定行的範例：



VS Code 編輯器，在 Python 檔案中選擇了第 2-3 行，Claude Code 面板顯示有關這些行的問題，帶有 @-提及參考

Step 3: 審查更改

當 Claude 想要編輯檔案時，它會顯示原始內容和建議更改的並排比較，然後要求許可。您可以接受、拒絕或告訴 Claude 改為做什麼。



VS Code 顯示 Claude 建議更改的差異，以及詢問是否進行編輯的許可提示

有關您可以使用 Claude Code 做什麼的更多想法，請參閱常見工作流程。

> **Tip:**
> 從命令面板執行「Claude Code: Open Walkthrough」以獲得基礎知識的引導式導覽。

使用提示框

提示框支援多項功能：

- **許可模式**：點擊提示框底部的模式指示器以切換模式。在正常模式下，Claude 在每個操作前要求許可。在 Plan Mode 中，Claude 描述它將做什麼，並在進行更改前等待批准。VS Code 會自動將計畫作為完整 markdown 文件開啟，您可以在其中添加內聯評論以在 Claude 開始前提供反饋。在自動接受模式下，Claude 進行編輯而不詢問。在 VS Code 設定中的 `claudeCode.initialPermissionMode` 下設定預設值。
- **命令菜單**：點擊 `/` 或輸入 `/` 以開啟命令菜單。選項包括附加檔案、切換模型、切換擴展思考和查看計畫使用情況（`/usage`）。「自訂」部分提供對 MCP servers、hooks、memory、permissions 和 plugins 的存取。帶有終端圖示的項目在整合終端中開啟。
- **上下文指示器**：提示框顯示您使用了多少 Claude 的 context window。Claude 在需要時會自動壓縮，或您可以手動執行 `/compact`。
- **擴展思考**：讓 Claude 花更多時間推理複雜問題。透過命令菜單（`/`）切換它。有關詳細資訊，請參閱擴展思考。
- **多行輸入**：按 `Shift+Enter` 添加新行而不發送。這也適用於問題對話框的「其他」自由文字輸入。

參考檔案和資料夾

使用 @-提及為 Claude 提供有關特定檔案或資料夾的上下文。當您輸入 `@` 後跟檔案或資料夾名稱時，Claude 會讀取該內容，並可以回答有關它的問題或對其進行更改。Claude Code 支援模糊匹配，因此您可以輸入部分名稱來找到您需要的內容：

```
> Explain the logic in @auth (fuzzy matches auth.js, AuthService.ts, etc.)
> What's in @src/components/ (include a trailing slash for folders)
```

對於大型 PDF，您可以要求 Claude 讀取特定頁面而不是整個檔案：單一頁面、範圍（如第 1-10 頁）或開放式範圍（如第 3 頁起）。

當您在編輯器中選擇文字時，Claude 可以自動看到您突出顯示的程式碼。提示框頁腳顯示選擇了多少行。按 `Option+K`（Mac）/ `Alt+K`（Windows/Linux）以插入帶有檔案路徑和行號的 @-提及（例如 `@app.ts#5-10`）。點擊選擇指示器以切換 Claude 是否可以看到您突出顯示的文字 - 眼睛斜線圖示表示選擇對 Claude 隱藏。

您也可以在將檔案拖動到提示框時按住 `Shift` 以將它們添加為附件。點擊任何附件上的 X 以將其從上下文中移除。

恢復過去的對話

點擊 Claude Code 面板頂部的下拉菜單以存取您的對話歷史記錄。您可以按關鍵字搜尋或按時間瀏覽（今天、昨天、過去 7 天等）。點擊任何對話以使用完整訊息歷史記錄恢復它。將滑鼠懸停在工作階段上以顯示重新命名和移除操作：重新命名以給它一個描述性標題，或移除以將其從清單中刪除。有關恢復工作階段的更多資訊，請參閱常見工作流程。

從 Claude.ai 恢復遠端工作階段

如果您使用網路上的 Claude Code，您可以直接在 VS Code 中恢復這些遠端工作階段。這需要使用 **Claude.ai Subscription** 登入，而不是 Anthropic Console。

Step 1: 開啟過去的對話

點擊 Claude Code 面板頂部的**過去的對話**下拉菜單。

Step 2: 選擇遠端標籤

對話框顯示兩個標籤：本地和遠端。點擊**遠端**以查看來自 claude.ai 的工作階段。

Step 3: 選擇要恢復的工作階段

瀏覽或搜尋您的遠端工作階段。點擊任何工作階段以下載它並在本地繼續對話。

> **Note:**
>
> 只有使用 GitHub 儲存庫啟動的網路工作階段才會出現在’遠端’標籤中。恢復會在本地載入對話歷史記錄；更改不會同步回 claude.ai。

自訂您的工作流程

一旦您啟動並執行，您可以重新定位 Claude 面板、執行多個工作階段或切換到終端模式。

選擇 Claude 的位置

您可以拖動 Claude 面板以在 VS Code 中重新定位它。抓住面板的標籤或標題列並將其拖動到：

- **次要邊欄**：視窗的右側。在您編碼時保持 Claude 可見。
- **主要邊欄**：左側邊欄，帶有資源管理器、搜尋等圖示。
- **編輯器區域**：將 Claude 作為標籤開啟，與您的檔案並排。適用於側面任務。

> **Tip:**
>
> 將邊欄用於您的主要 Claude 工作階段，並為側面任務開啟其他標籤。Claude 會記住您的首選位置。活動列工作階段清單圖示與 Claude 面板分開：工作階段清單在活動列中始終可見，而 Claude 面板圖示只有在面板停靠到左側邊欄時才會出現在那裡。

執行多個對話

使用命令面板中的**在新標籤中開啟**或**在新視窗中開啟**來開始其他對話。每個對話維護自己的歷史記錄和上下文，允許您並行處理不同的任務。

使用標籤時，spark 圖示上的小彩色點表示狀態：藍色表示許可請求待處理，橙色表示 Claude 在標籤隱藏時完成。

切換到終端模式

預設情況下，擴充功能開啟圖形聊天面板。如果您更喜歡 CLI 風格的介面，請開啟使用終端設定並勾選該框。

您也可以開啟 VS Code 設定（Mac 上的 `Cmd+,` 或 Windows/Linux 上的 `Ctrl+,`），前往’擴充功能’→‘Claude Code’，然後勾選**使用終端**。

管理 plugins

VS Code 擴充功能包含用於安裝和管理 plugins 的圖形介面。在提示框中輸入 `/plugins` 以開啟**管理 plugins** 介面。

安裝 plugins

plugin 對話框顯示兩個標籤：**Plugins** 和 **Marketplaces**。

在 Plugins 標籤中：

- **已安裝的 plugins** 出現在頂部，帶有切換開關以啟用或停用它們
- **可用的 plugins** 來自您配置的市場，出現在下方
- 搜尋以按名稱或描述篩選 plugins
- 點擊任何可用 plugin 上的**安裝**

當您安裝 plugin 時，選擇安裝範圍：

- **為您安裝**：在您的所有專案中可用（使用者範圍）
- **為此專案安裝**：與專案協作者共享（專案範圍）
- **本地安裝**：僅適用於您，僅在此儲存庫中（本地範圍）

管理市場

切換到 **Marketplaces** 標籤以添加或移除 plugin 來源：

- 輸入 GitHub 儲存庫、URL 或本地路徑以添加新市場
- 點擊重新整理圖示以更新市場的 plugin 清單
- 點擊垃圾桶圖示以移除市場

進行更改後，橫幅會提示您重新啟動 Claude Code 以應用更新。

> **Note:**
>
> VS Code 中的 plugin 管理在幕後使用相同的 CLI 命令。您在擴充功能中配置的 plugins 和市場也可在 CLI 中使用，反之亦然。

有關 plugin 系統的更多資訊，請參閱 Plugins 和 Plugin 市場。

使用 Chrome 自動化瀏覽器任務

將 Claude 連接到您的 Chrome 瀏覽器以測試網路應用程式、使用控制台日誌進行除錯，以及在不離開 VS Code 的情況下自動化瀏覽器工作流程。這需要 Claude in Chrome 擴充功能 版本 1.0.36 或更高版本。

在提示框中輸入 `@browser` 後跟您想要 Claude 做的事情：

```
@browser go to localhost:3000 and check the console for errors
```

您也可以開啟附件菜單以選擇特定的瀏覽器工具，如開啟新標籤或讀取頁面內容。

Claude 為瀏覽器任務開啟新標籤並共享您的瀏覽器登入狀態，因此它可以存取您已登入的任何網站。

有關設定說明、完整功能清單和故障排除，請參閱使用 Claude Code 與 Chrome。

VS Code 命令和快捷鍵

開啟命令面板（Mac 上的 `Cmd+Shift+P` 或 Windows/Linux 上的 `Ctrl+Shift+P`）並輸入’Claude Code’以查看 Claude Code 擴充功能的所有可用 VS Code 命令。

某些快捷鍵取決於哪個面板’獲得焦點’（接收鍵盤輸入）。當您的游標在程式碼檔案中時，編輯器獲得焦點。當您的游標在 Claude 的提示框中時，Claude 獲得焦點。使用 `Cmd+Esc` / `Ctrl+Esc` 在它們之間切換。

> **Note:**
>
> 這些是用於控制擴充功能的 VS Code 命令。並非所有內建 Claude Code 命令都在擴充功能中可用。有關詳細資訊，請參閱 VS Code 擴充功能與 Claude Code CLI。

命令	快捷鍵	描述
Focus Input	`Cmd+Esc`（Mac）/ `Ctrl+Esc`（Windows/Linux）	在編輯器和 Claude 之間切換焦點
Open in Side Bar	-	在左側邊欄中開啟 Claude
Open in Terminal	-	在終端模式中開啟 Claude
Open in New Tab	`Cmd+Shift+Esc`（Mac）/ `Ctrl+Shift+Esc`（Windows/Linux）	將新對話作為編輯器標籤開啟
Open in New Window	-	在單獨的視窗中開啟新對話
New Conversation	`Cmd+N`（Mac）/ `Ctrl+N`（Windows/Linux）	開始新對話（需要 Claude 獲得焦點）
Insert @-Mention Reference	`Option+K`（Mac）/ `Alt+K`（Windows/Linux）	插入對目前檔案和選擇的參考（需要編輯器獲得焦點）
Show Logs	-	檢視擴充功能除錯日誌
Logout	-	登出您的 Anthropic 帳戶

配置設定

擴充功能有兩種類型的設定：

- **擴充功能設定**在 VS Code 中：控制擴充功能在 VS Code 中的行為。使用 `Cmd+,`（Mac）或 `Ctrl+,`（Windows/Linux）開啟，然後前往’擴充功能’→‘Claude Code’。您也可以輸入 `/` 並選擇**一般配置**以開啟設定。
- **Claude Code 設定**在 `~/.claude/settings.json` 中：在擴充功能和 CLI 之間共享。用於允許的命令、環境變數、hooks 和 MCP servers。有關詳細資訊，請參閱設定。

> **Tip:**

將 `"$schema": "https://json.schemastore.org/claude-code-settings.json"` 添加到您的 `settings.json` 以在 VS Code 中直接獲得所有可用設定的自動完成和內聯驗證。

擴充功能設定

設定	預設值	描述
`selectedModel`	`default`	新對話的模型。使用 `/model` 按工作階段更改。
`useTerminal`	`false`	以終端模式而不是圖形面板啟動 Claude
`initialPermissionMode`	`default`	控制批准提示：`default`（每次詢問）、`plan`、`acceptEdits` 或 `bypassPermissions`
`preferredLocation`	`panel`	Claude 開啟的位置：`sidebar`（右側）或 `panel`（新標籤）
`autosave`	`true`	Claude 讀取或寫入檔案前自動儲存檔案
`useCtrlEnterToSend`	`false`	使用 Ctrl/Cmd+Enter 而不是 Enter 發送提示
`enableNewConversationShortcut`	`true`	啟用 Cmd/Ctrl+N 以開始新對話
`hideOnboarding`	`false`	隱藏入門檢查清單（畢業帽圖示）
`respectGitIgnore`	`true`	從檔案搜尋中排除 .gitignore 模式
`environmentVariables`	`[]`	為 Claude 程序設定環境變數。改為使用 Claude Code 設定以進行共享配置。
`disableLoginPrompt`	`false`	跳過身份驗證提示（用於第三方提供者設定）。

設定	預設值	描述
`allowDangerouslySkipPermissions`	`false`	繞過所有許可提示。**謹慎使用。**
`claudeProcessWrapper`	-	用於啟動 Claude 程序的可執行檔路徑

VS Code 擴充功能與 Claude Code CLI

Claude Code 既可作為 VS Code 擴充功能（圖形面板）也可作為 CLI（終端中的命令列介面）使用。某些功能僅在 CLI 中可用。如果您需要 CLI 專用功能，請在 VS Code 的整合終端中執行 `claude`。

功能	CLI	VS Code 擴充功能
命令和 skills	全部	子集（輸入 `/` 以查看可用的）
MCP server 配置	是	部分（透過 CLI 添加 servers；使用聊天面板中的 `/mcp` 管理現有 servers）
Checkpoints	是	是
`!` bash 快捷方式	是	否
Tab 完成	是	否

使用 checkpoints 進行 Rewind

VS Code 擴充功能支援 checkpoints，它們追蹤 Claude 的檔案編輯並讓您 rewind 到之前的狀態。將滑鼠懸停在任何訊息上以顯示 rewind 按鈕，然後從三個選項中選擇：

- **從此處分支對話**：從此訊息開始新的對話分支，同時保持所有程式碼更改完整
- **將程式碼 Rewind 到此處**：將檔案更改恢復到對話中的此點，同時保持完整的對話歷史記錄
- **分支對話並 Rewind 程式碼**：開始新的對話分支並將檔案更改恢復到此點

有關 checkpoints 如何工作及其限制的完整詳細資訊，請參閱 Checkpointing。

在 VS Code 中執行 CLI

若要在 VS Code 中使用 CLI，請開啟整合終端（Windows/Linux 上的 `Ctrl+`` 或 Mac 上的 `Cmd+`` ）並執行 `claude` 。CLI 會自動與您的 IDE 整合，以獲得差異檢視和診斷共享等功能。

如果使用外部終端，請在 Claude Code 內執行 `/ide` 以將其連接到 VS Code。

在擴充功能和 CLI 之間切換

擴充功能和 CLI 共享相同的對話歷史記錄。若要在 CLI 中繼續擴充功能對話，請在終端中執行 `claude --resume` 。這會開啟一個互動式選擇器，您可以在其中搜尋並選擇您的對話。

在提示中包含終端輸出

使用 `@terminal:name` 在您的提示中參考終端輸出，其中 `name` 是終端的標題。這讓 Claude 可以看到命令輸出、錯誤訊息或日誌，而無需複製貼上。

監控背景程序

當 Claude 執行長時間執行的命令時，擴充功能在狀態列中顯示進度。但是，與 CLI 相比，背景任務的可見性有限。為了獲得更好的可見性，讓 Claude 輸出命令，以便您可以在 VS Code 的整合終端中執行它。

使用 MCP 連接到外部工具

MCP（Model Context Protocol）servers 讓 Claude 存取外部工具、資料庫和 API。

若要添加 MCP server，請開啟整合終端（ `Ctrl+`` 或 `Cmd+`` ）並執行：

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

配置後，要求 Claude 使用工具（例如「Review PR #456」）。

若要在不離開 VS Code 的情況下管理 MCP servers，請在聊天面板中輸入 `/mcp` 。MCP 管理對話框讓您啟用或停用 servers、重新連接到 server 以及管理 OAuth 身份驗證。有關可用 servers，請參閱 MCP 文件。

使用 git

Claude Code 與 git 整合以幫助直接在 VS Code 中進行版本控制工作流程。要求 Claude 提交更改、建立 pull requests 或跨分支工作。

建立提交和 pull requests

Claude 可以暫存更改、編寫提交訊息並根據您的工作建立 pull requests：

```
> commit my changes with a descriptive message
> create a pr for this feature
> summarize the changes I've made to the auth module
```

建立 pull requests 時，Claude 會根據實際程式碼更改生成描述，並可以添加有關測試或實現決策的上下文。

使用 git worktrees 進行並行任務

使用 `--worktree`（`-w`）標誌以隔離的 worktree 啟動 Claude，該 worktree 具有自己的檔案和分支：

```
claude --worktree feature-auth
```

每個 worktree 維護獨立的檔案狀態，同時共享 git 歷史記錄。這可防止 Claude 實例在處理不同任務時相互干擾。有關更多詳細資訊，請參閱使用 Git worktrees 執行並行 Claude Code 工作階段。

使用第三方提供者

預設情況下，Claude Code 直接連接到 Anthropic 的 API。如果您的組織使用 Amazon Bedrock、Google Vertex AI 或 Microsoft Foundry 來存取 Claude，請配置擴充功能以改為使用您的提供者：

Step 1: 停用登入提示

開啟停用登入提示設定並勾選該框。

您也可以開啟 VS Code 設定（Mac 上的 `Cmd+,` 或 Windows/Linux 上的 `Ctrl+,`），搜尋’Claude Code login’，然後勾選**停用登入提示**。

Step 2: 配置您的提供者

遵循您的提供者的設定指南：

- Amazon Bedrock 上的 Claude Code
- Google Vertex AI 上的 Claude Code
- Microsoft Foundry 上的 Claude Code

這些指南涵蓋在 `~/.claude/settings.json` 中配置您的提供者，這確保您的設定在 VS Code 擴充功能和 CLI 之間共享。

安全和隱私

您的程式碼保持私密。Claude Code 處理您的程式碼以提供協助，但不使用它來訓練模型。有關資料處理和如何選擇退出日誌記錄的詳細資訊，請參閱資料和隱私。

啟用自動編輯許可後，Claude Code 可以修改 VS Code 配置檔案（如 `settings.json` 或 `tasks.json`），VS Code 可能會自動執行。為了在處理不受信任的程式碼時降低風險：

- 為不受信任的工作區啟用 VS Code 受限模式
- 使用手動批准模式而不是自動接受進行編輯
- 在接受更改前仔細審查它們

修復常見問題

擴充功能無法安裝

- 確保您有相容的 VS Code 版本（1.98.0 或更高版本）
- 檢查 VS Code 是否有權限安裝擴充功能
- 嘗試從 VS Code Marketplace 直接安裝

Spark 圖示不可見

當您開啟檔案時，Spark 圖示會出現在**編輯器工具列**（編輯器右上角）。如果您看不到它：

1. **開啟檔案**：該圖示需要開啟檔案。僅開啟資料夾是不夠的。
2. **檢查 VS Code 版本**：需要 1.98.0 或更高版本（幫助 → 關於）
3. **重新啟動 VS Code**：從命令面板執行「Developer: Reload Window」
4. **停用衝突的擴充功能**：暫時停用其他 AI 擴充功能（Cline、Continue 等）
5. **檢查工作區信任**：擴充功能在受限模式下不工作

或者，點擊**狀態列**（右下角）中的「 Claude Code」。即使沒有開啟檔案，這也有效。您也可以使用**命令面板**（`Cmd+Shift+P` / `Ctrl+Shift+P`）並輸入「Claude Code」。

Claude Code 從不回應

如果 Claude Code 沒有回應您的提示：

1. **檢查您的網際網路連接**：確保您有穩定的網際網路連接
2. **開始新對話**：嘗試開始新的對話以查看問題是否持續
3. **嘗試 CLI**：從終端執行 `claude` 以查看您是否獲得更詳細的錯誤訊息

如果問題持續，請在 GitHub 上提交問題，並提供有關錯誤的詳細資訊。

卸載擴充功能

若要卸載 Claude Code 擴充功能：

1. 開啟擴充功能檢視（Mac 上的 `Cmd+Shift+X` 或 Windows/Linux 上的 `Ctrl+Shift+X`）
2. 搜尋「Claude Code」
3. 點擊**卸載**

若要也移除擴充功能資料並重設所有設定：

```
rm -rf ~/.vscode/globalStorage/anthropic.claude-code
```

如需其他幫助，請參閱故障排除指南。

後續步驟

現在您已在 VS Code 中設定了 Claude Code：

- 探索常見工作流程以充分利用 Claude Code
- 設定 MCP servers 以使用外部工具擴展 Claude 的功能。使用 CLI 添加 servers，然後在聊天面板中使用 `/mcp` 管理它們。
- 配置 Claude Code 設定以自訂允許的命令、hooks 等。這些設定在擴充功能和 CLI 之間共享。

JetBrains IDEs

> 使用 Claude Code 與 JetBrains IDEs（包括 IntelliJ、PyCharm、WebStorm 等）整合

Claude Code 透過專用外掛程式與 JetBrains IDEs 整合，提供互動式差異檢視、選擇內容共享等功能。

支援的 IDEs

Claude Code 外掛程式適用於大多數 JetBrains IDEs，包括：

- IntelliJ IDEA
- PyCharm
- Android Studio
- WebStorm
- PhpStorm
- GoLand

功能

- **快速啟動**：使用 `Cmd+Esc`（Mac）或 `Ctrl+Esc`（Windows/Linux）直接從編輯器開啟 Claude Code，或點擊 UI 中的 Claude Code 按鈕
- **差異檢視**：程式碼變更可直接在 IDE 差異檢視器中顯示，而不是在終端機中
- **選擇內容共享**：IDE 中的目前選擇/分頁會自動與 Claude Code 共享
- **檔案參考快捷方式**：使用 `Cmd+Option+K`（Mac）或 `Alt+Ctrl+K`（Linux/Windows）插入檔案參考（例如 @File#L1-99）
- **診斷共享**：IDE 中的診斷錯誤（lint、語法等）會在您工作時自動與 Claude 共享

安裝

Marketplace 安裝

從 JetBrains marketplace 尋找並安裝 Claude Code 外掛程式，然後重新啟動您的 IDE。

如果您還未安裝 Claude Code，請參閱我們的快速入門指南以取得安裝說明。

> **Note:**
> 安裝外掛程式後，您可能需要完全重新啟動 IDE 才能使其生效。

使用方式

從您的 IDE

從 IDE 的整合終端機執行 `claude`，所有整合功能將處於活躍狀態。

從外部終端機

在任何外部終端機中使用 `/ide` 命令，將 Claude Code 連接到您的 JetBrains IDE 並啟動所有功能：

```
claude
```

```
/ide
```

如果您希望 Claude 能夠存取與 IDE 相同的檔案，請從與 IDE 專案根目錄相同的目錄啟動 Claude Code。

設定

Claude Code 設定

透過 Claude Code 的設定來設定 IDE 整合：

1. 執行 `claude`
2. 輸入 `/config` 命令
3. 將差異工具設定為 `auto` 以進行自動 IDE 偵測

外掛程式設定

透過前往 **Settings → Tools → Claude Code [Beta]** 來設定 Claude Code 外掛程式：

一般設定

- **Claude 命令**：指定自訂命令以執行 Claude（例如 `claude`、`/usr/local/bin/claude` 或 `npx @anthropic/claude`）
- **抑制找不到 Claude 命令的通知**：略過有關找不到 Claude 命令的通知

- **啟用使用 Option+Enter 進行多行提示**（僅限 macOS）：啟用時，Option+Enter 會在 Claude Code 提示中插入新行。如果遇到 Option 鍵被意外捕獲的問題，請停用此選項（需要終端機重新啟動）
- **啟用自動更新**：自動檢查並安裝外掛程式更新（在重新啟動時套用）

> **Tip:**
>
> 對於 WSL 使用者：將 `wsl -d Ubuntu -- bash -lic "claude"` 設定為您的 Claude 命令（將 `Ubuntu` 替換為您的 WSL 發行版名稱）

ESC 鍵設定

如果 ESC 鍵無法在 JetBrains 終端機中中斷 Claude Code 操作：

1. 前往 **Settings → Tools → Terminal**
2. 執行下列其中一項：
   - 取消勾選「使用 Escape 將焦點移至編輯器」，或
   - 點擊「設定終端機快捷鍵」並刪除「切換焦點至編輯器」快捷方式
3. 套用變更

這將允許 ESC 鍵正確中斷 Claude Code 操作。

特殊設定

遠端開發

> **Warning:**
>
> 使用 JetBrains 遠端開發時，您必須透過 **Settings → Plugin (Host)** 在遠端主機上安裝外掛程式。

外掛程式必須安裝在遠端主機上，而不是在您的本機用戶端機器上。

WSL 設定

> **Warning:**
>
> WSL 使用者可能需要額外設定才能使 IDE 偵測正常運作。請參閱我們的 WSL 疑難排解指南以取得詳細的設定說明。

WSL 設定可能需要：

- 適當的終端機設定
- 網路模式調整
- 防火牆設定更新

疑難排解

外掛程式無法運作

- 確保您從專案根目錄執行 Claude Code
- 檢查 JetBrains 外掛程式在 IDE 設定中是否已啟用
- 完全重新啟動 IDE（您可能需要執行多次）
- 對於遠端開發，確保外掛程式已安裝在遠端主機上

IDE 未被偵測

- 驗證外掛程式已安裝並啟用
- 完全重新啟動 IDE
- 檢查您是否從整合終端機執行 Claude Code
- 對於 WSL 使用者，請參閱 WSL 疑難排解指南

找不到命令

如果點擊 Claude 圖示顯示「找不到命令」：

1. 驗證 Claude Code 已安裝：`npm list -g @anthropic-ai/claude-code`
2. 在外掛程式設定中設定 Claude 命令路徑
3. 對於 WSL 使用者，使用設定部分中提到的 WSL 命令格式

安全考量

當 Claude Code 在啟用自動編輯權限的 JetBrains IDE 中執行時，它可能能夠修改可由您的 IDE 自動執行的 IDE 設定檔。這可能會增加在自動編輯模式下執行 Claude Code 的風險，並允許繞過 Claude Code 對 bash 執行的權限提示。

在 JetBrains IDEs 中執行時，請考慮：

- 對編輯使用手動核准模式
- 特別注意確保 Claude 僅與受信任的提示一起使用
- 注意 Claude Code 有權限修改的檔案

如需其他協助，請參閱我們的疑難排解指南。

使用 Claude Code Desktop

> 充分利用 Claude Code Desktop：具有 Git 隔離的並行會話、視覺化差異檢查、應用程式預覽、PR 監控、權限模式、連接器和企業配置。

Claude Desktop 應用程式中的 Code 標籤讓您可以透過圖形介面而不是終端機來使用 Claude Code。

Desktop 在標準 Claude Code 體驗的基礎上增加了這些功能：

- 視覺化差異檢查，包含內嵌註解
- 即時應用程式預覽，搭配開發伺服器
- GitHub PR 監控，具有自動修復和自動合併
- 並行會話，具有自動 Git worktree 隔離
- 排程任務，按照定期排程執行 Claude
- 連接器，用於 GitHub、Slack、Linear 等
- 本機、SSH 和雲端環境

> **Tip:**
>
> 初次使用 Desktop？從開始使用開始安裝應用程式並進行第一次編輯。

本頁涵蓋使用程式碼、管理會話、擴展 Claude Code、排程任務和配置。它還包括 CLI 比較和疑難排解。

開始會話

在發送第一條訊息之前，在提示區域中配置四項內容：

- **環境**：選擇 Claude 執行的位置。選擇 **Local** 用於您的機器、**Remote** 用於 Anthropic 託管的雲端會話，或用於您管理的遠端機器的 **SSH 連線**。請參閱環境配置。
- **專案資料夾**：選擇 Claude 工作的資料夾或儲存庫。對於遠端會話，您可以新增多個儲存庫。
- **模型**：從傳送按鈕旁的下拉式選單中選擇模型。會話開始後，模型會被鎖定。
- **權限模式**：從模式選擇器中選擇 Claude 擁有多少自主權。您可以在會話期間變更此設定。

輸入您的任務並按 **Enter** 開始。每個會話都會追蹤自己的上下文並獨立進行變更。

使用程式碼

為 Claude 提供正確的上下文，控制它自主執行的程度，並檢查它所做的變更。

使用提示框

輸入您希望 Claude 執行的操作，然後按 **Enter** 傳送。Claude 會讀取您的專案檔案、進行變更，並根據您的權限模式執行命令。您可以隨時中斷 Claude：按一下停止按鈕或輸入您的更正並按 **Enter**。Claude 會停止正在執行的操作並根據您的輸入進行調整。

提示框旁的 **+** 按鈕可讓您存取檔案附件、skills、連接器和plugins。

將檔案和上下文新增至提示

提示框支援兩種方式來引入外部上下文：

- **@mention 檔案**：輸入 `@` 後跟檔案名稱，將檔案新增至對話上下文。Claude 隨後可以讀取和參考該檔案。
- **附加檔案**：使用附件按鈕將影像、PDF 和其他檔案附加到您的提示，或直接將檔案拖放到提示中。這對於分享錯誤的螢幕截圖、設計模型或參考文件很有用。

選擇權限模式

權限模式控制 Claude 在會話期間擁有多少自主權：它是否在編輯檔案、執行命令或兩者之前詢問。您可以隨時使用傳送按鈕旁的模式選擇器切換模式。從「詢問權限」開始，以查看 Claude 確切執行的操作，然後隨著您變得更加熟悉，移至「自動接受編輯」或 Plan Mode。

模式	設定金鑰	行為
詢問權限	`default`	Claude 在編輯檔案或執行命令之前詢問。您會看到差異，並可以接受或拒絕每項變更。建議新使用者使用。
自動接受編輯	`acceptEdits`	Claude 自動接受檔案編輯，但在執行終端機命令之前仍會詢問。當您信任檔案變更並希望更快速地迭代時，請使用此選項。
Plan Mode	`plan`	Claude 分析您的程式碼並建立計畫，而不修改檔案或執行命令。適合您想要先檢查方法的複雜任務。

模式	設定金鑰	行為
略過權限	`bypassPermissions`	Claude 執行時不會出現任何權限提示，相當於 CLI 中的 `--dangerously-skip-permissions`。在「設定」→「Claude Code」下的「允許略過權限模式」中啟用。僅在沙箱容器或 VM 中使用。企業管理員可以停用此選項。

`dontAsk` 權限模式僅在 CLI 中可用。

> **Tip:**
>
> 在 Plan Mode 中開始複雜任務，以便 Claude 在進行變更之前規劃方法。批准計畫後，切換到’自動接受編輯’或’詢問權限’以執行它。有關此工作流程的更多資訊，請參閱先探索，然後計畫，然後編碼。

遠端會話支援’自動接受編輯’和 Plan Mode。’詢問權限’不可用，因為遠端會話預設會自動接受檔案編輯，’略過權限’不可用，因為遠端環境已經是沙箱化的。

企業管理員可以限制哪些權限模式可用。有關詳細資訊，請參閱企業配置。

預覽您的應用程式

Claude 可以啟動開發伺服器並開啟嵌入式瀏覽器來驗證其變更。這適用於前端網路應用程式以及後端伺服器：Claude 可以測試 API 端點、檢視伺服器日誌，並對其發現的問題進行迭代。在大多數情況下，Claude 在編輯專案檔案後會自動啟動伺服器。您也可以隨時要求 Claude 進行預覽。預設情況下，Claude 自動驗證每次編輯後的變更。

從預覽面板，您可以：

- 直接在嵌入式瀏覽器中與執行中的應用程式互動
- 觀看 Claude 自動驗證自己的變更：它會擷取螢幕截圖、檢查 DOM、按一下元素、填寫表單，並修復它發現的問題
- 從會話工具列中的 **Preview** 下拉式選單啟動或停止伺服器
- 透過在下拉式選單中選擇 **Persist sessions**，在伺服器重新啟動時保留 Cookie 和本機儲存，這樣您就不必在開發期間重新登入
- 編輯伺服器配置或一次停止所有伺服器

Claude 根據您的專案建立初始伺服器配置。如果您的應用程式使用自訂開發命令，請編輯 `.claude/launch.json` 以符合您的設定。有關完整參考，請參閱配置預覽伺服器。

若要清除已儲存的會話資料，請在’設定’→‘Claude Code’中切換 **Persist preview sessions** 關閉。若要完全停用預覽，請在’設定’→’Claude Code’中切換 **Preview** 關閉。

使用差異檢查檢查變更

Claude 對您的程式碼進行變更後，差異檢查可讓您在建立提取請求之前逐個檔案檢查修改。

當 Claude 變更檔案時，會出現一個差異統計指示器，顯示新增和移除的行數，例如 `+12 -1`。按一下此指示器以開啟差異檢視器，該檢視器在左側顯示檔案清單，在右側顯示每個檔案的變更。

若要對特定行進行註解，請按一下差異中的任何行以開啟註解框。輸入您的回饋並按 **Enter** 新增註解。在多行新增註解後，一次提交所有註解：

- **macOS**：按 **Cmd+Enter**
- **Windows**：按 **Ctrl+Enter**

Claude 會讀取您的註解並進行要求的變更，這些變更會顯示為您可以檢查的新差異。

檢查您的程式碼

在差異檢查中，按一下右上角工具列中的 **Review code**，要求 Claude 在您提交之前評估變更。Claude 會檢查目前的差異，並直接在差異檢查中留下註解。您可以回應任何註解或要求 Claude 進行修訂。

檢查著重於高信號問題：編譯錯誤、明確的邏輯錯誤、安全漏洞和明顯的錯誤。它不會標記樣式、格式、預先存在的問題或 linter 會捕捉的任何內容。

監控提取請求狀態

開啟提取請求後，CI 狀態列會出現在會話中。Claude Code 使用 GitHub CLI 來輪詢檢查結果並顯示失敗。

- **自動修復**：啟用後，Claude 會透過讀取失敗輸出並進行迭代，自動嘗試修復失敗的 CI 檢查。
- **自動合併**：啟用後，Claude 會在所有檢查通過後合併 PR。合併方法是壓縮。自動合併必須在您的 GitHub 儲存庫設定中啟用才能運作。

使用 CI 狀態列中的 **Auto-fix** 和 **Auto-merge** 切換來啟用任一選項。Claude Code 也會在 CI 完成時傳送桌面通知。

> **Note:**
>
> PR 監控需要在您的機器上安裝並驗證 GitHub CLI（`gh`）。如果未安裝 `gh`，Desktop 會在您第一次嘗試建立 PR 時提示您安裝它。

