Claude Code

Documentazione Completa

By Anthropic

Data di produzione：2026-03-17

Uno strumento di codifica agentico nel tuo terminale,
che comprende il tuo codice e ti aiuta a programmare più velocemente.

Compilato da code.claude.com/docs/it
Per l'ultima versione, visita il sito web.

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

Table of Contents

Part 1: Getting Started

Panoramica di Claude Code

> *Claude Code è uno strumento di codifica agentivo che legge la tua base di codice, modifica i file, esegue comandi e si integra con i tuoi strumenti di sviluppo. Disponibile nel tuo terminale, IDE, app desktop e browser.*

Claude Code è un assistente di codifica alimentato da IA che ti aiuta a costruire funzionalità, correggere bug e automatizzare attività di sviluppo. Comprende l'intera tua base di codice e può lavorare su più file e strumenti per portare a termine le cose.

Inizia

Scegli il tuo ambiente per iniziare. La maggior parte delle superfici richiede un abbonamento Claude o un account Anthropic Console. Il Terminal CLI e VS Code supportano anche provider di terze parti.

Terminal

La CLI completa per lavorare con Claude Code direttamente nel tuo terminale. Modifica file, esegui comandi e gestisci l'intero progetto dalla riga di comando.

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.

Quindi avvia Claude Code in qualsiasi progetto:

```
cd your-project
claude
```

Ti verrà chiesto di accedere al primo utilizzo. È tutto! Continua con la Guida rapida →

> **Tip:**

> Vedi configurazione avanzata per le opzioni di installazione, gli aggiornamenti manuali o le istruzioni di disinstallazione. Visita risoluzione dei problemi se riscontri problemi.

VS Code

L'estensione VS Code fornisce diff inline, @-mention, revisione del piano e cronologia delle conversazioni direttamente nel tuo editor.

- Installa per VS Code
- Installa per Cursor

Oppure cerca "Claude Code" nella visualizzazione Estensioni (`Cmd+Shift+X` su Mac, `Ctrl+Shift+X` su Windows/Linux). Dopo l'installazione, apri il Riquadro comandi (`Cmd+Shift+P` / `Ctrl+Shift+P`), digita "Claude Code" e seleziona **Apri in nuova scheda**.

Inizia con VS Code →

App desktop

Un'app standalone per eseguire Claude Code al di fuori del tuo IDE o terminale. Rivedi i diff visivamente, esegui più sessioni affiancate, pianifica attività ricorrenti e avvia sessioni cloud.

Scarica e installa:

- macOS (Intel e Apple Silicon)
- Windows (x64)
- Windows ARM64 (solo sessioni remote)

Dopo l'installazione, avvia Claude, accedi e fai clic sulla scheda **Code** per iniziare a codificare. È richiesto un abbonamento a pagamento.

Scopri di più sull'app desktop →

Web

Esegui Claude Code nel tuo browser senza configurazione locale. Avvia attività a lunga esecuzione e torna quando sono completate, lavora su repository che non hai localmente o esegui più attività in parallelo. Disponibile su browser desktop e sull'app Claude iOS.

Inizia a codificare su claude.ai/code.

Inizia sul web →

JetBrains

Un plugin per IntelliJ IDEA, PyCharm, WebStorm e altri IDE JetBrains con visualizzazione diff interattiva e condivisione del contesto di selezione.

Installa il plugin Claude Code dal JetBrains Marketplace e riavvia il tuo IDE.

Inizia con JetBrains →

Cosa puoi fare

Ecco alcuni dei modi in cui puoi utilizzare Claude Code:

Claude Code gestisce i compiti noiosi che consumano la tua giornata: scrivere test per il codice non testato, correggere errori di lint in un progetto, risolvere conflitti di merge, aggiornare le dipendenze e scrivere note di rilascio.

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

Descrivi quello che vuoi in linguaggio naturale. Claude Code pianifica l'approccio, scrive il codice su più file e verifica che funzioni.

Per i bug, incolla un messaggio di errore o descrivi il sintomo. Claude Code traccia il problema attraverso la tua base di codice, identifica la causa principale e implementa una correzione. Vedi flussi di lavoro comuni per altri esempi.

Claude Code funziona direttamente con git. Mette in stage le modifiche, scrive messaggi di commit, crea branch e apre pull request.

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

In CI, puoi automatizzare la revisione del codice e il triage dei problemi con GitHub Actions o GitLab CI/CD.

Il Model Context Protocol (MCP) è uno standard aperto per connettere gli strumenti di IA alle fonti di dati esterne. Con MCP, Claude Code può leggere i tuoi documenti di progettazione in Google Drive, aggiornare i ticket in Jira, estrarre dati da Slack o utilizzare i tuoi strumenti personalizzati.

`CLAUDE.md` è un file markdown che aggiungi alla radice del tuo progetto che Claude Code legge all'inizio di ogni sessione. Usalo per impostare standard di codifica, decisioni architettoniche, librerie preferite e liste di controllo di revisione. Claude costruisce anche memoria automatica mentre lavora, salvando insegnamenti come comandi di build e intuizioni di debug tra le sessioni senza che tu debba scrivere nulla.

Crea comandi personalizzati per pacchettizzare flussi di lavoro ripetibili che il tuo team può condividere, come `/review-pr` o `/deploy-staging`.

Hooks ti permettono di eseguire comandi shell prima o dopo le azioni di Claude Code, come la formattazione automatica dopo ogni modifica di file o l'esecuzione di lint prima di un commit.

Genera più agenti Claude Code che lavorano su diverse parti di un'attività contemporaneamente. Un agente principale coordina il lavoro, assegna sottoattività e unisce i risultati.

Per flussi di lavoro completamente personalizzati, l'Agent SDK ti permette di costruire i tuoi agenti alimentati dagli strumenti e dalle capacità di Claude Code, con controllo completo sull'orchestrazione, l'accesso agli strumenti e i permessi.

Claude Code è componibile e segue la filosofia Unix. Pipe i log in esso, eseguilo in CI o concatenalo con altri strumenti:

```
## 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"
```

Vedi il riferimento CLI per l'insieme completo di comandi e flag.

Le sessioni non sono legate a una singola superficie. Sposta il lavoro tra gli ambienti mentre il tuo contesto cambia:

- Allontanati dalla tua scrivania e continua a lavorare dal tuo telefono o da qualsiasi browser con Controllo remoto
- Avvia un'attività a lunga esecuzione sul web o app iOS, quindi trascinala nel tuo terminale con `/teleport`
- Trasferisci una sessione di terminale all'app Desktop con `/desktop` per la revisione visiva dei diff
- Instrada le attività dalla chat del team: menziona `@Claude` in Slack con una segnalazione di bug e ottieni una pull request in cambio

Usa Claude Code ovunque

Ogni superficie si connette allo stesso motore Claude Code sottostante, quindi i tuoi file CLAUDE.md, le impostazioni e i server MCP funzionano su tutti loro.

Oltre agli ambienti Terminal, VS Code, JetBrains, Desktop e Web sopra, Claude Code si integra con flussi di lavoro CI/CD, chat e browser:

Voglio...	Opzione migliore
Continuare una sessione locale dal mio telefono o da un altro dispositivo	Controllo remoto
Avviare un'attività localmente, continuare su mobile	Web o app Claude iOS
Automatizzare le revisioni PR e il triage dei problemi	GitHub Actions o GitLab CI/CD
Ottenere revisione automatica del codice su ogni PR	GitHub Code Review
Instradare le segnalazioni di bug da Slack alle pull request	Slack
Eseguire il debug di applicazioni web live	Chrome
Costruire agenti personalizzati per i tuoi flussi di lavoro	Agent SDK

Passaggi successivi

Una volta installato Claude Code, queste guide ti aiutano ad approfondire.

- Guida rapida: percorri il tuo primo vero compito, dall'esplorazione di una base di codice al commit di una correzione
- Archivia istruzioni e memorie: dai a Claude istruzioni persistenti con file CLAUDE.md e memoria automatica
- Flussi di lavoro comuni e best practice: modelli per ottenere il massimo da Claude Code
- Impostazioni: personalizza Claude Code per il tuo flusso di lavoro
- Risoluzione dei problemi: soluzioni per i problemi comuni

- code.claude.com: demo, prezzi e dettagli del prodotto

Guida rapida

> *Benvenuto in Claude Code!*

Questa guida rapida ti permetterà di utilizzare l'assistenza alla codifica basata su IA in pochi minuti. Alla fine, comprenderai come utilizzare Claude Code per le attività di sviluppo comuni.

Prima di iniziare

Assicurati di avere:

- Un terminale o un prompt dei comandi aperto
  - Se non hai mai utilizzato il terminale prima, consulta la guida del terminale
- Un progetto di codice con cui lavorare
- Un abbonamento Claude (Pro, Max, Teams o Enterprise), un account Claude Console o accesso tramite un provider cloud supportato

> **Note:**
>
> Questa guida copre il CLI del terminale. Claude Code è disponibile anche sul web, come app desktop, in VS Code e IDE JetBrains, in Slack e in CI/CD con GitHub Actions e GitLab. Vedi tutte le interfacce.

Passaggio 1: Installa 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.

Passaggio 2: Accedi al tuo account

Claude Code richiede un account per essere utilizzato. Quando avvii una sessione interattiva con il comando `claude`, dovrai effettuare l'accesso:

```
claude
## Ti verrà richiesto di accedere al primo utilizzo
```

```
/login
## Segui i prompt per accedere con il tuo account
```

Puoi accedere utilizzando uno di questi tipi di account:

- Claude Pro, Max, Teams o Enterprise (consigliato)
- Claude Console (accesso API con crediti prepagati). Al primo accesso, uno spazio di lavoro "Claude Code" viene creato automaticamente nella Console per il tracciamento centralizzato dei costi.
- Amazon Bedrock, Google Vertex AI o Microsoft Foundry (provider cloud aziendali)

Una volta effettuato l'accesso, le tue credenziali vengono archiviate e non dovrai accedere di nuovo. Per cambiare account in seguito, utilizza il comando `/login`.

Passaggio 3: Avvia la tua prima sessione

Apri il tuo terminale in qualsiasi directory del progetto e avvia Claude Code:

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

Vedrai la schermata di benvenuto di Claude Code con le informazioni della tua sessione, le conversazioni recenti e gli ultimi aggiornamenti. Digita `/help` per i comandi disponibili o `/resume` per continuare una conversazione precedente.

> **Tip:**
>
> Dopo aver effettuato l'accesso (Passaggio 2), le tue credenziali vengono archiviate nel tuo sistema. Scopri di più in Gestione delle credenziali.

Passaggio 4: Fai la tua prima domanda

Iniziamo con la comprensione della tua base di codice. Prova uno di questi comandi:

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

Claude analizzerà i tuoi file e fornirà un riepilogo. Puoi anche fare domande più specifiche:

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

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

```
explain the folder structure
```

Puoi anche chiedere a Claude informazioni sulle sue stesse capacità:

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

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

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

> **Note:**
>
> Claude Code legge i file del tuo progetto secondo le necessità. Non devi aggiungere manualmente il contesto.

Passaggio 5: Fai il tuo primo cambio di codice

Ora facciamo in modo che Claude Code faccia un po' di codifica vera. Prova un'attività semplice:

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

Claude Code farà:

1. Trovare il file appropriato

2. Mostrarti le modifiche proposte
3. Chiedere la tua approvazione
4. Effettuare la modifica

> **Note:**
>
> Claude Code chiede sempre il permesso prima di modificare i file. Puoi approvare le singole modifiche o abilitare la modalità “Accetta tutto” per una sessione.

Passaggio 6: Usa Git con Claude Code

Claude Code rende le operazioni Git conversazionali:

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

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

Puoi anche richiedere operazioni Git più complesse:

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

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

```
help me resolve merge conflicts
```

Passaggio 7: Correggi un bug o aggiungi una funzionalità

Claude è abile nel debug e nell’implementazione di funzionalità.

Descrivi quello che vuoi in linguaggio naturale:

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

O correggi i problemi esistenti:

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

Claude Code farà:

- Individuare il codice rilevante
- Comprendere il contesto
- Implementare una soluzione
- Eseguire i test se disponibili

Passaggio 8: Prova altri flussi di lavoro comuni

Ci sono diversi modi per lavorare con Claude:

Refactoring del codice

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

Scrivi test

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

Aggiorna la documentazione

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

Revisione del codice

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

> **Tip:**
>
> Parla a Claude come faresti con un collega disponibile. Descrivi quello che vuoi ottenere e ti aiuterà a raggiungerlo.

Comandi essenziali

Ecco i comandi più importanti per l'uso quotidiano:

Comando	Cosa fa	Esempio
`claude`	Avvia la modalità interattiva	`claude`
`claude "task"`	Esegui un'attività una tantum	`claude "fix the build error"`
`claude -p "query"`	Esegui una query una tantum, quindi esci	`claude -p "explain this function"`
`claude -c`	Continua la conversazione più recente nella directory corrente	`claude -c`
`claude -r`	Riprendi una conversazione precedente	`claude -r`
`claude commit`	Crea un commit Git	`claude commit`
`/clear`	Cancella la cronologia delle conversazioni	`/clear`
`/help`	Mostra i comandi disponibili	`/help`
`exit` o Ctrl+C	Esci da Claude Code	`exit`

Vedi il riferimento CLI per un elenco completo dei comandi.

Suggerimenti professionali per i principianti

Per ulteriori informazioni, vedi best practices e flussi di lavoro comuni.

Invece di: "fix the bug"

Prova: "fix the login bug where users see a blank screen after entering wrong credentials"

Suddividi i compiti complessi in passaggi:

```
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
```

Prima di apportare modifiche, lascia che Claude comprenda il tuo codice:

```
analyze the database schema
```

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

- Premi `?` per vedere tutte le scorciatoie da tastiera disponibili
- Usa Tab per il completamento dei comandi
- Premi ↑ per la cronologia dei comandi
- Digita `/` per vedere tutti i comandi e le skills

Cosa fare dopo?

Ora che hai imparato le nozioni di base, esplora funzionalità più avanzate:

- Come funziona Claude Code Comprendi il loop agentico, gli strumenti integrati e come Claude Code interagisce con il tuo progetto
- Best practices Ottieni risultati migliori con prompt efficaci e configurazione del progetto
- Flussi di lavoro comuni Guide passo dopo passo per attività comuni
- Estendi Claude Code Personalizza con CLAUDE.md, skills, hooks, MCP e altro

Ottenere aiuto

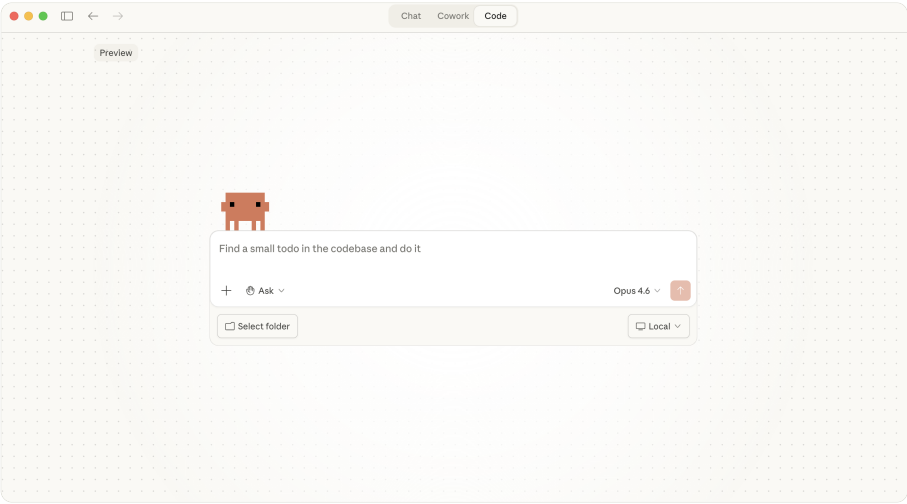
- **In Claude Code**: Digita `/help` o chiedi “how do I…”
- **Documentazione**: Sei qui! Sfoglia altre guide
- **Community**: Unisciti al nostro Discord per suggerimenti e supporto

Iniziare con l'app desktop

> *Installa Claude Code su desktop e avvia la tua prima sessione di codifica*

L'app desktop ti offre Claude Code con un'interfaccia grafica: revisione visiva dei diff, anteprima live dell'app, monitoraggio dei PR di GitHub con merge automatico, sessioni parallele con isolamento Git worktree, attività pianificate e la possibilità di eseguire attività in remoto. Non è richiesto il terminale.

Questa pagina illustra l'installazione dell'app e l'avvio della tua prima sessione. Se sei già configurato, consulta Usa Claude Code Desktop per il riferimento completo.



L'interfaccia di Claude Code Desktop che mostra la scheda Code selezionata, con una casella di prompt, il selettore della modalità di autorizzazione impostato su Chiedi autorizzazioni, il selettore del modello, il selettore della cartella e l'opzione Ambiente locale

L'app desktop ha tre schede:

- **Chat**: Conversazione generale senza accesso ai file, simile a claude.ai.
- **Cowork**: Un agente autonomo in background che lavora su attività in una VM cloud con il suo ambiente. Può funzionare indipendentemente mentre fai altro.
- **Code**: Un assistente di codifica interattivo con accesso diretto ai tuoi file locali. Rivedi e approvi ogni modifica in tempo reale.

Chat e Cowork sono trattati negli articoli di supporto di Claude Desktop. Questa pagina si concentra sulla scheda **Code**.

> **Note:**
>
> Claude Code richiede un abbonamento Pro, Max, Teams o Enterprise.

Installa

Step 1: Scarica l'app

Scarica Claude per la tua piattaforma.

- macOS Build universale per Intel e Apple Silicon
- Windows Per processori x64

Per Windows ARM64, scarica qui.

Linux non è attualmente supportato.

Step 2: Accedi

Avvia Claude dalla cartella Applicazioni (macOS) o dal menu Start (Windows). Accedi con il tuo account Anthropic.

Step 3: Apri la scheda Code

Fai clic sulla scheda **Code** al centro in alto. Se facendo clic su Code ti viene chiesto di eseguire l'upgrade, devi prima sottoscrivere un piano a pagamento. Se ti viene chiesto di accedere online, completa l'accesso e riavvia l'app. Se vedi un errore 403, consulta risoluzione dei problemi di autenticazione.

L'app desktop include Claude Code. Non è necessario installare Node.js o la CLI separatamente. Per utilizzare `claude` dal terminale, installa la CLI separatamente. Consulta Iniziare con la CLI.

Avvia la tua prima sessione

Con la scheda Code aperta, scegli un progetto e dai a Claude qualcosa da fare.

Step 1: Scegli un ambiente e una cartella

Seleziona **Local** per eseguire Claude sulla tua macchina utilizzando direttamente i tuoi file. Fai clic su **Select folder** e scegli la directory del tuo progetto.

> **Tip:**
>
> Inizia con un piccolo progetto che conosci bene. È il modo più veloce per vedere cosa può fare Claude Code. Su Windows, Git deve essere installato affinché le sessioni locali funzionino. La maggior parte dei Mac include Git per impostazione predefinita.

Puoi anche selezionare:

- **Remote**: Esegui sessioni sull'infrastruttura cloud di Anthropic che continuano anche se chiudi l'app. Le sessioni remote utilizzano la stessa infrastruttura di Claude Code sul web.
- **SSH**: Connettiti a una macchina remota tramite SSH (i tuoi server, VM cloud o dev container). Claude Code deve essere installato sulla macchina remota.

Step 2: Scegli un modello

Seleziona un modello dal menu a discesa accanto al pulsante di invio. Consulta modelli per un confronto tra Opus, Sonnet e Haiku. Non puoi cambiare il modello dopo l'avvio della sessione.

Step 3: Dì a Claude cosa fare

Digita cosa vuoi che Claude faccia:

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

Una sessione è una conversazione con Claude sul tuo codice. Ogni sessione traccia il suo proprio contesto e le sue modifiche, quindi puoi lavorare su più attività senza che si interferiscano a vicenda.

Step 4: Rivedi e accetta le modifiche

Per impostazione predefinita, la scheda Code inizia in modalità Chiedi autorizzazioni, dove Claude propone modifiche e attende la tua approvazione prima di applicarle. Vedrai:

1. Una visualizzazione diff che mostra esattamente cosa cambierà in ogni file
2. Pulsanti Accetta/Rifiuta per approvare o rifiutare ogni modifica
3. Aggiornamenti in tempo reale mentre Claude lavora sulla tua richiesta

Se rifiuti una modifica, Claude ti chiederà come vorresti procedere diversamente. I tuoi file non vengono modificati finché non accetti.

E adesso?

Hai fatto la tua prima modifica. Per il riferimento completo su tutto ciò che Desktop può fare, consulta Usa Claude Code Desktop. Ecco alcune cose da provare dopo.

Interrompi e guida. Puoi interrompere Claude in qualsiasi momento. Se sta andando nella direzione sbagliata, fai clic sul pulsante di arresto o digita la tua correzione e premi **Invio**. Claude smette di fare quello che sta facendo e si adatta in base al tuo input. Non devi aspettare che finisca o ricominciare da capo.

Dai a Claude più contesto. Digita `@filename` nella casella di prompt per inserire un file specifico nella conversazione, allega immagini e PDF utilizzando il pulsante di allegato, o trascina e rilascia i file direttamente nel prompt. Più contesto ha Claude, migliori sono i risultati. Consulta Aggiungi file e contesto.

Usa skills per attività ripetibili. Digita `/` o fai clic su **+** → **Slash commands** per sfogliare comandi incorporati, skills personalizzate e skills dei plugin. Le skills sono prompt riutilizzabili che puoi invocare quando ne hai bisogno, come liste di controllo per la revisione del codice o passaggi di distribuzione.

Rivedi le modifiche prima di eseguire il commit. Dopo che Claude modifica i file, appare un indicatore `+12 -1`. Fai clic su di esso per aprire la visualizzazione diff, rivedi le modifiche file per file e commenta righe specifiche. Claude legge i tuoi commenti e revisionali. Fai clic su **Review code** per far valutare a Claude i diff stessi e lasciare suggerimenti inline.

Regola quanto controllo hai. La tua modalità di autorizzazione controlla l'equilibrio. Chiedi autorizzazioni (predefinito) richiede approvazione prima di ogni modifica. Auto accept edits accetta automaticamente le modifiche ai file per un'iterazione più veloce. Plan mode consente a Claude di mappare un approccio senza toccare alcun file, il che è utile prima di un grande refactor.

Aggiungi plugin per più funzionalità. Fai clic sul pulsante **+** accanto alla casella di prompt e seleziona **Plugins** per sfogliare e installare plugin che aggiungono skills, agenti, MCP servers e altro.