管理會話

每個會話都是一個獨立的對話，具有自己的上下文和變更。您可以並行執行多個會話或將工作傳送到雲端。

使用會話並行工作

按一下側邊欄中的 **+ New session** 以並行處理多個任務。對於 Git 儲存庫，每個會話都會使用 Git worktrees 獲得自己的隔離專案副本，因此一個會話中的變更不會影響其他會話，直到您提交它們。

Worktrees 預設儲存在 `<project-root>/.claude/worktrees/` 中。您可以在’設定’→’Claude Code’下的’Worktree location’中將其變更為自訂目錄。您也可以設定一個分支前綴，該前綴會被加到每個 worktree 分支名稱前面，這對於保持 Claude 建立的分支井然有序很有用。完成後，若要移除 worktree，請將滑鼠懸停在側邊欄中的會話上，然後按一下封存圖示。

> **Note:**
>
> 會話隔離需要 Git。大多數 Mac 預設包含 Git。在終端機中執行 `git --version` 進行檢查。在 Windows 上，Code 標籤需要 Git 才能運作：下載 Git for Windows、安裝它，然後重新啟動應用程式。如果您遇到 Git 錯誤，請嘗試 Cowork 會話來幫助排除您的設定問題。

使用側邊欄頂部的篩選圖示按狀態（Active、Archived）和環境（Local、Cloud）篩選會話。若要重新命名會話或檢查上下文使用情況，請按一下活動會話頂部工具列中的會話標題。當上下文填滿時，Claude 會自動總結對話並繼續工作。您也可以輸入 `/compact` 來更早觸發總結並釋放上下文空間。有關壓縮如何運作的詳細資訊，請參閱上下文視窗。

遠端執行長時間執行的任務

對於大型重構、測試套件、遷移或其他長時間執行的任務，在啟動會話時選擇 **Remote** 而不是 **Local**。遠端會話在 Anthropic 的雲端基礎設施上執行，即使您關閉應用程式或關閉電腦，也會繼續執行。隨時檢查以查看進度或引導 Claude 朝不同方向發展。您也可以從 claude.ai/code 或 Claude iOS 應用程式監控遠端會話。

遠端會話也支援多個儲存庫。選擇雲端環境後，按一下儲存庫藥丸旁的 **+** 按鈕，將其他儲存庫新增到會話中。每個儲存庫都有自己的分支選擇器。這對於跨越多個程式碼庫的任務很有用，例如更新共用程式庫及其使用者。

有關遠端會話如何運作的更多資訊，請參閱網路上的 Claude Code。

在另一個表面繼續

Continue in 選單可從會話工具列右下角的 VS Code 圖示存取，可讓您將會話移至另一個表面：

- **Claude Code on the Web**：將您的本機會話傳送到遠端繼續執行。Desktop 推送您的分支、產生對話摘要，並使用完整上下文建立新的遠端會話。然後您可以選擇封存本機會話或保留它。這需要乾淨的工作樹，不適用於 SSH 會話。
- **Your IDE**：在目前工作目錄的支援 IDE 中開啟您的專案。

擴展 Claude Code

連接外部服務、新增可重複使用的工作流程、自訂 Claude 的行為，並配置預覽伺服器。

連接外部工具

對於本機和 SSH 會話，按一下提示框旁的 **+** 按鈕，然後選擇 **Connectors** 以新增 Google Calendar、Slack、GitHub、Linear、Notion 等整合。您可以在會話之前或期間新增連接器。遠端會話不提供連接器。

若要管理或斷開連接器，請在桌面應用程式中前往’設定’→‘Connectors’，或從提示框中的’Connectors’選單中選擇 **Manage connectors**。

連接後，Claude 可以讀取您的日曆、傳送訊息、建立問題，並直接與您的工具互動。您可以詢問 Claude 在您的會話中配置了哪些連接器。

連接器是MCP servers，具有圖形設定流程。使用它們可以快速與支援的服務整合。對於’連接器’中未列出的整合，透過設定檔案手動新增 MCP servers。您也可以建立自訂連接器。

使用 skills

Skills 擴展 Claude 可以執行的操作。Claude 在相關時自動載入它們，或者您可以直接呼叫一個：在提示框中輸入 `/` 或按一下 **+** 按鈕並選擇 **Slash commands** 以瀏覽可用的內容。這包括內建命令、您的自訂 skills、來自您程式碼庫的專案 skills，以及來自任何已安裝 plugins 的 skills。選擇一個，它會在輸入欄位中突出顯示。在其後輸入您的任務並照常傳送。

安裝 plugins

Plugins 是可重複使用的套件，可將 skills、agents、hooks、MCP servers 和 LSP 配置新增到 Claude Code。您可以從桌面應用程式安裝 plugins，而無需使用終端機。

對於本機和 SSH 會話，按一下提示框旁的 **+** 按鈕，然後選擇 **Plugins** 以查看您已安裝的 plugins 及其命令。若要新增 plugin，從子選單中選擇 **Add plugin** 以開啟 plugin 瀏覽器，該瀏覽器顯示來自您配置的市場的可用 plugins，包括官方 Anthropic 市場。選擇 **Manage plugins** 以啟用、停用或解除安裝 plugins。

Plugins 可以限定於您的使用者帳戶、特定專案或僅限本機。遠端會話不提供 Plugins。有關完整的 plugin 參考（包括建立您自己的 plugins），請參閱 plugins。

配置預覽伺服器

Claude 會自動偵測您的開發伺服器設定，並將配置儲存在啟動會話時選擇的資料夾根目錄中的 `.claude/launch.json`。Preview 使用此資料夾作為其工作目錄，因此如果您選擇了父資料夾，具有自己開發伺服器的子資料夾將不會自動偵測。若要使用子資料夾的伺服器，請直接在該資料夾中啟動會話，或手動新增配置。

若要自訂伺服器的啟動方式，例如使用 `yarn dev` 而不是 `npm run dev` 或變更連接埠，請手動編輯檔案或按一下 Preview 下拉式選單中的 **Edit configuration** 以在您的程式碼編輯器中開啟它。該檔案支援帶註解的 JSON。

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000
    }
  ]
}
```

您可以定義多個配置以從同一專案執行不同的伺服器，例如前端和 API。請參閱下面的範例。

自動驗證變更

啟用 `autoVerify` 時，Claude 會在編輯檔案後自動驗證程式碼變更。它會擷取螢幕截圖、檢查錯誤，並在完成回應之前確認變更有效。

自動驗證預設為開啟。透過將 `"autoVerify": false` 新增到 `.claude/launch.json` 來按專案停用它，或從 **Preview** 下拉式選單切換它。

```
{
  "version": "0.0.1",
  "autoVerify": false,
  "configurations": [...]
}
```

停用後，預覽工具仍然可用，您可以隨時要求 Claude 進行驗證。自動驗證使其在每次編輯後自動進行。

配置欄位

`configurations` 陣列中的每個項目接受以下欄位：

欄位	類型	描述
`name`	string	此伺服器的唯一識別碼
`runtimeExecutable`	string	要執行的命令，例如 `npm`、`yarn` 或 `node`
`runtimeArgs`	string[]	傳遞給 `runtimeExecutable` 的引數，例如 `["run", "dev"]`
`port`	number	您的伺服器監聽的連接埠。預設為 3000
`cwd`	string	相對於您的專案根目錄的工作目錄。預設為專案根目錄。使用 `${workspaceFolder}` 明確參考專案根目錄
`env`	object	其他環境變數作為鍵值對，例如 `{ "NODE_ENV": "development" }`。不要在此處放置機密，因為此檔案會提交到您的儲存庫。在您的 shell 設定檔中設定的機密會自動繼承。

欄位	類型	描述
`autoPort`	boolean	如何處理連接埠衝突。請參閱下面
`program`	string	使用 `node` 執行的指令碼。請參閱何時使用 `program` 與 `runtimeExecutable`
`args`	string[]	傳遞給 `program` 的引數。僅在設定 `program` 時使用

何時使用 `program` 與 `runtimeExecutable`

使用 `runtimeExecutable` 搭配 `runtimeArgs` 透過套件管理器啟動開發伺服器。例如，`"runtimeExecutable": "npm"` 搭配 `"runtimeArgs": ["run", "dev"]` 執行 `npm run dev`。

當您有想要直接使用 `node` 執行的獨立指令碼時，使用 `program`。例如，`"program": "server.js"` 執行 `node server.js`。使用 `args` 傳遞其他標誌。

連接埠衝突

`autoPort` 欄位控制當您偏好的連接埠已在使用中時會發生什麼：

- `true`：Claude 自動尋找並使用空閒連接埠。適合大多數開發伺服器。
- `false`：Claude 失敗並出現錯誤。當您的伺服器必須使用特定連接埠時使用此選項，例如 OAuth 回呼或 CORS 允許清單。
- **未設定（預設）**：Claude 詢問伺服器是否需要該確切連接埠，然後儲存您的答案。

當 Claude 選擇不同的連接埠時，它會透過 `PORT` 環境變數將指派的連接埠傳遞給您的伺服器。

範例

這些配置顯示不同專案類型的常見設定：

Next.js

此配置使用 Yarn 在連接埠 3000 上執行 Next.js 應用程式：

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "web",
      "runtimeExecutable": "yarn",
      "runtimeArgs": ["dev"],
      "port": 3000
    }
  ]
}
```

Multiple servers

對於具有前端和 API 伺服器的 monorepo，定義多個配置。前端使用 `autoPort: true`，因此如果 3000 被佔用，它會選擇空閒連接埠，而 API 伺服器需要確切的連接埠 8080：

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "frontend",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "cwd": "apps/web",
      "port": 3000,
      "autoPort": true
    },
    {
      "name": "api",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "start"],
      "cwd": "server",
      "port": 8080,
      "env": { "NODE_ENV": "development" },
      "autoPort": false
    }
  ]
}
```

Node.js script

若要直接執行 Node.js 指令碼而不是使用套件管理器命令，請使用 `program` 欄位：

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "server",
      "program": "server.js",
      "args": ["--verbose"],
      "port": 4000
    }
  ]
}
```

排程定期任務

排程任務會在您選擇的時間和頻率自動啟動新的本機會話。使用它們進行定期工作，例如每日程式碼檢查、相依性更新檢查或從您的日曆和收件箱提取的早晨簡報。

任務在您的機器上執行，因此桌面應用程式必須開啟且您的電腦保持清醒才能觸發它們。有關錯過的執行和追趕行為的詳細資訊，請參閱排程任務如何執行。

> **Note:**
>
> 預設情況下，排程任務針對您的工作目錄所處的任何狀態執行，包括未提交的變更。在提示輸入中啟用 worktree 切換，以為每次執行提供自己的隔離 Git worktree，與並行會話的方式相同。

若要建立排程任務，請按一下側邊欄中的 **Schedule**，然後按 **+ New task**。配置這些欄位：

欄位	描述
Name	任務的識別碼。轉換為小寫 kebab-case 並用作磁碟上的資料夾名稱。在您的任務中必須是唯一的。
Description	任務清單中顯示的簡短摘要。
Prompt	任務執行時傳送給 Claude 的指示。以您在提示框中編寫任何訊息的相同方式編寫此內容。提示輸入還包括模型、權限模式、工作資料夾和 worktree 的控制項。
Frequency	任務執行的頻率。請參閱下面的頻率選項。

您也可以透過在任何會話中描述您想要的內容來建立任務。例如，「設定一個每天早上 9 點執行的每日程式碼檢查。」

頻率選項

- **Manual**：無排程，僅在您按一下 **Run now** 時執行。適合儲存您按需觸發的提示
- **Hourly**：每小時執行一次。每個任務從整點開始獲得最多 10 分鐘的固定偏移，以錯開 API 流量
- **Daily**：顯示時間選擇器，預設為當地時間上午 9:00
- **Weekdays**：與 Daily 相同，但跳過星期六和星期日
- **Weekly**：顯示時間選擇器和日期選擇器

對於選擇器不提供的間隔（每 15 分鐘、每月第一天等），在任何 Desktop 會話中詢問 Claude 設定排程。使用純文字；例如，「排程一個任務每 6 小時執行一次所有測試。」

排程任務如何執行

排程任務在您的機器上本機執行。Desktop 在應用程式開啟時每分鐘檢查一次排程，並在任務到期時啟動新會話，獨立於您開啟的任何手動會話。每個任務在排程時間後獲得最多 10 分鐘的固定延遲，以錯開 API 流量。延遲是確定性的：同一任務始終在相同的偏移處啟動。

當任務觸發時，您會收到桌面通知，新會話會出現在側邊欄的 **Scheduled** 部分下。開啟它以查看 Claude 執行的操作、檢查變更或回應權限提示。會話的工作方式與任何其他會話相同：Claude 可以編輯檔案、執行命令、建立提交和開啟提取請求。

任務僅在桌面應用程式執行且您的電腦清醒時執行。如果您的電腦在排程時間內進入睡眠狀態，執行會被跳過。若要防止閒置睡眠，請在「設定」下的 **Desktop app → General** 中啟用 **Keep computer awake**。關閉筆記本電腦蓋仍會使其進入睡眠狀態。

錯過的執行

當應用程式啟動或您的電腦喚醒時，Desktop 會檢查每個任務是否在過去七天內錯過了任何執行。如果有，Desktop 會為最近錯過的時間啟動恰好一次追趕執行，並丟棄任何較舊的執行。每日任務錯過六天會在喚醒時執行一次。Desktop 會在追趕執行啟動時顯示通知。

編寫提示時請記住這一點。排程在上午 9 點的任務可能在晚上 11 點執行，如果您的電腦整天都在睡眠狀態。如果時間很重要，請在提示本身中新增護欄，例如：「僅檢查今天的提交。如果已過下午 5 點，請跳過檢查，只發佈錯過內容的摘要。」

排程任務的權限

每個任務都有自己的權限模式，您在建立或編輯任務時設定。來自 `~/.claude/settings.json` 的允許規則也適用於排程任務會話。如果任務在詢問模式下執行並需要執行它沒有權限的工具，執行會停滯，直到您批准它。會話保持在側邊欄中開啟，以便您稍後可以回答。

若要避免停滯，請在建立任務後按一下 **Run now**，觀察權限提示，並為每個提示選擇「always allow」。該任務的未來執行會自動批准相同的工具，而無需提示。您可以從任務的詳細資訊頁面檢查和撤銷這些批准。

管理排程任務

按一下 **Schedule** 清單中的任務以開啟其詳細資訊頁面。從這裡您可以：

- **Run now**：立即啟動任務，無需等待下一個排程時間
- **Toggle repeats**：暫停或恢復排程執行，而無需刪除任務
- **Edit**：變更提示、頻率、資料夾或其他設定
- **Review history**：查看每次過去的執行，包括因您的電腦睡眠而被跳過的執行
- **Review allowed permissions**：從 **Always allowed** 面板查看和撤銷此任務的已儲存工具批准
- **Delete**：移除任務並封存它建立的所有會話

您也可以透過在任何 Desktop 會話中詢問 Claude 來管理任務。例如，「暫停我的 dependency-audit 任務」、「刪除 standup-prep 任務」或「顯示我的排程任務。」

若要在磁碟上編輯任務的提示，請開啟 `~/.claude/scheduled-tasks/<task-name>/SKILL.md`（或在設定 `CLAUDE_CONFIG_DIR` 時在其下）。該檔案使用 YAML frontmatter 用於 `name` 和 `description`，提示作為正文。變更在下一次執行時生效。排程、資料夾、模型和啟用狀態不在此檔案中：透過編輯表單或詢問 Claude 來變更它們。

環境配置

您在啟動會話時選擇的環境決定了 Claude 執行的位置以及您如何連接：

- **Local**：在您的機器上執行，直接存取您的檔案
- **Remote**：在 Anthropic 的雲端基礎設施上執行。即使您關閉應用程式，會話也會繼續。
- **SSH**：在您透過 SSH 連接的遠端機器上執行，例如您自己的伺服器、雲端 VM 或開發容器

本機會話

本機會話從您的 shell 繼承環境變數。如果您需要其他變數，請在您的 shell 設定檔（例如 `~/.zshrc` 或 `~/.bashrc`）中設定它們，並重新啟動桌面應用程式。有關支援的變數的完整清單，請參閱環境變數。

擴展思考預設啟用，這改進了複雜推理任務的效能，但使用額外的 tokens。若要完全停用思考，請在您的 shell 設定檔中設定 `MAX_THINKING_TOKENS=0`。在 Opus 上，除了 `0` 外，`MAX_THINKING_TOKENS` 會被忽略，因為自適應推理控制思考深度。

遠端會話

遠端會話即使在您關閉應用程式後也會在背景繼續。使用量計入您的訂閱計畫限制，沒有單獨的計算費用。

您可以建立具有不同網路存取級別和環境變數的自訂雲端環境。在啟動遠端會話時選擇環境下拉式選單，然後選擇 **Add environment**。有關配置網路存取和環境變數的詳細資訊，請參閱雲端環境。

SSH 會話

SSH 會話讓您在遠端機器上執行 Claude Code，同時使用桌面應用程式作為您的介面。這對於使用位於雲端 VM、開發容器或具有特定硬體或相依性的伺服器上的程式碼庫很有用。

若要新增 SSH 連線，請在啟動會話之前按一下環境下拉式選單，然後選擇 **+ Add SSH connection**。對話框要求：

- **Name**：此連線的友善標籤
- **SSH Host**：`user@hostname` 或在 `~/.ssh/config` 中定義的主機
- **SSH Port**：如果留空則預設為 22，或使用您的 SSH 配置中的連接埠
- **Identity File**：您的私鑰的路徑，例如 `~/.ssh/id_rsa`。留空以使用預設金鑰或您的 SSH 配置。

新增後，連線會出現在環境下拉式選單中。選擇它以在該機器上啟動會話。Claude 在遠端機器上執行，可以存取其檔案和工具。

Claude Code 必須安裝在遠端機器上。連接後，SSH 會話支援權限模式、連接器、plugins 和 MCP servers。

企業配置

Teams 或 Enterprise 計畫上的組織可以透過管理員主控台控制、受管設定檔案和裝置管理原則來管理桌面應用程式行為。

管理員主控台控制

這些設定透過管理員設定主控台配置：

- **啟用或停用 Code 標籤**：控制您組織中的使用者是否可以在桌面應用程式中存取 Claude Code
- **停用略過權限模式**：防止您組織中的使用者啟用略過權限模式
- **停用網路上的 Claude Code**：為您的組織啟用或停用遠端會話

受管設定

受管設定會覆蓋專案和使用者設定，並在 Desktop 產生 CLI 會話時套用。您可以在組織的受管設定檔案中設定這些金鑰，或透過管理員主控台遠端推送它們。

金鑰	描述
`disableBypassPermissionsMode`	設定為 `"disable"` 以防止使用者啟用略過權限模式。請參閱受管設定。

有關受管專用設定（包括 `allowManagedPermissionRulesOnly` 和 `allowManagedHooksOnly`）的完整清單，請參閱受管專用設定。

透過管理員主控台上傳的遠端受管設定目前僅適用於 CLI 和 IDE 會話。對於 Desktop 特定的限制，請使用上面的管理員主控台控制。

裝置管理原則

IT 團隊可以透過 macOS 上的 MDM 或 Windows 上的群組原則來管理桌面應用程式。可用的原則包括啟用或停用 Claude Code 功能、控制自動更新和設定自訂部署 URL。

- **macOS**：使用 Jamf 或 Kandji 等工具透過 `com.anthropic.Claude` 偏好設定網域配置
- **Windows**：透過 `SOFTWARE\Policies\Claude` 的登錄配置

驗證和 SSO

企業組織可以要求所有使用者進行 SSO。有關計畫級別的詳細資訊，請參閱驗證，有關 SAML 和 OIDC 配置，請參閱設定 SSO。

資料處理

Claude Code 在本機會話中本機處理您的程式碼，或在遠端會話中在 Anthropic 的雲端基礎設施上處理。對話和程式碼上下文會傳送到 Anthropic 的 API 進行處理。有關資料保留、隱私和合規性的詳細資訊，請參閱資料處理。

部署

Desktop 可以透過企業部署工具分發：

- **macOS**：透過 MDM（例如 Jamf 或 Kandji）使用 `.dmg` 安裝程式分發
- **Windows**：透過 MSIX 套件或 `.exe` 安裝程式部署。有關企業部署選項（包括無聲安裝），請參閱為 Windows 部署 Claude Desktop

有關網路配置（例如代理設定、防火牆允許清單和 LLM 閘道），請參閱網路配置。

有關完整的企業配置參考，請參閱企業配置指南。

來自 CLI？

如果您已經使用 Claude Code CLI，Desktop 會執行相同的基礎引擎，但具有圖形介面。您可以在同一機器上同時執行兩者，甚至在同一專案上執行。每個都維護單獨的會話歷史記錄，但它們透過 CLAUDE.md 檔案共用配置和專案記憶。

若要將 CLI 會話移至 Desktop，請在終端機中執行 `/desktop`。Claude 會儲存您的會話並在桌面應用程式中開啟它，然後退出 CLI。此命令僅在 macOS 和 Windows 上可用。

> **Tip:**
>
> 何時使用 Desktop 與 CLI：當您想要視覺化差異檢查、檔案附件或側邊欄中的會話管理時，請使用 Desktop。當您需要指令碼、自動化、第三方提供者或偏好終端機工作流程時，請使用 CLI。

CLI 標誌等效項

此表顯示常見 CLI 標誌的桌面應用程式等效項。未列出的標誌沒有桌面等效項，因為它們是為指令碼或自動化設計的。

CLI	Desktop 等效項
`--model sonnet`	傳送按鈕旁的模型下拉式選單，在啟動會話之前
`--resume`, `--continue`	按一下側邊欄中的會話
`--permission-mode`	傳送按鈕旁的模式選擇器
`--dangerously-skip-permissions`	略過權限模式。在「設定」→「Claude Code」→「Allow bypass permissions mode」中啟用。企業管理員可以停用此設定。
`--add-dir`	在遠端會話中使用 + 按鈕新增多個儲存庫

CLI	Desktop 等效項
`--allowedTools`, `--disallowedTools`	在 Desktop 中不可用
`--verbose`	不可用。檢查系統日誌：macOS 上的 Console.app、Windows 上的「事件檢視器」→「Windows 日誌」→「應用程式」
`--print`, `--output-format`	不可用。Desktop 僅限互動。
`ANTHROPIC_MODEL` env var	傳送按鈕旁的模型下拉式選單
`MAX_THINKING_TOKENS` env var	在 shell 設定檔中設定；適用於本機會話。請參閱環境配置。

共用配置

Desktop 和 CLI 讀取相同的配置檔案，因此您的設定會轉移：

- **CLAUDE.md** 檔案在您的專案中由兩者使用
- **MCP servers** 在 `~/.claude.json` 或 `.mcp.json` 中配置的在兩者中都有效
- **Hooks** 和 **skills** 在設定中定義的適用於兩者
- **Settings** 在 `~/.claude.json` 和 `~/.claude/settings.json` 中是共用的。`settings.json` 中的權限規則、允許的工具和其他設定適用於 Desktop 會話。
- **Models**：Sonnet、Opus 和 Haiku 在兩者中都可用。在 Desktop 中，在啟動會話之前從傳送按鈕旁的下拉式選單中選擇模型。您無法在活動會話期間變更模型。

> **Note:**
>
> **MCP servers：桌面聊天應用程式與 Claude Code**：在 `claude_desktop_config.json` 中為 Claude Desktop 聊天應用程式配置的 MCP servers 與 Claude Code 分開，不會出現在 Code 標籤中。若要在 Claude Code 中使用 MCP servers，請在 `~/.claude.json` 或您的專案的 `.mcp.json` 檔案中配置它們。有關詳細資訊，請參閱 MCP 配置。

功能比較

此表比較 CLI 和 Desktop 之間的核心功能。有關 CLI 標誌的完整清單，請參閱 CLI 參考。

功能	CLI	Desktop
權限模式	所有模式，包括 `dontAsk`	詢問權限、自動接受編輯、Plan Mode 和透過「設定」的略過權限
`--dangerously-skip-permissions`	CLI 標誌	略過權限模式。在「設定」→「Claude Code」→「Allow bypass permissions mode」中啟用
第三方提供者	Bedrock、Vertex、Foundry	不可用。Desktop 直接連接到 Anthropic 的 API。
MCP servers	在設定檔案中配置	本機和 SSH 會話的連接器 UI，或設定檔案
Plugins	`/plugin` 命令	plugin 管理器 UI
@mention 檔案	文字型	帶自動完成
檔案附件	不可用	影像、PDF
會話隔離	`--worktree` 標誌	自動 worktrees
多個會話	單獨的終端機	側邊欄標籤
定期任務	cron 工作、CI 管道	排程任務
指令碼和自動化	`--print`、Agent SDK	不可用

Desktop 中不可用的內容

以下功能僅在 CLI 或 VS Code 擴充功能中可用：

- **第三方提供者**：Desktop 直接連接到 Anthropic 的 API。改用 CLI 搭配 Bedrock、Vertex 或 Foundry。
- **Linux**：桌面應用程式僅在 macOS 和 Windows 上可用。
- **內嵌程式碼建議**：Desktop 不提供自動完成樣式的建議。它透過對話提示和明確的程式碼變更進行工作。
- **Agent teams**：多代理協調可透過 CLI 和 Agent SDK 使用，不在 Desktop 中。

疑難排解

檢查您的版本

若要查看您執行的桌面應用程式版本：

- **macOS**：按一下選單列中的 **Claude**，然後按 **About Claude**
- **Windows**：按一下 **Help**，然後按 **About**

按一下版本號碼將其複製到您的剪貼簿。

Code 標籤中的 403 或驗證錯誤

如果在使用 Code 標籤時看到 `Error 403: Forbidden` 或其他驗證失敗：

1. 從應用程式選單登出並重新登入。這是最常見的修復。
2. 驗證您有有效的付費訂閱：Pro、Max、Teams 或 Enterprise。
3. 如果 CLI 有效但 Desktop 無效，請完全退出桌面應用程式（不只是關閉視窗），然後重新開啟並登入。
4. 檢查您的網際網路連線和代理設定。

啟動時螢幕空白或卡住

如果應用程式開啟但顯示空白或無反應的螢幕：

1. 重新啟動應用程式。
2. 檢查待處理的更新。應用程式在啟動時自動更新。
3. 在 Windows 上，檢查「事件檢視器」中的崩潰日誌，位於 **Windows 日誌 → 應用程式**。

「Failed to load session」

如果您看到 `Failed to load session`，選定的資料夾可能不再存在、Git 儲存庫可能需要未安裝的 Git LFS，或檔案權限可能阻止存取。嘗試選擇不同的資料夾或重新啟動應用程式。

會話找不到已安裝的工具

如果 Claude 找不到 `npm`、`node` 或其他 CLI 命令等工具，請驗證工具在您的常規終端機中有效、檢查您的 shell 設定檔是否正確設定 PATH，並重新啟動桌面應用程式以重新載入環境變數。

Git 和 Git LFS 錯誤

在 Windows 上，Git 是啟動本機會話的 Code 標籤所需的。如果您看到「Git is required」，請安裝 Git for Windows 並重新啟動應用程式。

如果您看到「Git LFS is required by this repository but is not installed」，請從 git-lfs.com 安裝 Git LFS，執行 `git lfs install`，然後重新啟動應用程式。

Windows 上的 MCP servers 無法運作

如果 MCP server 切換沒有回應或伺服器在 Windows 上無法連接，請檢查伺服器是否在您的設定中正確配置、重新啟動應用程式、驗證伺服器程序在工作管理員中執行，並檢查伺服器日誌以查看連接錯誤。

應用程式無法退出

- **macOS**：按 Cmd+Q。如果應用程式沒有回應，請使用 Cmd+Option+Esc 強制退出，選擇 Claude，然後按「強制退出」。
- **Windows**：使用 Ctrl+Shift+Esc 的工作管理員來結束 Claude 程序。

Windows 特定問題

- **安裝後 PATH 未更新**：開啟新的終端機視窗。PATH 更新僅適用於新的終端機會話。
- **並行安裝錯誤**：如果您看到有關另一個安裝進行中的錯誤，但實際上沒有，請嘗試以管理員身份執行安裝程式。
- **ARM64**：Windows ARM64 裝置完全支援。

Intel Mac 上的 Cowork 標籤不可用

Cowork 標籤在 macOS 上需要 Apple Silicon（M1 或更新版本）。在 Windows 上，Cowork 在所有支援的硬體上可用。Chat 和 Code 標籤在 Intel Mac 上正常運作。

在 CLI 中開啟時「Branch doesn’t exist yet」

遠端會話可以建立在您的本機機器上不存在的分支。按一下會話工具列中的分支名稱以複製它，然後在本機提取它：

```
git fetch origin <branch-name>
git checkout <branch-name>
```

仍然卡住？

- 在 GitHub Issues 上搜尋或提交錯誤
- 造訪 Claude 支援中心

提交錯誤時，請包括您的桌面應用程式版本、您的作業系統、確切的錯誤訊息和相關日誌。在 macOS 上，檢查 Console.app。在 Windows 上，檢查「事件檢視器」→「Windows 日誌」→「應用程式」。

在 Chrome 中使用 Claude Code（測試版）

> 將 Claude Code 連接到您的 Chrome 瀏覽器，以測試網頁應用程式、使用控制台日誌進行除錯、自動填充表單，以及從網頁中提取資料。

Claude Code 與 Claude in Chrome 瀏覽器擴充功能整合，為您提供從 CLI 或 VS Code 擴充功能 進行瀏覽器自動化的功能。建立您的程式碼，然後在瀏覽器中測試和除錯，無需切換上下文。

Claude 為瀏覽器任務開啟新標籤頁，並共享您瀏覽器的登入狀態，因此它可以存取您已登入的任何網站。瀏覽器操作在可見的 Chrome 視窗中即時執行。當 Claude 遇到登入頁面或 CAPTCHA 時，它會暫停並要求您手動處理。

> **Note:**
>
> Chrome 整合處於測試版，目前僅適用於 Google Chrome。尚不支援 Brave、Arc 或其他基於 Chromium 的瀏覽器。也不支援 WSL（Windows Subsystem for Linux）。

功能

連接 Chrome 後，您可以在單一工作流程中鏈接瀏覽器操作與編碼任務：

- **即時除錯**：直接讀取控制台錯誤和 DOM 狀態，然後修復導致它們的程式碼
- **設計驗證**：從 Figma 模型建立 UI，然後在瀏覽器中開啟以驗證它是否相符
- **網頁應用程式測試**：測試表單驗證、檢查視覺回歸或驗證使用者流程
- **已驗證的網頁應用程式**：與 Google Docs、Gmail、Notion 或您已登入的任何應用程式互動，無需 API 連接器
- **資料提取**：從網頁中提取結構化資訊並將其儲存在本地
- **任務自動化**：自動化重複的瀏覽器任務，如資料輸入、表單填充或多網站工作流程
- **工作階段錄製**：將瀏覽器互動錄製為 GIF，以記錄或分享發生的情況

先決條件

在使用 Claude Code 與 Chrome 之前，您需要：

- Google Chrome 瀏覽器
- Claude in Chrome 擴充功能 版本 1.0.36 或更高版本
- Claude Code 版本 2.0.73 或更高版本

- 直接 Anthropic 計畫（Pro、Max、Team 或 Enterprise）

> **Note:**
>
> Chrome 整合不適用於 Amazon Bedrock、Google Cloud Vertex AI 或 Microsoft Foundry 等第三方提供商。如果您只透過第三方提供商存取 Claude，則需要單獨的 claude.ai 帳戶才能使用此功能。

在 CLI 中開始

Step 1: 使用 Chrome 啟動 Claude Code

使用 `--chrome` 標誌啟動 Claude Code：

```
claude --chrome
```

您也可以透過執行 `/chrome` 在現有工作階段中啟用 Chrome。

Step 2: 要求 Claude 使用瀏覽器

此範例導航到頁面、與其互動並報告其發現，全部來自您的終端或編輯器：

```
Go to code.claude.com/docs, click on the search box,
type "hooks", and tell me what results appear
```

隨時執行 `/chrome` 以檢查連接狀態、管理權限或重新連接擴充功能。

對於 VS Code，請參閱 VS Code 中的瀏覽器自動化。

預設啟用 Chrome

為了避免每個工作階段都傳遞 `--chrome`，執行 `/chrome` 並選擇「預設啟用」。

在 VS Code 擴充功能 中，只要安裝了 Chrome 擴充功能，Chrome 就可用。無需額外標誌。

> **Note:**
>
> 在 CLI 中預設啟用 Chrome 會增加上下文使用量，因為瀏覽器工具始終被載入。如果您注意到上下文消耗增加，請停用此設定，並僅在需要時使用 `--chrome`。

管理網站權限

網站級權限繼承自 Chrome 擴充功能。在 Chrome 擴充功能設定中管理權限，以控制 Claude 可以瀏覽、點擊和輸入的網站。

範例工作流程

這些範例展示了將瀏覽器操作與編碼任務結合的常見方式。執行 `/mcp` 並選擇 `claude-in-chrome` 以查看可用瀏覽器工具的完整清單。

測試本地網頁應用程式

開發網頁應用程式時，要求 Claude 驗證您的變更是否正確運作：

```
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 導航到您的本地伺服器、與表單互動並報告其觀察結果。

使用控制台日誌進行除錯

Claude 可以讀取控制台輸出以幫助診斷問題。告訴 Claude 要尋找的模式，而不是要求所有控制台輸出，因為日誌可能很冗長：

```
Open the dashboard page and check the console for any errors when
the page loads.
```

Claude 讀取控制台訊息，可以篩選特定模式或錯誤類型。

自動填充表單

加快重複資料輸入任務的速度：

```
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 讀取您的本地檔案、導航網頁介面並為每筆記錄輸入資料。

在 Google Docs 中起草內容

使用 Claude 直接在您的文件中寫入，無需 API 設定：

```
Draft a project update based on the recent commits and add it to my
Google Doc at docs.google.com/document/d/abc123
```

Claude 開啟文件、點擊編輯器並輸入內容。這適用於您已登入的任何網頁應用程式：Gmail、Notion、Sheets 等。

從網頁中提取資料

從網站中提取結構化資訊：

```
Go to the product listings page and extract the name, price, and
availability for each item. Save the results as a CSV file.
```