Visualizza l'anteprima della tua app. Fai clic sul menu a discesa **Preview** per eseguire il tuo dev server direttamente nel desktop. Claude può visualizzare l'app in esecuzione, testare gli endpoint, ispezionare i log e iterare su ciò che vede. Consulta Visualizza l'anteprima della tua app.

Traccia la tua pull request. Dopo aver aperto un PR, Claude Code monitora i risultati dei controlli CI e può correggere automaticamente gli errori o unire il PR una volta che tutti i controlli passano. Consulta Monitora lo stato della pull request.

Metti Claude in programma. Configura attività pianificate per eseguire Claude automaticamente su base ricorrente: una revisione del codice giornaliera ogni mattina, un audit delle dipendenze settimanale o un briefing che estrae dai tuoi strumenti connessi.

Scala quando sei pronto. Apri sessioni parallele dalla barra laterale per lavorare su più attività contemporaneamente, ognuna nel suo Git worktree. Invia lavoro di lunga durata al cloud in modo che continui anche se chiudi l'app, o continua una sessione sul web o nel tuo IDE se un'attività richiede più tempo del previsto. Connetti strumenti esterni come GitHub, Slack e Linear per riunire il tuo flusso di lavoro.

Vieni dalla CLI?

Desktop esegue lo stesso motore della CLI con un'interfaccia grafica. Puoi eseguire entrambi contemporaneamente sullo stesso progetto e condividono la configurazione (file CLAUDE.md, MCP servers, hooks, skills e impostazioni). Per un confronto completo delle funzionalità, equivalenti di flag e cosa non è disponibile in Desktop, consulta Confronto CLI.

Cosa c'è dopo

- Usa Claude Code Desktop: modalità di autorizzazione, sessioni parallele, visualizzazione diff, connettori e configurazione aziendale
- Risoluzione dei problemi: soluzioni a errori comuni e problemi di configurazione
- Best practice: suggerimenti per scrivere prompt efficaci e ottenere il massimo da Claude Code
- Flussi di lavoro comuni: tutorial per il debug, il refactoring, i test e altro

Configurare Claude Code

> *Installa, autentica e inizia a utilizzare Claude Code sulla tua macchina di sviluppo.*

Requisiti di sistema

- **Sistema operativo**:
  - macOS 13.0+
  - Windows 10 1809+ o Windows Server 2019+ (vedere le note di configurazione)
  - Ubuntu 20.04+
  - Debian 10+
  - Alpine Linux 3.19+ (dipendenze aggiuntive richieste)
- **Hardware**: 4 GB+ di RAM
- **Rete**: Connessione Internet richiesta (vedere configurazione di rete)
- **Shell**: Funziona meglio in Bash o Zsh
- **Posizione**: Paesi supportati da Anthropic

Dipendenze aggiuntive

- **ripgrep**: Solitamente incluso con Claude Code. Se la ricerca non funziona, vedere risoluzione dei problemi di ricerca.
- **Node.js 18+**: Richiesto solo per installazione npm deprecata

Installazione

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.

Dopo il completamento del processo di installazione, accedi al tuo progetto e avvia Claude Code:

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

Se riscontri problemi durante l'installazione, consulta la guida alla risoluzione dei problemi.

> **Tip:**
>
> Esegui `claude doctor` dopo l'installazione per verificare il tipo di installazione e la versione.

Configurazione specifica della piattaforma

Windows: Esegui Claude Code in modo nativo (richiede Git Bash) o all'interno di WSL. Sia WSL 1 che WSL 2 sono supportati, ma WSL 1 ha supporto limitato e non supporta funzionalità come il sandboxing dello strumento Bash.

Alpine Linux e altre distribuzioni basate su musl/uClibc:

Il programma di installazione nativo su Alpine e altre distribuzioni basate su musl/uClibc richiede `libgcc`, `libstdc++` e `ripgrep`. Installa questi utilizzando il gestore di pacchetti della tua distribuzione, quindi imposta `USE_BUILTIN_RIPGREP=0`.

Su Alpine:

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

Autenticazione

Per i singoli utenti

1. **Piano Claude Pro o Max** (consigliato): Sottoscrivi il piano Pro o Max di Claude per un abbonamento unificato che include sia Claude Code che Claude sul web. Gestisci il tuo account in un unico posto e accedi con il tuo account Claude.ai.
2. **Claude Console**: Connettiti tramite la Claude Console e completa il processo OAuth. Richiede fatturazione attiva nella Console Anthropic. Uno spazio di lavoro "Claude Code" viene creato automaticamente per il tracciamento dell'utilizzo e la gestione dei costi. Non puoi creare chiavi API per lo spazio di lavoro Claude Code; è dedicato esclusivamente all'utilizzo di Claude Code.

Per team e organizzazioni

1. **Claude for Teams o Enterprise** (consigliato): Sottoscrivi Claude for Teams o Claude for Enterprise per fatturazione centralizzata, gestione del team e accesso sia a Claude Code che a Claude sul web. I membri del team accedono con i loro account Claude.ai.
2. **Claude Console con fatturazione del team**: Configura un'organizzazione Claude Console condivisa con fatturazione del team. Invita i membri del team e assegna ruoli per il tracciamento dell'utilizzo.
3. **Provider cloud**: Configura Claude Code per utilizzare Amazon Bedrock, Google Vertex AI o Microsoft Foundry per distribuzioni con la tua infrastruttura cloud esistente.

Installa una versione specifica

Il programma di installazione nativo accetta un numero di versione specifico o un canale di rilascio (`latest` o `stable`). Il canale che scegli al momento dell'installazione diventa il tuo predefinito per gli aggiornamenti automatici. Vedere Configurare il canale di rilascio per ulteriori informazioni.

Per installare la versione più recente (predefinita):

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
```

Per installare la versione stabile:

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
```

Per installare un numero di versione specifico:

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
```

Integrità binaria e firma del codice

- I checksum SHA256 per tutte le piattaforme sono pubblicati nei manifesti di rilascio, attualmente ubicati in `https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json` (esempio: sostituisci `{VERSION}` con `2.0.30`)
- I binari firmati sono distribuiti per le seguenti piattaforme:
  - macOS: Firmato da “Anthropic PBC” e notarizzato da Apple
  - Windows: Firmato da “Anthropic, PBC”

Installazione NPM (deprecata)

L'installazione NPM è deprecata. Utilizza il metodo di installazione nativa quando possibile. Per migrare un'installazione npm esistente a nativa, esegui `claude install`.

Installazione npm globale

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

> **Warning:**
>
> NON utilizzare `sudo npm install -g` poiché ciò può portare a problemi di autorizzazione e rischi di sicurezza. Se riscontri errori di autorizzazione, vedere risoluzione dei problemi di autorizzazione per le soluzioni consigliate.

Configurazione di Windows

Opzione 1: Claude Code all'interno di WSL

- Sia WSL 1 che WSL 2 sono supportati
- WSL 2 supporta il sandboxing per una sicurezza migliorata. WSL 1 non supporta il sandboxing.

Opzione 2: Claude Code su Windows nativo con Git Bash

- Richiede Git for Windows
- Per le installazioni Git portatili, specifica il percorso del tuo `bash.exe`:

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

Aggiorna Claude Code

Aggiornamenti automatici

Claude Code si aggiorna automaticamente per assicurarti di avere le funzionalità più recenti e le correzioni di sicurezza.

- **Controlli degli aggiornamenti**: Eseguiti all'avvio e periodicamente durante l'esecuzione
- **Processo di aggiornamento**: Scarica e installa automaticamente in background
- **Notifiche**: Vedrai una notifica quando gli aggiornamenti vengono installati

- **Applicazione degli aggiornamenti**: Gli aggiornamenti hanno effetto la prossima volta che avvii Claude Code

> **Note:**
>
> Le installazioni Homebrew e WinGet non si aggiornano automaticamente. Utilizza `brew upgrade claude-code` o `winget upgrade Anthropic.ClaudeCode` per aggiornare manualmente.
>
> **Problema noto:** Claude Code potrebbe notificarti gli aggiornamenti prima che la nuova versione sia disponibile in questi gestori di pacchetti. Se un aggiornamento non riesce, attendi e riprova più tardi.

Configurare il canale di rilascio

Configura quale canale di rilascio Claude Code segue sia per gli aggiornamenti automatici che per `claude update` con l'impostazione `autoUpdatesChannel`:

- `"latest"` (predefinito): Ricevi nuove funzionalità non appena vengono rilasciate
- `"stable"`: Utilizza una versione che è tipicamente circa una settimana più vecchia, saltando i rilasci con regressioni importanti

Configura questo tramite `/config` → **Canale di aggiornamento automatico**, o aggiungilo al tuo file settings.json:

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

Per distribuzioni aziendali, puoi applicare un canale di rilascio coerente in tutta l'organizzazione utilizzando impostazioni gestite.

Disabilita gli aggiornamenti automatici

Imposta la variabile di ambiente `DISABLE_AUTOUPDATER` nella tua shell o nel file settings.json:

```
export DISABLE_AUTOUPDATER=1
```

Aggiorna manualmente

```
claude update
```

Disinstalla Claude Code

Se hai bisogno di disinstallare Claude Code, segui le istruzioni per il tuo metodo di installazione.

Installazione nativa

Rimuovi il binario Claude Code e i file di versione:

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"
```

Installazione Homebrew

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

Installazione WinGet

```
winget uninstall Anthropic.ClaudeCode
```

Installazione NPM

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

Pulisci i file di configurazione (facoltativo)

> **Warning:**
>
> La rimozione dei file di configurazione eliminerà tutte le tue impostazioni, gli strumenti consentiti, le configurazioni del server MCP e la cronologia della sessione.

Per rimuovere le impostazioni e i dati memorizzati nella cache di Claude Code:

macOS, Linux, WSL:

```
## Rimuovi le impostazioni utente e lo stato
rm -rf ~/.claude
rm ~/.claude.json

## Rimuovi le impostazioni specifiche del progetto (esegui dalla directory del tuo progetto)
rm -rf .claude
rm -f .mcp.json
```

Windows PowerShell:

```
## Rimuovi le impostazioni utente e lo stato
Remove-Item -Path "$env:USERPROFILE\.claude" -Recurse -Force
Remove-Item -Path "$env:USERPROFILE\.claude.json" -Force

## Rimuovi le impostazioni specifiche del progetto (esegui dalla directory del tuo progetto)
Remove-Item -Path ".claude" -Recurse -Force
Remove-Item -Path ".mcp.json" -Force
```

Windows CMD:

```
REM Rimuovi le impostazioni utente e lo stato
rmdir /s /q "%USERPROFILE%\.claude"
del "%USERPROFILE%\.claude.json"

REM Rimuovi le impostazioni specifiche del progetto (esegui dalla directory del
tuo progetto)
rmdir /s /q ".claude"
del ".mcp.json"
```

Come funziona Claude Code

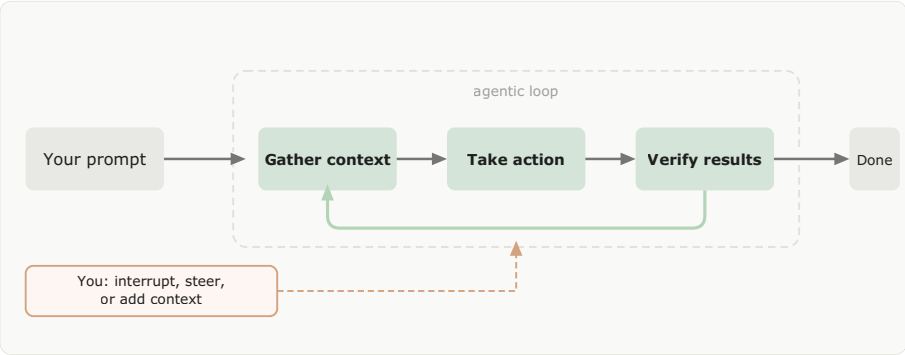
> *Comprendi il ciclo agentico, gli strumenti integrati e come Claude Code interagisce con il tuo progetto.*

Claude Code è un assistente agentico che funziona nel tuo terminale. Sebbene eccella nella programmazione, può aiutarti con qualsiasi cosa tu possa fare dalla riga di comando: scrivere documentazione, eseguire build, cercare file, ricercare argomenti e altro ancora.