Claude 導航到頁面、讀取內容並將資料編譯成結構化格式。

執行多網站工作流程

協調多個網站之間的任務：

```
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 跨標籤頁工作以收集資訊並完成工作流程。

錄製演示 GIF

建立瀏覽器互動的可共享錄製：

```
Record a GIF showing how to complete the checkout flow, from adding
an item to the cart through to the confirmation page.
```

Claude 錄製互動序列並將其儲存為 GIF 檔案。

故障排除

未偵測到擴充功能

如果 Claude Code 顯示「未偵測到 Chrome 擴充功能」：

1. 驗證 Chrome 擴充功能已安裝並在 `chrome://extensions` 中啟用

2. 透過執行 `claude --version` 驗證 Claude Code 是否為最新版本
3. 檢查 Chrome 是否正在執行
4. 執行 `/chrome` 並選擇「重新連接擴充功能」以重新建立連接
5. 如果問題仍然存在，請重新啟動 Claude Code 和 Chrome

第一次啟用 Chrome 整合時，Claude Code 會安裝原生訊息主機設定檔。Chrome 在啟動時讀取此檔案，因此如果擴充功能在您的第一次嘗試中未被偵測到，請重新啟動 Chrome 以取得新設定。

如果連接仍然失敗，請驗證主機設定檔是否存在於：

- **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**：在 Windows 登錄中檢查 `HKCU\Software\Google\Chrome\NativeMessagingHosts\`

瀏覽器無回應

如果 Claude 的瀏覽器命令停止工作：

1. 檢查是否有模態對話框（警告、確認、提示）阻止頁面。JavaScript 對話框會阻止瀏覽器事件並防止 Claude 接收命令。手動關閉對話框，然後告訴 Claude 繼續。
2. 要求 Claude 建立新標籤頁並重試
3. 透過在 `chrome://extensions` 中停用並重新啟用 Chrome 擴充功能來重新啟動它

長工作階段期間連接中斷

Chrome 擴充功能的服務工作者可能在延長的工作階段期間進入閒置狀態，這會中斷連接。如果瀏覽器工具在一段時間不活動後停止工作，請執行 `/chrome` 並選擇「重新連接擴充功能」。

Windows 特定問題

在 Windows 上，您可能會遇到：

- **命名管道衝突 (EADDRINUSE)**：如果另一個程序正在使用相同的命名管道，請重新啟動 Claude Code。關閉任何可能使用 Chrome 的其他 Claude Code 工作階段。
- **原生訊息主機錯誤**：如果原生訊息主機在啟動時崩潰，請嘗試重新安裝 Claude Code 以重新產生主機設定。

常見錯誤訊息

以下是最常遇到的錯誤及其解決方法：

錯誤	原因	修復
「瀏覽器擴充功能未連接」	原生訊息主機無法到達擴充功能	重新啟動 Chrome 和 Claude Code，然後執行 `/chrome` 以重新連接
「未偵測到擴充功能」	Chrome 擴充功能未安裝或已停用	在 `chrome://extensions` 中安裝或啟用擴充功能
「沒有可用的標籤頁」	Claude 在標籤頁準備好之前嘗試操作	要求 Claude 建立新標籤頁並重試
「接收端不存在」	擴充功能服務工作者進入閒置狀態	執行 `/chrome` 並選擇「重新連接擴充功能」

另請參閱

- 在 VS Code 中使用 Claude Code：VS Code 擴充功能中的瀏覽器自動化
- CLI 參考：命令列標誌，包括 `--chrome`
- 常見工作流程：更多使用 Claude Code 的方式
- 資料和隱私：Claude Code 如何處理您的資料
- Claude in Chrome 入門：Chrome 擴充功能的完整文件，包括快捷鍵、排程和權限

Slack 中的 Claude Code

> 直接從您的 Slack 工作區委派編碼任務

Slack 中的 Claude Code 將 Claude Code 的強大功能直接帶入您的 Slack 工作區。當您提及 `@Claude` 並附帶編碼任務時，Claude 會自動檢測意圖並在網路上建立 Claude Code 工作階段，讓您無需離開團隊對話即可委派開發工作。

此整合建立在現有的 Claude for Slack 應用程式基礎上，但為編碼相關請求添加了智能路由到網路上的 Claude Code。

使用案例

- **錯誤調查和修復**：要求 Claude 在 Slack 頻道中報告錯誤時立即調查和修復。
- **快速代碼審查和修改**：讓 Claude 根據團隊反饋實現小功能或重構代碼。
- **協作調試**：當團隊討論提供關鍵背景資訊（例如錯誤重現或用戶報告）時，Claude 可以使用該資訊來指導其調試方法。
- **並行任務執行**：在 Slack 中啟動編碼任務，同時繼續其他工作，完成時接收通知。

先決條件

在使用 Slack 中的 Claude Code 之前，請確保您具有以下條件：

要求	詳情
Claude 計畫	Pro、Max、Team 或 Enterprise，具有 Claude Code 存取權限（高級席位）
網路上的 Claude Code	必須啟用對網路上的 Claude Code 的存取
GitHub 帳戶	連接到網路上的 Claude Code，至少有一個存儲庫已驗證
Slack 驗證	您的 Slack 帳戶通過 Claude 應用程式連接到您的 Claude 帳戶

在 Slack 中設定 Claude Code

Step 1: 在 Slack 中安裝 Claude 應用程式

工作區管理員必須從 Slack 應用程式市場安裝 Claude 應用程式。訪問 Slack 應用程式市場並點擊「Add to Slack」以開始安裝過程。

Step 2: 連接您的 Claude 帳戶

應用程式安裝後，驗證您的個人 Claude 帳戶：

1. 通過點擊您的應用程式部分中的「Claude」在 Slack 中打開 Claude 應用程式
2. 導航到應用程式首頁標籤
3. 點擊「Connect」以將您的 Slack 帳戶與您的 Claude 帳戶連接
4. 在您的瀏覽器中完成驗證流程

Step 3: 配置網路上的 Claude Code

確保您的網路上的 Claude Code 已正確配置：

- 訪問 claude.ai/code 並使用您連接到 Slack 的同一帳戶登入
- 如果尚未連接，請連接您的 GitHub 帳戶
- 驗證至少一個您希望 Claude 使用的存儲庫

Step 4: 選擇您的路由模式

連接帳戶後，配置 Claude 如何在 Slack 中處理您的訊息。導航到 Slack 中的 Claude 應用程式首頁以找到**路由模式**設定。

模式	行為
僅代碼	Claude 將所有 @mentions 路由到 Claude Code 工作階段。最適合使用 Claude in Slack 專門用於開發任務的團隊。
代碼 + 聊天	Claude 分析每條訊息並智能地在 Claude Code（用於編碼任務）和 Claude Chat（用於寫作、分析和一般問題）之間路由。最適合希望為所有類型工作提供單一 @Claude 入口點的團隊。

> **Note:**
>
> 在代碼 + 聊天模式中，如果 Claude 將訊息路由到聊天但您想要編碼工作階段，您可以點擊「Retry as Code」以改為建立 Claude Code 工作階段。同樣，如果它被路由到代碼但您想要聊天工作階段，您可以在該執行緒中選擇該選項。

工作原理

自動檢測

當您在 Slack 頻道或執行緒中提及 @Claude 時，Claude 會自動分析您的訊息以確定它是否是編碼任務。如果 Claude 檢測到編碼意圖，它將把您的請求路由到網路上的 Claude Code，而不是作為常規聊天助手回應。

您也可以明確告訴 Claude 將請求作為編碼任務處理，即使它沒有自動檢測到。

> **Note:**
> Slack 中的 Claude Code 僅在頻道（公開或私人）中工作。它在直接訊息 (DM) 中不起作用。

背景資訊收集

來自執行緒：當您在執行緒中 @mention Claude 時，它會從該執行緒中的所有訊息收集背景資訊以理解完整對話。

來自頻道：當直接在頻道中提及時，Claude 會查看最近的頻道訊息以獲取相關背景資訊。

此背景資訊幫助 Claude 理解問題、選擇適當的存儲庫並指導其任務方法。

> **Warning:**
> 當在 Slack 中調用 @Claude 時，Claude 會獲得對對話背景資訊的存取權限以更好地理解您的請求。Claude 可能會遵循背景資訊中其他訊息的指示，因此用戶應確保僅在受信任的 Slack 對話中使用 Claude。

工作階段流程

1. **啟動**：您 @mention Claude 並提出編碼請求
2. **檢測**：Claude 分析您的訊息並檢測編碼意圖
3. **工作階段建立**：在 claude.ai/code 上建立新的 Claude Code 工作階段
4. **進度更新**：Claude 在工作進行時向您的 Slack 執行緒發佈狀態更新
5. **完成**：完成後，Claude @mentions 您並提供摘要和操作按鈕
6. **審查**：點擊「View Session」以查看完整記錄，或點擊「Create PR」以開啟拉取請求

用戶介面元素

應用程式首頁

應用程式首頁標籤顯示您的連接狀態，並允許您連接或斷開您的 Claude 帳戶與 Slack 的連接。

訊息操作

- **View Session**：在您的瀏覽器中打開完整的 Claude Code 工作階段，您可以在其中查看所有執行的工作、繼續工作階段或提出其他請求。
- **Create PR**：直接從工作階段的更改建立拉取請求。

- **Retry as Code**：如果 Claude 最初作為聊天助手回應但您想要編碼工作階段，點擊此按鈕以將請求重試為 Claude Code 任務。
- **Change Repo**：允許您選擇不同的存儲庫，如果 Claude 選擇不正確。

存儲庫選擇

Claude 根據您的 Slack 對話中的背景資訊自動選擇存儲庫。如果多個存儲庫可能適用，Claude 可能會顯示一個下拉菜單，允許您選擇正確的存儲庫。

存取和權限

用戶級別存取

存取類型	要求
Claude Code 工作階段	每個用戶在其自己的 Claude 帳戶下運行工作階段
使用情況和速率限制	工作階段計入個人用戶的計畫限制
存儲庫存取	用戶只能存取他們個人連接的存儲庫
工作階段歷史記錄	工作階段出現在您的 Claude Code 歷史記錄中，位於 claude.ai/code

工作區管理員權限

Slack 工作區管理員控制 Claude 應用程式是否可以在工作區中安裝。然後個人用戶使用他們自己的 Claude 帳戶進行驗證以使用此整合。

在何處可以存取什麼

在 Slack 中：您將看到狀態更新、完成摘要和操作按鈕。完整記錄被保留並始終可存取。

在網路上：完整的 Claude Code 工作階段，包含完整對話歷史記錄、所有代碼更改、文件操作以及繼續工作階段或建立拉取請求的能力。

最佳實踐

撰寫有效的請求

- **具體明確**：在相關時包括文件名、函數名或錯誤訊息。
- **提供背景資訊**：如果從對話中不清楚，請提及存儲庫或專案。
- **定義成功**：解釋「完成」的樣子——Claude 應該編寫測試嗎？更新文檔？建立拉取請求？

- **使用執行緒**：在討論錯誤或功能時在執行緒中回覆，以便 Claude 可以收集完整背景資訊。

何時使用 Slack 與網路

在以下情況下使用 Slack：背景資訊已存在於 Slack 討論中、您想要非同步啟動任務或與需要可見性的隊友協作。

直接在網路上使用：當您需要上傳文件、想要在開發期間進行實時互動或正在處理更長、更複雜的任務時。

故障排除

工作階段未啟動

1. 驗證您的 Claude 帳戶已在 Claude 應用程式首頁中連接
2. 檢查您是否已啟用網路上的 Claude Code 存取
3. 確保您至少有一個 GitHub 存儲庫連接到 Claude Code

存儲庫未顯示

1. 在 claude.ai/code 的網路上的 Claude Code 中連接存儲庫
2. 驗證您對該存儲庫的 GitHub 權限
3. 嘗試斷開並重新連接您的 GitHub 帳戶

選擇了錯誤的存儲庫

1. 點擊「Change Repo」按鈕以選擇不同的存儲庫
2. 在您的請求中包括存儲庫名稱以獲得更準確的選擇

驗證錯誤

1. 在應用程式首頁中斷開並重新連接您的 Claude 帳戶
2. 確保您在瀏覽器中登入正確的 Claude 帳戶
3. 檢查您的 Claude 計畫是否包括 Claude Code 存取

工作階段過期

1. 工作階段在網路上的 Claude Code 歷史記錄中保持可存取
2. 您可以從 claude.ai/code 繼續或參考過去的工作階段

目前限制

- **僅 GitHub**：目前支持 GitHub 上的存儲庫。

- **一次一個拉取請求**：每個工作階段可以建立一個拉取請求。
- **速率限制適用**：工作階段使用您的個人 Claude 計畫的速率限制。
- **需要網路存取**：用戶必須具有網路上的 Claude Code 存取；沒有它的用戶將只獲得標準 Claude 聊天回應。

相關資源

- 網路上的 Claude Code 了解更多關於網路上的 Claude Code
- Claude for Slack Claude for Slack 一般文檔
- Slack 應用程式市場 從 Slack 市場安裝 Claude 應用程式
- Claude 幫助中心 獲取額外支援

Claude Code 網頁版

> 在安全的雲端基礎設施上非同步執行 Claude Code 任務

> **Note:**
>
> Claude Code 網頁版目前處於研究預覽階段。

什麼是 Claude Code 網頁版？

Claude Code 網頁版讓開發者可以從 Claude 應用程式啟動 Claude Code。這非常適合：

- **回答問題**：詢問程式碼架構以及功能如何實現
- **修復錯誤和例行任務**：定義明確的任務，不需要頻繁調整
- **並行工作**：同時處理多個錯誤修復
- **不在本機上的儲存庫**：處理您未在本機簽出的程式碼
- **後端變更**：Claude Code 可以編寫測試，然後編寫程式碼來通過這些測試

Claude Code 也可在 Claude 應用程式中用於 iOS 和 Android，用於隨時啟動任務和監控進行中的工作。

您可以從終端使用 `--remote` 在網頁上啟動新任務，或將網頁工作階段傳送回終端以在本機繼續。若要在執行 Claude Code 時使用網頁介面而不是雲端基礎設施，請參閱遠端控制。

誰可以使用 Claude Code 網頁版？

Claude Code 網頁版在研究預覽中可供以下人員使用：

- **Pro 使用者**
- **Max 使用者**
- **Team 使用者**
- **Enterprise 使用者**（具有高級席位或 Chat + Claude Code 席位）

開始使用

1. 造訪 claude.ai/code
2. 連接您的 GitHub 帳戶
3. 在您的儲存庫中安裝 Claude GitHub 應用程式

4. 選擇您的預設環境
5. 提交您的編碼任務
6. 在差異檢視中檢查變更，使用評論進行迭代，然後建立拉取請求

運作方式

當您在 Claude Code 網頁版上啟動任務時：

1. **儲存庫複製**：您的儲存庫被複製到 Anthropic 管理的虛擬機器
2. **環境設定**：Claude 準備一個安全的雲端環境，包含您的程式碼，然後執行您的設定指令碼（如果已配置）
3. **網路配置**：根據您的設定配置網際網路存取
4. **任務執行**：Claude 分析程式碼、進行變更、執行測試並檢查其工作
5. **完成**：您會收到完成通知，可以使用變更建立拉取請求
6. **結果**：變更被推送到分支，準備好建立拉取請求

使用差異檢視檢查變更

差異檢視讓您在建立拉取請求之前確切看到 Claude 變更了什麼。與其點擊「建立 PR」在 GitHub 中檢查變更，不如直接在應用程式中檢查差異，並與 Claude 迭代，直到變更準備好。

當 Claude 對檔案進行變更時，會出現一個差異統計指示器，顯示新增和移除的行數（例如 `+12 -1`）。選擇此指示器以開啟差異檢視器，該檢視器在左側顯示檔案清單，在右側顯示每個檔案的變更。

從差異檢視中，您可以：

- 逐個檔案檢查變更
- 對特定變更進行評論以請求修改
- 根據您看到的內容與 Claude 繼續迭代

這讓您可以通過多輪反饋來精煉變更，而無需建立草稿 PR 或切換到 GitHub。

在網頁和終端之間移動任務

您可以從終端在網頁上啟動新任務，或將網頁工作階段拉入終端以在本機繼續。網頁工作階段即使在您關閉筆記型電腦後仍會保留，您可以從任何地方（包括 Claude 行動應用程式）監控它們。

> **Note:**

> 工作階段交接是單向的：您可以將網頁工作階段拉入終端，但無法將現有終端工作階段推送到網頁。`--remote` 旗標為您目前的儲存庫建立一個新的網頁工作階段。

從終端到網頁

使用 `--remote` 旗標從命令列啟動網頁工作階段：

```
claude --remote "Fix the authentication bug in src/auth/login.ts"
```

這會在 claude.ai 上建立一個新的網頁工作階段。任務在雲端執行，而您繼續在本機工作。使用 `/tasks` 檢查進度，或在 claude.ai 或 Claude 行動應用程式上開啟工作階段以直接互動。從那裡，您可以引導 Claude、提供反饋或回答問題，就像任何其他對話一樣。

遠端任務的提示

在本機規劃，在遠端執行：對於複雜任務，在規劃模式下啟動 Claude 以協作制定方法，然後將工作發送到網頁：

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

在規劃模式下，Claude 只能讀取檔案和探索程式碼庫。一旦您對計畫感到滿意，為自主執行啟動遠端工作階段：

```
claude --remote "Execute the migration plan in docs/migration-plan.md"
```

此模式讓您可以控制策略，同時讓 Claude 在雲端自主執行。

並行執行任務：每個 `--remote` 命令建立自己的網頁工作階段，獨立執行。您可以啟動多個任務，它們都會在單獨的工作階段中同時執行：

```
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"
```

使用 `/tasks` 監控所有工作階段。當工作階段完成時，您可以從網頁介面建立拉取請求，或傳送工作階段到終端以繼續工作。

從網頁到終端

有幾種方法可以將網頁工作階段拉入終端：

- **使用 `/teleport`**：在 Claude Code 中，執行 `/teleport`（或 `/tp`）以查看您的網頁工作階段的互動式選擇器。如果您有未提交的變更，系統會提示您先隱藏它們。
- **使用 `--teleport`**：從命令列，執行 `claude --teleport` 以獲得互動式工作階段選擇器，或執行 `claude --teleport <session-id>` 以直接恢復特定工作階段。
- **從 `/tasks`**：執行 `/tasks` 以查看您的背景工作階段，然後按 `t` 傳送到其中一個
- **從網頁介面**：點擊「在 CLI 中開啟」以複製可貼到終端的命令

當您傳送工作階段時，Claude 驗證您在正確的儲存庫中，從遠端工作階段取得並簽出分支，並將完整的對話歷史記錄載入到終端。

傳送的要求

傳送在恢復工作階段之前檢查這些要求。如果任何要求未滿足，您會看到錯誤或被提示解決問題。

要求	詳細資訊
乾淨的 git 狀態	您的工作目錄必須沒有未提交的變更。如果需要，傳送會提示您隱藏變更。
正確的儲存庫	您必須從同一儲存庫的簽出執行 `--teleport`，而不是分支。
分支可用	網頁工作階段中的分支必須已推送到遠端。傳送會自動取得並簽出它。
相同帳戶	您必須驗證到網頁工作階段中使用的相同 Claude.ai 帳戶。

共享工作階段

若要共享工作階段，請根據下面的帳戶類型切換其可見性。之後，按原樣共享工作階段連結。打開您共享工作階段的收件者將在載入時看到工作階段的最新狀態，但收件者的頁面不會即時更新。

從 Enterprise 或 Teams 帳戶共享

對於 Enterprise 和 Teams 帳戶，兩個可見性選項是**私人**和**Team**。Team 可見性使工作階段對您的 Claude.ai 組織的其他成員可見。儲存庫存取驗證預設啟用，基於連接到收件者帳戶的 GitHub 帳戶。您帳戶的顯示名稱對所有有存取權限的收件者可見。Slack 中的 Claude 工作階段會自動以 Team 可見性共享。

從 Max 或 Pro 帳戶共享

對於 Max 和 Pro 帳戶，兩個可見性選項是**私人**和**公開**。公開可見性使工作階段對任何登入 claude.ai 的使用者可見。

在共享之前檢查您的工作階段是否包含敏感內容。工作階段可能包含來自私人 GitHub 儲存庫的程式碼和認證。儲存庫存取驗證預設未啟用。

通過進入「設定」>「Claude Code」>「共享設定」來啟用儲存庫存取驗證和/或從共享工作階段中隱藏您的名稱。

管理工作階段

封存工作階段

您可以封存工作階段以保持工作階段清單的組織。封存的工作階段隱藏在預設工作階段清單中，但可以通過篩選封存的工作階段來檢視。

若要封存工作階段，請在側邊欄中將滑鼠懸停在工作階段上，然後點擊封存圖示。

刪除工作階段

刪除工作階段會永久移除工作階段及其資料。此操作無法撤銷。您可以通過兩種方式刪除工作階段：

- **從側邊欄**：篩選封存的工作階段，然後將滑鼠懸停在您要刪除的工作階段上，並點擊刪除圖示
- **從工作階段功能表**：開啟工作階段，點擊工作階段標題旁的下拉式功能表，然後選擇**刪除**

在刪除工作階段之前，系統會要求您確認。

雲端環境

預設映像

我們建立並維護一個通用映像，預先安裝了常見的工具鏈和語言生態系統。此映像包括：

- 流行的程式設計語言和執行時
- 常見的建置工具和套件管理器
- 測試框架和 linters

檢查可用工具

若要查看環境中預先安裝的內容，請要求 Claude Code 執行：

```
check-tools
```

此命令顯示：

- 程式設計語言及其版本
- 可用的套件管理器
- 已安裝的開發工具

特定語言的設定

通用映像包括以下的預先配置環境：

- **Python**：Python 3.x，包含 pip、poetry 和常見的科學庫
- **Node.js**：最新 LTS 版本，包含 npm、yarn、pnpm 和 bun
- **Ruby**：版本 3.1.6、3.2.6、3.3.6（預設：3.3.6），包含 gem、bundler 和 rbenv 用於版本管理
- **PHP**：版本 8.4.14
- **Java**：OpenJDK，包含 Maven 和 Gradle
- **Go**：最新穩定版本，包含模組支援
- **Rust**：Rust 工具鏈，包含 cargo
- **C++**：GCC 和 Clang 編譯器

資料庫

通用映像包括以下資料庫：

- **PostgreSQL**：版本 16
- **Redis**：版本 7.0

環境配置

當您在 Claude Code 網頁版中啟動工作階段時，以下是幕後發生的情況：

1. **環境準備**：我們複製您的儲存庫並執行任何已配置的設定指令碼。儲存庫將使用您 GitHub 儲存庫上的預設分支進行複製。如果您想簽出特定分支，可以在提示中指定。
2. **網路配置**：我們為代理配置網際網路存取。網際網路存取預設受限，但您可以根據需要配置環境以無網際網路或完全網際網路存取。
3. **Claude Code 執行**：Claude Code 執行以完成您的任務，編寫程式碼、執行測試並檢查其工作。您可以通過網頁介面在整個工作階段中引導和引導 Claude。Claude 尊重您在 `CLAUDE.md` 中定義的上下文。
4. **結果**：當 Claude 完成其工作時，它將推送分支到遠端。您將能夠為分支建立拉取請求。

> **Note:**
>
> Claude 完全通過環境中可用的終端和 CLI 工具進行操作。它使用通用映像中的預先安裝工具以及您通過 hooks 或依賴管理安裝的任何其他工具。

若要新增環境：選擇目前環境以開啟環境選擇器，然後選擇「新增環境」。這將開啟一個對話框，您可以在其中指定環境名稱、網路存取級別、環境變數和設定指令碼。

若要更新現有環境：選擇目前環境，在環境名稱的右側，然後選擇設定按鈕。這將開啟一個對話框，您可以在其中更新環境名稱、網路存取、環境變數和設定指令碼。

若要從終端選擇預設環境：如果您配置了多個環境，執行 `/remote-env` 以選擇使用 `--remote` 從終端啟動網頁工作階段時要使用的環境。使用單一環境，此命令顯示您目前的配置。

> **Note:**
>
> 環境變數必須指定為鍵值對，採用 `.env` 格式。例如：
>
> ```
> API_KEY=your_api_key
> DEBUG=true
> ```

設定指令碼

設定指令碼是一個 Bash 指令碼，在新的雲端工作階段啟動時執行，在 Claude Code 啟動之前。使用設定指令碼來安裝依賴項、配置工具或準備雲端環境需要的任何東西，這些東西不在預設映像中。

指令碼在 Ubuntu 24.04 上以 root 身份執行，因此 `apt install` 和大多數語言套件管理器都可以工作。

> **Tip:**
>
> 在將其新增到指令碼之前，請要求 Claude 在雲端工作階段中執行 `check-tools` 以檢查已安裝的內容。

若要新增設定指令碼，請開啟環境設定對話框，並在**設定指令碼**欄位中輸入您的指令碼。

此範例安裝 `gh` CLI，它不在預設映像中：

```
#!/bin/bash
apt update && apt install -y gh
```

設定指令碼僅在建立新工作階段時執行。恢復現有工作階段時會跳過它們。

如果指令碼以非零值退出，工作階段將無法啟動。將 `|| true` 附加到非關鍵命令以避免在不穩定的安裝上阻止工作階段。

> **Note:**
>
> 安裝套件的設定指令碼需要網路存取才能到達登錄。預設網路存取允許連接到常見套件登錄，包括 npm、PyPI、RubyGems 和 crates.io。如果您的環境禁用了網路存取，指令碼將無法安裝套件。

設定指令碼與 SessionStart hooks

使用設定指令碼來安裝雲端需要但您的筆記型電腦已有的東西，例如語言執行時或 CLI 工具。使用 SessionStart hook 進行應在任何地方執行的專案設定，雲端和本機，例如 `npm install`。

兩者都在工作階段開始時執行，但它們屬於不同的位置：

	設定指令碼	SessionStart hooks
附加到	雲端環境	您的儲存庫
配置在	雲端環境 UI	儲存庫中的 `.claude/settings.json`
執行	在 Claude Code 啟動之前，僅在新工作階段上	在 Claude Code 啟動之後，在每個工作階段上，包括已恢復的
範圍	僅雲端環境	本機和雲端

SessionStart hooks 也可以在本機使用者級別 `~/.claude/settings.json` 中定義，但使用者級別設定不會轉移到雲端工作階段。在雲端中，只有提交到儲存庫的 hooks 執行。

依賴管理

自訂環境映像和快照尚不支援。使用設定指令碼在工作階段啟動時安裝套件，或使用 SessionStart hooks 進行應在本機環境中也執行的依賴安裝。SessionStart hooks 有已知限制。

若要使用設定指令碼配置自動依賴安裝，請開啟環境設定並新增指令碼：

```
#!/bin/bash
npm install
pip install -r requirements.txt
```

或者，您可以在儲存庫的 `.claude/settings.json` 檔案中使用 SessionStart hooks 進行依賴安裝，這應該在本機環境中也執行：

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/scripts/install_pkgs.sh"
          }
        ]
      }
    ]
  }
}
```

在 `scripts/install_pkgs.sh` 建立對應的指令碼：

```
#!/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
```

使其可執行：`chmod +x scripts/install_pkgs.sh`

保留環境變數

SessionStart hooks 可以通過寫入 `CLAUDE_ENV_FILE` 環境變數中指定的檔案來為後續 Bash 命令保留環境變數。有關詳細資訊，請參閱 hooks 參考中的 SessionStart hooks。

依賴管理限制

- **Hooks 對所有工作階段執行**：SessionStart hooks 在本機和遠端環境中執行。沒有 hook 配置來將 hook 限制為僅遠端工作階段。若要跳過本機執行，請檢查指令碼中的 `CLAUDE_CODE_REMOTE` 環境變數，如上所示。
- **需要網路存取**：安裝命令需要網路存取才能到達套件登錄。如果您的環境配置為「無網際網路」存取，這些 hooks 將失敗。使用「有限」（預設）或「完全」網路存取。預設允許清單包括常見登錄，例如 npm、PyPI、RubyGems 和 crates.io。
- **代理相容性**：遠端環境中的所有出站流量都通過安全代理。某些套件管理器無法與此代理正確配合使用。Bun 是一個已知的例子。
- **在每個工作階段啟動時執行**：Hooks 在每次工作階段啟動或恢復時執行，增加啟動延遲。通過在重新安裝之前檢查依賴項是否已存在來保持安裝指令碼快速。

網路存取和安全性

網路政策

GitHub 代理

為了安全起見，所有 GitHub 操作都通過專用代理服務進行，該服務透明地處理所有 git 互動。在沙箱內，git 用戶端使用自訂建置的限定認證進行驗證。此代理：

- 安全地管理 GitHub 驗證 - git 用戶端在沙箱內使用限定認證，代理驗證並將其轉換為您的實際 GitHub 驗證令牌
- 限制 git push 操作到目前工作分支以確保安全
- 啟用無縫複製、取得和拉取請求操作，同時維護安全邊界

安全代理

環境在 HTTP/HTTPS 網路代理後面執行，用於安全和濫用防止目的。所有出站網際網路流量都通過此代理，該代理提供：

- 防止惡意請求
- 速率限制和濫用防止
- 增強安全性的內容篩選

存取級別

預設情況下，網路存取限制為允許清單域。

您可以配置自訂網路存取，包括禁用網路存取。

預設允許的域

使用「有限」網路存取時，預設允許以下域：

Anthropic 服務

- api.anthropic.com
- statsig.anthropic.com
- platform.claude.com
- code.claude.com
- claude.ai

版本控制

- 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

容器登錄

- 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

雲端平台

- 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

套件管理器 - JavaScript/Node

- registry.npmjs.org
- www.npmjs.com
- www.npmjs.org
- npmjs.com
- npmjs.org
- yarnpkg.com
- registry.yarnpkg.com

套件管理器 - Python

- pypi.org
- www.pypi.org
- files.pythonhosted.org
- pythonhosted.org
- test.pypi.org
- pypi.python.org
- pypa.io
- www.pypa.io

套件管理器 - 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

套件管理器 - Rust

- crates.io
- www.crates.io
- index.crates.io
- static.crates.io
- rustup.rs
- static.rust-lang.org
- www.rust-lang.org

套件管理器 - Go

- proxy.golang.org
- sum.golang.org
- index.golang.org
- golang.org
- www.golang.org
- goproxy.io
- pkg.go.dev

套件管理器 - 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

套件管理器 - 其他語言

- 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 發行版

- archive.ubuntu.com
- security.ubuntu.com
- ubuntu.com
- www.ubuntu.com
- *.ubuntu.com
- ppa.launchpad.net
- launchpad.net
- www.launchpad.net

開發工具和平台

- 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

雲端服務和監控

- statsig.com
- www.statsig.com
- api.statsig.com
- sentry.io
- *.sentry.io
- http-intake.logs.datadoghq.com
- *.datadoghq.com
- *.datadoghq.eu

內容傳遞和鏡像

- sourceforge.net
- *.sourceforge.net
- packagecloud.io
- *.packagecloud.io

架構和配置

- json-schema.org
- www.json-schema.org
- json.schemastore.org
- www.schemastore.org

Model Context Protocol

- *.modelcontextprotocol.io

> **Note:**
>
> 標記為 `*` 的域表示萬用字元子域匹配。例如，`*.gcr.io` 允許存取 `gcr.io` 的任何子域。

自訂網路存取的安全最佳實踐

1. **最小權限原則**：僅啟用所需的最小網路存取
2. **定期審計**：定期檢查允許的域
3. **使用 HTTPS**：始終優先使用 HTTPS 端點而不是 HTTP

安全性和隔離

Claude Code 網頁版提供強大的安全保證：

- **隔離的虛擬機器**：每個工作階段在隔離的 Anthropic 管理的虛擬機器中執行
- **網路存取控制**：網路存取預設受限，可以禁用

> **Note:**
>
> 在禁用網路存取的情況下執行時，Claude Code 被允許與 Anthropic API 通訊，這可能仍然允許資料離開隔離的 Claude Code 虛擬機器。

- **認證保護**：敏感認證（例如 git 認證或簽署金鑰）永遠不在沙箱內與 Claude Code 一起。驗證通過使用限定認證的安全代理進行處理
- **安全分析**：程式碼在隔離的虛擬機器內進行分析和修改，然後建立拉取請求

定價和速率限制

Claude Code 網頁版與您帳戶內所有其他 Claude 和 Claude Code 使用共享速率限制。並行執行多個任務將按比例消耗更多速率限制。

限制

- **儲存庫驗證**：您只能在驗證到相同帳戶時將工作階段從網頁移動到本機
- **平台限制**：Claude Code 網頁版僅適用於在 GitHub 中託管的程式碼。GitLab 和其他非 GitHub 儲存庫無法與雲端工作階段一起使用

最佳實踐

1. **自動化環境設定**：使用設定指令碼在 Claude Code 啟動之前安裝依賴項和配置工具。對於更高級的場景，配置 SessionStart hooks。
2. **記錄要求**：在 `CLAUDE.md` 檔案中清楚地指定依賴項和命令。如果您有 `AGENTS.md` 檔案，可以在 `CLAUDE.md` 中使用 `@AGENTS.md` 來源它以維護單一事實來源。

相關資源

- Hooks 配置
- 設定參考
- 安全性
- 資料使用

Part 9: Security & Privacy

安全性

> 了解 Claude Code 的安全防護措施和安全使用的最佳實踐。

我們如何處理安全性

安全基礎

您的程式碼安全至關重要。Claude Code 以安全為核心進行構建，按照 Anthropic 的全面安全計畫開發。在 Anthropic Trust Center 了解更多資訊並存取資源（SOC 2 Type 2 報告、ISO 27001 證書等）。

基於權限的架構

Claude Code 預設使用嚴格的唯讀權限。當需要額外操作時（編輯檔案、執行測試、執行命令），Claude Code 會請求明確的權限。使用者可以控制是否批准一次性操作或自動允許操作。