Questa guida copre l'architettura principale, le capacità integrate e suggerimenti per lavorare efficacemente. Per procedure dettagliate, consulta Flussi di lavoro comuni. Per funzionalità di estensibilità come skills, MCP e hooks, consulta Estendi Claude Code.

Il ciclo agentico

Quando dai a Claude un compito, lavora attraverso tre fasi: **raccogliere contesto**, **intraprendere azioni** e **verificare i risultati**. Queste fasi si mescolano insieme. Claude utilizza strumenti durante tutto il processo, sia cercando file per comprendere il tuo codice, modificando per apportare modifiche, o eseguendo test per verificare il suo lavoro.



Il ciclo agentico: il tuo prompt porta Claude a raccogliere contesto, intraprendere azioni, verificare i risultati e ripetere fino al completamento dell'attività. Puoi interrompere in qualsiasi momento.

Il ciclo si adatta a quello che chiedi. Una domanda sulla tua base di codice potrebbe richiedere solo la raccolta di contesto. Una correzione di bug cicla attraverso tutte e tre le fasi ripetutamente. Un refactoring potrebbe comportare una verifica estesa. Claude decide cosa richiede ogni passaggio in base a quello che ha imparato dal passaggio precedente, concatenando dozzine di azioni insieme e correggendo il corso lungo il percorso.

Anche tu fai parte di questo ciclo. Puoi interrompere in qualsiasi momento per indirizzare Claude in una direzione diversa, fornire contesto aggiuntivo o chiedergli di provare un approccio diverso. Claude lavora autonomamente ma rimane reattivo al tuo input.

Il ciclo agentico è alimentato da due componenti: modelli che ragionano e strumenti che agiscono. Claude Code funge da **harness agentico** intorno a Claude: fornisce gli strumenti, la gestione del contesto e l'ambiente di esecuzione che trasformano un modello di linguaggio in un agente di codifica capace.

Modelli

Claude Code utilizza i modelli Claude per comprendere il tuo codice e ragionare sui compiti. Claude può leggere codice in qualsiasi linguaggio, comprendere come i componenti si collegano e capire cosa deve cambiare per raggiungere il tuo obiettivo. Per compiti complessi, suddivide il lavoro in passaggi, li esegue e si adatta in base a quello che impara.

Sono disponibili più modelli con diversi compromessi. Sonnet gestisce bene la maggior parte dei compiti di codifica. Opus fornisce un ragionamento più forte per decisioni architettoniche complesse. Cambia con `/model` durante una sessione o inizia con `claude --model <name>`.

Quando questa guida dice "Claude sceglie" o "Claude decide", è il modello che sta facendo il ragionamento.

Strumenti

Gli strumenti sono ciò che rende Claude Code agentico. Senza strumenti, Claude può solo rispondere con testo. Con gli strumenti, Claude può agire: leggere il tuo codice, modificare file, eseguire comandi, cercare il web e interagire con servizi esterni. Ogni utilizzo di uno strumento restituisce informazioni che si alimentano nel ciclo, informando la decisione successiva di Claude.

Gli strumenti integrati generalmente rientrano in cinque categorie, ognuna rappresentante un diverso tipo di agenzia.

Categoria	Cosa Claude può fare
Operazioni su file	Leggere file, modificare codice, creare nuovi file, rinominare e riorganizzare
Ricerca	Trovare file per pattern, cercare contenuto con regex, esplorare basi di codice

Categoria	Cosa Claude può fare
Esecuzione	Eseguire comandi shell, avviare server, eseguire test, usare git
Web	Cercare il web, recuperare documentazione, cercare messaggi di errore
Intelligenza del codice	Vedere errori di tipo e avvisi dopo le modifiche, saltare alle definizioni, trovare riferimenti (richiede plugin di intelligenza del codice)

Queste sono le capacità principali. Claude ha anche strumenti per generare subagents, farti domande e altri compiti di orchestrazione. Consulta Strumenti disponibili per Claude per l'elenco completo.

Claude sceglie quali strumenti utilizzare in base al tuo prompt e a quello che impara lungo il percorso. Quando dici "correggi i test falliti", Claude potrebbe:

1. Eseguire la suite di test per vedere cosa sta fallendo
2. Leggere l'output dell'errore
3. Cercare i file sorgente rilevanti
4. Leggere quei file per comprendere il codice
5. Modificare i file per correggere il problema
6. Eseguire di nuovo i test per verificare

Ogni utilizzo di uno strumento fornisce a Claude nuove informazioni che informano il passaggio successivo. Questo è il ciclo agentico in azione.

Estensione delle capacità di base: Gli strumenti integrati sono la fondazione. Puoi estendere ciò che Claude sa con skills, connetterti a servizi esterni con MCP, automatizzare flussi di lavoro con hooks e delegare compiti a subagents. Queste estensioni formano un livello sopra il ciclo agentico principale. Consulta Estendi Claude Code per una guida sulla scelta dell'estensione giusta per le tue esigenze.

A cosa Claude può accedere

Questa guida si concentra sul terminale. Claude Code funziona anche in VS Code, IDE JetBrains e altri ambienti.

Quando esegui `claude` in una directory, Claude Code ottiene accesso a:

- **Il tuo progetto.** File nella tua directory e sottodirectory, più file altrove con la tua autorizzazione.
- **Il tuo terminale.** Qualsiasi comando che potresti eseguire: strumenti di build, git, gestori di pacchetti, utilità di sistema, script. Se puoi farlo dalla riga di comando, Claude può farlo anche lui.
- **Il tuo stato git.** Ramo corrente, modifiche non committate e cronologia dei commit recenti.
- **Il tuo CLAUDE.md.** Un file markdown dove memorizzi istruzioni specifiche del progetto, convenzioni e contesto che Claude dovrebbe conoscere ogni sessione.
- **Auto memory.** Apprendimenti che Claude salva automaticamente mentre lavori, come pattern di progetto e le tue preferenze. Le prime 200 righe di MEMORY.md vengono caricate all'inizio di ogni sessione.
- **Estensioni che configuri.** Server MCP per servizi esterni, skills per flussi di lavoro, subagents per lavoro delegato e Claude in Chrome per l'interazione del browser.

Poiché Claude vede l'intero tuo progetto, può lavorare su di esso. Quando chiedi a Claude di "correggere il bug di autenticazione", cerca file rilevanti, legge più file per comprendere il contesto, apporta modifiche coordinate su di essi, esegue test per verificare la correzione e committa le modifiche se lo chiedi. Questo è diverso dagli assistenti di codice inline che vedono solo il file corrente.

Ambienti e interfacce

Il ciclo agentico, gli strumenti e le capacità descritti sopra sono gli stessi ovunque tu usi Claude Code. Quello che cambia è dove il codice viene eseguito e come interagisci con esso.

Ambienti di esecuzione

Claude Code funziona in tre ambienti, ognuno con diversi compromessi per dove il tuo codice viene eseguito.

Ambiente	Dove viene eseguito il codice	Caso d'uso
Locale	La tua macchina	Predefinito. Accesso completo ai tuoi file, strumenti e ambiente

Ambiente	Dove viene eseguito il codice	Caso d'uso
Cloud	VM gestite da Anthropic	Delegare compiti, lavorare su repo che non hai localmente
Remote Control	La tua macchina, controllata da un browser	Usa l'interfaccia web mantenendo tutto locale

Interfacce

Puoi accedere a Claude Code tramite il terminale, l'app desktop, estensioni IDE, claude.ai/code, Remote Control, Slack e pipeline CI/CD. L'interfaccia determina come vedi e interagisci con Claude, ma il ciclo agentico sottostante è identico. Consulta Usa Claude Code ovunque per l'elenco completo.

Lavora con le sessioni

Claude Code salva la tua conversazione localmente mentre lavori. Ogni messaggio, utilizzo di strumento e risultato viene archiviato, il che consente il rewind, la ripresa e il fork delle sessioni. Prima che Claude apporta modifiche al codice, crea anche uno snapshot dei file interessati in modo da poter ripristinare se necessario.

Le sessioni sono indipendenti. Ogni nuova sessione inizia con una finestra di contesto fresca, senza la cronologia della conversazione dalle sessioni precedenti. Claude può persistere gli apprendimenti tra le sessioni utilizzando auto memory e puoi aggiungere le tue istruzioni persistenti in CLAUDE.md.

Lavora tra i rami

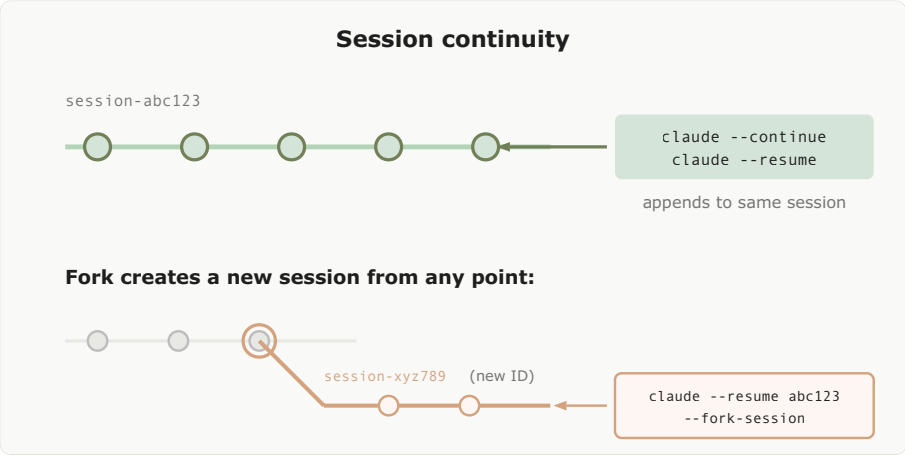
Ogni conversazione di Claude Code è una sessione legata alla tua directory corrente. Quando riprendi, vedi solo le sessioni da quella directory.

Claude vede i file del tuo ramo corrente. Quando cambi ramo, Claude vede i file del nuovo ramo, ma la cronologia della tua conversazione rimane la stessa. Claude ricorda quello che hai discusso anche dopo il cambio di ramo.

Poiché le sessioni sono legate alle directory, puoi eseguire sessioni Claude parallele utilizzando git worktrees, che creano directory separate per singoli rami.

Riprendi o fai il fork delle sessioni

Quando riprendi una sessione con `claude --continue` o `claude --resume`, riprendi da dove hai lasciato utilizzando lo stesso ID di sessione. I nuovi messaggi si aggiungono alla conversazione esistente. La tua cronologia completa della conversazione viene ripristinata, ma i permessi con ambito di sessione non lo sono. Dovrai approvarli di nuovo.