我們設計 Claude Code 以實現透明和安全。例如，我們要求在執行 bash 命令前進行批准，讓您擁有直接控制權。這種方法使使用者和組織能夠直接配置權限。

有關詳細的權限配置，請參閱 Permissions。

內建保護

為了降低代理系統中的風險：

- **沙箱化 bash 工具**：Sandbox bash 命令具有檔案系統和網路隔離，減少權限提示同時保持安全性。使用 `/sandbox` 啟用以定義 Claude Code 可以自主工作的邊界
- **寫入存取限制**：Claude Code 只能寫入啟動它的資料夾及其子資料夾——它無法在沒有明確權限的情況下修改父目錄中的檔案。雖然 Claude Code 可以讀取工作目錄外的檔案（對於存取系統庫和依賴項很有用），但寫入操作嚴格限制在專案範圍內，建立了清晰的安全邊界
- **提示疲勞緩解**：支援按使用者、按程式碼庫或按組織允許列表常用的安全命令
- **Accept Edits 模式**：批量接受多個編輯，同時為具有副作用的命令保持權限提示

使用者責任

Claude Code 只擁有您授予它的權限。您負責在批准前審查建議的程式碼和命令的安全性。

防止提示注入

提示注入是一種技術，攻擊者試圖通過插入惡意文本來覆蓋或操縱 AI 助手的指令。Claude Code 包括針對這些攻擊的多項防護措施：

核心保護

- **權限系統**：敏感操作需要明確批准
- **上下文感知分析**：通過分析完整請求來檢測潛在有害的指令
- **輸入淨化**：通過處理使用者輸入來防止命令注入
- **命令黑名單**：預設阻止從網路獲取任意內容的風險命令，如 `curl` 和 `wget`。明確允許時，請注意 permission pattern limitations

隱私保護

我們實施了多項保護措施來保護您的資料，包括：

- 敏感資訊的有限保留期（請參閱 Privacy Center 了解更多）
- 對使用者會話資料的受限存取
- 使用者對資料訓練偏好的控制。消費者使用者可以隨時更改其 privacy settings。

有關完整詳情，請查閱我們的 Commercial Terms of Service（適用於 Team、Enterprise 和 API 使用者）或 Consumer Terms（適用於 Free、Pro 和 Max 使用者）以及 Privacy Policy。

額外保護措施

- **網路請求批准**：進行網路請求的工具預設需要使用者批准
- **隔離的上下文視窗**：Web fetch 使用單獨的上下文視窗以避免注入潛在的惡意提示
- **信任驗證**：首次程式碼庫執行和新的 MCP servers 需要信任驗證
  - 注意：使用 `-p` 標誌以非互動方式執行時，信任驗證被禁用
- **命令注入檢測**：即使之前已允許列表，可疑的 bash 命令也需要手動批准
- **故障關閉匹配**：不匹配的命令預設需要手動批准
- **自然語言描述**：複雜的 bash 命令包括使用者理解的說明
- **安全的認證儲存**：API 金鑰和令牌已加密。請參閱 Credential Management

> **Warning:**
>
> **Windows WebDAV 安全風險**：在 Windows 上執行 Claude Code 時，我們建議不要啟用 WebDAV 或允許 Claude Code 存取可能包含 WebDAV 子目錄的路徑，如 `\\*`。WebDAV 已被 Microsoft 棄用，原因是安全風險。啟用 WebDAV 可能允許 Claude Code 觸發對遠端主機的網路請求，繞過權限系統。

使用不受信任內容的最佳實踐：

1. 在批准前審查建議的命令
2. 避免直接將不受信任的內容傳送給 Claude
3. 驗證對關鍵檔案的建議更改
4. 使用虛擬機器 (VM) 執行指令碼和進行工具呼叫，特別是在與外部網路服務互動時
5. 使用 `/bug` 報告可疑行為

> **Warning:**
>
> 雖然這些保護措施大大降低了風險，但沒有任何系統完全免疫所有攻擊。在使用任何 AI 工具時，始終保持良好的安全實踐。

MCP 安全性

Claude Code 允許使用者配置 Model Context Protocol (MCP) servers。允許的 MCP servers 列表在您的原始程式碼中配置，作為 Claude Code 設定的一部分，工程師將其簽入原始碼控制。

我們鼓勵編寫您自己的 MCP servers 或使用來自您信任的提供者的 MCP servers。您能夠為 MCP servers 配置 Claude Code 權限。Anthropic 不管理或審計任何 MCP servers。

IDE 安全性

有關在 IDE 中執行 Claude Code 的更多資訊，請參閱 VS Code security and privacy。

雲端執行安全性

使用 Claude Code on the web 時，會實施額外的安全控制：

- **隔離的虛擬機器**：每個雲端會話在隔離的、由 Anthropic 管理的 VM 中執行
- **網路存取控制**：網路存取預設受限，可以配置為禁用或僅允許特定網域
- **認證保護**：身份驗證通過安全代理進行處理，該代理在沙箱內使用範圍限定的認證，然後轉換為您的實際 GitHub 身份驗證令牌

- **分支限制**：Git push 操作限制在目前工作分支
- **審計日誌**：雲端環境中的所有操作都被記錄以用於合規和審計目的
- **自動清理**：會話完成後，雲端環境會自動終止

有關雲端執行的更多詳情，請參閱 Claude Code on the web。

Remote Control 會話的工作方式不同：網路介面連接到在您本地機器上執行的 Claude Code 程序。所有程式碼執行和檔案存取保持本地，任何本地 Claude Code 會話期間流動的相同資料通過 TLS 上的 Anthropic API 傳輸。不涉及雲端 VM 或沙箱化。連接使用多個短期、範圍狹窄的認證，每個認證限制於特定目的並獨立過期，以限制任何單一洩露認證的影響範圍。

安全最佳實踐

使用敏感程式碼

- 在批准前審查所有建議的更改
- 為敏感儲存庫使用專案特定的權限設定
- 考慮使用 devcontainers 以獲得額外隔離
- 使用 `/permissions` 定期審計您的權限設定

團隊安全性

- 使用 managed settings 強制執行組織標準
- 通過版本控制共享已批准的權限配置
- 培訓團隊成員了解安全最佳實踐
- 通過 OpenTelemetry metrics 監控 Claude Code 使用情況
- 使用 `ConfigChange` hooks 審計或阻止會話期間的設定更改

報告安全問題

如果您在 Claude Code 中發現安全漏洞：

1. 不要公開披露
2. 通過我們的 HackerOne program 報告
3. 包括詳細的重現步驟
4. 在公開披露前留出時間讓我們解決問題

相關資源

- Sandboxing - bash 命令的檔案系統和網路隔離
- Permissions - 配置權限和存取控制

- Monitoring usage - 追蹤和審計 Claude Code 活動
- Development containers - 安全、隔離的環境
- Anthropic Trust Center - 安全認證和合規性

Sandboxing

> 了解 Claude Code 的沙箱化 bash 工具如何提供檔案系統和網路隔離，以實現更安全、更自主的代理執行。

概述

Claude Code 具有原生沙箱化功能，為代理執行提供更安全的環境，同時減少對持續權限提示的需求。沙箱化不是要求每個 bash 命令的權限，而是預先建立定義的邊界，讓 Claude Code 能夠以降低風險的方式更自由地工作。

沙箱化 bash 工具使用作業系統級別的原語來強制執行檔案系統和網路隔離。

為什麼沙箱化很重要

傳統的基於權限的安全性需要對 bash 命令進行持續的使用者批准。雖然這提供了控制，但可能導致：

- **批准疲勞**：重複點擊「批准」可能導致使用者對他們批准的內容關注度降低
- **生產力降低**：持續的中斷會減慢開發工作流程
- **自主性受限**：當等待批准時，Claude Code 無法高效工作

沙箱化通過以下方式解決這些挑戰：

1. **定義清晰的邊界**：精確指定 Claude Code 可以存取的目錄和網路主機
2. **減少權限提示**：沙箱內的安全命令不需要批准
3. **維持安全性**：嘗試存取沙箱外的資源會觸發立即通知
4. **啟用自主性**：Claude Code 可以在定義的限制內更獨立地運行

> **Warning:**
>
> 有效的沙箱化需要**同時**進行檔案系統和網路隔離。沒有網路隔離，受損的代理可能會洩露敏感檔案，如 SSH 金鑰。沒有檔案系統隔離，受損的代理可能會後門系統資源以獲得網路存取。配置沙箱化時，重要的是確保您配置的設定不會在這些系統中建立繞過。

它如何運作

檔案系統隔離

沙箱化 bash 工具將檔案系統存取限制在特定目錄：

- **預設寫入行為**：對目前工作目錄及其子目錄的讀取和寫入存取
- **預設讀取行為**：對整個電腦的讀取存取，除了某些被拒絕的目錄
- **被阻止的存取**：無法在沒有明確權限的情況下修改目前工作目錄外的檔案
- **可配置**：通過設定定義自訂允許和拒絕的路徑

您可以使用設定中的 `sandbox.filesystem.allowWrite` 授予對其他路徑的寫入存取。這些限制在作業系統級別強制執行（macOS 上的 Seatbelt，Linux 上的 bubblewrap），因此它們適用於所有子流程命令，包括 `kubectl`、`terraform` 和 `npm` 等工具，而不僅僅是 Claude 的檔案工具。

網路隔離

網路存取通過在沙箱外運行的代理伺服器進行控制：

- **域名限制**：只能存取已批准的域名
- **使用者確認**：新的域名請求會觸發權限提示（除非啟用了 `allowManagedDomainsOnly`，它會自動阻止非允許的域名）
- **自訂代理支援**：進階使用者可以在出站流量上實施自訂規則
- **全面覆蓋**：限制適用於所有指令碼、程式和由命令產生的子流程

作業系統級別的強制執行

沙箱化 bash 工具利用作業系統安全原語：

- **macOS**：使用 Seatbelt 進行沙箱強制執行
- **Linux**：使用 bubblewrap 進行隔離
- **WSL2**：使用 bubblewrap，與 Linux 相同

不支援 WSL1，因為 bubblewrap 需要僅在 WSL2 中可用的核心功能。

這些作業系統級別的限制確保由 Claude Code 命令產生的所有子流程都繼承相同的安全邊界。

入門

先決條件

在 **macOS** 上，沙箱化使用內建的 Seatbelt 框架開箱即用。

在 **Linux 和 WSL2** 上，首先安裝所需的套件：

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

啟用沙箱化

您可以通過執行 `/sandbox` 命令來啟用沙箱化：

```
/sandbox
```

這會開啟一個選單，您可以在其中選擇沙箱模式。如果缺少所需的依賴項（例如 Linux 上的 `bubblewrap` 或 `socat`），選單會顯示您平台的安裝說明。

沙箱模式

Claude Code 提供兩種沙箱模式：

自動允許模式：Bash 命令將嘗試在沙箱內運行，並自動允許而無需權限。無法沙箱化的命令（例如需要存取非允許主機的網路存取的命令）會回退到常規權限流程。您配置的明確詢問/拒絕規則始終被尊重。

常規權限模式：所有 bash 命令都通過標準權限流程進行，即使沙箱化也是如此。這提供了更多控制，但需要更多批准。

在兩種模式中，沙箱強制執行相同的檔案系統和網路限制。區別僅在於沙箱化命令是自動批准還是需要明確權限。

> **Info:**
>
> 自動允許模式獨立於您的權限模式設定工作。即使您不在「接受編輯」模式中，當啟用自動允許時，沙箱化 bash 命令也會自動運行。這意味著在沙箱邊界內修改檔案的 bash 命令將執行而不提示，即使檔案編輯工具通常需要批准。

配置沙箱化

通過您的 `settings.json` 檔案自訂沙箱行為。有關完整配置參考，請參閱 Settings。

授予子流程對特定路徑的寫入存取

預設情況下，沙箱化命令只能寫入目前工作目錄。如果子流程命令（如 `kubectl`、`terraform` 或 `npm`）需要寫入專案目錄外，請使用 `sandbox.filesystem.allowWrite` 授予對特定路徑的存取：

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "allowWrite": ["~/.kube", "//tmp/build"]
    }
  }
}
```

這些路徑在作業系統級別強制執行，因此在沙箱內運行的所有命令（包括其子流程）都尊重它們。當工具需要對特定位置的寫入存取時，這是推薦的方法，而不是使用 `excludedCommands` 將工具排除在沙箱外。

當在多個 settings scopes 中定義 `allowWrite`（或 `denyWrite` / `denyRead`）時，陣列被**合併**，這意味著來自每個範圍的路徑被組合，而不是被替換。例如，如果受管設定允許寫入 `//opt/company-tools`，而使用者在其個人設定中新增 `~/.kube`，則兩個路徑都包含在最終沙箱配置中。這意味著使用者和專案可以擴展清單而無需複製或覆蓋由更高優先級範圍設定的路徑。

路徑前綴控制路徑的解析方式：

前綴	含義	範例
`//`	從檔案系統根目錄的絕對路徑	`//tmp/build` 變成 `/tmp/build`
`~/`	相對於主目錄	`~/.kube` 變成 `$HOME/.kube`
`/`	相對於設定檔案的目錄	`/build` 變成 `$SETTINGS_DIR/build`
`./` 或無前綴	相對路徑（由沙箱執行時解析）	`./output`

您也可以使用 `sandbox.filesystem.denyWrite` 和 `sandbox.filesystem.denyRead` 拒絕寫入或讀取存取。這些與來自 `Edit(...)` 和 `Read(...)` 權限規則的任何路徑合併。

Tip:

並非所有命令都與沙箱化開箱即用相容。一些可能幫助您充分利用沙箱的注意事項：

- 許多 CLI 工具需要存取某些主機。當您使用這些工具時，它們將請求權限以存取某些主機。授予權限將允許它們現在和將來存取這些主機，使它們能夠在沙箱內安全執行。
- `watchman` 與在沙箱中運行不相容。如果您正在執行 `jest`，請考慮使用 `jest --no-watchman`
- `docker` 與在沙箱中運行不相容。考慮在 `excludedCommands` 中指定 `docker` 以強制其在沙箱外運行。

Note:

Claude Code 包含一個有意的逃生艙機制，允許命令在必要時在沙箱外運行。當命令因沙箱限制而失敗時（例如網路連接問題或不相容的工具），Claude 會被提示分析失敗，並可能使用 `dangerouslyDisableSandbox` 參數重試命令。使用此參數的命令通過需要使用者權限執行的常規 Claude Code 權限流程進行。這允許 Claude Code 處理某些工具或網路操作無法在沙箱約束內運作的邊界情況。

您可以通過在 sandbox settings 中設定 `"allowUnsandboxedCommands": false` 來禁用此逃生艙。禁用時，`dangerouslyDisableSandbox` 參數被完全忽略，所有命令必須沙箱化運行或在 `excludedCommands` 中明確列出。

安全優勢

防止提示注入

即使攻擊者通過提示注入成功操縱 Claude Code 的行為，沙箱也確保您的系統保持安全：

檔案系統保護：

- 無法修改關鍵配置檔案，如 `~/.bashrc`
- 無法修改 `/bin/` 中的系統級檔案
- 無法讀取在您的 Claude 權限設定 中被拒絕的檔案

網路保護：

- 無法將資料洩露到攻擊者控制的伺服器
- 無法從未授權的域名下載惡意指令碼
- 無法對未批准的服務進行意外的 API 呼叫
- 無法聯繫任何未明確允許的域名

監控和控制：

- 所有在沙箱外的存取嘗試都在作業系統級別被阻止
- 當邊界被測試時，您會收到立即通知
- 您可以選擇拒絕、允許一次或永久更新您的配置

減少攻擊面

沙箱化限制了以下可能造成的損害：

- **惡意依賴項**：具有有害程式碼的 NPM 套件或其他依賴項
- **受損指令碼**：具有安全漏洞的構建指令碼或工具
- **社交工程**：欺騙使用者執行危險命令的攻擊
- **提示注入**：欺騙 Claude 執行危險命令的攻擊

透明操作

當 Claude Code 嘗試存取沙箱外的網路資源時：

1. 操作在作業系統級別被阻止
2. 您會收到立即通知
3. 您可以選擇：
    - 拒絕請求
    - 允許一次
    - 更新您的沙箱配置以永久允許它

安全限制

- 網路沙箱化限制：網路過濾系統通過限制流程允許連接的域名來運作。它不會以其他方式檢查通過代理的流量，使用者負責確保他們在其策略中只允許受信任的域名。

> **Warning:**
>
> 使用者應該意識到允許廣泛域名（如 `github.com`）可能帶來的潛在風險，這可能允許資料洩露。此外，在某些情況下，可能可以通過 domain fronting 繞過網路過濾。

- Unix 套接字特權提升：`allowUnixSockets` 配置可能會無意中授予對強大系統服務的存取，這可能導致沙箱繞過。例如，如果它用於允許存取 `/var/run/docker.sock`，這將有效地通過利用 docker 套接字授予對主機系統的存取。鼓勵使用者仔細考慮他們通過沙箱允許的任何 unix 套接字。

- 檔案系統權限提升：過於寬泛的檔案系統寫入權限可能導致特權提升攻擊。允許寫入包含 `$PATH` 中可執行檔案的目錄、系統配置目錄或使用者 shell 配置檔案（`.bashrc`、`.zshrc`）可能導致當其他使用者或系統流程存取這些檔案時在不同安全上下文中執行程式碼。
- Linux 沙箱強度：Linux 實現提供強大的檔案系統和網路隔離，但包含一個 `enableWeakerNestedSandbox` 模式，使其能夠在 Docker 環境中工作而無需特權命名空間。此選項大大削弱了安全性，應僅在其他隔離被強制執行的情況下使用。

沙箱化與權限的關係

沙箱化和 permissions 是協同工作的互補安全層：

- **權限**控制 Claude Code 可以使用哪些工具，並在任何工具運行之前進行評估。它們適用於所有工具：Bash、Read、Edit、WebFetch、MCP 和其他工具。
- **沙箱化**提供作業系統級別的強制執行，限制 Bash 命令在檔案系統和網路級別可以存取的內容。它僅適用於 Bash 命令及其子流程。

檔案系統和網路限制通過沙箱設定和權限規則進行配置：

- 使用 `sandbox.filesystem.allowWrite` 授予子流程對工作目錄外路徑的寫入存取
- 使用 `sandbox.filesystem.denyWrite` 和 `sandbox.filesystem.denyRead` 阻止子流程對特定路徑的存取
- 使用 `Read` 和 `Edit` 拒絕規則阻止對特定檔案或目錄的存取
- 使用 `WebFetch` 允許/拒絕規則控制域名存取
- 使用沙箱 `allowedDomains` 控制 Bash 命令可以到達的域名

來自 `sandbox.filesystem` 設定和權限規則的路徑被合併到最終沙箱配置中。

此 repository 包含常見部署場景的入門設定配置，包括沙箱特定的範例。使用這些作為起點，並根據您的需求進行調整。

進階用法

自訂代理配置

對於需要進階網路安全的組織，您可以實施自訂代理以：

- 解密和檢查 HTTPS 流量
- 應用自訂過濾規則
- 記錄所有網路請求
- 與現有安全基礎設施整合

```
{
  "sandbox": {
    "network": {
      "httpProxyPort": 8080,
      "socksProxyPort": 8081
    }
  }
}
```

與現有安全工具的整合

沙箱化 bash 工具與以下工具配合使用：

- **權限規則**：與 permission settings 結合以實現深度防禦
- **開發容器**：與 devcontainers 一起使用以獲得額外隔離
- **企業策略**：通過 managed settings 強制執行沙箱配置

最佳實踐

1. **從限制性開始**：從最小權限開始，根據需要擴展
2. **監控日誌**：檢查沙箱違規嘗試以了解 Claude Code 的需求
3. **使用環境特定配置**：開發與生產環境的不同沙箱規則
4. **與權限結合**：將沙箱化與 IAM 策略一起使用以實現全面安全
5. **測試配置**：驗證您的沙箱設定不會阻止合法工作流程

開源

沙箱執行時可作為開源 npm 套件供您在自己的代理專案中使用。這使更廣泛的 AI 代理社群能夠構建更安全、更安全的自主系統。這也可以用於沙箱化您可能希望運行的其他程式。例如，要沙箱化 MCP 伺服器，您可以執行：

```
npx @anthropic-ai/sandbox-runtime <command-to-sandbox>
```

有關實現詳情和原始程式碼，請訪問 GitHub repository。

限制

- **效能開銷**：最小，但某些檔案系統操作可能稍慢

- **相容性**：某些需要特定系統存取模式的工具可能需要配置調整，或甚至可能需要在沙箱外運行
- **平台支援**：支援 macOS、Linux 和 WSL2。不支援 WSL1。計劃提供原生 Windows 支援。

另請參閱

- Security - 全面的安全功能和最佳實踐
- Permissions - 權限配置和存取控制
- Settings - 完整配置參考
- CLI reference - 命令列選項

Checkpointing

> 自動追蹤並回溯 Claude 的編輯，快速恢復不想要的變更。

Claude Code 會自動追蹤 Claude 在您工作時所做的檔案編輯，讓您可以快速復原變更並回溯到先前的狀態，以防任何事情出現偏差。

Checkpointing 如何運作

當您與 Claude 合作時，checkpointing 會自動在每次編輯前捕捉您程式碼的狀態。這個安全網讓您可以放心地進行雄心勃勃的大規模任務，因為您隨時可以回到先前的程式碼狀態。

自動追蹤

Claude Code 追蹤其檔案編輯工具所做的所有變更：

- 每個使用者提示都會建立一個新的 checkpoint
- Checkpoints 在工作階段之間持續存在，因此您可以在恢復的對話中存取它們
- 在 30 天後自動清理（可配置）

回溯變更

按兩次 `Esc`（`Esc` + `Esc`）或使用 `/rewind` 命令來開啟回溯選單。您可以選擇恢復：

- **僅對話**：回溯到使用者訊息，同時保留程式碼變更
- **僅程式碼**：還原檔案變更，同時保留對話
- **程式碼和對話**：將兩者都恢復到工作階段中的先前點

常見使用案例

Checkpoints 在以下情況特別有用：

- **探索替代方案**：嘗試不同的實作方法，而不會失去您的起點
- **從錯誤中恢復**：快速復原引入錯誤或破壞功能的變更
- **反覆迭代功能**：實驗變化，同時知道您可以回到工作狀態

限制

Bash 命令變更未被追蹤

Checkpointing 不追蹤由 bash 命令修改的檔案。例如，如果 Claude Code 執行：

```
rm file.txt
mv old.txt new.txt
cp source.txt dest.txt
```

這些檔案修改無法透過回溯復原。只有透過 Claude 的檔案編輯工具所做的直接檔案編輯才會被追蹤。

外部變更未被追蹤

Checkpointing 只追蹤在目前工作階段中已編輯的檔案。您在 Claude Code 外部手動對檔案所做的變更，以及來自其他並行工作階段的編輯通常不會被捕捉，除非它們碰巧修改了與目前工作階段相同的檔案。

不是版本控制的替代品

Checkpoints 設計用於快速的工作階段級恢復。對於永久版本歷史和協作：

- 繼續使用版本控制（例如 Git）進行提交、分支和長期歷史
- Checkpoints 補充但不替代適當的版本控制
- 將 checkpoints 視為「本地復原」，將 Git 視為「永久歷史」

另請參閱

- Interactive mode - 快捷鍵和工作階段控制
- Built-in commands - 使用 `/rewind` 存取 checkpoints
- CLI reference - 命令列選項

資料使用

> 了解 Anthropic 對 Claude 資料使用政策

資料政策

資料訓練政策

消費者使用者（免費、Pro 和 Max 方案）：我們讓您可以選擇是否允許您的資料用於改進未來的 Claude 模型。當此設定開啟時，我們將使用來自免費、Pro 和 Max 帳戶的資料來訓練新模型（包括當您從這些帳戶使用 Claude Code 時）。

商業使用者：（Team 和 Enterprise 方案、API、第三方平台和 Claude Gov）維持現有政策：除非客戶選擇向我們提供資料以改進模型（例如，開發者合作夥伴計畫），否則 Anthropic 不會在商業條款下使用發送至 Claude Code 的程式碼或提示來訓練生成模型。

開發者合作夥伴計畫

如果您明確選擇加入向我們提供訓練材料的方法，例如透過開發者合作夥伴計畫，我們可能會使用所提供的材料來訓練我們的模型。組織管理員可以明確選擇為其組織加入開發者合作夥伴計畫。請注意，此計畫僅適用於 Anthropic 第一方 API，不適用於 Bedrock 或 Vertex 使用者。

使用 `/bug` 命令的回饋

如果您選擇使用 `/bug` 命令向我們發送有關 Claude Code 的回饋，我們可能會使用您的回饋來改進我們的產品和服務。透過 `/bug` 共享的文字記錄會保留 5 年。

工作階段品質調查

當您在 Claude Code 中看到「Claude 在此工作階段中表現如何？」提示時，回應此調查（包括選擇「關閉」），只會記錄您的數字評分（1、2、3 或關閉）。我們不會作為此調查的一部分收集或儲存任何對話文字記錄、輸入、輸出或其他工作階段資料。與豎起大拇指/向下大拇指回饋或 `/bug` 報告不同，此工作階段品質調查是一個簡單的產品滿意度指標。您對此調查的回應不會影響您的資料訓練偏好設定，也不能用於訓練我們的 AI 模型。

若要停用這些調查，請設定 `CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1`。使用第三方提供者（Bedrock、Vertex、Foundry）或停用遙測時，調查也會自動停用。

資料保留

Anthropic 根據您的帳戶類型和偏好設定保留 Claude Code 資料。

消費者使用者（免費、Pro 和 Max 方案）：

- 允許資料用於模型改進的使用者：5 年保留期，以支持模型開發和安全改進
- 不允許資料用於模型改進的使用者：30 天保留期
- 隱私設定可以隨時在 claude.ai/settings/data-privacy-controls 變更。

商業使用者（Team、Enterprise 和 API）：

- 標準：30 天保留期
- 零資料保留：適用於 Claude for Enterprise 上的 Claude Code。ZDR 按組織啟用；每個新組織必須由您的帳戶團隊單獨啟用 ZDR
- 本機快取：Claude Code 用戶端可能會在本機儲存工作階段長達 30 天，以啟用工作階段繼續（可配置）

您可以隨時刪除網路上的個別 Claude Code 工作階段。刪除工作階段會永久移除工作階段的事件資料。如需有關如何刪除工作階段的說明，請參閱管理工作階段。

在我們的隱私中心了解更多有關資料保留實踐的資訊。

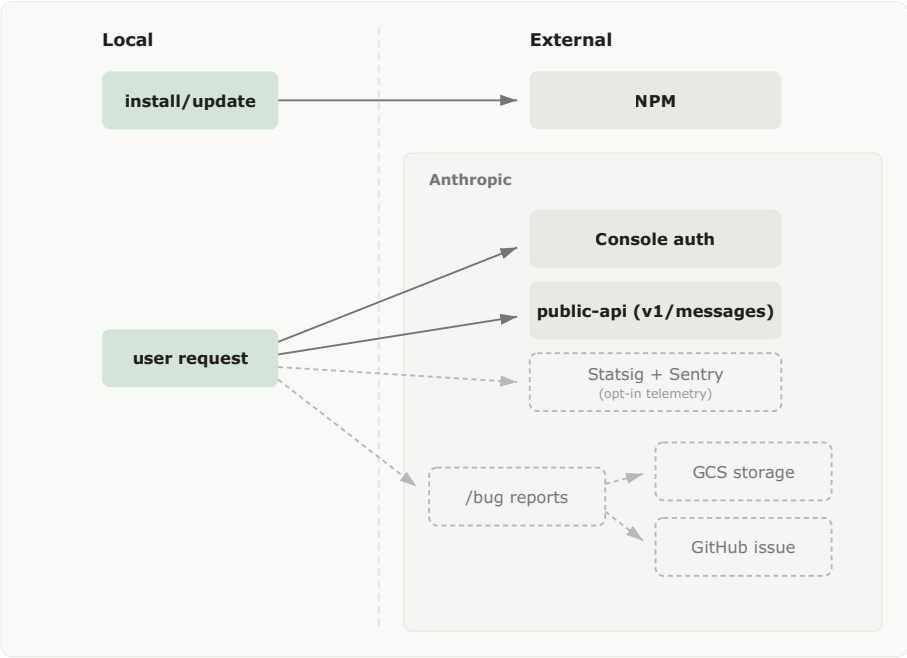
如需完整詳細資訊，請查閱我們的商業服務條款（適用於 Team、Enterprise 和 API 使用者）或消費者條款（適用於免費、Pro 和 Max 使用者）和隱私政策。

資料存取

對於所有第一方使用者，您可以了解更多有關為本機 Claude Code 和遠端 Claude Code 記錄的資料。遠端控制工作階段遵循本機資料流，因為所有執行都在您的機器上進行。請注意，對於遠端 Claude Code，Claude 會存取您啟動 Claude Code 工作階段的儲存庫。Claude 不會存取您已連接但尚未在其中啟動工作階段的儲存庫。

本機 Claude Code：資料流和相依性

下圖顯示 Claude Code 在安裝和正常操作期間如何連接到外部服務。實線表示必需的連接，而虛線表示可選或使用者啟動的資料流。



顯示 Claude Code 外部連接的圖表：安裝/更新連接到 NPM，使用者請求連接到 Anthropic 服務，包括 Console 驗證、public-api，以及可選的 Statsig、Sentry 和錯誤報告

Claude Code 從 NPM 安裝。Claude Code 在本機執行。為了與 LLM 互動，Claude Code 透過網路發送資料。此資料包括所有使用者提示和模型輸出。資料在傳輸中透過 TLS 加密，在靜止時未加密。Claude Code 與大多數流行的 VPN 和 LLM 代理相容。

Claude Code 建立在 Anthropic 的 API 上。有關我們 API 的安全控制詳細資訊，包括我們的 API 記錄程序，請參閱 Anthropic 信任中心中提供的合規性文件。

雲端執行：資料流和相依性

使用網路上的 Claude Code 時，工作階段在 Anthropic 管理的虛擬機器中執行，而不是在本機執行。在雲端環境中：

- **程式碼和資料儲存：**您的儲存庫被複製到隔離的 VM。程式碼和工作階段資料受您帳戶類型的保留和使用政策約束（請參閱上面的資料保留部分）
- **認證：**GitHub 驗證透過安全代理進行；您的 GitHub 認證永遠不會進入沙箱
- **網路流量：**所有出站流量都透過安全代理進行，用於稽核記錄和濫用防止
- **工作階段資料：**提示、程式碼變更和輸出遵循與本機 Claude Code 使用相同的資料政策

有關雲端執行的安全詳細資訊，請參閱安全性。

遙測服務

Claude Code 從使用者的機器連接到 Statsig 服務，以記錄延遲、可靠性和使用模式等操作指標。此記錄不包括任何程式碼或檔案路徑。資料在傳輸中使用 TLS 加密，在靜止時使用 256 位 AES 加密。在 Statsig 安全文件中閱讀更多資訊。若要選擇退出 Statsig 遙測，請設定 `DISABLE_TELEMETRY` 環境變數。

Claude Code 從使用者的機器連接到 Sentry 進行操作錯誤記錄。資料在傳輸中使用 TLS 加密，在靜止時使用 256 位 AES 加密。在 Sentry 安全文件中閱讀更多資訊。若要選擇退出錯誤記錄，請設定 `DISABLE_ERROR_REPORTING` 環境變數。

當使用者執行 `/bug` 命令時，他們的完整對話歷史記錄（包括程式碼）的副本會發送到 Anthropic。資料在傳輸中和靜止時加密。可選地，在我們的公開儲存庫中建立 Github 問題。若要選擇退出錯誤報告，請設定 `DISABLE_BUG_COMMAND` 環境變數。

按 API 提供者的預設行為

根據預設，當使用 Bedrock、Vertex 或 Foundry 時，我們會停用所有非必要流量（包括錯誤報告、遙測、錯誤報告功能和工作階段品質調查）。您也可以透過設定 `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` 環境變數一次選擇退出所有這些。以下是完整的預設行為：

服務	Claude API	Vertex API	Bedrock API	Foundry API
Statsig（指標）	預設開啟。`DISABLE_TELEMETRY=1` 以停用。	預設關閉。`CLAUDE_CODE_USE_VERTEX` 必須為 1。	預設關閉。`CLAUDE_CODE_USE_BEDROCK` 必須為 1。	預設關閉。`CLAUDE_CODE_USE_FOUNDRY` 必須為 1。
Sentry（錯誤）	預設開啟。`DISABLE_ERROR_REPORTING=1` 以停用。	預設關閉。`CLAUDE_CODE_USE_VERTEX` 必須為 1。	預設關閉。`CLAUDE_CODE_USE_BEDROCK` 必須為 1。	預設關閉。`CLAUDE_CODE_USE_FOUNDRY` 必須為 1。
Claude API（`/bug` 報告）	預設開啟。`DISABLE_BUG_COMMAND=1` 以停用。	預設關閉。`CLAUDE_CODE_USE_VERTEX` 必須為 1。	預設關閉。`CLAUDE_CODE_USE_BEDROCK` 必須為 1。	預設關閉。`CLAUDE_CODE_USE_FOUNDRY` 必須為 1。

服務	Claude API	Vertex API	Bedrock API	Foundry API
工作階段品質調查	預設開啟。`CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1` 以停用。	預設關閉。`CLAUDE_CODE_USE_VERTEX` 必須為 1。	預設關閉。`CLAUDE_CODE_USE_BEDROCK` 必須為 1。	預設關閉。`CLAUDE_CODE_USE_FOUNDRY` 必須為 1。

所有環境變數都可以簽入 `settings.json`（閱讀更多）。

法律和合規

> Claude Code 的法律協議、合規認證和安全資訊。

法律協議

許可證

您對 Claude Code 的使用受以下條款約束：

- 商業條款 - 適用於 Team、Enterprise 和 Claude API 使用者
- 消費者條款 - 適用於 Free、Pro 和 Max 使用者

商業協議

無論您是直接使用 Claude API（1P）還是通過 AWS Bedrock 或 Google Vertex（3P）存取，您現有的商業協議將適用於 Claude Code 使用，除非我們已相互同意另行安排。

合規

醫療保健合規（BAA）

如果客戶與我們簽訂了業務關聯協議（BAA），並想使用 Claude Code，如果客戶已執行 BAA 並已啟用零資料保留（ZDR），BAA 將自動擴展以涵蓋 Claude Code。BAA 將適用於該客戶通過 Claude Code 流動的 API 流量。

安全和信任

信任和安全

您可以在 Anthropic 信任中心和透明度中心找到更多資訊。

安全漏洞報告

Anthropic 通過 HackerOne 管理我們的安全計畫。使用此表單報告漏洞。

零數據保留

> 了解 Claude for Enterprise 上 Claude Code 的零數據保留 (ZDR)，包括範圍、禁用功能以及如何請求啟用。

零數據保留 (ZDR) 在通過 Claude for Enterprise 使用 Claude Code 時可用。啟用 ZDR 後，Claude Code 會話期間生成的提示和模型回應會實時處理，並在返回回應後不會由 Anthropic 存儲，除非需要遵守法律或防止濫用。

Claude for Enterprise 上的 ZDR 使企業客戶能夠使用 Claude Code 並實現零數據保留，同時獲得管理功能：

- 每個用戶的成本控制
- 分析儀表板
- 服務器管理的設置
- 審計日誌

Claude for Enterprise 上 Claude Code 的 ZDR 僅適用於 Anthropic 的直接平台。對於在 AWS Bedrock、Google Vertex AI 或 Microsoft Foundry 上的 Claude 部署，請參考這些平台的數據保留政策。

ZDR 範圍

ZDR 涵蓋 Claude for Enterprise 上的 Claude Code 推理。

> **Warning:**
>
> ZDR 在每個組織的基礎上啟用。每個新組織都需要由您的 Anthropic 帳戶團隊單獨啟用 ZDR。ZDR 不會自動應用於在同一帳戶下創建的新組織。請聯繫您的帳戶團隊為任何新組織啟用 ZDR。

ZDR 涵蓋的內容

ZDR 涵蓋通過 Claude for Enterprise 上的 Claude Code 進行的模型推理調用。當您在終端中使用 Claude Code 時，您發送的提示和 Claude 生成的回應不會由 Anthropic 保留。無論使用哪個 Claude 模型，這都適用。

ZDR 不涵蓋的內容

即使對於啟用了 ZDR 的組織，ZDR 也不適用於以下內容。這些功能遵循標準數據保留政策：

功能	詳情
claude.ai 上的聊天	通過 Claude for Enterprise 網絡界面的聊天對話不受 ZDR 保護。
Cowork	Cowork 會話不受 ZDR 保護。
Claude Code 分析	不存儲提示或模型回應，但收集生產力元數據，例如帳戶電子郵件和使用統計信息。對於 ZDR 組織，貢獻指標不可用；分析儀表板僅顯示使用指標。
用戶和座位管理	管理數據（例如帳戶電子郵件和座位分配）根據標準政策保留。
第三方集成	由第三方工具、MCP servers 或其他外部集成處理的數據不受 ZDR 保護。請獨立審查這些服務的數據處理實踐。

ZDR 下禁用的功能

當為 Claude for Enterprise 上的 Claude Code 組織啟用 ZDR 時，某些需要存儲提示或完成的功能會在後端級別自動禁用：

功能	原因
Web 上的 Claude Code	需要服務器端存儲對話歷史記錄。
Desktop 應用程序的遠程會話	需要包含提示和完成的持久會話數據。
反饋提交（`/feedback`）	提交反饋會將對話數據發送給 Anthropic。

這些功能在後端被阻止，無論客戶端顯示如何。如果您在啟動期間在 Claude Code 終端中看到禁用的功能，嘗試使用它會返回一個錯誤，指示組織的政策不允許該操作。

如果未來的功能需要存儲提示或完成，它們也可能被禁用。

政策違規的數據保留

即使啟用了 ZDR，Anthropic 也可能在法律要求或解決使用政策違規時保留數據。如果會話因政策違規而被標記，Anthropic 可能會保留相關的輸入和輸出長達 2 年，與 Anthropic 的標準 ZDR 政策一致。

請求 ZDR

要為 Claude for Enterprise 上的 Claude Code 請求 ZDR，請聯繫您的 Anthropic 帳戶團隊。您的帳戶團隊將在內部提交請求，Anthropic 將在確認符合條件後在您的組織上審查並啟用 ZDR。所有啟用操作都會被審計記錄。

如果您目前通過按使用量付費的 API 密鑰使用 Claude Code 的 ZDR，您可以過渡到 Claude for Enterprise 以獲得管理功能的訪問權限，同時為 Claude Code 保持 ZDR。請聯繫您的帳戶團隊以協調遷移。

Part 10: Enterprise & Monitoring

使用分析追蹤團隊使用情況

> 在分析儀表板中查看 Claude Code 使用指標、追蹤採用情況並衡量工程速度。

Claude Code 提供分析儀表板，幫助組織了解開發人員使用模式、追蹤貢獻指標，並衡量 Claude Code 對工程速度的影響。根據您的計劃訪問儀表板：

計劃	儀表板 URL	包含內容	了解更多
Claude for Teams / Enterprise	claude.ai/analytics/claude-code	使用指標、與 GitHub 整合的貢獻指標、排行榜、數據匯出	詳情
API (Claude Console)	platform.claude.com/claude-code	使用指標、支出追蹤、團隊洞察	詳情

訪問 Teams 和 Enterprise 的分析

導航至 claude.ai/analytics/claude-code。管理員和所有者可以查看儀表板。

Teams 和 Enterprise 儀表板包括：

- **使用指標**：已接受的代碼行數、建議接受率、每日活躍用戶和會話
- **貢獻指標**：使用 Claude Code 協助發布的 PR 和代碼行數，具有 GitHub 整合
- **排行榜**：按 Claude Code 使用情況排名的頂級貢獻者
- **數據匯出**：將貢獻數據下載為 CSV 格式以進行自訂報告

啟用貢獻指標

> **Note:**
>
> 貢獻指標處於公開測試版，可在 Claude for Teams 和 Claude for Enterprise 計劃上使用。這些指標僅涵蓋您 claude.ai 組織內的用戶。通過 Claude Console API 或第三方整合的使用不包括在內。

使用和採用數據適用於所有 Claude for Teams 和 Claude for Enterprise 帳戶。貢獻指標需要額外設置以連接您的 GitHub 組織。

您需要所有者角色來配置分析設置。GitHub 管理員必須安裝 GitHub 應用程序。

> **Warning:**
> 啟用了 Zero Data Retention 的組織無法使用貢獻指標。分析儀表板將僅顯示使用指標。

Step 1: 安裝 GitHub 應用程序

GitHub 管理員在您組織的 GitHub 帳戶上安裝 Claude GitHub 應用程序，位址為 github.com/apps/claude。

Step 2: 啟用 Claude Code 分析

Claude 所有者導航至 claude.ai/admin-settings/claude-code 並啟用 Claude Code 分析功能。

Step 3: 啟用 GitHub 分析

在同一頁面上，啟用’GitHub 分析’切換。

Step 4: 使用 GitHub 進行身份驗證

完成 GitHub 身份驗證流程並選擇要包含在分析中的 GitHub 組織。

啟用後，數據通常在 24 小時內出現，並進行每日更新。如果沒有數據出現，您可能會看到以下消息之一：

- **「需要 GitHub 應用程序」**：安裝 GitHub 應用程序以查看貢獻指標
- **「數據處理進行中」**：幾天後重新檢查，如果數據未出現，請確認 GitHub 應用程序已安裝

貢獻指標支持 GitHub Cloud 和 GitHub Enterprise Server。

查看摘要指標

> **Note:**
> 這些指標故意保守，代表對 Claude Code 實際影響的低估。只有高度確信 Claude Code 參與的代碼行和 PR 才會被計算。

儀表板在頂部顯示這些摘要指標：

- **帶有 CC 的 PR**：包含至少一行使用 Claude Code 編寫的代碼的已合併拉取請求的總計數
- **帶有 CC 的代碼行**：所有已合併 PR 中使用 Claude Code 協助編寫的代碼行總數。僅計算「有效行」：規範化後超過 3 個字符的行，不包括空行和僅包含括號或瑣碎標點符號的行。
- **帶有 Claude Code 的 PR (%)**：包含 Claude Code 協助代碼的所有已合併 PR 的百分比
- **建議接受率**：用戶接受 Claude Code 代碼編輯建議的次數百分比，包括 Edit、Write 和 NotebookEdit 工具使用
- **已接受的代碼行**：Claude Code 編寫且用戶在其會話中已接受的代碼行總數。這不包括被拒絕的建議，也不追蹤後續刪除。

探索圖表

儀表板包括多個圖表以可視化一段時間內的趨勢。

追蹤採用

採用圖表顯示每日使用趨勢：

- **用戶**：每日活躍用戶
- **會話**：每天活躍 Claude Code 會話的數量

衡量每個用戶的 PR

此圖表顯示一段時間內的個人開發人員活動：

- **每個用戶的 PR**：每天合併的 PR 總數除以每日活躍用戶
- **用戶**：每日活躍用戶

使用此功能了解隨著 Claude Code 採用增加，個人生產力如何變化。

查看拉取請求細分

拉取請求圖表顯示已合併 PR 的每日細分：

- **帶有 CC 的 PR**：包含 Claude Code 協助代碼的拉取請求
- **不帶 CC 的 PR**：不包含 Claude Code 協助代碼的拉取請求

切換至**代碼行**視圖以按代碼行而不是 PR 計數查看相同的細分。

查找頂級貢獻者

排行榜顯示按貢獻量排名的前 10 名用戶。在以下之間切換：

- **拉取請求**：顯示每個用戶的帶有 Claude Code 的 PR 與所有 PR
- **代碼行**：顯示每個用戶的帶有 Claude Code 的行與所有行

點擊**匯出所有用戶**以將所有用戶的完整貢獻數據下載為 CSV 文件。匯出包括所有用戶，而不僅僅是顯示的前 10 名。

PR 歸因

啟用貢獻指標後，Claude Code 會分析已合併的拉取請求，以確定哪些代碼是使用 Claude Code 協助編寫的。這是通過將 Claude Code 會話活動與每個 PR 中的代碼進行匹配來完成的。

標記標準

如果 PR 包含在 Claude Code 會話期間編寫的至少一行代碼，則將其標記為「帶有 Claude Code」。系統使用保守匹配：只有高度確信 Claude Code 參與的代碼才被計為協助。

歸因過程

當拉取請求被合併時：

1. 從 PR diff 中提取添加的行
2. 識別在時間窗口內編輯匹配文件的 Claude Code 會話
3. 使用多種策略將 PR 行與 Claude Code 輸出進行匹配
4. 計算 AI 協助行和總行的指標

在比較之前，行被規範化：空格被修剪、多個空格被折疊、引號被標準化、文本被轉換為小寫。

包含 Claude Code 協助行的已合併拉取請求在 GitHub 中被標記為 `claude-code-assisted`。

時間窗口

PR 合併日期前 21 天至後 2 天的會話被考慮用於歸因匹配。

排除的文件

某些文件會自動從分析中排除，因為它們是自動生成的：

- 鎖定文件：package-lock.json、yarn.lock、Cargo.lock 及類似文件
- 生成的代碼：Protobuf 輸出、構建工件、縮小的文件
- 構建目錄：dist/、build/、node_modules/、target/

- 測試夾具：快照、盒帶、模擬數據
- 超過 1,000 個字符的行，可能是縮小或生成的

歸因說明

在解釋歸因數據時，請記住這些額外詳情：

- 由開發人員大幅重寫的代碼，差異超過 20%，不歸因於 Claude Code
- 21 天窗口外的會話不被考慮
- 該算法在執行歸因時不考慮 PR 源或目標分支

從分析中獲得最大收益

使用貢獻指標來展示 ROI、識別採用模式並找到可以幫助他人入門的團隊成員。

監控採用

追蹤採用圖表和用戶計數以識別：

- 可以分享最佳實踐的活躍用戶
- 整個組織的整體採用趨勢
- 可能表示摩擦或問題的使用下降

衡量 ROI

貢獻指標幫助回答「這個工具值得投資嗎？」，使用來自您自己代碼庫的數據：

- 隨著採用增加，追蹤一段時間內每個用戶的 PR 變化
- 比較使用和不使用 Claude Code 發布的 PR 和代碼行
- 與 DORA 指標、衝刺速度或其他工程 KPI 一起使用，以了解採用 Claude Code 帶來的變化

識別超級用戶

排行榜幫助您找到具有高 Claude Code 採用率的團隊成員，他們可以：

- 與團隊分享提示技術和工作流程
- 提供有關什麼運作良好的反饋
- 幫助新用戶入門

以編程方式訪問數據

要通過 GitHub 查詢此數據，請搜索標記為 `claude-code-assisted` 的 PR。

訪問 API 客戶的分析

使用 Claude Console 的 API 客戶可以在 platform.claude.com/claude-code 訪問分析。您需要 UsageView 權限才能訪問儀表板，該權限授予開發人員、計費、管理員、所有者和主要所有者角色。

> **Note:**
>
> GitHub 整合的貢獻指標目前不適用於 API 客戶。Console 儀表板僅顯示使用和支出指標。

Console 儀表板顯示：

- **已接受的代碼行**：Claude Code 編寫且用戶在其會話中已接受的代碼行總數。這不包括被拒絕的建議，也不追蹤後續刪除。
- **建議接受率**：用戶接受代碼編輯工具使用的次數百分比，包括 Edit、Write 和 NotebookEdit 工具。
- **活動**：圖表上顯示的每日活躍用戶和會話。
- **支出**：每日 API 成本（以美元計）以及用戶計數。

查看團隊洞察

團隊洞察表顯示每個用戶的指標：

- **成員**：所有已向 Claude Code 進行身份驗證的用戶。API 密鑰用戶按密鑰標識符顯示，OAuth 用戶按電子郵件地址顯示。
- **本月支出**：每個用戶當前月份的每用戶 API 成本總計。
- **本月代碼行**：每個用戶當前月份已接受代碼行的每用戶總計。

> **Note:**
>
> Console 儀表板中的支出數字是用於分析目的的估計值。有關實際成本，請參閱您的計費頁面。

相關資源

- 使用 OpenTelemetry 進行監控：將實時指標和事件匯出到您的可觀測性堆棧
- 有效管理成本：設置支出限制並優化令牌使用
- 權限：配置角色和權限

監控

> 了解如何為 Claude Code 啟用和配置 OpenTelemetry。

Claude Code 支援 OpenTelemetry (OTel) 指標和事件，用於監控和可觀測性。

所有指標都是透過 OpenTelemetry 的標準指標協議匯出的時間序列資料，事件則透過 OpenTelemetry 的日誌/事件協議匯出。使用者有責任確保其指標和日誌後端已正確配置，且聚合粒度符合其監控需求。

快速開始

使用環境變數配置 OpenTelemetry：

```
## 1. 啟用遙測
export CLAUDE_CODE_ENABLE_TELEMETRY=1

## 2. 選擇匯出器（兩者都是可選的 - 僅配置您需要的）
export OTEL_METRICS_EXPORTER=otlp       # 選項：otlp、prometheus、console
export OTEL_LOGS_EXPORTER=otlp          # 選項：otlp、console

## 3. 配置 OTLP 端點（用於 OTLP 匯出器）
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## 4. 設定身份驗證（如果需要）
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer your-token"

## 5. 用於除錯：減少匯出間隔
export OTEL_METRIC_EXPORT_INTERVAL=10000  # 10 秒（預設：60000ms）
export OTEL_LOGS_EXPORT_INTERVAL=5000     # 5 秒（預設：5000ms）

## 6. 執行 Claude Code
claude
```

> **Note:**

> 預設匯出間隔為指標 60 秒和日誌 5 秒。在設定期間，您可能希望使用較短的間隔用於除錯目的。記得在生產環境中重設這些值。

如需完整配置選項，請參閱 OpenTelemetry 規範。

管理員配置

管理員可以透過受管設定檔為所有使用者配置 OpenTelemetry 設定。這允許在整個組織中集中控制遙測設定。請參閱設定優先順序以了解有關如何應用設定的更多資訊。

受管設定配置範例：

```
{
  "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:**
>
> 受管設定可以透過 MDM（行動裝置管理）或其他裝置管理解決方案進行分發。在受管設定檔中定義的環境變數具有高優先順序，使用者無法覆蓋。

配置詳情

常見配置變數

環境變數	描述	範例值
`CLAUDE_CODE_ENABLE_TELEMETRY`	啟用遙測收集（必需）	`1`
`OTEL_METRICS_EXPORTER`	指標匯出器類型（逗號分隔）	`console`、`otlp`、`prometheus`

環境變數	描述	範例值
`OTEL_LOGS_EXPORTER`	日誌/事件匯出器類型（逗號分隔）	`console`、`otlp`
`OTEL_EXPORTER_OTLP_PROTOCOL`	OTLP 匯出器的協議（所有訊號）	`grpc`、`http/json`、`http/protobuf`
`OTEL_EXPORTER_OTLP_ENDPOINT`	OTLP 收集器端點（所有訊號）	`http://localhost:4317`
`OTEL_EXPORTER_OTLP_METRICS_PROTOCOL`	指標協議（覆蓋一般設定）	`grpc`、`http/json`、`http/protobuf`
`OTEL_EXPORTER_OTLP_METRICS_ENDPOINT`	OTLP 指標端點（覆蓋一般設定）	`http://localhost:4318/v1/metrics`
`OTEL_EXPORTER_OTLP_LOGS_PROTOCOL`	日誌協議（覆蓋一般設定）	`grpc`、`http/json`、`http/protobuf`
`OTEL_EXPORTER_OTLP_LOGS_ENDPOINT`	OTLP 日誌端點（覆蓋一般設定）	`http://localhost:4318/v1/logs`
`OTEL_EXPORTER_OTLP_HEADERS`	OTLP 的身份驗證標頭	`Authorization=Bearer token`
`OTEL_EXPORTER_OTLP_METRICS_CLIENT_KEY`	mTLS 身份驗證的用戶端金鑰	用戶端金鑰檔案的路徑
`OTEL_EXPORTER_OTLP_METRICS_CLIENT_CERTIFICATE`	mTLS 身份驗證的用戶端憑證	用戶端憑證檔案的路徑
`OTEL_METRIC_EXPORT_INTERVAL`	匯出間隔（毫秒）（預設：60000）	`5000`、`60000`
`OTEL_LOGS_EXPORT_INTERVAL`	日誌匯出間隔（毫秒）（預設：5000）	`1000`、`10000`
`OTEL_LOG_USER_PROMPTS`	啟用使用者提示內容的日誌記錄（預設：停用）	`1` 以啟用

環境變數	描述	範例值
`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS`	重新整理動態標頭的間隔（預設：1740000ms / 29 分鐘）	`900000`

指標基數控制

以下環境變數控制指標中包含哪些屬性以管理基數：

環境變數	描述	預設值	停用範例
`OTEL_METRICS_INCLUDE_SESSION_ID`	在指標中包含 session.id 屬性	`true`	`false`
`OTEL_METRICS_INCLUDE_VERSION`	在指標中包含 app.version 屬性	`false`	`true`
`OTEL_METRICS_INCLUDE_ACCOUNT_UUID`	在指標中包含 user.account_uuid 屬性	`true`	`false`

這些變數有助於控制指標的基數，這會影響指標後端中的儲存需求和查詢效能。較低的基數通常意味著更好的效能和更低的儲存成本，但分析的資料粒度較低。

動態標頭

對於需要動態身份驗證的企業環境，您可以配置指令碼來動態產生標頭：

設定配置

新增至您的 `.claude/settings.json`：

```
{
  "otelHeadersHelper": "/bin/generate_opentelemetry_headers.sh"
}
```

指令碼需求

指令碼必須輸出有效的 JSON，其中包含代表 HTTP 標頭的字串鍵值對：

```
#!/bin/bash
## 範例：多個標頭
echo "{\"Authorization\": \"Bearer $(get-token.sh)\", \"X-API-Key\": \"$(get-api-key.sh)\"}"
```

重新整理行為

標頭協助程式指令碼在啟動時執行，之後定期執行以支援權杖重新整理。預設情況下，指令碼每 29 分鐘執行一次。使用 `CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS` 環境變數自訂間隔。

多團隊組織支援

具有多個團隊或部門的組織可以使用 `OTEL_RESOURCE_ATTRIBUTES` 環境變數新增自訂屬性以區分不同的群組：

```
## 新增自訂屬性以進行團隊識別
export OTEL_RESOURCE_ATTRIBUTES="department=engineering,team.id=platform,cost_center=eng-123"
```

這些自訂屬性將包含在所有指標和事件中，允許您：

- 按團隊或部門篩選指標
- 追蹤每個成本中心的成本
- 建立團隊特定的儀表板
- 為特定團隊設定警報

> **Warning:**
>
> **OTEL_RESOURCE_ATTRIBUTES 的重要格式要求：**
>
> `OTEL_RESOURCE_ATTRIBUTES` 環境變數遵循 W3C Baggage 規範，具有嚴格的格式要求：
>
> - **不允許空格**：值不能包含空格。例如，`user.organizationName=My Company` 無效
> - **格式**：必須是逗號分隔的鍵=值對：`key1=value1,key2=value2`
> - **允許的字元**：僅限 US-ASCII 字元，不包括控制字元、空格、雙引號、逗號、分號和反斜線
> - **特殊字元**：允許範圍外的字元必須進行百分比編碼
>
> **範例：**

```
## ❌無效 - 包含空格
export OTEL_RESOURCE_ATTRIBUTES="org.name=John's Organization"

## ✅有效 - 改用底線或駝峰式大小寫
export OTEL_RESOURCE_ATTRIBUTES="org.name=Johns_Organization"
export OTEL_RESOURCE_ATTRIBUTES="org.name=JohnsOrganization"

## ✅有效 - 如果需要，對特殊字元進行百分比編碼
export OTEL_RESOURCE_ATTRIBUTES="org.name=John%27s%20Organization"
```

注意：將值用引號括起來不會逃逸空格。例如，`org.name="My Company"` 會產生字面值 `"My Company"`（包括引號），而不是 `My Company`。

配置範例

```
## 控制台除錯（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

## 多個匯出器
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console,otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=http/json

## 指標和日誌的不同端點/後端
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

## 僅指標（無事件/日誌）
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

## 僅事件/日誌（無指標）
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
```

可用的指標和事件

標準屬性

所有指標和事件共享這些標準屬性：

屬性	描述	控制者
`session.id`	唯一的工作階段識別碼	`OTEL_METRICS_INCLUDE_SESSION_ID`（預設：true）
`app.version`	目前的 Claude Code 版本	`OTEL_METRICS_INCLUDE_VERSION`（預設：false）
`organization.id`	組織 UUID（已驗證時）	可用時始終包含
`user.account_uuid`	帳戶 UUID（已驗證時）	`OTEL_METRICS_INCLUDE_ACCOUNT_UUID`（預設：true）
`terminal.type`	終端機類型（例如，`iTerm.app`、`vscode`、`cursor`、`tmux`）	偵測到時始終包含

指標

Claude Code 匯出以下指標：

指標名稱	描述	單位
`claude_code.session.count`	啟動的 CLI 工作階段計數	count
`claude_code.lines_of_code.count`	修改的程式碼行數計數	count
`claude_code.pull_request.count`	建立的提取請求數	count
`claude_code.commit.count`	建立的 git 提交數	count

指標名稱	描述	單位
`claude_code.cost.usage`	Claude Code 工作階段的成本	USD
`claude_code.token.usage`	使用的權杖數	tokens
`claude_code.code_edit_tool.decision`	程式碼編輯工具權限決定的計數	count
`claude_code.active_time.total`	總活躍時間（秒）	s

指標詳情

工作階段計數器

在每個工作階段開始時遞增。

屬性：

- 所有標準屬性

程式碼行計數器

在新增或移除程式碼時遞增。

屬性：

- 所有標準屬性
- `type`：（`"added"`、`"removed"`）

提取請求計數器

透過 Claude Code 建立提取請求時遞增。

屬性：

- 所有標準屬性

提交計數器

透過 Claude Code 建立 git 提交時遞增。

屬性：

- 所有標準屬性

成本計數器

在每個 API 請求後遞增。

屬性：

- 所有標準屬性
- `model`：模型識別碼（例如，“claude-sonnet-4-5-20250929”）

權杖計數器

在每個 API 請求後遞增。

屬性：

- 所有標準屬性
- `type`：（`"input"`、`"output"`、`"cacheRead"`、`"cacheCreation"`）
- `model`：模型識別碼（例如，“claude-sonnet-4-5-20250929”）

程式碼編輯工具決定計數器

當使用者接受或拒絕 Edit、Write 或 NotebookEdit 工具使用時遞增。

屬性：

- 所有標準屬性
- `tool`：工具名稱（`"Edit"`、`"Write"`、`"NotebookEdit"`）
- `decision`：使用者決定（`"accept"`、`"reject"`）
- `language`：編輯檔案的程式設計語言（例如，`"TypeScript"`、`"Python"`、`"JavaScript"`、`"Markdown"`）。對於無法識別的副檔名，傳回 `"unknown"`。

活躍時間計數器

追蹤實際花費在主動使用 Claude Code 上的時間（非閒置時間）。此指標在使用者互動期間遞增，例如輸入提示或接收回應。

屬性：

- 所有標準屬性

事件

Claude Code 透過 OpenTelemetry 日誌/事件匯出以下事件（當配置 `OTEL_LOGS_EXPORTER` 時）：

使用者提示事件

當使用者提交提示時記錄。

事件名稱：`claude_code.user_prompt`

屬性：

- 所有標準屬性
- `event.name`：`"user_prompt"`
- `event.timestamp`：ISO 8601 時間戳
- `prompt_length`：提示的長度
- `prompt`：提示內容（預設為編輯，使用 `OTEL_LOG_USER_PROMPTS=1` 啟用）

工具結果事件

當工具完成執行時記錄。

事件名稱：`claude_code.tool_result`

屬性：

- 所有標準屬性
- `event.name`：`"tool_result"`
- `event.timestamp`：ISO 8601 時間戳
- `tool_name`：工具的名稱
- `success`：`"true"` 或 `"false"`
- `duration_ms`：執行時間（毫秒）
- `error`：錯誤訊息（如果失敗）
- `decision`：`"accept"` 或 `"reject"`
- `source`：決定來源 - `"config"`、`"user_permanent"`、`"user_temporary"`、`"user_abort"` 或 `"user_reject"`
- `tool_parameters`：包含工具特定參數的 JSON 字串（如果可用）
  - 對於 Bash 工具：包括 `bash_command`、`full_command`、`timeout`、`description`、`sandbox`

API 請求事件

為每個 API 請求記錄到 Claude。

事件名稱：`claude_code.api_request`

屬性：

- 所有標準屬性
- `event.name`：`"api_request"`
- `event.timestamp`：ISO 8601 時間戳
- `model`：使用的模型（例如，“claude-sonnet-4-5-20250929”）
- `cost_usd`：估計成本（美元）
- `duration_ms`：請求持續時間（毫秒）
- `input_tokens`：輸入權杖數
- `output_tokens`：輸出權杖數
- `cache_read_tokens`：從快取讀取的權杖數
- `cache_creation_tokens`：用於快取建立的權杖數

API 錯誤事件

當 API 請求到 Claude 失敗時記錄。

事件名稱：`claude_code.api_error`

屬性：

- 所有標準屬性
- `event.name`：`"api_error"`
- `event.timestamp`：ISO 8601 時間戳
- `model`：使用的模型（例如，“claude-sonnet-4-5-20250929”）
- `error`：錯誤訊息
- `status_code`：HTTP 狀態碼（如果適用）
- `duration_ms`：請求持續時間（毫秒）
- `attempt`：嘗試次數（用於重試的請求）

工具決定事件

當做出工具權限決定（接受/拒絕）時記錄。

事件名稱：`claude_code.tool_decision`

屬性：

- 所有標準屬性
- `event.name`：`"tool_decision"`
- `event.timestamp`：ISO 8601 時間戳

- `tool_name`：工具的名稱（例如，“Read”、“Edit”、“Write”、“NotebookEdit”）
- `decision`：`"accept"` 或 `"reject"`
- `source`：決定來源 - `"config"`、`"user_permanent"`、`"user_temporary"`、`"user_abort"` 或 `"user_reject"`

解釋指標和事件資料

Claude Code 匯出的指標提供了對使用模式和生產力的寶貴見解。以下是您可以建立的一些常見視覺化和分析：

使用情況監控

指標	分析機會
`claude_code.token.usage`	按 `type`（輸入/輸出）、使用者、團隊或模型進行細分
`claude_code.session.count`	追蹤一段時間內的採用和參與度
`claude_code.lines_of_code.count`	透過追蹤程式碼新增/移除來衡量生產力
`claude_code.commit.count` & `claude_code.pull_request.count`	了解對開發工作流程的影響

成本監控

`claude_code.cost.usage` 指標有助於：

- 追蹤跨團隊或個人的使用趨勢
- 識別高使用量工作階段以進行最佳化

> **Note:**
>
> 成本指標是近似值。如需官方帳單資料，請參閱您的 API 提供者（Claude Console、AWS Bedrock 或 Google Cloud Vertex）。

警報和分段

要考慮的常見警報：

- 成本尖峰
- 異常的權杖消耗
- 來自特定使用者的高工作階段量

所有指標都可以按 `user.account_uuid`、`organization.id`、`session.id`、`model` 和 `app.version` 進行分段。

事件分析

事件資料提供了對 Claude Code 互動的詳細見解：

工具使用模式：分析工具結果事件以識別：

- 最常使用的工具
- 工具成功率
- 平均工具執行時間
- 按工具類型的錯誤模式

效能監控：追蹤 API 請求持續時間和工具執行時間以識別效能瓶頸。

後端考量

您選擇的指標和日誌後端決定了您可以執行的分析類型：

對於指標

- **時間序列資料庫（例如，Prometheus）**：速率計算、聚合指標
- **欄式存儲（例如，ClickHouse）**：複雜查詢、唯一使用者分析
- **功能完整的可觀測性平台（例如，Honeycomb、Datadog）**：進階查詢、視覺化、警報

對於事件/日誌

- **日誌聚合系統（例如，Elasticsearch、Loki）**：全文搜尋、日誌分析
- **欄式存儲（例如，ClickHouse）**：結構化事件分析
- **功能完整的可觀測性平台（例如，Honeycomb、Datadog）**：指標和事件之間的關聯

對於需要日活躍使用者/週活躍使用者/月活躍使用者 (DAU/WAU/MAU) 指標的組織，請考慮支援高效唯一值查詢的後端。

服務資訊

所有指標和事件都使用以下資源屬性匯出：

- `service.name`：`claude-code`
- `service.version`：目前的 Claude Code 版本
- `os.type`：作業系統類型（例如，`linux`、`darwin`、`windows`）
- `os.version`：作業系統版本字串
- `host.arch`：主機架構（例如，`amd64`、`arm64`）
- `wsl.version`：WSL 版本號（僅在 Windows Subsystem for Linux 上執行時出現）
- 計量器名稱：`com.anthropic.claude_code`

ROI 測量資源

如需有關測量 Claude Code 投資回報率的綜合指南，包括遙測設定、成本分析、生產力指標和自動化報告，請參閱 Claude Code ROI 測量指南。此儲存庫提供現成可用的 Docker Compose 配置、Prometheus 和 OpenTelemetry 設定，以及用於產生與 Linear 等工具整合的生產力報告的範本。

安全性/隱私考量

- 遙測是選擇加入的，需要明確配置
- 敏感資訊（如 API 金鑰或檔案內容）永遠不會包含在指標或事件中
- 使用者提示內容預設為編輯 - 僅記錄提示長度。若要啟用使用者提示日誌記錄，請設定 `OTEL_LOG_USER_PROMPTS=1`

在 Amazon Bedrock 上監控 Claude Code

如需 Amazon Bedrock 上 Claude Code 使用情況監控的詳細指南，請參閱 Claude Code 監控實作 (Bedrock)。

企業部署概述

> 了解 Claude Code 如何與各種第三方服務和基礎設施整合，以滿足企業部署需求。

本頁面提供可用部署選項的概述，並幫助您為組織選擇正確的配置。

提供商比較

功能 Anthropic Amazon Bedrock Google Vertex AI Microsoft Foundry

區域 支援的國家 多個 AWS 區域 多個 GCP 區域 多個 Azure 區域

提示快取 預設啟用 預設啟用 預設啟用 預設啟用

身份驗證 API 金鑰 API 金鑰或 AWS 認證 GCP 認證 API 金鑰或 Microsoft Entra ID

成本追蹤 儀表板 AWS Cost Explorer GCP Billing Azure Cost Management

企業功能 團隊、使用情況監控 IAM 政策、CloudTrail IAM 角色、Cloud Audit Logs RBAC 政策、Azure Monitor

雲端提供商

- Amazon Bedrock 透過 AWS 基礎設施使用 Claude 模型，支援 API 金鑰或基於 IAM 的身份驗證和 AWS 原生監控
- Google Vertex AI 透過 Google Cloud Platform 存取 Claude 模型，具有企業級安全性和合規性
- Microsoft Foundry 透過 Azure 存取 Claude，支援 API 金鑰或 Microsoft Entra ID 身份驗證和 Azure 計費

企業基礎設施

- 企業網路 配置 Claude Code 以與組織的代理伺服器和 SSL/TLS 需求相容
- LLM Gateway 部署集中式模型存取，具有使用情況追蹤、預算編制和稽核日誌

配置概述

Claude Code 支援靈活的配置選項，允許您組合不同的提供商和基礎設施：

> **Note:**

了解以下差異：

- **企業代理**：用於路由流量的 HTTP/HTTPS 代理（透過 `HTTPS_PROXY` 或 `HTTP_PROXY` 設定）
- **LLM Gateway**：處理身份驗證並提供提供商相容端點的服務（透過 `ANTHROPIC_BASE_URL`、`ANTHROPIC_BEDROCK_BASE_URL` 或 `ANTHROPIC_VERTEX_BASE_URL` 設定）

兩種配置可以同時使用。

使用 Bedrock 搭配企業代理

透過企業 HTTP/HTTPS 代理路由 Bedrock 流量：

```
## 啟用 Bedrock
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1

## 配置企業代理
export HTTPS_PROXY='https://proxy.example.com:8080'
```

使用 Bedrock 搭配 LLM Gateway

使用提供 Bedrock 相容端點的閘道服務：

```
## 啟用 Bedrock
export CLAUDE_CODE_USE_BEDROCK=1

## 配置 LLM 閘道
export ANTHROPIC_BEDROCK_BASE_URL='https://your-llm-gateway.com/bedrock'
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1  # 如果閘道處理 AWS 身份驗證
```