Continuità della sessione: resume continua la stessa sessione, fork crea un nuovo ramo con un nuovo ID.

Per creare un ramo e provare un approccio diverso senza influenzare la sessione originale, utilizza il flag `--fork-session`:

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

Questo crea un nuovo ID di sessione preservando la cronologia della conversazione fino a quel punto. La sessione originale rimane invariata. Come resume, le sessioni forkate non ereditano i permessi con ambito di sessione.

Stessa sessione in più terminali: Se riprendi la stessa sessione in più terminali, entrambi i terminali scrivono nello stesso file di sessione. I messaggi da entrambi vengono intercalati, come due persone che scrivono nello stesso quaderno. Nulla si corrompe, ma la conversazione diventa confusa. Ogni terminale vede solo i suoi messaggi durante la sessione, ma se riprendi quella sessione in seguito, vedrai tutto intercalato. Per il lavoro parallelo dallo stesso punto di partenza, utilizza `--fork-session` per dare a ogni terminale la sua sessione pulita.

La finestra di contesto

La finestra di contesto di Claude contiene la cronologia della tua conversazione, i contenuti dei file, gli output dei comandi, CLAUDE.md, le skills caricate e le istruzioni di sistema. Man mano che lavori, il contesto si riempie. Claude compatta automaticamente, ma le istruzioni dall'inizio della conversazione possono andare perse. Metti le regole persistenti in CLAUDE.md ed esegui `/context` per vedere cosa sta usando lo spazio.

Quando il contesto si riempie

Claude Code gestisce il contesto automaticamente mentre ti avvicini al limite. Cancella prima gli output degli strumenti più vecchi, quindi riassume la conversazione se necessario. Le tue richieste e i frammenti di codice chiave vengono preservati; le istruzioni dettagliate dall'inizio della conversazione possono andare perse. Metti le regole persistenti in CLAUDE.md piuttosto che fare affidamento sulla cronologia della conversazione.

Per controllare cosa viene preservato durante la compattazione, aggiungi una sezione "Compact Instructions" a CLAUDE.md o esegui `/compact` con un focus (come `/compact focus on the API changes`).

Esegui `/context` per vedere cosa sta usando lo spazio. I server MCP aggiungono definizioni di strumenti a ogni richiesta, quindi pochi server possono consumare un contesto significativo prima di iniziare a lavorare. Esegui `/mcp` per controllare i costi per server.

Gestisci il contesto con skills e subagents

Oltre alla compattazione, puoi utilizzare altre funzionalità per controllare cosa viene caricato nel contesto.

Skills si caricano su richiesta. Claude vede le descrizioni delle skills all'inizio della sessione, ma il contenuto completo si carica solo quando una skill viene utilizzata. Per le skills che invochi manualmente, imposta `disable-model-invocation: true` per mantenere le descrizioni fuori dal contesto fino a quando non ne hai bisogno.

Subagents ottengono il loro contesto fresco, completamente separato dalla tua conversazione principale. Il loro lavoro non gonfia il tuo contesto. Quando finito, restituiscono un riassunto. Questo isolamento è il motivo per cui i subagents aiutano con le sessioni lunghe.

Consulta costi del contesto per quello che ogni funzionalità costa e riduci l'utilizzo dei token per suggerimenti sulla gestione del contesto.

Rimani al sicuro con checkpoint e permessi

Claude ha due meccanismi di sicurezza: i checkpoint ti permettono di annullare le modifiche ai file e i permessi controllano cosa Claude può fare senza chiedere.

Annulla le modifiche con i checkpoint

Ogni modifica ai file è reversibile. Prima che Claude modifichi qualsiasi file, crea uno snapshot dei contenuti attuali. Se qualcosa va storto, premi `Esc` due volte per tornare a uno stato precedente, o chiedi a Claude di annullare.

I checkpoint sono locali alla tua sessione, separati da git. Coprono solo le modifiche ai file. Le azioni che influenzano i sistemi remoti (database, API, distribuzioni) non possono essere checkpointed, motivo per cui Claude chiede prima di eseguire comandi con effetti collaterali esterni.

Controlla cosa Claude può fare

Premi `Shift+Tab` per scorrere le modalità di permesso:

- **Predefinito**: Claude chiede prima delle modifiche ai file e dei comandi shell
- **Auto-accept edits**: Claude modifica i file senza chiedere, chiede ancora per i comandi
- **Plan mode**: Claude utilizza solo strumenti di sola lettura, creando un piano che puoi approvare prima dell'esecuzione

Puoi anche consentire comandi specifici in `.claude/settings.json` in modo che Claude non chieda ogni volta. Questo è utile per comandi affidabili come `npm test` o `git status`. Le impostazioni possono essere scoped da politiche a livello di organizzazione fino alle preferenze personali. Consulta Permessi per i dettagli.

Lavora efficacemente con Claude Code

Questi suggerimenti ti aiutano a ottenere risultati migliori da Claude Code.

Chiedi aiuto a Claude Code

Claude Code può insegnarti come usarlo. Fai domande come "come configuro gli hooks?" o "qual è il modo migliore per strutturare il mio CLAUDE.md?" e Claude spiegherà.

I comandi integrati ti guidano anche attraverso la configurazione:

- `/init` ti guida attraverso la creazione di un CLAUDE.md per il tuo progetto
- `/agents` ti aiuta a configurare subagents personalizzati

- `/doctor` diagnostica i problemi comuni con la tua installazione

È una conversazione

Claude Code è conversazionale. Non hai bisogno di prompt perfetti. Inizia con quello che vuoi, quindi affina:

```
Correggi il bug di login
```

[Claude indaga, prova qualcosa]

```
Non è del tutto giusto. Il problema è nella gestione della sessione.
```

[Claude adatta l'approccio]

Quando il primo tentativo non è giusto, non ricominciare da capo. Itera.

Interrompi e indirizza

Puoi interrompere Claude in qualsiasi momento. Se sta andando nella direzione sbagliata, digita semplicemente la tua correzione e premi Invio. Claude smetterà quello che sta facendo e adatterà il suo approccio in base al tuo input. Non devi aspettare che finisca o ricominciare da capo.

Sii specifico all'inizio

Più preciso è il tuo prompt iniziale, meno correzioni avrai bisogno. Fai riferimento a file specifici, menziona vincoli e indica pattern di esempio.

```
Il flusso di checkout è rotto per gli utenti con carte scadute.
Controlla src/payments/ per il problema, specialmente l'aggiornamento del token.
Scrivi prima un test fallito, poi correggilo.
```

I prompt vaghi funzionano, ma passerai più tempo a indirizzare. I prompt specifici come quello sopra spesso hanno successo al primo tentativo.

Dai a Claude qualcosa da verificare

Claude funziona meglio quando può verificare il suo lavoro. Includi casi di test, incolla screenshot dell'interfaccia utente prevista o definisci l'output che desideri.

```
Implementa validateEmail. Casi di test: 'user@example.com' → true,
'invalid' → false, 'user@.com' → false. Esegui i test dopo.
```

Per il lavoro visivo, incolla uno screenshot del design e chiedi a Claude di confrontare la sua implementazione con esso.

Esplora prima di implementare

Per problemi complessi, separa la ricerca dalla codifica. Utilizza la modalità piano (`Shift+Tab` due volte) per analizzare prima la base di codice:

```
Leggi src/auth/ e comprendi come gestiamo le sessioni.
Quindi crea un piano per aggiungere il supporto OAuth.
```

Rivedi il piano, affinalo attraverso la conversazione, quindi lascia che Claude implementi. Questo approccio a due fasi produce risultati migliori rispetto al passare direttamente al codice.

Delega, non dettare

Pensa di delegare a un collega capace. Fornisci contesto e direzione, quindi affida a Claude di capire i dettagli:

```
Il flusso di checkout è rotto per gli utenti con carte scadute.
Il codice rilevante è in src/payments/. Puoi indagare e correggerlo?
```

Non hai bisogno di specificare quali file leggere o quali comandi eseguire. Claude lo capisce.

Cosa c'è dopo

- Estendi con funzionalità Aggiungi Skills, connessioni MCP e comandi personalizzati
- Flussi di lavoro comuni Guide passo dopo passo per compiti tipici

Part 2: Core Usage

Estendi Claude Code

> *Comprendi quando utilizzare CLAUDE.md, Skills, subagents, hooks, MCP e plugins.*

Claude Code combina un modello che ragiona sul vostro codice con strumenti integrati per operazioni su file, ricerca, esecuzione e accesso web. Gli strumenti integrati coprono la maggior parte dei compiti di codifica. Questa guida copre il livello di estensione: funzionalità che aggiungete per personalizzare ciò che Claude conosce, collegarlo a servizi esterni e automatizzare i flussi di lavoro.

> **Note:**
>
> Per informazioni su come funziona il ciclo agentico principale, consultate How Claude Code works.

Siete nuovi a Claude Code? Iniziate con CLAUDE.md per le convenzioni del progetto. Aggiungete altre estensioni secondo le vostre necessità.

Panoramica

Le estensioni si collegano a diverse parti del ciclo agentico:

- **CLAUDE.md** aggiunge contesto persistente che Claude vede in ogni sessione
- **Skills** aggiungono conoscenze riutilizzabili e flussi di lavoro invocabili
- **MCP** collega Claude a servizi e strumenti esterni
- **Subagents** eseguono i loro propri cicli in contesto isolato, restituendo riassunti
- **Agent teams** coordinano più sessioni indipendenti con compiti condivisi e messaggistica peer-to-peer
- **Hooks** vengono eseguiti completamente al di fuori del ciclo come script deterministici
- **Plugins** e **marketplaces** confezionano e distribuiscono queste funzionalità

Skills sono l'estensione più flessibile. Una skill è un file markdown contenente conoscenze, flussi di lavoro o istruzioni. Potete invocare skills con un comando come `/deploy`, oppure Claude può caricarle automaticamente quando rilevante. Le skills possono essere eseguite nella vostra conversazione attuale o in un contesto isolato tramite subagents.

Abbinate le funzionalità al vostro obiettivo

Le funzionalità vanno dal contesto sempre attivo che Claude vede in ogni sessione, alle capacità su richiesta che voi o Claude potete invocare, all'automazione in background che viene eseguita su eventi specifici. La tabella seguente mostra ciò che è disponibile e quando ogni funzionalità ha senso.

Funzionalità	Cosa fa	Quando utilizzarla	Esempio
CLAUDE.md	Contesto persistente caricato in ogni conversazione	Convenzioni del progetto, regole "fai sempre X"	"Usa pnpm, non npm. Esegui i test prima di fare il commit."
Skill	Istruzioni, conoscenze e flussi di lavoro che Claude può utilizzare	Contenuto riutilizzabile, documenti di riferimento, compiti ripetibili	`/deploy` esegue la vostra checklist di distribuzione; skill di documentazione API con pattern di endpoint
Subagent	Contesto di esecuzione isolato che restituisce risultati riassunti	Isolamento del contesto, compiti paralleli, worker specializzati	Compito di ricerca che legge molti file ma restituisce solo i risultati chiave
Agent teams	Coordinare più sessioni Claude Code indipendenti	Ricerca parallela, sviluppo di nuove funzionalità, debug con ipotesi concorrenti	Generare revisori per controllare sicurezza, prestazioni e test simultaneamente
MCP	Collegamento a servizi esterni	Dati o azioni esterne	Interrogare il vostro database, inviare a Slack, controllare un browser
Hook	Script deterministico che viene eseguito su eventi	Automazione prevedibile, nessun LLM coinvolto	Eseguire ESLint dopo ogni modifica di file

Plugins sono il livello di confezionamento. Un plugin raggruppa skills, hooks, subagents e server MCP in una singola unità installabile. Le skills dei plugin hanno uno spazio dei nomi (come `/my-plugin:review`) in modo che più plugin possano coesistere. Utilizzate i plugin quando desiderate riutilizzare la stessa configurazione su più repository o distribuire ad altri tramite un **marketplace**.