使用 Foundry 搭配企業代理

透過企業 HTTP/HTTPS 代理路由 Azure 流量：

```
## 啟用 Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1
export ANTHROPIC_FOUNDRY_RESOURCE=your-resource
export ANTHROPIC_FOUNDRY_API_KEY=your-api-key  # 或省略以使用 Entra ID 身份驗證

## 配置企業代理
export HTTPS_PROXY='https://proxy.example.com:8080'
```

使用 Foundry 搭配 LLM Gateway

使用提供 Azure 相容端點的閘道服務：

```
## 啟用 Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1

## 配置 LLM 閘道
export ANTHROPIC_FOUNDRY_BASE_URL='https://your-llm-gateway.com'
export CLAUDE_CODE_SKIP_FOUNDRY_AUTH=1  # 如果閘道處理 Azure 身份驗證
```

使用 Vertex AI 搭配企業代理

透過企業 HTTP/HTTPS 代理路由 Vertex AI 流量：

```
## 啟用 Vertex
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
export ANTHROPIC_VERTEX_PROJECT_ID=your-project-id

## 配置企業代理
export HTTPS_PROXY='https://proxy.example.com:8080'
```

使用 Vertex AI 搭配 LLM Gateway

將 Google Vertex AI 模型與 LLM 閘道結合以進行集中式管理：

```
## 啟用 Vertex
export CLAUDE_CODE_USE_VERTEX=1

## 配置 LLM 閘道
export ANTHROPIC_VERTEX_BASE_URL='https://your-llm-gateway.com/vertex'
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1  # 如果閘道處理 GCP 身份驗證
```

身份驗證配置

Claude Code 在需要時使用 `ANTHROPIC_AUTH_TOKEN` 作為 `Authorization` 標頭。`SKIP_AUTH` 標誌（`CLAUDE_CODE_SKIP_BEDROCK_AUTH`、`CLAUDE_CODE_SKIP_VERTEX_AUTH`）用於 LLM 閘道場景，其中閘道處理提供商身份驗證。

選擇正確的部署配置

選擇部署方法時，請考慮以下因素：

直接提供商存取

最適合以下組織：

- 希望最簡單的設定
- 已有 AWS 或 GCP 基礎設施
- 需要提供商原生監控和合規性

企業代理

最適合以下組織：

- 有現有的企業代理需求
- 需要流量監控和合規性
- 必須透過特定網路路徑路由所有流量

LLM Gateway

最適合以下組織：

- 需要跨團隊的使用情況追蹤
- 希望在模型之間動態切換
- 需要自訂速率限制或預算
- 需要集中式身份驗證管理

除錯

除錯部署時：

- 使用 `claude /status` 斜線命令。此命令提供對任何應用的身份驗證、代理和 URL 設定的可觀測性。
- 設定環境變數 `export ANTHROPIC_LOG=debug` 以記錄請求。

組織的最佳實踐

1. 投資文件和記憶

我們強烈建議投資文件，以便 Claude Code 了解您的程式碼庫。組織可以在多個級別部署 CLAUDE.md 檔案：

- **組織範圍**：部署到系統目錄，如 `/Library/Application Support/ClaudeCode/CLAUDE.md`（macOS），用於公司範圍的標準
- **存放庫級別**：在存放庫根目錄中建立 `CLAUDE.md` 檔案，包含專案架構、建置命令和貢獻指南。將這些簽入原始碼控制，以便所有使用者受益

  了解更多。

2. 簡化部署

如果您有自訂開發環境，我們發現建立「一鍵」安裝 Claude Code 的方式是在組織中推動採用的關鍵。

3. 從引導式使用開始

鼓勵新使用者嘗試 Claude Code 進行程式碼庫問答，或在較小的錯誤修復或功能請求上使用。要求 Claude Code 制定計畫。檢查 Claude 的建議，如果偏離軌道，請提供反饋。隨著時間推移，當使用者更好地理解這種新範例時，他們將更有效地讓 Claude Code 更自主地運行。

4. 配置安全政策

安全團隊可以配置受管權限，以確定 Claude Code 允許和不允許執行的操作，這些操作無法被本地配置覆蓋。了解更多。

5. 利用 MCP 進行整合

MCP 是為 Claude Code 提供更多資訊的絕佳方式，例如連接到票證管理系統或錯誤日誌。我們建議一個中央團隊配置 MCP 伺服器，並將 `.mcp.json` 配置簽入程式碼庫，以便所有使用者受益。了解更多。

在 Anthropic，我們信任 Claude Code 為每個 Anthropic 程式碼庫的開發提供動力。我們希望您享受使用 Claude Code 就像我們一樣。

後續步驟

- 設定 Amazon Bedrock 進行 AWS 原生部署
- 配置 Google Vertex AI 進行 GCP 部署
- 設定 Microsoft Foundry 進行 Azure 部署
- 配置企業網路 以滿足網路需求
- 部署 LLM Gateway 進行企業管理
- 設定 以了解配置選項和環境變數

設定伺服器管理的設定 (公開測試版)

> 透過伺服器傳遞的設定在 Claude.ai 上為您的組織集中設定 Claude Code，無需裝置管理基礎設施。

伺服器管理的設定允許管理員透過 Claude.ai 上的網頁介面集中設定 Claude Code。Claude Code 用戶端在使用者使用其組織認證進行身份驗證時會自動接收這些設定。

此方法適用於沒有裝置管理基礎設施的組織，或需要為非受管裝置上的使用者管理設定的組織。

> **Note:**
>
> 伺服器管理的設定處於公開測試版，適用於 Claude for Teams 和 Claude for Enterprise 客戶。功能可能在正式推出前進行演變。

需求

若要使用伺服器管理的設定，您需要：

- Claude for Teams 或 Claude for Enterprise 方案
- Claude for Teams 的 Claude Code 版本 2.1.38 或更新版本，或 Claude for Enterprise 的版本 2.1.30 或更新版本
- 對 `api.anthropic.com` 的網路存取

在伺服器管理和端點管理的設定之間選擇

Claude Code 支援兩種集中設定方法。伺服器管理的設定從 Anthropic 的伺服器傳遞設定。端點管理的設定 透過原生作業系統原則 (macOS 受管偏好設定、Windows 登錄) 或受管設定檔直接部署到裝置。

方法	最適合	安全模型
伺服器管理的設定	沒有 MDM 的組織，或非受管裝置上的使用者	在身份驗證時從 Anthropic 伺服器傳遞的設定
端點管理的設定	具有 MDM 或端點管理的組織	透過 MDM 設定檔、登錄原則或受管設定檔部署到裝置的設定

如果您的裝置已在 MDM 或端點管理解決方案中註冊，端點管理的設定提供更強的安全保證，因為設定檔可以在作業系統層級受到保護，防止使用者修改。

設定伺服器管理的設定

Step 1: 開啟管理員主控台

在 Claude.ai 中，導覽至 **Admin Settings > Claude Code > Managed settings**。

Step 2: 定義您的設定

將您的設定新增為 JSON。支援 `settings.json` 中提供的所有設定，包括 僅限受管的設定，例如 `disableBypassPermissionsMode`。

此範例強制執行權限拒絕清單並防止使用者繞過權限：

```
{
  "permissions": {
    "deny": [
      "Bash(curl *)",
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(./secrets/**)"
    ]
  },
  "disableBypassPermissionsMode": "disable"
}
```

Step 3: 儲存並部署

儲存您的變更。Claude Code 用戶端在下次啟動或每小時輪詢週期時會接收更新的設定。

驗證設定傳遞

若要確認設定正在套用，請要求使用者重新啟動 Claude Code。如果設定包含觸發安全核准對話方塊的設定，使用者會在啟動時看到描述受管設定的提示。您也可以透過讓使用者執行 `/permissions` 來驗證受管權限規則是否有效，以檢視其有效的權限規則。

存取控制

以下角色可以管理伺服器管理的設定：

- **主要擁有者**
- **擁有者**

限制對受信任人員的存取，因為設定變更會套用到組織中的所有使用者。

目前的限制

伺服器管理的設定在測試版期間有以下限制：

- 設定統一套用到組織中的所有使用者。尚不支援每個群組的設定。
- MCP 伺服器設定 無法透過伺服器管理的設定分發。

設定傳遞

設定優先順序

伺服器管理的設定和端點管理的設定都佔據 Claude Code 設定階層中的最高層級。沒有其他設定層級可以覆蓋它們，包括命令列引數。當兩者都存在時，伺服器管理的設定優先，端點管理的設定不會被使用。

擷取和快取行為

Claude Code 在啟動時從 Anthropic 的伺服器擷取設定，並在活動工作階段期間每小時輪詢更新。

首次啟動而無快取設定：

- Claude Code 非同步擷取設定
- 如果擷取失敗，Claude Code 會在沒有受管設定的情況下繼續
- 在設定載入之前有一個簡短的視窗，其中限制尚未強制執行

後續啟動並有快取設定：

- 快取設定在啟動時立即套用
- Claude Code 在背景擷取新鮮設定
- 快取設定透過網路故障持續存在

Claude Code 自動套用設定更新而無需重新啟動，除了進階設定（例如 OpenTelemetry 設定）需要完整重新啟動才能生效。

安全核准對話方塊

某些可能造成安全風險的設定需要明確的使用者核准才能套用：

- **Shell 命令設定**：執行 shell 命令的設定
- **自訂環境變數**：不在已知安全允許清單中的變數
- **Hook 設定**：任何 hook 定義

當這些設定存在時，使用者會看到安全對話方塊，說明正在設定的內容。使用者必須核准才能繼續。如果使用者拒絕設定，Claude Code 會結束。

> **Note:**
>
> 在使用 `-p` 旗標的非互動模式中，Claude Code 會略過安全對話方塊並在沒有使用者核准的情況下套用設定。

平台可用性

伺服器管理的設定需要直接連線到 `api.anthropic.com`，在使用第三方模型提供者時無法使用：

- Amazon Bedrock
- Google Vertex AI
- Microsoft Foundry
- 透過 `ANTHROPIC_BASE_URL` 或 LLM 閘道 的自訂 API 端點

稽核記錄

設定變更的稽核記錄事件可透過合規性 API 或稽核記錄匯出取得。請聯絡您的 Anthropic 帳戶團隊以取得存取權。

稽核事件包括執行的動作類型、執行動作的帳戶和裝置，以及對先前和新值的參考。

安全考量

伺服器管理的設定提供集中式原則強制執行，但它們作為用戶端控制運作。在非受管裝置上，具有管理員或 sudo 存取權的使用者可以修改 Claude Code 二進位檔、檔案系統或網路設定。

情況	行為
使用者編輯快取的設定檔	篡改的檔案在啟動時套用，但正確的設定會在下次伺服器擷取時還原
使用者刪除快取的設定檔	首次啟動行為發生：設定非同步擷取，有一個簡短的未強制執行視窗
API 無法使用	如果可用，快取設定會套用，否則受管設定在下次成功擷取之前不會強制執行
使用者使用不同的組織進行身份驗證	不會為受管組織外的帳戶傳遞設定

情況	行為
使用者設定非預設的 `ANTHROPIC_BASE_URL`	使用第三方 API 提供者時，伺服器管理的設定會被略過

若要偵測執行時期設定變更，請使用 `ConfigChange` hooks 來記錄修改或在未授權的變更生效前阻止它們。

如需更強的強制執行保證，請在已在 MDM 解決方案中註冊的裝置上使用端點管理的設定。

另請參閱

用於管理 Claude Code 設定的相關頁面：

- Settings：完整的設定參考，包括所有可用的設定
- Endpoint-managed settings：由 IT 部門部署到裝置的受管設定
- Authentication：設定使用者對 Claude Code 的存取
- Security：安全保護措施和最佳實踐

Part 11: Cloud Providers

Amazon Bedrock 上的 Claude Code

> 了解如何透過 Amazon Bedrock 設定 Claude Code，包括設定、IAM 設定和故障排除。

先決條件

在使用 Bedrock 設定 Claude Code 之前，請確保您具有：

- 已啟用 Bedrock 存取的 AWS 帳戶
- 在 Bedrock 中存取所需的 Claude 模型（例如 Claude Sonnet 4.6）
- 已安裝並設定 AWS CLI（選用 - 僅在您沒有其他取得認證機制時才需要）
- 適當的 IAM 權限

> **Note:**
>
> 如果您要將 Claude Code 部署給多個使用者，請固定您的模型版本，以防止在 Anthropic 發佈新模型時發生中斷。

設定

1. 提交使用案例詳細資訊

Anthropic 模型的首次使用者必須在叫用模型之前提交使用案例詳細資訊。這是每個帳戶執行一次的操作。

1. 確保您具有正確的 IAM 權限（請參閱下面的更多資訊）
2. 導覽至 Amazon Bedrock 主控台
3. 選取**聊天/文字遊樂場**
4. 選擇任何 Anthropic 模型，系統將提示您填寫使用案例表單

2. 設定 AWS 認證

Claude Code 使用預設的 AWS SDK 認證鏈。使用以下其中一種方法設定您的認證：

選項 A：AWS CLI 設定

```
aws configure
```

選項 B：環境變數（存取金鑰）

```
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
```

選項 C：環境變數（SSO 設定檔）

```
aws sso login --profile=<your-profile-name>

export AWS_PROFILE=your-profile-name
```

選項 D：AWS 管理主控台認證

```
aws login
```

深入了解 `aws login`。

選項 E：Bedrock API 金鑰

```
export AWS_BEARER_TOKEN_BEDROCK=your-bedrock-api-key
```

Bedrock API 金鑰提供了一種更簡單的驗證方法，無需完整的 AWS 認證。深入了解 Bedrock API 金鑰。

進階認證設定

Claude Code 支援 AWS SSO 和公司身分提供者的自動認證重新整理。將這些設定新增至您的 Claude Code 設定檔（請參閱設定以了解檔案位置）。

當 Claude Code 偵測到您的 AWS 認證已過期（基於本機時間戳記或當 Bedrock 傳回認證錯誤時），它將自動執行您設定的 `awsAuthRefresh` 和/或 `awsCredentialExport` 命令以取得新認證，然後重試請求。

範例設定

```
{
  "awsAuthRefresh": "aws sso login --profile myprofile",
  "env": {
    "AWS_PROFILE": "myprofile"
  }
}
```

設定說明

`awsAuthRefresh`：用於修改 `.aws` 目錄的命令，例如更新認證、SSO 快取或設定檔。命令的輸出會顯示給使用者，但不支援互動式輸入。這適用於瀏覽器型 SSO 流程，其中 CLI 顯示 URL 或代碼，您在瀏覽器中完成驗證。

`awsCredentialExport`：僅在您無法修改 `.aws` 且必須直接傳回認證時使用。輸出會被無聲地擷取，不會顯示給使用者。命令必須以此格式輸出 JSON：

```
{
  "Credentials": {
    "AccessKeyId": "value",
    "SecretAccessKey": "value",
    "SessionToken": "value"
  }
}
```

3. 設定 Claude Code

設定下列環境變數以啟用 Bedrock：

```
## 啟用 Bedrock 整合
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1  # 或您偏好的區域

## 選用：覆寫小型/快速模型 (Haiku) 的區域
export ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION=us-west-2
```

為 Claude Code 啟用 Bedrock 時，請記住以下事項：

- `AWS_REGION` 是必需的環境變數。Claude Code 不會從 `.aws` 設定檔讀取此設定。

- 使用 Bedrock 時，`/login` 和 `/logout` 命令會被停用，因為驗證是透過 AWS 認證處理的。
- 您可以使用設定檔來設定環境變數，例如 `AWS_PROFILE`，您不想將其洩露給其他程序。請參閱設定以取得更多資訊。

4. 固定模型版本

> **Warning:**
>
> 為每個部署固定特定的模型版本。如果您使用模型別名（`sonnet`、`opus`、`haiku`）而不固定版本，Claude Code 可能會嘗試使用您的 Bedrock 帳戶中不可用的較新模型版本，在 Anthropic 發佈更新時破壞現有使用者。

將這些環境變數設定為特定的 Bedrock 模型 ID：

```
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'
```

這些變數使用跨區域推論設定檔 ID（帶有 `us.` 前綴）。如果您使用不同的區域前綴或應用程式推論設定檔，請相應調整。如需目前和舊版模型 ID，請參閱模型概觀。請參閱模型設定以取得完整的環境變數清單。

未設定固定變數時，Claude Code 使用這些預設模型：

模型類型	預設值
主要模型	`global.anthropic.claude-sonnet-4-6`
小型/快速模型	`us.anthropic.claude-haiku-4-5-20251001-v1:0`

若要進一步自訂模型，請使用以下其中一種方法：

```
## 使用推論設定檔 ID
export ANTHROPIC_MODEL='global.anthropic.claude-sonnet-4-6'
export ANTHROPIC_SMALL_FAST_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'

## 使用應用程式推論設定檔 ARN
export ANTHROPIC_MODEL='arn:aws:bedrock:us-east-2:your-account-id:application-inference-profile/your-model-id'

## 選用：如果需要，停用 prompt caching
export DISABLE_PROMPT_CACHING=1
```

> **Note:**
>
> Prompt caching 可能不適用於所有區域。

將每個模型版本對應至推論設定檔

`ANTHROPIC_DEFAULT_*_MODEL` 環境變數為每個模型系列設定一個推論設定檔。如果您的組織需要在 `/model` 選擇器中公開同一系列的多個版本，每個版本都路由到其自己的應用程式推論設定檔 ARN，請改用設定檔中的 `modelOverrides` 設定。

此範例將三個 Opus 版本對應至不同的 ARN，以便使用者可以在它們之間切換，而無需繞過您組織的推論設定檔：

```
{
  "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"
  }
}
```

當使用者在 `/model` 中選取其中一個版本時，Claude Code 會使用對應的 ARN 呼叫 Bedrock。沒有覆寫的版本會回退到內建的 Bedrock 模型 ID 或在啟動時發現的任何相符推論設定檔。請參閱覆寫每個版本的模型 ID，以了解覆寫如何與 `availableModels` 和其他模型設定互動的詳細資訊。

IAM 設定

建立具有 Claude Code 所需權限的 IAM 政策：

```
{
  "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"
        }
      }
    }
  ]
}
```

如需更嚴格的權限，您可以將資源限制為特定的推論設定檔 ARN。

如需詳細資訊，請參閱 Bedrock IAM 文件。

> **Note:**
> 為 Claude Code 建立專用的 AWS 帳戶，以簡化成本追蹤和存取控制。

AWS Guardrails

Amazon Bedrock Guardrails 可讓您為 Claude Code 實施內容篩選。在 Amazon Bedrock 主控台中建立 Guardrail，發佈版本，然後將 Guardrail 標頭新增至您的設定檔。如果您使用跨區域推論設定檔，請在 Guardrail 上啟用跨區域推論。

範例設定：

```
{
  "env": {
    "ANTHROPIC_CUSTOM_HEADERS": "X-Amzn-Bedrock-GuardrailIdentifier: your-guardrail-id\nX-Amzn-Bedrock-GuardrailVersion: 1"
  }
}
```

故障排除

如果您遇到區域問題：

- 檢查模型可用性：`aws bedrock list-inference-profiles --region your-region`
- 切換至支援的區域：`export AWS_REGION=us-east-1`
- 考慮使用推論設定檔進行跨區域存取

如果您收到「不支援隨需輸送量」的錯誤：

- 將模型指定為推論設定檔 ID

Claude Code 使用 Bedrock Invoke API，不支援 Converse API。

其他資源

- Bedrock 文件
- Bedrock 定價
- Bedrock 推論設定檔
- Amazon Bedrock 上的 Claude Code：快速設定指南
- Claude Code 監控實施 (Bedrock)

Google Vertex AI 上的 Claude Code

> 了解如何透過 Google Vertex AI 設定 Claude Code，包括設定、IAM 設定和故障排除。

先決條件

在使用 Vertex AI 設定 Claude Code 之前，請確保您具有：

- 已啟用計費的 Google Cloud Platform (GCP) 帳戶
- 已啟用 Vertex AI API 的 GCP 專案
- 存取所需的 Claude 模型（例如 Claude Sonnet 4.5）
- 已安裝並設定 Google Cloud SDK (`gcloud`)
- 在所需的 GCP 區域中分配的配額

區域設定

Claude Code 可與 Vertex AI 全球和區域端點搭配使用。

> **Note:**
>
> Vertex AI 可能不支援所有區域上的 Claude Code 預設模型。您可能需要切換到支援的區域或模型。

> **Note:**
>
> Vertex AI 可能不支援全球端點上的 Claude Code 預設模型。您可能需要切換到區域端點或支援的模型。

設定

1. 啟用 Vertex AI API

在您的 GCP 專案中啟用 Vertex AI API：

```
## 設定您的專案 ID
gcloud config set project YOUR-PROJECT-ID

## 啟用 Vertex AI API
gcloud services enable aiplatform.googleapis.com
```

2. 要求模型存取

要求在 Vertex AI 中存取 Claude 模型：

1. 導覽至 Vertex AI Model Garden
2. 搜尋「Claude」模型
3. 要求存取所需的 Claude 模型（例如 Claude Sonnet 4.5）
4. 等待核准（可能需要 24-48 小時）

3. 設定 GCP 認證

Claude Code 使用標準的 Google Cloud 驗證。

如需詳細資訊，請參閱 Google Cloud 驗證文件。

> **Note:**
>
> 進行驗證時，Claude Code 將自動使用 `ANTHROPIC_VERTEX_PROJECT_ID` 環境變數中的專案 ID。若要覆寫此設定，請設定下列其中一個環境變數： `GCLOUD_PROJECT` 、 `GOOGLE_CLOUD_PROJECT` 或 `GOOGLE_APPLICATION_CREDENTIALS` 。

4. 設定 Claude Code

設定下列環境變數：

```
## 啟用 Vertex AI 整合
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=global
export ANTHROPIC_VERTEX_PROJECT_ID=YOUR-PROJECT-ID

## 選用：如需要，停用 prompt caching
export DISABLE_PROMPT_CACHING=1

## 當 CLOUD_ML_REGION=global 時，覆寫不支援模型的區域
export VERTEX_REGION_CLAUDE_3_5_HAIKU=us-east5

## 選用：覆寫其他特定模型的區域
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:**
>
> 當您指定 `cache_control` 暫時旗標時，Prompt caching 會自動支援。若要停用它，請設定 `DISABLE_PROMPT_CACHING=1`。如需提高速率限制，請聯絡 Google Cloud 支援。

> **Note:**
>
> 使用 Vertex AI 時，`/login` 和 `/logout` 命令已停用，因為驗證是透過 Google Cloud 認證處理的。

5. 模型設定

Claude Code 為 Vertex AI 使用這些預設模型：

模型類型	預設值
主要模型	`claude-sonnet-4-5@20250929`
小型/快速模型	`claude-haiku-4-5@20251001`

> **Note:**
>
> 對於 Vertex AI 使用者，Claude Code 不會自動從 Haiku 3.5 升級到 Haiku 4.5。若要手動切換到較新的 Haiku 模型，請將 `ANTHROPIC_DEFAULT_HAIKU_MODEL` 環境變數設定為完整模型名稱（例如 `claude-haiku-4-5@20251001`）。

若要自訂模型：

```
export ANTHROPIC_MODEL='claude-opus-4-1@20250805'
export ANTHROPIC_SMALL_FAST_MODEL='claude-haiku-4-5@20251001'
```

IAM 設定

指派所需的 IAM 權限：

`roles/aiplatform.user` 角色包含所需的權限：

- `aiplatform.endpoints.predict` - 模型叫用和權杖計數所需

如需更嚴格的權限，請建立只包含上述權限的自訂角色。

如需詳細資訊，請參閱 Vertex IAM 文件。

> **Note:**
>
> 我們建議為 Claude Code 建立專用的 GCP 專案，以簡化成本追蹤和存取控制。

1M token context window

Claude Sonnet 4 和 Sonnet 4.5 在 Vertex AI 上支援 1M token context window。

> **Note:**
>
> 1M token context window 目前處於測試版。若要使用擴展的 context window，請在您的 Vertex AI 要求中包含 `context-1m-2025-08-07` 測試版標頭。

故障排除

如果您遇到配額問題：

- 透過 Cloud Console 檢查目前配額或要求增加配額

如果您遇到「找不到模型」404 錯誤：

- 確認模型在 Model Garden 中已啟用
- 驗證您有權存取指定的區域
- 如果使用 `CLOUD_ML_REGION=global`，請檢查您的模型是否在 Model Garden 中的「支援的功能」下支援全球端點。對於不支援全球端點的模型，請執行下列其中一項：
  - 透過 `ANTHROPIC_MODEL` 或 `ANTHROPIC_SMALL_FAST_MODEL` 指定支援的模型，或
  - 使用 `VERTEX_REGION_<MODEL_NAME>` 環境變數設定區域端點

如果您遇到 429 錯誤：

- 對於區域端點，請確保主要模型和小型/快速模型在您選定的區域中受支援
- 考慮切換到 `CLOUD_ML_REGION=global` 以獲得更好的可用性

其他資源

- Vertex AI 文件
- Vertex AI 定價
- Vertex AI 配額和限制

Claude Code on Microsoft Foundry

> 了解如何透過 Microsoft Foundry 配置 Claude Code，包括設定、配置和故障排除。

先決條件

在使用 Microsoft Foundry 配置 Claude Code 之前，請確保您具有：

- 具有 Microsoft Foundry 存取權限的 Azure 訂閱
- 建立 Microsoft Foundry 資源和部署的 RBAC 權限
- 已安裝並配置 Azure CLI（選用 - 僅在您沒有其他取得認證機制時才需要）

設定

1. 佈建 Microsoft Foundry 資源

首先，在 Azure 中建立 Claude 資源：

1. 瀏覽至 Microsoft Foundry 入口網站
2. 建立新資源，並記下您的資源名稱
3. 為 Claude 模型建立部署：
    - Claude Opus
    - Claude Sonnet
    - Claude Haiku

2. 配置 Azure 認證

Claude Code 支援 Microsoft Foundry 的兩種驗證方法。選擇最適合您安全性要求的方法。

選項 A：API 金鑰驗證

1. 在 Microsoft Foundry 入口網站中瀏覽至您的資源
2. 前往 **端點和金鑰** 部分
3. 複製 **API 金鑰**
4. 設定環境變數：

```
export ANTHROPIC_FOUNDRY_API_KEY=your-azure-api-key
```

選項 B：Microsoft Entra ID 驗證

當未設定 `ANTHROPIC_FOUNDRY_API_KEY` 時，Claude Code 會自動使用 Azure SDK 預設認證鏈。這支援多種方法來驗證本機和遠端工作負載。

在本機環境中，您通常可以使用 Azure CLI：

```
az login
```

> **Note:**
>
> 使用 Microsoft Foundry 時，`/login` 和 `/logout` 命令已停用，因為驗證是透過 Azure 認證處理的。

3. 配置 Claude Code

設定下列環境變數以啟用 Microsoft Foundry。請注意，您的部署名稱會設定為 Claude Code 中的模型識別碼（如果使用建議的部署名稱，可能是選用的）。

```
## 啟用 Microsoft Foundry 整合
export CLAUDE_CODE_USE_FOUNDRY=1

## Azure 資源名稱（將 {resource} 替換為您的資源名稱）
export ANTHROPIC_FOUNDRY_RESOURCE={resource}
## 或提供完整的基礎 URL：
## export ANTHROPIC_FOUNDRY_BASE_URL=https://{resource}.services.ai.azure.com

## 將模型設定為您資源的部署名稱
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'
```

如需模型配置選項的更多詳細資訊，請參閱模型配置。

Azure RBAC 配置

`Azure AI User` 和 `Cognitive Services User` 預設角色包含叫用 Claude 模型所需的所有權限。

如需更嚴格的權限，請建立具有以下內容的自訂角色：

```
{
  "permissions": [
    {
      "dataActions": [
        "Microsoft.CognitiveServices/accounts/providers/*"
      ]
    }
  ]
}
```

如需詳細資訊，請參閱 Microsoft Foundry RBAC 文件。

故障排除

如果您收到錯誤「Failed to get token from azureADTokenProvider: ChainedTokenCredential authentication failed」：

- 在環境中配置 Entra ID，或設定 `ANTHROPIC_FOUNDRY_API_KEY`。

其他資源

- Microsoft Foundry 文件
- Microsoft Foundry 模型
- Microsoft Foundry 定價

LLM gateway 配置

> 了解如何配置 Claude Code 以使用 LLM gateway 解決方案。涵蓋 gateway 要求、身份驗證配置、模型選擇和提供商特定的端點設置。

LLM gateway 提供了 Claude Code 和模型提供商之間的集中代理層，通常提供：

- **集中身份驗證** - 單一 API 密鑰管理點
- **使用情況追蹤** - 監控跨團隊和項目的使用情況
- **成本控制** - 實施預算和速率限制
- **審計日誌** - 追蹤所有模型交互以進行合規性檢查
- **模型路由** - 無需更改代碼即可在提供商之間切換

Gateway 要求

為了讓 LLM gateway 與 Claude Code 配合使用，它必須滿足以下要求：

API 格式

gateway 必須向客戶端公開以下至少一種 API 格式：

1. **Anthropic Messages**: `/v1/messages`, `/v1/messages/count_tokens`
   - 必須轉發請求標頭：`anthropic-beta`、`anthropic-version`
2. **Bedrock InvokeModel**: `/invoke`, `/invoke-with-response-stream`
   - 必須保留請求正文字段：`anthropic_beta`、`anthropic_version`
3. **Vertex rawPredict**: `:rawPredict`、`:streamRawPredict`、`/count-tokens:rawPredict`
   - 必須轉發請求標頭：`anthropic-beta`、`anthropic-version`

未能轉發標頭或保留正文字段可能會導致功能減少或無法使用 Claude Code 功能。

> **Note:**
>
> Claude Code 根據 API 格式確定要啟用的功能。使用 Bedrock 或 Vertex 的 Anthropic Messages 格式時，您可能需要設置環境變數 `CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS=1`。

配置

模型選擇

默認情況下，Claude Code 將為選定的 API 格式使用標準模型名稱。

如果您在 gateway 中配置了自定義模型名稱，請使用 模型配置 中記錄的環境變數來匹配您的自定義名稱。

LiteLLM 配置

> **Note:**
>
> LiteLLM 是第三方代理服務。Anthropic 不認可、維護或審計 LiteLLM 的安全性或功能。本指南僅供參考，可能會過時。請自行決定是否使用。

先決條件

- Claude Code 已更新至最新版本
- LiteLLM Proxy Server 已部署且可訪問
- 通過您選擇的提供商訪問 Claude 模型

基本 LiteLLM 設置

配置 Claude Code：

身份驗證方法

靜態 API 密鑰

使用固定 API 密鑰的最簡單方法：

```
## 在環境中設置
export ANTHROPIC_AUTH_TOKEN=sk-litellm-static-key

## 或在 Claude Code 設置中
{
  "env": {
    "ANTHROPIC_AUTH_TOKEN": "sk-litellm-static-key"
  }
}
```

此值將作為 `Authorization` 標頭發送。

使用幫助程序的動態 API 密鑰

用於輪換密鑰或按用戶身份驗證：

1. 創建 API 密鑰幫助程序腳本：

```
#!/bin/bash
## ~/bin/get-litellm-key.sh

## 示例：從保管庫獲取密鑰
vault kv get -field=api_key secret/litellm/claude-code

## 示例：生成 JWT 令牌
jwt encode \
  --secret="${JWT_SECRET}" \
  --exp="+1h" \
  '{"user":"'${USER}'","team":"engineering"}'
```

1. 配置 Claude Code 設置以使用幫助程序：

```
{
  "apiKeyHelper": "~/bin/get-litellm-key.sh"
}
```

1. 設置令牌刷新間隔：

```
## 每小時刷新一次（3600000 毫秒）
export CLAUDE_CODE_API_KEY_HELPER_TTL_MS=3600000
```

此值將作為 `Authorization` 和 `X-Api-Key` 標頭發送。`apiKeyHelper` 的優先級低於 `ANTHROPIC_AUTH_TOKEN` 或 `ANTHROPIC_API_KEY`。

統一端點（推薦）

使用 LiteLLM 的 Anthropic 格式端點：

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000
```

統一端點相對於傳遞端點的優勢：

- 負載均衡
- 故障轉移
- 對成本追蹤和最終用戶追蹤的一致支持

提供商特定的傳遞端點（替代方案）

通過 LiteLLM 的 Claude API

使用 傳遞端點：

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000/anthropic
```

通過 LiteLLM 的 Amazon Bedrock

使用 傳遞端點：

```
export ANTHROPIC_BEDROCK_BASE_URL=https://litellm-server:4000/bedrock
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1
export CLAUDE_CODE_USE_BEDROCK=1
```

通過 LiteLLM 的 Google Vertex AI

使用 傳遞端點：

```
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
```

有關更多詳細信息，請參閱 LiteLLM 文檔。

其他資源

- LiteLLM 文檔
- Claude Code 設置
- 企業網絡配置
- 第三方集成概述

企業網路配置

> 為企業環境配置 Claude Code，支援代理伺服器、自訂憑證授權單位 (CA) 和相互傳輸層安全性 (mTLS) 驗證。

Claude Code 透過環境變數支援各種企業網路和安全配置。這包括透過公司代理伺服器路由流量、信任自訂憑證授權單位 (CA)，以及使用相互傳輸層安全性 (mTLS) 憑證進行驗證以增強安全性。

> **Note:**
>
> 本頁面顯示的所有環境變數也可以在 `settings.json` 中配置。

代理配置

環境變數

Claude Code 遵守標準代理環境變數：

```
## HTTPS 代理（建議）
export HTTPS_PROXY=https://proxy.example.com:8080

## HTTP 代理（如果 HTTPS 不可用）
export HTTP_PROXY=http://proxy.example.com:8080

## 繞過特定請求的代理 - 空格分隔格式
export NO_PROXY="localhost 192.168.1.1 example.com .example.com"
## 繞過特定請求的代理 - 逗號分隔格式
export NO_PROXY="localhost,192.168.1.1,example.com,.example.com"
## 繞過所有請求的代理
export NO_PROXY="*"
```

> **Note:**
>
> Claude Code 不支援 SOCKS 代理。

基本驗證

如果您的代理需要基本驗證，請在代理 URL 中包含認證資訊：