Confrontate funzionalità simili

Alcune funzionalità possono sembrare simili. Ecco come distinguerle.

Skill vs Subagent

Skills e subagents risolvono problemi diversi:

- **Skills** sono contenuti riutilizzabili che potete caricare in qualsiasi contesto
- **Subagents** sono worker isolati che vengono eseguiti separatamente dalla vostra conversazione principale

Aspetto	Skill	Subagent
Cosa è	Istruzioni, conoscenze o flussi di lavoro riutilizzabili	Worker isolato con il suo proprio contesto
Vantaggio principale	Condividere contenuti tra contesti	Isolamento del contesto. Il lavoro avviene separatamente, solo il riassunto ritorna
Migliore per	Materiale di riferimento, flussi di lavoro invocabili	Compiti che leggono molti file, lavoro parallelo, worker specializzati

Le skills possono essere di riferimento o di azione. Le skills di riferimento forniscono conoscenze che Claude utilizza durante la vostra sessione (come la vostra guida di stile API). Le skills di azione dicono a Claude di fare qualcosa di specifico (come `/deploy` che esegue il vostro flusso di lavoro di distribuzione).

Utilizzate un subagent quando avete bisogno di isolamento del contesto o quando la vostra finestra di contesto si sta riempiendo. Il subagent potrebbe leggere dozzine di file o eseguire ricerche estese, ma la vostra conversazione principale riceve solo un riassunto. Poiché il lavoro del subagent non consuma il vostro contesto principale, questo è utile anche quando non avete bisogno che il lavoro intermedio rimanga visibile. I subagents personalizzati possono avere le loro proprie istruzioni e possono precaricare skills.

Possono combinarsi. Un subagent può precaricare skills specifiche (campo `skills:`). Una skill può essere eseguita in contesto isolato utilizzando `context: fork`. Consultate Skills per i dettagli.

CLAUDE.md vs Skill

Entrambi memorizzano istruzioni, ma si caricano diversamente e servono scopi diversi.

Aspetto	CLAUDE.md	Skill
Si carica	Ogni sessione, automaticamente	Su richiesta
Può includere file	Sì, con importazioni `@path`	Sì, con importazioni `@path`
Può attivare flussi di lavoro	No	Sì, con `/<name>`
Migliore per	Regole “fai sempre X”	Materiale di riferimento, flussi di lavoro invocabili

Mettetelo in CLAUDE.md se Claude dovrebbe sempre saperlo: convenzioni di codifica, comandi di build, struttura del progetto, regole “non fare mai X”.

Mettetelo in una skill se è materiale di riferimento di cui Claude ha bisogno a volte (documentazione API, guide di stile) o un flusso di lavoro che attivate con `/<name>` (deploy, review, release).

Regola pratica: Mantenete CLAUDE.md sotto 200 righe. Se sta crescendo, spostate il contenuto di riferimento in skills o dividetelo in file `.claude/rules/`.

CLAUDE.md vs Rules vs Skills

Tutti e tre memorizzano istruzioni, ma si caricano diversamente:

Aspetto	CLAUDE.md	`.claude/rules/`	Skill
Si carica	Ogni sessione	Ogni sessione, o quando vengono aperti file corrispondenti	Su richiesta, quando invocato o rilevante
Ambito	Intero progetto	Può essere limitato a percorsi di file	Specifico del compito

Aspetto	CLAUDE.md	`.claude/rules/`	Skill
Migliore per	Convenzioni e comandi di build principali	Linee guida specifiche del linguaggio o della directory	Materiale di riferimento, flussi di lavoro ripetibili

Utilizzate CLAUDE.md per istruzioni di cui ogni sessione ha bisogno: comandi di build, convenzioni di test, architettura del progetto.

Utilizzate rules per mantenere CLAUDE.md focalizzato. Le rules con frontmatter `paths` si caricano solo quando Claude lavora con file corrispondenti, risparmiando contesto.

Utilizzate skills per contenuti di cui Claude ha bisogno solo a volte, come documentazione API o una checklist di distribuzione che attivate con `/<name>`.

Subagent vs Agent team

Entrambi parallelizzano il lavoro, ma sono architettonicamente diversi:

- **Subagents** vengono eseguiti all'interno della vostra sessione e riportano i risultati al vostro contesto principale
- **Agent teams** sono sessioni Claude Code indipendenti che comunicano tra loro

Aspetto	Subagent	Agent team
Contesto	Finestra di contesto propria; i risultati ritornano al chiamante	Finestra di contesto propria; completamente indipendente
Comunicazione	Riporta i risultati solo all'agente principale	I compagni di squadra si messaggiano direttamente
Coordinamento	L'agente principale gestisce tutto il lavoro	Elenco di compiti condiviso con auto-coordinamento
Migliore per	Compiti focalizzati dove conta solo il risultato	Lavoro complesso che richiede discussione e collaborazione
Costo del token	Inferiore: i risultati sono riassunti al contesto principale	Superiore: ogni compagno di squadra è un'istanza Claude separata

Utilizzate un subagent quando avete bisogno di un worker veloce e focalizzato: ricercare una domanda, verificare un'affermazione, rivedere un file. Il subagent fa il lavoro e restituisce un riassunto. La vostra conversazione principale rimane pulita.

Utilizzate un agent team quando i compagni di squadra hanno bisogno di condividere i risultati, sfidare l'uno l'altro e coordinarsi in modo indipendente. Gli agent teams sono migliori per la ricerca con ipotesi concorrenti, la revisione del codice parallela e lo sviluppo di nuove funzionalità dove ogni compagno di squadra possiede un pezzo separato.

Punto di transizione: Se state eseguendo subagents paralleli ma raggiungete limiti di contesto, o se i vostri subagents hanno bisogno di comunicare tra loro, gli agent teams sono il passo naturale successivo.

> **Note:**
>
> Gli agent teams sono sperimentali e disabilitati per impostazione predefinita. Consultate agent teams per la configurazione e le limitazioni attuali.

MCP vs Skill

MCP collega Claude a servizi esterni. Le skills estendono ciò che Claude conosce, incluso come utilizzare efficacemente quei servizi.

Aspetto	MCP	Skill
Cosa è	Protocollo per il collegamento a servizi esterni	Conoscenze, flussi di lavoro e materiale di riferimento
Fornisce	Accesso a strumenti e dati	Conoscenze, flussi di lavoro, materiale di riferimento
Esempi	Integrazione Slack, query di database, controllo del browser	Checklist di revisione del codice, flusso di lavoro di distribuzione, guida di stile API

Questi risolvono problemi diversi e funzionano bene insieme:

MCP dà a Claude la capacità di interagire con sistemi esterni. Senza MCP, Claude non può interrogare il vostro database o inviare a Slack.

Skills danno a Claude conoscenze su come utilizzare efficacemente quegli strumenti, più flussi di lavoro che potete attivare con `/<name>`. Una skill potrebbe includere lo schema del database del vostro team e i pattern di query, o un flusso di lavoro `/post-to-slack` con le regole di formattazione dei messaggi del vostro team.

Esempio: Un server MCP collega Claude al vostro database. Una skill insegna a Claude il vostro modello di dati, i pattern di query comuni e quali tabelle utilizzare per diversi compiti.

Comprendete come le funzionalità si stratificano

Le funzionalità possono essere definite a più livelli: a livello di utente, per progetto, tramite plugins o tramite politiche gestite. Potete anche annidare file CLAUDE.md in sottodirectory o posizionare skills in pacchetti specifici di un monorepo. Quando la stessa funzionalità esiste a più livelli, ecco come si stratificano:

- **I file CLAUDE.md** sono additivi: tutti i livelli contribuiscono contenuti al contesto di Claude simultaneamente. I file dalla vostra directory di lavoro e sopra si caricano all'avvio; le sottodirectory si caricano mentre lavorate in esse. Quando le istruzioni entrano in conflitto, Claude usa il giudizio per riconciliarle, con istruzioni più specifiche che tipicamente hanno la precedenza. Consultate come i file CLAUDE.md si caricano.
- **Skills e subagents** si sovrascrivono per nome: quando lo stesso nome esiste a più livelli, una definizione vince in base alla priorità (gestito > utente > progetto per skills; gestito > flag CLI > progetto > utente > plugin per subagents). Le skills dei plugin sono con spazio dei nomi per evitare conflitti. Consultate scoperta delle skills e ambito del subagent.
- **I server MCP** si sovrascrivono per nome: locale > progetto > utente. Consultate ambito MCP.
- **Hooks** si uniscono: tutti gli hooks registrati si attivano per i loro eventi corrispondenti indipendentemente dalla fonte. Consultate hooks.

Combinate le funzionalità

Ogni estensione risolve un problema diverso: CLAUDE.md gestisce il contesto sempre attivo, le skills gestiscono conoscenze e flussi di lavoro su richiesta, MCP gestisce connessioni esterne, i subagents gestiscono l'isolamento e gli hooks gestiscono l'automazione. Le configurazioni reali le combinano in base al vostro flusso di lavoro.

Ad esempio, potreste utilizzare CLAUDE.md per convenzioni del progetto, una skill per il vostro flusso di lavoro di distribuzione, MCP per connettervi al vostro database e un hook per eseguire il linting dopo ogni modifica. Ogni funzionalità gestisce ciò che fa meglio.

Pattern	Come funziona	Esempio
Skill + MCP	MCP fornisce la connessione; una skill insegna a Claude come utilizzarla bene	MCP si connette al vostro database, una skill documenta lo schema e i pattern di query
Skill + Subagent	Una skill genera subagents per il lavoro parallelo	La skill `/audit` avvia subagents di sicurezza, prestazioni e stile che lavorano in contesto isolato
CLAUDE.md + Skills	CLAUDE.md contiene regole sempre attive; le skills contengono materiale di riferimento caricato su richiesta	CLAUDE.md dice “segui le nostre convenzioni API,” una skill contiene la guida di stile API completa
Hook + MCP	Un hook attiva azioni esterne tramite MCP	L’hook post-modifica invia una notifica Slack quando Claude modifica file critici

Comprendete i costi del contesto

Ogni funzionalità che aggiungete consuma parte del contesto di Claude. Troppo può riempire la vostra finestra di contesto, ma può anche aggiungere rumore che rende Claude meno efficace; le skills potrebbero non attivarsi correttamente, o Claude potrebbe perdere traccia delle vostre convenzioni. Comprendere questi compromessi vi aiuta a costruire una configurazione efficace.