```
export HTTPS_PROXY=http://username:password@proxy.example.com:8080
```

> **Warning:**
> 避免在指令碼中硬編碼密碼。改用環境變數或安全認證儲存。

> **Tip:**
> 對於需要進階驗證（NTLM、Kerberos 等）的代理，請考慮使用支援您的驗證方法的 LLM Gateway 服務。

自訂 CA 憑證

如果您的企業環境使用自訂 CA 進行 HTTPS 連線（無論是透過代理還是直接 API 存取），請配置 Claude Code 以信任它們：

```
export NODE_EXTRA_CA_CERTS=/path/to/ca-cert.pem
```

mTLS 驗證

對於需要用戶端憑證驗證的企業環境：

```
## 用於驗證的用戶端憑證
export CLAUDE_CODE_CLIENT_CERT=/path/to/client-cert.pem

## 用戶端私密金鑰
export CLAUDE_CODE_CLIENT_KEY=/path/to/client-key.pem

## 選用：加密私密金鑰的密碼
export CLAUDE_CODE_CLIENT_KEY_PASSPHRASE="your-passphrase"
```

網路存取需求

Claude Code 需要存取以下 URL：

- `api.anthropic.com`：Claude API 端點
- `claude.ai`：claude.ai 帳戶的驗證
- `platform.claude.com`：Anthropic Console 帳戶的驗證

確保這些 URL 在您的代理配置和防火牆規則中被列入白名單。這在容器化或受限網路環境中使用 Claude Code 時特別重要。

其他資源

- Claude Code 設定
- 環境變數參考
- 疑難排解指南

Part 12: Environment Setup

開發容器

> 了解 Claude Code 開發容器，適合需要一致、安全環境的團隊。

參考 devcontainer 設置 和相關的 Dockerfile 提供了一個預先配置的開發容器，您可以按原樣使用或根據需要自訂。此 devcontainer 與 Visual Studio Code Dev Containers 擴充功能 和類似工具相容。

容器的增強安全措施（隔離和防火牆規則）允許您執行 `claude --dangerously-skip-permissions` 以繞過權限提示，實現無人值守操作。

> **Warning:**
>
> 雖然 devcontainer 提供了大量保護，但沒有任何系統完全免疫所有攻擊。 當使用 `--dangerously-skip-permissions` 執行時，devcontainer 不會阻止惡意專案從 devcontainer 中可存取的任何內容（包括 Claude Code 認證）進行資料外洩。 我們建議僅在使用受信任的儲存庫進行開發時使用 devcontainer。 始終保持良好的安全實踐並監控 Claude 的活動。

主要功能

- **生產就緒的 Node.js**：基於 Node.js 20 構建，包含必要的開發依賴項
- **設計安全**：自訂防火牆限制網路存取，僅允許必要的服務
- **開發人員友善的工具**：包括 git、ZSH 及生產力增強功能、fzf 等
- **無縫 VS Code 整合**：預先配置的擴充功能和最佳化設定
- **工作階段持久性**：在容器重新啟動之間保留命令歷史記錄和配置
- **隨處可用**：相容於 macOS、Windows 和 Linux 開發環境

4 步快速入門

1. 安裝 VS Code 和 Remote - Containers 擴充功能
2. 複製 Claude Code 參考實現 儲存庫
3. 在 VS Code 中開啟儲存庫
4. 出現提示時，點擊「在容器中重新開啟」（或使用命令面板：Cmd+Shift+P → 「Remote-Containers: Reopen in Container」）

配置詳解

devcontainer 設置包含三個主要元件：

- **devcontainer.json**：控制容器設定、擴充功能和磁碟區掛載
- **Dockerfile**：定義容器映像和已安裝的工具
- **init-firewall.sh**：建立網路安全規則

安全功能

容器透過其防火牆配置實現多層安全方法：

- **精確存取控制**：將出站連線限制為僅白名單網域（npm 登錄、GitHub、Claude API 等）
- **允許的出站連線**：防火牆允許出站 DNS 和 SSH 連線
- **預設拒絕原則**：阻止所有其他外部網路存取
- **啟動驗證**：在容器初始化時驗證防火牆規則
- **隔離**：建立與主系統分離的安全開發環境

自訂選項

devcontainer 配置設計為適應您的需求：

- 根據您的工作流程新增或移除 VS Code 擴充功能
- 針對不同的硬體環境修改資源配置
- 調整網路存取權限
- 自訂 shell 配置和開發人員工具

使用案例範例

安全的客戶端工作

使用 devcontainer 隔離不同的客戶端專案，確保程式碼和認證不會在環境之間混合。

團隊入職

新團隊成員可以在幾分鐘內獲得完全配置的開發環境，所有必要的工具和設定都已預先安裝。

一致的 CI/CD 環境

在 CI/CD 管道中鏡像您的 devcontainer 配置，以確保開發和生產環境相符。

相關資源

- VS Code devcontainer 文件
- Claude Code 安全最佳實踐
- 企業網路配置

優化您的終端機設置

> Claude Code 在終端機配置正確時效果最佳。請遵循這些指南來優化您的體驗。

主題和外觀

Claude 無法控制您終端機的主題。這由您的終端機應用程式處理。您可以隨時透過 `/config` 命令將 Claude Code 的主題與您的終端機相匹配。

如需進一步自訂 Claude Code 介面本身，您可以配置自訂狀態列以在終端機底部顯示上下文資訊，例如目前的模型、工作目錄或 git 分支。

換行符

您有多個選項可以在 Claude Code 中輸入換行符：

- **快速逃脫**：輸入 `\` 後跟 Enter 以建立新行
- **Shift+Enter**：在 iTerm2、WezTerm、Ghostty 和 Kitty 中開箱即用
- **鍵盤快捷鍵**：在其他終端機中設置按鍵綁定以插入新行

為其他終端機設置 Shift+Enter

在 Claude Code 中執行 `/terminal-setup` 以自動為 VS Code、Alacritty、Zed 和 Warp 配置 Shift+Enter。

> **Note:**
>
> `/terminal-setup` 命令僅在需要手動配置的終端機中可見。如果您使用 iTerm2、WezTerm、Ghostty 或 Kitty，您將看不到此命令，因為 Shift+Enter 已經原生運作。

設置 Option+Enter（VS Code、iTerm2 或 macOS Terminal.app）

對於 Mac Terminal.app：

1. 開啟設定 → 設定檔 → 鍵盤
2. 勾選「使用 Option 作為 Meta 鍵」

對於 iTerm2 和 VS Code 終端機：

1. 開啟設定 → 設定檔 → 按鍵
2. 在「一般」下，將左/右 Option 鍵設置為「Esc+」

通知設置

透過適當的通知配置，永遠不會錯過 Claude 完成任務的時刻：

iTerm 2 系統通知

對於 iTerm 2 任務完成時的警報：

1. 開啟 iTerm 2 偏好設定
2. 導覽至設定檔 → 終端機
3. 啟用「靜音鈴聲」和篩選警報 → 「傳送逃脫序列產生的警報」
4. 設置您偏好的通知延遲

請注意，這些通知是 iTerm 2 特有的，在預設的 macOS Terminal 中不可用。

自訂通知掛鉤

對於進階通知處理，您可以建立通知掛鉤以執行您自己的邏輯。

處理大型輸入

使用大量程式碼或長指令時：

- **避免直接貼上**：Claude Code 可能難以處理非常長的貼上內容
- **使用基於檔案的工作流程**：將內容寫入檔案並要求 Claude 讀取它
- **注意 VS Code 的限制**：VS Code 終端機特別容易截斷長貼上

Vim 模式

Claude Code 支援可透過 `/vim` 啟用或透過 `/config` 配置的 Vim 按鍵綁定子集。

支援的子集包括：

- 模式切換：`Esc`（至 NORMAL）、`i` / `I`、`a` / `A`、`o` / `O`（至 INSERT）
- 導覽：`h` / `j` / `k` / `l`、`w` / `e` / `b`、`0` / `$` / `^`、`gg` / `G`、`f` / `F` / `t` / `T` 搭配 `;` / `,` 重複
- 編輯：`x`、`dw` / `de` / `db` / `dd` / `D`、`cw` / `ce` / `cb` / `cc` / `C`、`.`（重複）
- 複製/貼上：`yy` / `Y`、`yw` / `ye` / `yb`、`p` / `P`
- 文字物件：`iw` / `aw`、`iW` / `aW`、`i"` / `a"`、`i'` / `a'`、`i(` / `a(`、`i[` / `a[`、`i{` / `a{`
- 縮排：`>>` / `<<`
- 行操作：`J`（合併行）

請參閱互動模式以取得完整參考。

自訂您的狀態列

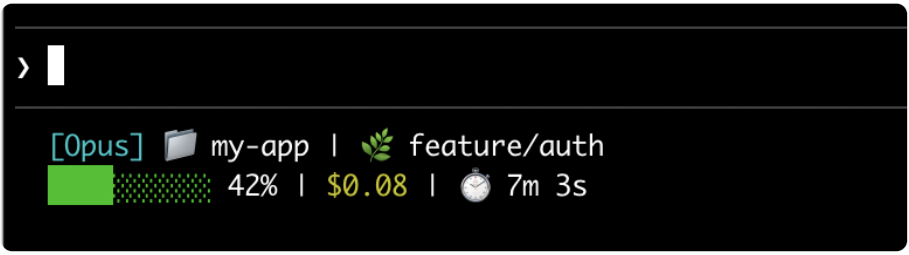
> 設定自訂狀態列以監控 Claude Code 中的上下文視窗使用情況、成本和 git 狀態

狀態列是 Claude Code 底部的可自訂列，可執行您設定的任何 shell 指令碼。它透過 stdin 接收 JSON 工作階段資料，並顯示您的指令碼列印的任何內容，為您提供上下文使用情況、成本、git 狀態或任何其他您想追蹤的內容的持久、一目瞭然的檢視。

狀態列在以下情況下很有用：

- 您想在工作時監控上下文視窗使用情況
- 您需要追蹤工作階段成本
- 您在多個工作階段中工作，需要區分它們
- 您想讓 git 分支和狀態始終可見

以下是一個多行狀態列的範例，在第一行顯示 git 資訊，在第二行顯示顏色編碼的上下文列。



多行狀態列，在第一行顯示模型名稱、目錄、git 分支，在第二行顯示上下文使用進度列、成本和持續時間

本頁面介紹設定基本狀態列、說明資料如何從 Claude Code 流向您的指令碼、列出您可以顯示的所有欄位，並提供常見模式的現成範例，例如 git 狀態、成本追蹤和進度列。

設定狀態列

使用 `/statusline` 命令讓 Claude Code 為您產生指令碼，或手動建立指令碼並將其新增到您的設定。

使用 /statusline 命令

`/statusline` 命令接受描述您想顯示內容的自然語言指令。Claude Code 在 `~/.claude/` 中產生指令碼檔案並自動更新您的設定：

```
/statusline show model name and context percentage with a progress bar
```

手動設定狀態列

將 `statusLine` 欄位新增到您的使用者設定（`~/.claude/settings.json`，其中 `~` 是您的主目錄）或專案設定。將 `type` 設定為 `"command"`，並將 `command` 指向指令碼路徑或內聯 shell 命令。如需建立指令碼的完整逐步說明，請參閱逐步建立狀態列。

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh",
    "padding": 2
  }
}
```

`command` 欄位在 shell 中執行，因此您也可以使用內聯命令而不是指令碼檔案。此範例使用 `jq` 解析 JSON 輸入並顯示模型名稱和上下文百分比：

```
{
  "statusLine": {
    "type": "command",
    "command": "jq -r '\"[\\(.model.display_name)] \\(.context_window.used_percent
age // 0)% context\"'"
  }
}
```

可選的 `padding` 欄位為狀態列內容新增額外的水平間距（以字元為單位）。預設為 `0`。此填充是在介面的內建間距之外，因此它控制相對縮排而不是距離終端邊緣的絕對距離。

停用狀態列

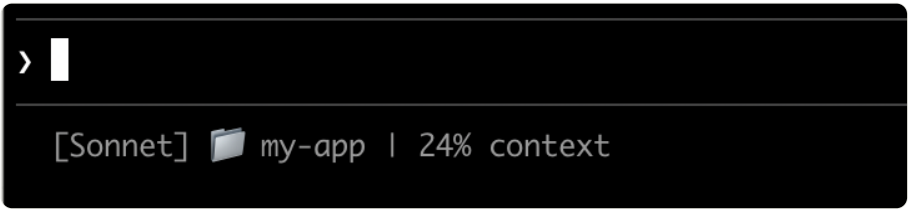
執行 `/statusline` 並要求它移除或清除您的狀態列（例如 `/statusline delete`、`/statusline clear`、`/statusline remove it`）。您也可以手動從 settings.json 中刪除 `statusLine` 欄位。

逐步建立狀態列

此逐步說明透過手動建立顯示目前模型、工作目錄和上下文視窗使用百分比的狀態列來展示幕後發生的情況。

> **Note:**
>
> 使用 `/statusline` 和您想要的內容描述會自動為您設定所有這些。

這些範例使用 Bash 指令碼，適用於 macOS 和 Linux。在 Windows 上，請參閱 Windows 設定以取得 PowerShell 和 Git Bash 範例。



狀態列顯示模型名稱、目錄和上下文百分比

Step 1: 建立讀取 JSON 並列印輸出的指令碼

Claude Code 透過 stdin 將 JSON 資料傳送到您的指令碼。此指令碼使用 `jq`（一個您可能需要安裝的命令列 JSON 解析器）來提取模型名稱、目錄和上下文百分比，然後列印格式化的行。

將此儲存到 `~/.claude/statusline.sh`（其中 `~` 是您的主目錄，例如 macOS 上的 `/Users/username` 或 Linux 上的 `/home/username`）：

```
#!/bin/bash
## Read JSON data that Claude Code sends to stdin
input=$(cat)

## Extract fields using jq
MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
## The "// 0" provides a fallback if the field is null
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## Output the status line - ${DIR##*/} extracts just the folder name
echo "[$MODEL] 📁 ${DIR##*/} | ${PCT}% context"
```

Step 2: 使其可執行

將指令碼標記為可執行，以便您的 shell 可以執行它：

```
chmod +x ~/.claude/statusline.sh
```

Step 3: 新增到設定

告訴 Claude Code 執行您的指令碼作為狀態列。將此設定新增到 `~/.claude/settings.json`，它將 `type` 設定為 `"command"`（意思是「執行此 shell 命令」）並將 `command` 指向您的指令碼：

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh"
  }
}
```

您的狀態列出現在介面底部。設定會自動重新載入，但變更在您與 Claude Code 的下一次互動之前不會出現。

狀態列如何運作

Claude Code 執行您的指令碼並透過 stdin 將JSON 工作階段資料傳送給它。您的指令碼讀取 JSON、提取所需內容並將文字列印到 stdout。Claude Code 顯示您的指令碼列印的任何內容。

何時更新

您的指令碼在每個新的助手訊息之後、權限模式變更時或 vim 模式切換時執行。更新在 300ms 處進行去抖動，這意味著快速變更會批次在一起，您的指令碼在事情穩定後執行一次。如果在您的指令碼仍在執行時觸發新的更新，則會取消進行中的執行。如果您編輯指令碼，變更在您與 Claude Code 的下一次互動觸發更新之前不會出現。

您的指令碼可以輸出什麼

- **多行**：每個 `echo` 或 `print` 陳述式顯示為單獨的行。請參閱多行範例。
- **顏色**：使用ANSI 逃逸碼，例如 `\033[32m` 表示綠色（終端必須支援它們）。請參閱 git 狀態範例。
- **連結**：使用 OSC 8 逃逸序列使文字可點擊（macOS 上為 Cmd+click，Windows/Linux 上為 Ctrl+click）。需要支援超連結的終端，例如 iTerm2、Kitty 或 WezTerm。請參閱可點擊連結範例。

> **Note:**
>
> 狀態列在本地執行，不消耗 API 令牌。在某些 UI 互動期間，它會暫時隱藏，包括自動完成建議、說明功能表和權限提示。

可用資料

Claude Code 透過 stdin 將以下 JSON 欄位傳送到您的指令碼：

欄位	描述
`model.id`, `model.display_name`	目前模型識別碼和顯示名稱
`cwd`, `workspace.current_dir`	目前工作目錄。兩個欄位包含相同的值；`workspace.current_dir` 因與 `workspace.project_dir` 一致而首選。
`workspace.project_dir`	啟動 Claude Code 的目錄，如果工作階段期間工作目錄變更，可能與 `cwd` 不同
`cost.total_cost_usd`	工作階段總成本（美元）

欄位	描述
`cost.total_duration_ms`	自工作階段開始以來的總掛鐘時間（毫秒）
`cost.total_api_duration_ms`	等待 API 回應所花費的總時間（毫秒）
`cost.total_lines_added`，`cost.total_lines_removed`	變更的程式碼行數
`context_window.total_input_tokens`，`context_window.total_output_tokens`	整個工作階段中的累積令牌計數
`context_window.context_window_size`	最大上下文視窗大小（令牌）。預設為 200000，或對於具有擴展上下文的模型為 1000000。
`context_window.used_percentage`	預先計算的已使用上下文視窗百分比
`context_window.remaining_percentage`	預先計算的剩餘上下文視窗百分比
`context_window.current_usage`	最後一次 API 呼叫中的令牌計數，在上下文視窗欄位中描述
`exceeds_200k_tokens`	最近 API 回應中的總令牌計數（輸入、快取和輸出令牌合併）是否超過 200k。這是一個固定閾值，與實際上下文視窗大小無關。
`session_id`	唯一工作階段識別碼
`transcript_path`	對話記錄檔案的路徑
`version`	Claude Code 版本
`output_style.name`	目前輸出樣式的名稱
`vim.mode`	啟用 vim 模式時的目前 vim 模式（`NORMAL` 或 `INSERT`）
`agent.name`	使用 `--agent` 旗標或設定代理設定時的代理名稱
`worktree.name`	作用中 worktree 的名稱。僅在 `--worktree` 工作階段期間出現
`worktree.path`	worktree 目錄的絕對路徑

欄位	描述
`worktree.branch`	worktree 的 Git 分支名稱（例如 `"worktree-my-feature"`）。對於基於 hook 的 worktree 不存在
`worktree.original_cwd`	Claude 進入 worktree 之前所在的目錄
`worktree.original_branch`	進入 worktree 之前簽出的 Git 分支。對於基於 hook 的 worktree 不存在

您的狀態列命令透過 stdin 接收此 JSON 結構：

```
{
  "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"
  }
}
```

可能不存在的欄位（不在 JSON 中）：

- `vim`：僅在啟用 vim 模式時出現
- `agent`：僅在使用 `--agent` 旗標或設定代理設定時出現
- `worktree`：僅在 `--worktree` 工作階段期間出現。存在時，`branch` 和 `original_branch` 對於基於 hook 的 worktree 也可能不存在

可能為 `null` 的欄位：

- `context_window.current_usage`：在工作階段中第一次 API 呼叫之前為 `null`
- `context_window.used_percentage`，`context_window.remaining_percentage`：在工作階段早期可能為 `null`

在您的指令碼中使用條件存取處理遺漏的欄位，並使用後備預設值處理 null 值。

上下文視窗欄位

`context_window` 物件提供兩種追蹤上下文使用情況的方式：

- **累積總計**（`total_input_tokens`，`total_output_tokens`）：整個工作階段中所有令牌的總和，用於追蹤總消耗
- **目前使用情況**（`current_usage`）：最後一次 API 呼叫中的令牌計數，使用此方式以獲得準確的上下文百分比，因為它反映實際上下文狀態

`current_usage` 物件包含：

- `input_tokens`：目前上下文中的輸入令牌
- `output_tokens`：產生的輸出令牌

- `cache_creation_input_tokens`：寫入快取的令牌
- `cache_read_input_tokens`：從快取讀取的令牌

`used_percentage` 欄位僅從輸入令牌計算：`input_tokens + cache_creation_input_tokens + cache_read_input_tokens`。它不包括 `output_tokens`。

如果您從 `current_usage` 手動計算上下文百分比，請使用相同的僅輸入公式以符合 `used_percentage`。

`current_usage` 物件在工作階段中第一次 API 呼叫之前為 `null`。

範例

這些範例展示常見的狀態列模式。若要使用任何範例：

1. 將指令碼儲存到檔案，例如 `~/.claude/statusline.sh`（或 `.py` / `.js`）
2. 使其可執行：`chmod +x ~/.claude/statusline.sh`
3. 將路徑新增到您的設定

Bash 範例使用 `jq` 來解析 JSON。Python 和 Node.js 具有內建的 JSON 解析。

上下文視窗使用情況

顯示目前模型和上下文視窗使用情況，帶有視覺進度列。每個指令碼從 stdin 讀取 JSON，提取 `used_percentage` 欄位，並建立一個 10 字元的列，其中填充的塊 (▓) 代表使用情況：

狀態列顯示模型名稱和帶有百分比的進度列

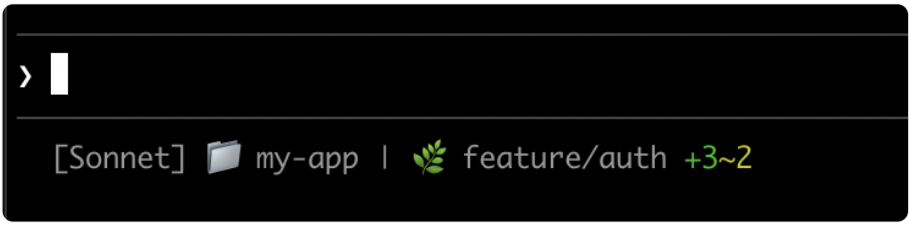
```bash
#!/bin/bash
## Read all of stdin into a variable
input=$(cat)

## Extract fields with jq, "// 0" provides fallback for null
MODEL=$(echo "$input" | jq -r '.model.display_name')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## Build progress bar: printf creates spaces, tr replaces with blocks
BAR_WIDTH=10
FILLED=$((PCT * BAR_WIDTH / 100))
EMPTY=$((BAR_WIDTH - FILLED))
BAR=""
[ "$FILLED" -gt 0 ] && BAR=$(printf "%${FILLED}s" | tr ' ' '▓')
[ "$EMPTY" -gt 0 ] && BAR="${BAR}$(printf "%${EMPTY}s" | tr ' ' '░')"

echo "[$MODEL] $BAR $PCT%"
```

```python
#!/usr/bin/env python3
import json, sys

## json.load reads and parses stdin in one step
data = json.load(sys.stdin)
model = data['model']['display_name']
## "or 0" handles null values
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)

## String multiplication builds the bar
filled = pct * 10 // 100
bar = '▓' * filled + '░' * (10 - filled)

print(f"[{model}] {bar} {pct}%")
```

```
#!/usr/bin/env node
// Node.js reads stdin asynchronously with events
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    // Optional chaining (?.) safely handles null fields
    const pct = Math.floor(data.context_window?.used_percentage || 0);

    // String.repeat() builds the bar
    const filled = Math.floor(pct * 10 / 100);
    const bar = '▓'.repeat(filled) + '░'.repeat(10 - filled);

    console.log(`[${model}] ${bar} ${pct}%`);
});
```

Git 狀態與顏色

顯示 git 分支，帶有暫存和修改檔案的顏色編碼指示器。此指令碼使用ANSI 逃逸碼表示終端顏色：`\033[32m` 是綠色，`\033[33m` 是黃色，`\033[0m` 重設為預設值。



狀態列顯示模型、目錄、git 分支和暫存和修改檔案的彩色指示器

每個指令碼檢查目前目錄是否是 git 儲存庫，計算暫存和修改檔案，並顯示顏色編碼的指示器：

```
#!/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
```

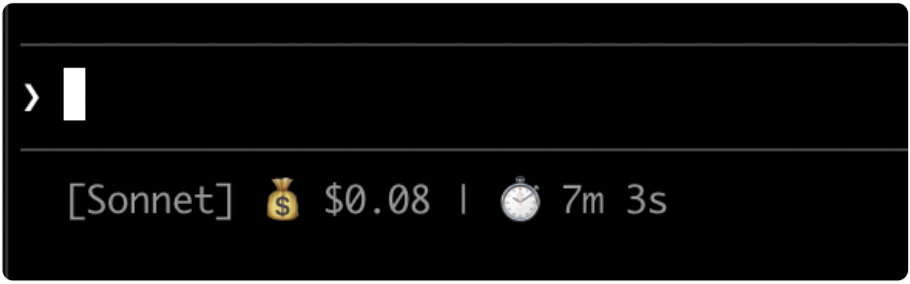
```
#!/usr/bin/env python3
import json, sys, subprocess, os

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}`);
    }
});
```

成本和持續時間追蹤

追蹤您工作階段的 API 成本和經過時間。`cost.total_cost_usd` 欄位累積目前工作階段中所有 API 呼叫的成本。`cost.total_duration_ms` 欄位測量自工作階段開始以來的總經過時間，而 `cost.total_api_duration_ms` 僅追蹤等待 API 回應所花費的時間。

每個指令碼將成本格式化為貨幣並將毫秒轉換為分鐘和秒：



狀態列顯示模型名稱、工作階段成本和持續時間

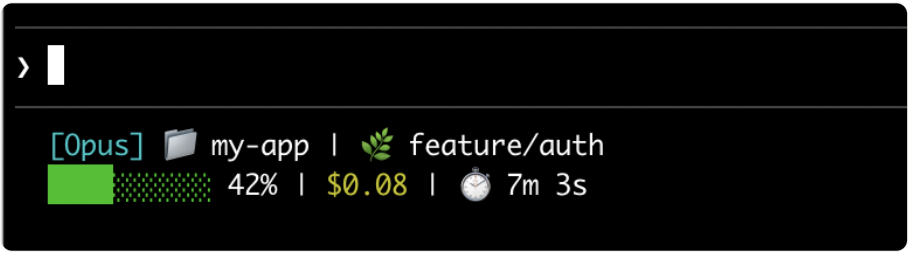
```bash
#!/bin/bash
input=$(cat)

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"
```

```python
#!/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`);
});
```

顯示多行

您的指令碼可以輸出多行以建立更豐富的顯示。每個 `echo` 陳述式在狀態區域中產生單獨的行。



多行狀態列，在第一行顯示模型名稱、目錄、git 分支，在第二行顯示上下文使用進度列、成本和持續時間

此範例結合了多種技術：基於閾值的顏色（70% 以下為綠色，70-89% 為黃色，90%+ 為紅色）、進度列和 git 分支資訊。每個 `print` 或 `echo` 陳述式建立單獨的行：

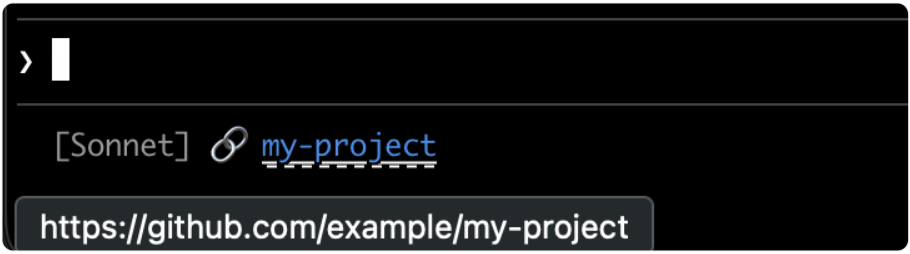
```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

CYAN='\033[36m'; GREEN='\033[32m'; YELLOW='\033[33m'; RED='\033[31m'; RESET='\033[
0m'

## Pick bar color based on context usage
if [ "$PCT" -ge 90 ]; then BAR_COLOR="$RED"
elif [ "$PCT" -ge 70 ]; then BAR_COLOR="$YELLOW"
else BAR_COLOR="$GREEN"; fi

FILLED=$((PCT / 10)); EMPTY=$((10 - FILLED))
BAR=$(printf "%${FILLED}s" | tr ' ' '▓')$(printf "%${EMPTY}s" | tr ' ' '░')

MINS=$((DURATION_MS / 60000)); SECS=$(((DURATION_MS % 60000) / 1000))

BRANCH=""
git rev-parse --git-dir > /dev/null 2>&1 && BRANCH=" | 🌿 $(git branch --show-
current 2>/dev/null)"

echo -e "${CYAN}[$MODEL]${RESET} 📁 ${DIR##*/}$BRANCH"
COST_FMT=$(printf '$%.2f' "$COST")
echo -e "${BAR_COLOR}${BAR}${RESET} ${PCT}% | ${YELLOW}${COST_FMT}${RESET} | ⏱️ $
{MINS}m ${SECS}s"
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

CYAN, GREEN, YELLOW, RED, RESET = '\033[36m', '\033[32m', '\033[33m', '\033[31m',
'\033[0m'

bar_color = RED if pct ≥ 90 else YELLOW if pct ≥ 70 else GREEN
filled = pct // 10
bar = '█' * filled + '░' * (10 - filled)

mins, secs = duration_ms // 60000, (duration_ms % 60000) // 1000

try:
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=Tru
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`);
});
```

可點擊連結

此範例建立指向您的 GitHub 儲存庫的可點擊連結。它讀取 git 遠端 URL，使用 `sed` 將 SSH 格式轉換為 HTTPS，並將儲存庫名稱包裝在 OSC 8 逃逸碼中。按住 Cmd (macOS) 或 Ctrl (Windows/Linux) 並點擊以在瀏覽器中開啟連結。



狀態列顯示指向 GitHub 儲存庫的可點擊連結

每個指令碼取得 git 遠端 URL，將 SSH 格式轉換為 HTTPS，並將儲存庫名稱包裝在 OSC 8 逃逸碼中。Bash 版本使用 `printf '%b'`，它比 `echo -e` 更可靠地跨不同 shell 解釋反斜杠逃逸：

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')

## Convert git SSH URL to HTTPS
REMOTE=$(git remote get-url origin 2>/dev/null | sed 's/git@github.com:/https:\/\/
github.com\//' | sed 's/\.git$//')

if [ -n "$REMOTE" ]; then
    REPO_NAME=$(basename "$REMOTE")
    # OSC 8 format: \e]8;;URL\a then TEXT then \e]8;;\a
    # printf %b interprets escape sequences reliably across shells
    printf '%b' "[$MODEL] 🔗 \e]8;;${REMOTE}\a${REPO_NAME}\e]8;;\a\n"
else
    echo "[$MODEL]"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, re, os

data = json.load(sys.stdin)
model = data['model']['display_name']

## Get git remote URL
try:
    remote = subprocess.check_output(
        ['git', 'remote', 'get-url', 'origin'],
        stderr=subprocess.DEVNULL, text=True
    ).strip()
    # Convert SSH to HTTPS format
    remote = re.sub(r'^git@github\.com:', 'https://github.com/', remote)
    remote = re.sub(r'\.git$', '', remote)
    repo_name = os.path.basename(remote)
    # OSC 8 escape sequences
    link = f"\033]8;;{remote}\a{repo_name}\033]8;;\a"
    print(f"[{model}] 🔗 {link}")
except:
    print(f"[{model}]")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;

    try {
        let remote = execSync('git remote get-url origin', { encoding: 'utf8', std
io: ['pipe', 'pipe', 'ignore'] }).trim();
        // Convert SSH to HTTPS format
        remote = remote.replace(/^git@github\.com:/, 'https://github.com/').replac
e(/\.git$/, '');
        const repoName = path.basename(remote);
        // OSC 8 escape sequences
        const link = `\x1b]8;;${remote}\x07${repoName}\x1b]8;;\x07`;
        console.log(`[${model}] 🔗 ${link}`);
    } catch {
        console.log(`[${model}]`);
    }
});
```

快取昂貴的操作

您的狀態列指令碼在活躍工作階段期間頻繁執行。`git status` 或 `git diff` 等命令可能很慢，特別是在大型儲存庫中。此範例將 git 資訊快取到臨時檔案，並且僅每 5 秒重新整理一次。

使用穩定的固定檔案名稱作為快取檔案，例如 `/tmp/statusline-git-cache`。每個狀態列呼叫作為新程序執行，因此基於程序的識別碼（如 `$$`、`os.getpid()` 或 `process.pid`）每次產生不同的值，快取永遠不會被重複使用。

每個指令碼在執行 git 命令之前檢查快取檔案是否遺漏或超過 5 秒：

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')

CACHE_FILE="/tmp/statusline-git-cache"
CACHE_MAX_AGE=5  # seconds

cache_is_stale() {
    [ ! -f "$CACHE_FILE" ] || \
    # stat -f %m is macOS, stat -c %Y is Linux
    [ $(($(date +%s) - $(stat -f %m "$CACHE_FILE" 2>/dev/null || stat -c %Y "$CACH
E_FILE" 2>/dev/null || echo 0))) -gt $CACHE_MAX_AGE ]
}

if cache_is_stale; then
    if git rev-parse --git-dir > /dev/null 2>&1; then
        BRANCH=$(git branch --show-current 2>/dev/null)
        STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
        MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')
        echo "$BRANCH|$STAGED|$MODIFIED" > "$CACHE_FILE"
    else
        echo "||" > "$CACHE_FILE"
    fi
fi

IFS='|' read -r BRANCH STAGED MODIFIED < "$CACHE_FILE"

if [ -n "$BRANCH" ]; then
    echo "[$MODEL] 📁 ${DIR##*/} | 🌿 $BRANCH +$STAGED ~$MODIFIED"
else
    echo "[$MODEL] 📁 ${DIR##*/}"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os, time

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])

CACHE_FILE = "/tmp/statusline-git-cache"
CACHE_MAX_AGE = 5  # seconds

def cache_is_stale():
    if not os.path.exists(CACHE_FILE):
        return True
    return time.time() - os.path.getmtime(CACHE_FILE) > CACHE_MAX_AGE

if cache_is_stale():
    try:
        subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subproce
ss.DEVNULL)
        branch = subprocess.check_output(['git', 'branch', '--show-current'],
text=True).strip()
        staged = subprocess.check_output(['git', 'diff', '--cached', '--
numstat'], text=True).strip()
        modified = subprocess.check_output(['git', 'diff', '--numstat'],
text=True).strip()
        staged_count = len(staged.split('\n')) if staged else 0
        modified_count = len(modified.split('\n')) if modified else 0
        with open(CACHE_FILE, 'w') as f:
            f.write(f"{branch}|{staged_count}|{modified_count}")
    except:
        with open(CACHE_FILE, 'w') as f:
            f.write("||")

with open(CACHE_FILE) as f:
    branch, staged, modified = f.read().strip().split('|')

if branch:
```

```
    print(f"[{model}] 📁 {directory} | 🌿 {branch} +{staged} ~{modified}")
else:
    print(f"[{model}] 📁 {directory}")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const fs = require('fs');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    const dir = path.basename(data.workspace.current_dir);

    const CACHE_FILE = '/tmp/statusline-git-cache';
    const CACHE_MAX_AGE = 5; // seconds

    const cacheIsStale = () => {
        if (!fs.existsSync(CACHE_FILE)) return true;
        return (Date.now() / 1000) - fs.statSync(CACHE_FILE).mtimeMs / 1000 >
CACHE_MAX_AGE;
    };

    if (cacheIsStale()) {
        try {
            execSync('git rev-parse --git-dir', { stdio: 'ignore' });
            const branch = execSync('git branch --show-current', { encoding: 'utf8
' }).trim();
            const staged = execSync('git diff --cached --numstat', { encoding: 'ut
f8' }).trim().split('\n').filter(Boolean).length;
            const modified = execSync('git diff --numstat', { encoding:
'utf8' }).trim().split('\n').filter(Boolean).length;
            fs.writeFileSync(CACHE_FILE, `${branch}|${staged}|${modified}`);
        } catch {
            fs.writeFileSync(CACHE_FILE, '||');
        }
    }

    const [branch, staged, modified] = fs.readFileSync(CACHE_FILE,
'utf8').trim().split('|');
```

```
    if (branch) {
        console.log(`[${model}] 📁 ${dir} | 🌿 ${branch} +${staged} ~${modified}
`);
    } else {
        console.log(`[${model}] 📁 ${dir}`);
    }
});
```

Windows 設定

在 Windows 上，Claude Code 透過 Git Bash 執行狀態列命令。您可以從該 shell 呼叫 PowerShell：

```
{
  "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]"
}
```

或直接執行 Bash 指令碼：

```
{
  "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]"
```

提示

- **使用模擬輸入測試**：`echo '{"model":{"display_name":"Opus"},"context_window":{"used_percentage":25}}' | ./statusline.sh`
- **保持輸出簡短**：狀態列寬度有限，因此長輸出可能會被截斷或換行不當
- **快取慢速操作**：您的指令碼在活躍工作階段期間頻繁執行，因此 `git status` 等命令可能會導致延遲。請參閱快取範例以瞭解如何處理此問題。

社群專案如 ccstatusline 和 starship-claude 提供具有主題和其他功能的預先建立設定。

疑難排解

狀態列未出現

- 驗證您的指令碼是否可執行：`chmod +x ~/.claude/statusline.sh`
- 檢查您的指令碼是否輸出到 stdout 而不是 stderr
- 手動執行您的指令碼以驗證它產生輸出
- 如果 `disableAllHooks` 在您的設定中設定為 `true`，狀態列也會被停用。移除此設定或將其設定為 `false` 以重新啟用。
- 執行 `claude --debug` 以記錄工作階段中第一次狀態列呼叫的結束代碼和 stderr
- 要求 Claude 讀取您的設定檔案並直接執行 `statusLine` 命令以顯示錯誤

狀態列顯示 `--` 或空值

- 欄位在第一次 API 回應完成之前可能為 `null`
- 在您的指令碼中使用後備（例如 jq 中的 `// 0`）處理 null 值
- 如果值在多個訊息後仍為空，請重新啟動 Claude Code

上下文百分比顯示意外值

- 使用 `used_percentage` 以獲得準確的上下文狀態，而不是累積總計
- `total_input_tokens` 和 `total_output_tokens` 在整個工作階段中累積，可能超過上下文視窗大小
- 上下文百分比可能與 `/context` 輸出不同，因為每個計算時間不同

OSC 8 連結不可點擊

- 驗證您的終端支援 OSC 8 超連結（iTerm2、Kitty、WezTerm）
- Terminal.app 不支援可點擊連結
- SSH 和 tmux 工作階段可能根據設定去除 OSC 序列
- 如果逃逸序列顯示為文字（如 `\e]8;;`），請使用 `printf '%b'` 而不是 `echo -e` 以獲得更可靠的逃逸處理

逃逸序列的顯示故障

- 複雜的逃逸序列（ANSI 顏色、OSC 8 連結）如果與其他 UI 更新重疊，偶爾會導致輸出亂碼
- 如果您看到損壞的文字，請嘗試將指令碼簡化為純文字輸出
- 帶有逃逸碼的多行狀態列比單行純文字更容易出現呈現問題

指令碼錯誤或掛起

- 以非零代碼結束或不產生輸出的指令碼會導致狀態列變為空白
- 慢速指令碼會阻止狀態列更新，直到它們完成。保持指令碼快速以避免過時輸出。
- 如果在慢速指令碼執行時觸發新的更新，進行中的指令碼會被取消
- 在設定之前使用模擬輸入獨立測試您的指令碼

通知共享狀態列行

- 系統通知（如 MCP 伺服器錯誤、自動更新和令牌警告）顯示在與您的狀態列相同行的右側
- 啟用詳細模式會在此區域新增令牌計數器
- 在狹窄的終端上，這些通知可能會截斷您的狀態列輸出

Part 13: Troubleshooting & Changelog

疑難排解

> 探索 Claude Code 安裝和使用中常見問題的解決方案。

疑難排解安裝問題

> **Tip:**
>
> 如果您想完全跳過終端，Claude Code Desktop 應用程式可讓您透過圖形介面安裝和使用 Claude Code。下載適用於 macOS 或 Windows 的版本，無需任何命令列設定即可開始編碼。

找到您看到的錯誤訊息或症狀：

您看到的內容	解決方案
`command not found: claude` 或 `'claude' is not recognized`	修復您的 PATH
`syntax error near unexpected token '<'`	安裝指令碼傳回 HTML
`curl: (56) Failure writing output to destination`	先下載指令碼，然後執行
Linux 上安裝期間 `Killed`	為低記憶體伺服器新增交換空間
`TLS connect error` 或 `SSL/TLS secure channel`	更新 CA 憑證
`Failed to fetch version` 或無法連接下載伺服器	檢查網路和代理設定
`irm is not recognized` 或 `&& is not valid`	為您的 shell 使用正確的命令
`Claude Code on Windows requires git-bash`	安裝或設定 Git Bash

您看到的內容	解決方案
`Error loading shared library`	您的系統安裝了錯誤的二進位變體
Linux 上的 `Illegal instruction`	架構不匹配
macOS 上的 `dyld: cannot load` 或 `Abort trap`	二進位不相容
`Invoke-Expression: Missing argument in parameter list`	安裝指令碼傳回 HTML
`App unavailable in region`	Claude Code 在您的國家/地區不可用。請參閱支援的國家/地區。
`unable to get local issuer certificate`	設定公司 CA 憑證
`OAuth error` 或 `403 Forbidden`	修復驗證

如果您的問題未列出，請執行這些診斷步驟。

偵錯安裝問題

檢查網路連線

安裝程式從 `storage.googleapis.com` 下載。驗證您可以連接到它：

```
curl -sI https://storage.googleapis.com
```

如果失敗，您的網路可能阻止了連線。常見原因：

- 公司防火牆或代理阻止 Google Cloud Storage
- 區域網路限制：嘗試使用 VPN 或替代網路
- TLS/SSL 問題：更新您系統的 CA 憑證，或檢查是否設定了 `HTTPS_PROXY`

如果您在公司代理後面，在安裝前設定 `HTTPS_PROXY` 和 `HTTP_PROXY` 為您的代理位址。如果您不知道代理 URL，請詢問您的 IT 團隊，或檢查您的瀏覽器代理設定。

此範例設定兩個代理變數，然後透過您的代理執行安裝程式：

```
export HTTP_PROXY=http://proxy.example.com:8080
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

驗證您的 PATH

如果安裝成功但執行 `claude` 時出現 `command not found` 或 `not recognized` 錯誤，安裝目錄不在您的 PATH 中。您的 shell 在 PATH 中列出的目錄中搜尋程式，安裝程式在 macOS/Linux 上將 `claude` 放在 `~/.local/bin/claude`，或在 Windows 上放在 `%USERPROFILE%\.local\bin\claude.exe`。

透過列出您的 PATH 項目並篩選 `local/bin` 來檢查安裝目錄是否在您的 PATH 中：

macOS/Linux

```
echo $PATH | tr ':' '\n' | grep local/bin
```

如果沒有輸出，該目錄遺失。將其新增到您的 shell 設定：

```
## Zsh (macOS 預設)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc

## Bash (Linux 預設)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

或者，關閉並重新開啟您的終端。

驗證修復是否有效：

```
claude --version
```

Windows PowerShell

```
$env:PATH -split ';' | Select-String 'local\\bin'
```

如果沒有輸出，將安裝目錄新增到您的使用者 PATH：

```
$currentPath = [Environment]::GetEnvironmentVariable('PATH', 'User')
[Environment]::SetEnvironmentVariable('PATH', "$currentPath;$env:USERPROFILE\.local\bin", 'User')
```

重新啟動您的終端以使變更生效。

驗證修復是否有效：

```
claude --version
```

Windows CMD

```
echo %PATH% | findstr /i "local\bin"
```

如果沒有輸出，開啟系統設定，前往環境變數，並將 `%USERPROFILE%\.local\bin` 新增到您的使用者 PATH 變數。重新啟動您的終端。

驗證修復是否有效：

```
claude --version
```

檢查衝突的安裝

多個 Claude Code 安裝可能導致版本不匹配或意外行為。檢查已安裝的內容：

macOS/Linux

列出在您的 PATH 中找到的所有 `claude` 二進位檔：

```
which -a claude
```

檢查原生安裝程式和 npm 版本是否存在：

```
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"
```

如果您找到多個安裝，只保留一個。建議使用 `~/.local/bin/claude` 的原生安裝。移除任何額外的安裝：

解除安裝 npm 全域安裝：

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

在 macOS 上移除 Homebrew 安裝：

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

檢查目錄權限

安裝程式需要對 `~/.local/bin/` 和 `~/.claude/` 的寫入存取權。如果安裝因權限錯誤而失敗，檢查這些目錄是否可寫：

```
test -w ~/.local/bin && echo "writable" || echo "not writable"
test -w ~/.claude && echo "writable" || echo "not writable"
```

如果任一目錄不可寫，建立安裝目錄並將您的使用者設定為擁有者：

```
sudo mkdir -p ~/.local/bin
sudo chown -R $(whoami) ~/.local
```

驗證二進位檔是否有效

如果 `claude` 已安裝但在啟動時崩潰或掛起，執行這些檢查以縮小原因範圍。

確認二進位檔存在且可執行：

```
ls -la $(which claude)
```

在 Linux 上，檢查遺失的共用程式庫。如果 `ldd` 顯示遺失的程式庫，您可能需要安裝系統套件。在 Alpine Linux 和其他基於 musl 的發行版上，請參閱 Alpine Linux 設定。

```
ldd $(which claude) | grep "not found"
```

執行快速健全性檢查，確認二進位檔可以執行：

```
claude --version
```

常見安裝問題

這些是最常遇到的安裝問題及其解決方案。

安裝指令碼傳回 HTML 而不是 shell 指令碼

執行安裝命令時，您可能會看到以下其中一個錯誤：

```
bash: line 1: syntax error near unexpected token `<'
bash: line 1: `<!DOCTYPE html>'
```

在 PowerShell 上，同樣的問題顯示為：

```
Invoke-Expression: Missing argument in parameter list.
```

這表示安裝 URL 傳回了 HTML 頁面而不是安裝指令碼。如果 HTML 頁面顯示「App unavailable in region」，Claude Code 在您的國家/地區不可用。請參閱支援的國家/地區。

否則，這可能由於網路問題、區域路由或臨時服務中斷而發生。

解決方案：

1. **使用替代安裝方法**：
   在 macOS 或 Linux 上，透過 Homebrew 安裝：

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

在 Windows 上，透過 WinGet 安裝：

```
winget install Anthropic.ClaudeCode
```

1. **幾分鐘後重試**：此問題通常是暫時的。等待並重試原始命令。

安裝後 `command not found: claude`

安裝完成但 `claude` 無法運作。確切的錯誤因平台而異：

平台	錯誤訊息
macOS	`zsh: command not found: claude`
Linux	`bash: claude: command not found`
Windows CMD	`'claude' is not recognized as an internal or external command`
PowerShell	`claude : The term 'claude' is not recognized as the name of a cmdlet`

這表示安裝目錄不在您的 shell 搜尋路徑中。請參閱驗證您的 PATH 以取得每個平台上的修復。

`curl: (56) Failure writing output to destination`

`curl ... | bash` 命令下載指令碼並使用管道 (`|`) 將其直接傳遞給 Bash 執行。此錯誤表示連線在指令碼完成下載前中斷。常見原因包括網路中斷、下載被中途阻止或系統資源限制。

解決方案：

1. **檢查網路穩定性**：Claude Code 二進位檔託管在 Google Cloud Storage 上。測試您是否可以連接到它：

```
curl -fsSL https://storage.googleapis.com -o /dev/null
```

如果命令無聲地完成，您的連線沒問題，問題可能是間歇性的。重試安裝命令。如果您看到錯誤，您的網路可能阻止了下載。

1. **嘗試替代安裝方法**：
   在 macOS 或 Linux 上：

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

在 Windows 上：

```
winget install Anthropic.ClaudeCode
```

TLS 或 SSL 連線錯誤

像 `curl: (35) TLS connect error`、`schannel: next InitializeSecurityContext failed` 或 PowerShell 的 `Could not establish trust relationship for the SSL/TLS secure channel` 這樣的錯誤表示 TLS 握手失敗。

解決方案：

1. **更新您的系統 CA 憑證**：
   在 Ubuntu/Debian 上：

```
sudo apt-get update && sudo apt-get install ca-certificates
```

在 macOS 上透過 Homebrew：

```
brew install ca-certificates
```

1. **在 Windows 上，在執行安裝程式前在 PowerShell 中啟用 TLS 1.2**：

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
irm https://claude.ai/install.ps1 | iex
```

1. **檢查代理或防火牆干擾**：執行 TLS 檢查的公司代理可能導致這些錯誤，包括 `unable to get local issuer certificate`。將 `NODE_EXTRA_CA_CERTS` 設定為您的公司 CA 憑證套件：

```
export NODE_EXTRA_CA_CERTS=/path/to/corporate-ca.pem
```

如果您沒有憑證檔案，請詢問您的 IT 團隊。您也可以嘗試直接連線以確認代理是原因。

`Failed to fetch version from storage.googleapis.com`

安裝程式無法連接到下載伺服器。這通常表示 `storage.googleapis.com` 在您的網路上被阻止。

解決方案：

1. **直接測試連線**：

```
curl -sI https://storage.googleapis.com
```

1. **如果在代理後面**，設定 `HTTPS_PROXY` 以便安裝程式可以透過它路由。請參閱代理設定以取得詳細資訊。

```
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

1. **如果在受限網路上**，嘗試不同的網路或 VPN，或使用替代安裝方法：
   在 macOS 或 Linux 上：

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

在 Windows 上：

```
winget install Anthropic.ClaudeCode
```

Windows：`irm` 或 `&&` 無法識別

如果您看到 `'irm' is not recognized` 或 `The token '&&' is not valid`，您執行的是錯誤的 shell 命令。

- **`irm` 無法識別**：您在 CMD 中，而不是 PowerShell。您有兩個選項：
  透過在開始功能表中搜尋「PowerShell」來開啟 PowerShell，然後執行原始安裝命令：

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

或留在 CMD 中並改用 CMD 安裝程式：

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

- `&&` **無效**：您在 PowerShell 中但執行了 CMD 安裝程式命令。使用 PowerShell 安裝程式：

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

低記憶體 Linux 伺服器上安裝被終止

如果您在 VPS 或雲端執行個體上的安裝期間看到 `Killed`：

```
Setting up Claude Code...
Installing Claude Code native build latest...
bash: line 142: 34803 Killed    "$binary_path" install ${TARGET:+"$TARGET"}
```

Linux OOM 殺手終止了該程序，因為系統記憶體不足。Claude Code 需要至少 4 GB 的可用 RAM。

解決方案：

1. **新增交換空間**（如果您的伺服器 RAM 有限）。交換使用磁碟空間作為溢出記憶體，讓安裝即使在低物理 RAM 的情況下也能完成。
   建立 2 GB 交換檔案並啟用它：

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```

然後重試安裝：

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

1. **關閉其他程序**以在安裝前釋放記憶體。
2. **使用更大的執行個體**（如果可能）。Claude Code 需要至少 4 GB 的 RAM。

Docker 中安裝掛起

在 Docker 容器中安裝 Claude Code 時，以 root 身份安裝到 `/` 可能導致掛起。

解決方案：

1. **在執行安裝程式前設定工作目錄**。從 `/` 執行時，安裝程式掃描整個檔案系統，導致過度的記憶體使用。設定 `WORKDIR` 將掃描限制在小目錄：

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

1. **增加 Docker 記憶體限制**（如果使用 Docker Desktop）：

```
docker build --memory=4g .
```

Windows：Claude Desktop 覆蓋 `claude` CLI 命令

如果您安裝了舊版本的 Claude Desktop，它可能在 `WindowsApps` 目錄中註冊一個 `Claude.exe`，其 PATH 優先級高於 Claude Code CLI。執行 `claude` 會開啟 Desktop 應用程式而不是 CLI。

更新 Claude Desktop 到最新版本以修復此問題。

Windows：「Claude Code on Windows requires git-bash」

Windows 上的 Claude Code 需要 Git for Windows，其中包括 Git Bash。

如果未安裝 Git，從 git-scm.com/downloads/win 下載並安裝它。在設定期間，選擇「Add to PATH」。安裝後重新啟動您的終端。

如果 Git 已安裝但 Claude Code 仍無法找到它，在您的 settings.json 檔案中設定路徑：

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

如果您的 Git 安裝在其他地方，透過在 PowerShell 中執行 `where.exe git` 找到路徑，並使用該目錄中的 `bin\bash.exe` 路徑。

Linux：安裝了錯誤的二進位變體（musl/glibc 不匹配）

如果在安裝後看到有關遺失共用程式庫的錯誤，如 `libstdc++.so.6` 或 `libgcc_s.so.1`，安裝程式可能為您的系統下載了錯誤的二進位變體。

```
Error loading shared library libstdc++.so.6: No such file or directory
```

這可能發生在已安裝 musl 交叉編譯套件的基於 glibc 的系統上，導致安裝程式誤偵測系統為 musl。

解決方案：

1. **檢查您的系統使用哪個 libc**：

```
ldd /bin/ls | head -1
```

如果它顯示 `linux-vdso.so` 或對 `/lib/x86_64-linux-gnu/` 的參考，您在 glibc 上。如果它顯示 `musl`，您在 musl 上。

1. **如果您在 glibc 上但得到了 musl 二進位檔**，移除安裝並重新安裝。您也可以從 GCS 儲存桶 `https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json` 手動下載正確的二進位檔。使用 `ldd /bin/ls` 和 `ls /lib/libc.musl*` 的輸出提交 GitHub 問題。
2. **如果您實際上在 musl 上**（Alpine Linux），安裝所需的套件：

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

Linux 上的 `Illegal instruction`

如果安裝程式列印 `Illegal instruction` 而不是 OOM `Killed` 訊息，下載的二進位檔與您的 CPU 架構不匹配。這通常發生在接收 x86 二進位檔的 ARM 伺服器上，或在缺少所需指令集的較舊 CPU 上。

```
bash: line 142: 2238232 Illegal instruction     "$binary_path" install ${TARGET:
+"$TARGET"}
```

解決方案：

1. **驗證您的架構**：

```
uname -m
```

`x86_64` 表示 64 位 Intel/AMD，`aarch64` 表示 ARM64。如果二進位檔不匹配，提交 GitHub 問題並附上輸出。

1. **嘗試替代安裝方法**，同時解決架構問題：

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

macOS 上的 `dyld: cannot load`

如果在安裝期間看到 `dyld: cannot load` 或 `Abort trap: 6`，二進位檔與您的 macOS 版本或硬體不相容。

```
dyld: cannot load 'claude-2.1.42-darwin-x64' (load command 0x80000034 is unknown)
Abort trap: 6
```

解決方案：

1. **檢查您的 macOS 版本**：Claude Code 需要 macOS 13.0 或更新版本。開啟 Apple 功能表並選擇「About This Mac」以檢查您的版本。
2. **更新 macOS**（如果您在較舊版本上）。二進位檔使用較舊 macOS 版本不支援的載入命令。
3. **嘗試 Homebrew** 作為替代安裝方法：

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

Windows 安裝問題：WSL 中的錯誤

您可能在 WSL 中遇到以下問題：

OS/平台偵測問題：如果您在安裝期間收到錯誤，WSL 可能使用 Windows `npm`。嘗試：

- 在安裝前執行 `npm config set os linux`
- 使用 `npm install -g @anthropic-ai/claude-code --force --no-os-check` 安裝。不要使用 `sudo`。

找不到 Node 錯誤：如果執行 `claude` 時看到 `exec: node: not found`，您的 WSL 環境可能使用 Windows 安裝的 Node.js。您可以使用 `which npm` 和 `which node` 確認這一點，它們應該指向以 `/usr/` 開頭的 Linux 路徑，而不是 `/mnt/c/`。要修復此問題，請嘗試透過您的 Linux 發行版的套件管理器或透過 `nvm` 安裝 Node。

nvm 版本衝突：如果您在 WSL 和 Windows 中都安裝了 nvm，在 WSL 中切換 Node 版本時可能會遇到版本衝突。這是因為 WSL 預設匯入 Windows PATH，導致 Windows nvm/npm 優先於 WSL 安裝。

您可以透過以下方式識別此問題：

- 執行 `which npm` 和 `which node` - 如果它們指向 Windows 路徑（以 `/mnt/c/` 開頭），則使用 Windows 版本
- 在 WSL 中使用 nvm 切換 Node 版本後遇到損壞的功能

要解決此問題，修復您的 Linux PATH 以確保 Linux node/npm 版本優先：

主要解決方案：確保 nvm 在您的 shell 中正確載入

最常見的原因是 nvm 未在非互動式 shell 中載入。將以下內容新增到您的 shell 設定檔（`~/.bashrc`、`~/.zshrc` 等）：

```
## 如果存在，載入 nvm
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

或在您的目前工作階段中直接執行：

```
source ~/.nvm/nvm.sh
```

替代方案：調整 PATH 順序

如果 nvm 正確載入但 Windows 路徑仍優先，您可以在 shell 設定中明確將 Linux 路徑前置到 PATH：

```
export PATH="$HOME/.nvm/versions/node/$(node -v)/bin:$PATH"
```

> **Warning:**
> 避免透過 `appendWindowsPath = false` 停用 Windows PATH 匯入，因為這會破壞從 WSL 呼叫 Windows 可執行檔的能力。同樣，如果您為 Windows 開發使用 Node.js，請避免從 Windows 解除安裝它。

WSL2 sandbox 設定

Sandboxing 在 WSL2 上受支援，但需要安裝其他套件。如果執行 `/sandbox` 時看到「Sandbox requires socat and bubblewrap」之類的錯誤，請安裝依賴項：

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

WSL1 不支援 sandboxing。如果您看到「Sandboxing requires WSL2」，您需要升級到 WSL2 或執行 Claude Code 而不進行 sandboxing。

安裝期間的權限錯誤

如果原生安裝程式因權限錯誤而失敗，目標目錄可能不可寫。請參閱檢查目錄權限。

如果您之前使用 npm 安裝並遇到 npm 特定的權限錯誤，請切換到原生安裝程式：

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

權限和驗證

這些部分涉及登入失敗、令牌問題和權限提示行為。

重複的權限提示

如果您發現自己重複批准相同的命令，您可以使用 `/permissions` 命令允許特定工具無需批准即可執行。請參閱權限文件。

驗證問題

如果您遇到驗證問題：

1. 執行 `/logout` 以完全登出
2. 關閉 Claude Code
3. 使用 `claude` 重新啟動並再次完成驗證程序

如果瀏覽器在登入期間未自動開啟，按 `c` 將 OAuth URL 複製到您的剪貼簿，然後手動將其貼到您的瀏覽器中。

OAuth 錯誤：無效代碼

如果您看到 `OAuth error: Invalid code. Please make sure the full code was copied`，登入代碼已過期或在複製貼上期間被截斷。

解決方案：

- 在瀏覽器開啟後按 Enter 重試並快速完成登入
- 輸入 `c` 以複製完整 URL（如果瀏覽器未自動開啟）
- 如果使用遠端/SSH 工作階段，瀏覽器可能在錯誤的機器上開啟。複製終端中顯示的 URL 並在您的本機瀏覽器中開啟它。

登入後 403 Forbidden

如果登入後看到 `API Error: 403 {"error":{"type":"forbidden","message":"Request not allowed"}}`：

- **Claude Pro/Max 使用者**：在 claude.ai/settings 驗證您的訂閱是否有效
- **Console 使用者**：確認您的帳戶已由您的管理員指派「Claude Code」或「Developer」角色
- **在代理後面**：公司代理可能干擾 API 請求。請參閱網路設定以取得代理設定。

OAuth 登入在 WSL2 中失敗

如果 WSL 無法開啟您的 Windows 瀏覽器，WSL2 中基於瀏覽器的登入可能失敗。設定 `BROWSER` 環境變數：

```
export BROWSER="/mnt/c/Program Files/Google/Chrome/Application/chrome.exe"
claude
```

或手動複製 URL：當登入提示出現時，按 `c` 複製 OAuth URL，然後將其貼到您的 Windows 瀏覽器中。

「未登入」或令牌已過期

如果 Claude Code 在工作階段後提示您再次登入，您的 OAuth 令牌可能已過期。

執行 `/login` 以重新驗證。如果這種情況經常發生，請檢查您的系統時鐘是否準確，因為令牌驗證取決於正確的時間戳。

設定檔位置

Claude Code 在多個位置儲存設定：

檔案	用途
`~/.claude/settings.json`	使用者設定（權限、hooks、模型覆蓋）
`.claude/settings.json`	專案設定（簽入原始碼控制）
`.claude/settings.local.json`	本機專案設定（未提交）
`~/.claude.json`	全域狀態（主題、OAuth、MCP 伺服器）
`.mcp.json`	專案 MCP 伺服器（簽入原始碼控制）
`managed-mcp.json`	受管 MCP 伺服器
受管設定	受管設定（伺服器管理、MDM/OS 層級原則或檔案型）

在 Windows 上，`~` 指您的使用者主目錄，例如 `C:\Users\YourName`。

有關設定這些檔案的詳細資訊，請參閱設定和 MCP。

重設設定

要將 Claude Code 重設為預設設定，您可以移除設定檔：

```
## 重設所有使用者設定和狀態
rm ~/.claude.json
rm -rf ~/.claude/

## 重設專案特定設定
rm -rf .claude/
rm .mcp.json
```

> **Warning:**
> 這將移除您的所有設定、MCP 伺服器設定和工作階段歷史記錄。

效能和穩定性

這些部分涵蓋與資源使用、回應性和搜尋行為相關的問題。

高 CPU 或記憶體使用率

Claude Code 設計用於與大多數開發環境搭配使用，但在處理大型程式碼庫時可能消耗大量資源。如果您遇到效能問題：

1. 定期使用 `/compact` 以減少上下文大小
2. 在主要任務之間關閉並重新啟動 Claude Code
3. 考慮將大型建置目錄新增到您的 `.gitignore` 檔案

命令掛起或凍結

如果 Claude Code 似乎無回應：

1. 按 Ctrl+C 嘗試取消目前操作
2. 如果無回應，您可能需要關閉終端並重新啟動

搜尋和探索問題

如果搜尋工具、`@file` 提及、自訂代理和自訂 skills 無法運作，請安裝系統 `ripgrep`：

```
## macOS (Homebrew)
brew install ripgrep

## Windows (winget)
winget install BurntSushi.ripgrep.MSVC

## Ubuntu/Debian
sudo apt install ripgrep

## Alpine Linux
apk add ripgrep

## Arch Linux
pacman -S ripgrep
```

然後在您的環境中設定 `USE_BUILTIN_RIPGREP=0`。

WSL 上的搜尋速度緩慢或結果不完整

在 WSL 上跨檔案系統工作時的磁碟讀取效能損失可能導致在 WSL 上使用 Claude Code 時搜尋結果少於預期。搜尋仍然有效，但傳回的結果少於原生檔案系統。

> **Note:**
>
> 在這種情況下，`/doctor` 將顯示搜尋為正常。

解決方案：

1. **提交更具體的搜尋**：透過指定目錄或檔案類型來減少搜尋的檔案數量：「在 auth-service 套件中搜尋 JWT 驗證邏輯」或「在 JS 檔案中尋找 md5 雜湊的使用」。
2. **將專案移到 Linux 檔案系統**：如果可能，確保您的專案位於 Linux 檔案系統（`/home/`）而不是 Windows 檔案系統（`/mnt/c/`）。
3. **改用原生 Windows**：考慮在 Windows 上原生執行 Claude Code 而不是透過 WSL，以獲得更好的檔案系統效能。

IDE 整合問題

如果 Claude Code 未連接到您的 IDE 或在 IDE 終端中行為異常，請嘗試以下解決方案。

WSL2 上未偵測到 JetBrains IDE

如果您在 WSL2 上使用 Claude Code 搭配 JetBrains IDE 並收到「No available IDEs detected」錯誤，這可能是由於 WSL2 的網路設定或 Windows 防火牆阻止連線。

WSL2 網路模式

WSL2 預設使用 NAT 網路，這可能會阻止 IDE 偵測。您有兩個選項：

選項 1：設定 Windows 防火牆（建議）

1. 找到您的 WSL2 IP 位址：

```
wsl hostname -I
## 範例輸出：172.21.123.45
```

1. 以管理員身份開啟 PowerShell 並建立防火牆規則：

```
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
```

根據步驟 1 中的 WSL2 子網調整 IP 範圍。

1. 重新啟動您的 IDE 和 Claude Code

選項 2：切換到鏡像網路

在您的 Windows 使用者目錄中新增到 `.wslconfig`：

```
[wsl2]
networkingMode=mirrored
```

然後從 PowerShell 使用 `wsl --shutdown` 重新啟動 WSL。

> **Note:**
>
> 這些網路問題僅影響 WSL2。WSL1 直接使用主機的網路，不需要這些設定。

有關其他 JetBrains 設定提示，請參閱 JetBrains IDE 指南。

報告 Windows IDE 整合問題

如果您在 Windows 上遇到 IDE 整合問題，建立問題並提供以下資訊：

- 環境類型：原生 Windows (Git Bash) 或 WSL1/WSL2
- WSL 網路模式（如適用）：NAT 或鏡像
- IDE 名稱和版本
- Claude Code 擴充功能/外掛程式版本
- Shell 類型：Bash、Zsh、PowerShell 等

JetBrains IDE 終端中的 Escape 鍵無法運作

如果您在 JetBrains 終端中使用 Claude Code 且 `Esc` 鍵無法如預期中斷代理，這可能是由於 JetBrains 預設快捷鍵的衝突。

要修復此問題：

1. 前往設定 → 工具 → 終端
2. 任一：
   - 取消勾選「Move focus to the editor with Escape」，或
   - 按一下「Configure terminal keybindings」並刪除「Switch focus to Editor」快捷鍵
3. 套用變更

這允許 `Esc` 鍵正確中斷 Claude Code 操作。

Markdown 格式化問題

Claude Code 有時會產生 markdown 檔案，其程式碼圍欄上缺少語言標籤，這可能影響 GitHub、編輯器和文件工具中的語法突出顯示和可讀性。

程式碼區塊中缺少語言標籤

如果您在產生的 markdown 中注意到像這樣的程式碼區塊：

````
```
function example() {
  return "hello";
}
```text
````

而不是像這樣的正確標籤區塊：

````
```javascript
function example() {
  return "hello";
}
```text
````

解決方案：

1. **要求 Claude 新增語言標籤**：要求「Add appropriate language tags to all code blocks in this markdown file」。
2. **使用後處理 hooks**：設定自動格式化 hooks 以偵測並新增遺失的語言標籤。請參閱編輯後自動格式化程式碼以取得 PostToolUse 格式化 hook 的範例。
3. **手動驗證**：產生 markdown 檔案後，檢查它們是否有正確的程式碼區塊格式化，如果需要，請要求更正。

不一致的間距和格式化

如果產生的 markdown 有過多的空白行或不一致的間距：

解決方案：

1. **要求格式化更正**：要求 Claude「Fix spacing and formatting issues in this markdown file」。
2. **使用格式化工具**：設定 hooks 以在產生的 markdown 檔案上執行 markdown 格式化程式（如 `prettier`）或自訂格式化指令碼。
3. **指定格式化偏好**：在您的提示或專案記憶體檔案中包含格式化要求。

減少 markdown 格式化問題

要最小化格式化問題：

- **在請求中明確**：要求「properly formatted markdown with language-tagged code blocks」
- **使用專案慣例**：在 `CLAUDE.md` 中記錄您偏好的 markdown 風格
- **設定驗證 hooks**：使用後處理 hooks 自動驗證和修復常見格式化問題

取得更多幫助

如果您遇到此處未涵蓋的問題：

1. 在 Claude Code 中使用 `/bug` 命令直接向 Anthropic 報告問題
2. 檢查 GitHub 儲存庫以了解已知問題

3. 執行 `/doctor` 以診斷問題。它檢查：
   - 安裝類型、版本和搜尋功能
   - 自動更新狀態和可用版本
   - 無效的設定檔（格式不正確的 JSON、不正確的類型）
   - MCP 伺服器設定錯誤
   - 快捷鍵設定問題
   - 上下文使用警告（大型 CLAUDE.md 檔案、高 MCP 令牌使用、無法連接的權限規則）
   - 外掛程式和代理載入錯誤
4. 直接詢問 Claude 其功能和特性 - Claude 內建存取其文件