Costo del contesto per funzionalità

Ogni funzionalità ha una strategia di caricamento e un costo di contesto diversi:

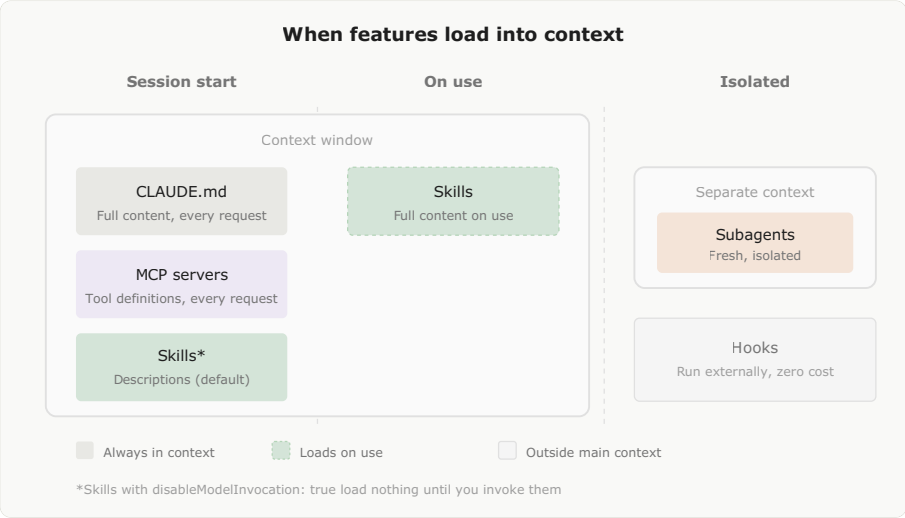
Funzionalità	Quando si carica	Cosa si carica	Costo del contesto
CLAUDE.md	Inizio della sessione	Contenuto completo	Ogni richiesta
Skills	Inizio della sessione + quando utilizzate	Descrizioni all’inizio, contenuto completo quando utilizzate	Basso (descrizioni ogni richiesta)*

Funzionalità	Quando si carica	Cosa si carica	Costo del contesto
Server MCP	Inizio della sessione	Tutte le definizioni degli strumenti e gli schemi JSON	Ogni richiesta
Subagents	Quando generati	Contesto fresco con skills specificate	Isolato dalla sessione principale
Hooks	Al trigger	Niente (viene eseguito esternamente)	Zero, a meno che l'hook non restituisca contesto aggiuntivo

*Per impostazione predefinita, le descrizioni delle skills si caricano all'inizio della sessione in modo che Claude possa decidere quando utilizzarle. Impostate `disable-model-invocation: true` nel frontmatter di una skill per nasconderla completamente a Claude fino a quando non la invocate manualmente. Questo riduce il costo del contesto a zero per le skills che invocate solo voi.

Comprendete come le funzionalità si caricano

Ogni funzionalità si carica in diversi punti della vostra sessione. Le schede seguenti spiegano quando ogni funzionalità si carica e cosa entra nel contesto.



Caricamento del contesto: CLAUDE.md e MCP si caricano all'inizio della sessione e rimangono in ogni richiesta. Le skills caricano descrizioni all'inizio, contenuto completo all'invocazione. I subagents ottengono contesto isolato. Gli hooks vengono eseguiti esternamente.

CLAUDE.md

Quando: Inizio della sessione

Cosa si carica: Contenuto completo di tutti i file CLAUDE.md (livelli gestito, utente e progetto).

Eredità: Claude legge i file CLAUDE.md dalla vostra directory di lavoro fino alla radice e scopre quelli annidati nelle sottodirectory mentre accede a quei file. Consultate Come i file CLAUDE.md si caricano per i dettagli.

> **Tip:**
>
> Mantenete CLAUDE.md sotto ~500 righe. Spostate il materiale di riferimento in skills, che si caricano su richiesta.

Skills

Le skills sono capacità extra nel toolkit di Claude. Possono essere materiale di riferimento (come una guida di stile API) o flussi di lavoro invocabili che attivate con `/<name>` (come `/deploy`). Claude Code viene fornito con skills raggruppate come `/simplify`, `/batch` e `/debug` che funzionano subito. Potete anche crearne di vostre. Claude utilizza le skills quando appropriato, oppure potete invocarne una direttamente.

Quando: Dipende dalla configurazione della skill. Per impostazione predefinita, le descrizioni si caricano all'inizio della sessione e il contenuto completo si carica quando utilizzate. Per le skills solo utente (`disable-model-invocation: true`), niente si carica fino a quando non le invocate.

Cosa si carica: Per le skills invocabili dal modello, Claude vede nomi e descrizioni in ogni richiesta. Quando invocate una skill con `/<name>` o Claude la carica automaticamente, il contenuto completo si carica nella vostra conversazione.

Come Claude sceglie le skills: Claude abbina il vostro compito alle descrizioni delle skills per decidere quali sono rilevanti. Se le descrizioni sono vaghe o si sovrappongono, Claude potrebbe caricare la skill sbagliata o perderne una che aiuterebbe. Per dire a Claude di utilizzare una skill specifica, invocatela con `/<name>`. Le skills con `disable-model-invocation: true` sono invisibili a Claude fino a quando non le invocate.

Costo del contesto: Basso fino a quando non vengono utilizzate. Le skills solo utente hanno costo zero fino a quando non vengono invocate.

Nei subagents: Le skills funzionano diversamente nei subagents. Invece del caricamento su richiesta, le skills passate a un subagent vengono completamente precaricate nel suo contesto all'avvio. I subagents non ereditano le skills dalla sessione principale; dovete specificarle esplicitamente.

> **Tip:**
>
> Utilizzate `disable-model-invocation: true` per le skills con effetti collaterali. Questo risparmia contesto e assicura che solo voi le attiviate.

Server MCP

Quando: Inizio della sessione.

Cosa si carica: Tutte le definizioni degli strumenti e gli schemi JSON dai server connessi.

Costo del contesto: Ricerca degli strumenti (abilitata per impostazione predefinita) carica gli strumenti MCP fino al 10% del contesto e rinvia il resto fino a quando non è necessario.

Nota di affidabilità: Le connessioni MCP possono fallire silenziosamente durante la sessione. Se un server si disconnette, i suoi strumenti scompaiono senza avviso. Claude potrebbe provare a utilizzare uno strumento che non esiste più. Se notate che Claude non riesce a utilizzare uno strumento MCP a cui poteva accedere in precedenza, controllate la connessione con `/mcp`.

> **Tip:**
>
> Eseguite `/mcp` per vedere i costi dei token per server. Disconnettete i server che non state utilizzando attivamente.

Subagents

Quando: Su richiesta, quando voi o Claude ne generate uno per un compito.

Cosa si carica: Contesto fresco e isolato contenente:

- Il prompt di sistema (condiviso con il genitore per l'efficienza della cache)
- Contenuto completo delle skills elencate nel campo `skills:` dell'agente
- CLAUDE.md e stato git (ereditati dal genitore)
- Qualsiasi contesto che l'agente principale passa nel prompt

Costo del contesto: Isolato dalla sessione principale. I subagents non ereditano la vostra cronologia di conversazione o le skills invocate.

> **Tip:**
>
> Utilizzate i subagents per il lavoro che non ha bisogno del vostro contesto di conversazione completo. Il loro isolamento previene il gonfiore della vostra sessione principale.

Hooks

Quando: Al trigger. Gli hooks si attivano su eventi del ciclo di vita specifici come esecuzione dello strumento, confini della sessione, invio del prompt, richieste di autorizzazione e compattazione. Consultate Hooks per l'elenco completo.

Cosa si carica: Niente per impostazione predefinita. Gli hooks vengono eseguiti come script esterni.

Costo del contesto: Zero, a meno che l'hook non restituisca output che viene aggiunto come messaggi alla vostra conversazione.

> **Tip:**
>
> Gli hooks sono ideali per effetti collaterali (linting, logging) che non hanno bisogno di influenzare il contesto di Claude.

Scopri di più

Ogni funzionalità ha la sua propria guida con istruzioni di configurazione, esempi e opzioni di configurazione.

- CLAUDE.md Memorizzate il contesto del progetto, le convenzioni e le istruzioni
- Skills Date a Claude competenze di dominio e flussi di lavoro riutilizzabili
- Subagents Delegate il lavoro a contesto isolato
- Agent teams Coordinate più sessioni che lavorano in parallelo
- MCP Collegate Claude a servizi esterni
- Hooks Automatizzate i flussi di lavoro con gli hooks
- Plugins Confezionate e condividete set di funzionalità
- Marketplaces Ospitate e distribuite raccolte di plugin

Modalità interattiva

> *Riferimento completo per le scorciatoie da tastiera, le modalità di input e le funzioni interattive nelle sessioni di Claude Code.*

Scorciatoie da tastiera

Note:

Le scorciatoie da tastiera possono variare a seconda della piattaforma e del terminale. Premere `?` per visualizzare le scorciatoie disponibili per il vostro ambiente.

Utenti macOS: Le scorciatoie con il tasto Option/Alt (`Alt+B` , `Alt+F` , `Alt+Y` , `Alt+M` , `Alt+P`) richiedono la configurazione di Option come Meta nel vostro terminale:

- **iTerm2**: impostazioni → Profili → Tasti → impostare il tasto Option sinistro/destro su "Esc+"
- **Terminal.app**: impostazioni → Profili → Tastiera → selezionare "Usa Option come Meta Key"
- **VS Code**: impostazioni → Profili → Tasti → impostare il tasto Option sinistro/destro su "Esc+"

Vedere Configurazione del terminale per i dettagli.

Controlli generali

Scorciatoia	Descrizione	Contesto
`Ctrl+C`	Annulla l'input corrente o la generazione	Interruzione standard
`Ctrl+F`	Termina tutti gli agenti in background. Premere due volte entro 3 secondi per confermare	Controllo agente background
`Ctrl+D`	Esci dalla sessione di Claude Code	Segnale EOF
`Ctrl+G`	Apri nell'editor di testo predefinito	Modifica il vostro prompt o la risposta personalizzata nell'editor di testo predefinito

Scorciatoia	Descrizione	Contesto
`Ctrl+L`	Cancella lo schermo del terminale	Mantiene la cronologia della conversazione
`Ctrl+O`	Attiva/disattiva l'output dettagliato	Mostra l'utilizzo dettagliato degli strumenti e l'esecuzione
`Ctrl+R`	Ricerca inversa nella cronologia dei comandi	Cerca i comandi precedenti in modo interattivo
`Ctrl+V` o `Cmd+V` (iTerm2) o `Alt+V` (Windows)	Incolla immagine dagli appunti	Incolla un'immagine o un percorso a un file immagine
`Ctrl+B`	Attività in esecuzione in background	Esegue i comandi bash e gli agenti in background. Gli utenti Tmux premono due volte
`Ctrl+T`	Attiva/disattiva l'elenco delle attività	Mostra o nascondi l'elenco delle attività nell'area di stato del terminale
`Frecce sinistra/destra`	Scorrere le schede della finestra di dialogo	Navigare tra le schede nelle finestre di dialogo delle autorizzazioni e nei menu
`Frecce su/giù`	Navigare nella cronologia dei comandi	Richiamare gli input precedenti
`Esc` + `Esc`	Riavvolgi o riassumi	Ripristina il codice e/o la conversazione a un punto precedente, o riassumi da un messaggio selezionato
`Shift+Tab` o `Alt+M` (alcune configurazioni)	Attiva/disattiva le modalità di autorizzazione	Passa tra la modalità Auto-Accept, Plan Mode e la modalità normale.
`Option+P` (macOS) o `Alt+P` (Windows/Linux)	Cambia modello	Cambia modelli senza cancellare il vostro prompt

Scorciatoia	Descrizione	Contesto
`Option+T` (macOS) o `Alt+T` (Windows/Linux)	Attiva/disattiva il pensiero esteso	Abilita o disabilita la modalità di pensiero esteso. Eseguire prima `/terminal-setup` per abilitare questa scorciatoia

Modifica del testo

Scorciatoia	Descrizione	Contesto
`Ctrl+K`	Elimina fino alla fine della riga	Memorizza il testo eliminato per l'incollamento
`Ctrl+U`	Elimina l'intera riga	Memorizza il testo eliminato per l'incollamento
`Ctrl+Y`	Incolla il testo eliminato	Incolla il testo eliminato con `Ctrl+K` o `Ctrl+U`
`Alt+Y` (dopo `Ctrl+Y`)	Scorrere la cronologia degli incollamenti	Dopo l'incollamento, scorrere il testo precedentemente eliminato. Richiede Option come Meta su macOS
`Alt+B`	Sposta il cursore indietro di una parola	Navigazione per parole. Richiede Option come Meta su macOS
`Alt+F`	Sposta il cursore in avanti di una parola	Navigazione per parole. Richiede Option come Meta su macOS

Tema e visualizzazione

Scorciatoia	Descrizione	Contesto
`Ctrl+T`	Attiva/disattiva l'evidenziazione della sintassi per i blocchi di codice	Funziona solo all'interno del menu di selezione `/theme`. Controlla se il codice nelle risposte di Claude utilizza la colorazione della sintassi

> **Note:**
>
> L'evidenziazione della sintassi è disponibile solo nella build nativa di Claude Code.

Input multilinea

Metodo	Scorciatoia	Contesto
Escape rapido	`\` + `Enter`	Funziona in tutti i terminali
Predefinito macOS	`Option+Enter`	Predefinito su macOS
Shift+Enter	`Shift+Enter`	Funziona immediatamente in iTerm2, WezTerm, Ghostty, Kitty
Sequenza di controllo	`Ctrl+J`	Carattere di avanzamento riga per multilinea
Modalità incolla	Incolla direttamente	Per blocchi di codice, log

> **Tip:**
>
> Shift+Enter funziona senza configurazione in iTerm2, WezTerm, Ghostty e Kitty. Per altri terminali (VS Code, Alacritty, Zed, Warp), eseguire `/terminal-setup` per installare il binding.

Comandi rapidi

Scorciatoia	Descrizione	Note
`/` all'inizio	Comando o skill	Vedere comandi integrati e skills
`!` all'inizio	Modalità Bash	Esegui i comandi direttamente e aggiungi l'output di esecuzione alla sessione
`@`	Menzione del percorso del file	Attiva l'autocompletamento del percorso del file

Comandi integrati

Digitare `/` in Claude Code per visualizzare tutti i comandi disponibili, oppure digitare `/` seguito da qualsiasi lettera per filtrare. Non tutti i comandi sono visibili a ogni utente. Alcuni dipendono dalla vostra piattaforma, dal piano o dall'ambiente. Ad esempio, `/desktop` appare solo su macOS e Windows, `/upgrade` e `/privacy-settings` sono disponibili solo per i piani Pro e Max, e `/terminal-setup` è nascosto quando il vostro terminale supporta nativamente i suoi keybindings.

Claude Code viene anche fornito con skills in bundle come `/simplify`, `/batch` e `/debug` che appaiono insieme ai comandi integrati quando digitate `/`. Per creare i vostri comandi, vedere skills.

Nella tabella sottostante, `<arg>` indica un argomento obbligatorio e `[arg]` indica uno facoltativo.

Comando	Scopo
`/add-dir <path>`	Aggiungi una nuova directory di lavoro alla sessione corrente
`/agents`	Gestisci le configurazioni agent
`/btw <question>`	Fai una rapida domanda laterale senza aggiungere alla conversazione
`/chrome`	Configura le impostazioni di Claude in Chrome
`/clear`	Cancella la cronologia della conversazione e libera il contesto. Alias: `/reset`, `/new`
`/compact [instructions]`	Compatta la conversazione con istruzioni di focus facoltative
`/config`	Apri l'interfaccia Impostazioni per regolare il tema, il modello, lo stile di output e altre preferenze. Alias: `/settings`
`/context`	Visualizza l'utilizzo del contesto corrente come una griglia colorata
`/copy`	Copia l'ultima risposta dell'assistente negli appunti. Quando sono presenti blocchi di codice, mostra un selettore interattivo per selezionare singoli blocchi o la risposta completa
`/cost`	Mostra le statistiche di utilizzo dei token. Vedere la guida al tracciamento dei costi per i dettagli specifici dell'abbonamento

Comando	Scopo
`/desktop`	Continua la sessione corrente nell'app Claude Code Desktop. Solo macOS e Windows. Alias: `/app`
`/diff`	Apri un visualizzatore diff interattivo che mostra le modifiche non sottoposte a commit e i diff per turno. Usa le frecce sinistra/destra per passare tra il diff git corrente e i singoli turni di Claude, e su/giù per sfogliare i file
`/doctor`	Diagnostica e verifica l'installazione e le impostazioni di Claude Code
`/exit`	Esci dalla CLI. Alias: `/quit`
`/export [filename]`	Esporta la conversazione corrente come testo semplice. Con un nome file, scrive direttamente in quel file. Senza, apre una finestra di dialogo per copiare negli appunti o salvare in un file
`/extra-usage`	Configura l'utilizzo extra per continuare a lavorare quando vengono raggiunti i limiti di velocità
`/fast [on|off]`	Attiva/disattiva la modalità veloce
`/feedback [report]`	Invia feedback su Claude Code. Alias: `/bug`
`/fork [name]`	Crea un fork della conversazione corrente a questo punto
`/help`	Mostra la guida e i comandi disponibili
`/hooks`	Gestisci le configurazioni hook per gli eventi degli strumenti
`/ide`	Gestisci le integrazioni IDE e mostra lo stato
`/init`	Inizializza il progetto con la guida `CLAUDE.md`
`/insights`	Genera un rapporto che analizza le vostre sessioni di Claude Code, incluse le aree del progetto, i modelli di interazione e i punti di attrito
`/install-github-app`	Configura l'app Claude GitHub Actions per un repository. Vi guida nella selezione di un repo e nella configurazione dell'integrazione
`/install-slack-app`	Installa l'app Claude Slack. Apre un browser per completare il flusso OAuth
`/keybindings`	Apri o crea il vostro file di configurazione delle scorciatoie da tastiera
`/login`	Accedi al vostro account Anthropic

Comando	Scopo
`/logout`	Esci dal vostro account Anthropic
`/mcp`	Gestisci le connessioni ai server MCP e l'autenticazione OAuth
`/memory`	Modifica i file di memoria `CLAUDE.md`, abilita o disabilita la memoria automatica e visualizza le voci di memoria automatica
`/mobile`	Mostra il codice QR per scaricare l'app mobile Claude. Alias: `/ios`, `/android`
`/model [model]`	Seleziona o cambia il modello di IA. Per i modelli che lo supportano, usa le frecce sinistra/destra per regolare il livello di sforzo. Il cambio ha effetto immediato senza aspettare il completamento della risposta corrente
`/passes`	Condividi una settimana gratuita di Claude Code con gli amici. Visibile solo se il vostro account è idoneo
`/permissions`	Visualizza o aggiorna le autorizzazioni. Alias: `/allowed-tools`
`/plan`	Entra direttamente in plan mode dal prompt
`/plugin`	Gestisci i plugins di Claude Code
`/pr-comments [PR]`	Recupera e visualizza i commenti da una pull request di GitHub. Rileva automaticamente il PR per il ramo corrente, oppure passa un URL o un numero di PR. Richiede la CLI `gh`
`/privacy-settings`	Visualizza e aggiorna le vostre impostazioni di privacy. Disponibile solo per gli abbonati ai piani Pro e Max
`/release-notes`	Visualizza il changelog completo, con la versione più recente più vicina al vostro prompt
`/reload-plugins`	Ricarica tutti i plugins attivi per applicare le modifiche in sospeso senza riavviare. Segnala cosa è stato caricato e nota le modifiche che richiedono un riavvio
`/remote-control`	Rendi questa sessione disponibile per il controllo remoto da claude.ai. Alias: `/rc`
`/remote-env`	Configura l'ambiente remoto predefinito per le sessioni teleport
`/rename [name]`	Rinomina la sessione corrente. Senza un nome, ne genera uno automaticamente dalla cronologia della conversazione

Comando	Scopo
`/resume [session]`	Riprendi una conversazione per ID o nome, oppure apri il selettore di sessione. Alias: `/continue`
`/review`	Deprecato. Installa invece il `code-review` plugin: `claude plugin install code-review@claude-code-marketplace`
`/rewind`	Riavvolgi la conversazione e/o il codice a un punto precedente, o riassumi da un messaggio selezionato. Vedere checkpointing. Alias: `/checkpoint`
`/sandbox`	Attiva/disattiva la modalità sandbox. Disponibile solo su piattaforme supportate
`/security-review`	Analizza le modifiche in sospeso sul ramo corrente per le vulnerabilità di sicurezza. Esamina il diff git e identifica i rischi come iniezione, problemi di autenticazione e esposizione dei dati
`/skills`	Elenca le skills disponibili
`/stats`	Visualizza l'utilizzo giornaliero, la cronologia delle sessioni, le serie e le preferenze dei modelli
`/status`	Apri l'interfaccia Impostazioni (scheda Stato) che mostra la versione, il modello, l'account e la connettività
`/statusline`	Configura la status line di Claude Code. Descrivi cosa desideri, oppure esegui senza argomenti per auto-configurare dal vostro prompt della shell
`/stickers`	Ordina gli adesivi di Claude Code
`/tasks`	Elenca e gestisci le attività in background
`/terminal-setup`	Configura i keybindings del terminale per Shift+Enter e altre scorciatoie. Visibile solo nei terminali che ne hanno bisogno, come VS Code, Alacritty o Warp
`/theme`	Cambia il tema del colore. Include varianti chiare e scure, temi accessibili ai daltonici (daltonizzati) e temi ANSI che utilizzano la tavolozza dei colori del vostro terminale
`/upgrade`	Apri la pagina di upgrade per passare a un livello di piano superiore
`/usage`	Mostra i limiti di utilizzo del piano e lo stato del limite di velocità
`/vim`	Attiva/disattiva tra le modalità di modifica Vim e Normale

Prompt MCP

I server MCP possono esporre prompt che appaiono come comandi. Questi utilizzano il formato `/mcp__<server>__<prompt>` e vengono scoperti dinamicamente dai server connessi. Vedere Prompt MCP per i dettagli.

Modalità editor Vim

Abilita la modifica in stile Vim con il comando `/vim` o configura permanentemente tramite `/config`.

Cambio di modalità

Comando	Azione	Dalla modalità
`Esc`	Entra in modalità NORMAL	INSERT
`i`	Inserisci prima del cursore	NORMAL
`I`	Inserisci all'inizio della riga	NORMAL
`a`	Inserisci dopo il cursore	NORMAL
`A`	Inserisci alla fine della riga	NORMAL
`o`	Apri riga sotto	NORMAL
`O`	Apri riga sopra	NORMAL

Navigazione (modalità NORMAL)

Comando	Azione
`h` / `j` / `k` / `l`	Sposta sinistra/giù/su/destra
`w`	Parola successiva
`e`	Fine della parola
`b`	Parola precedente
`0`	Inizio della riga
`$`	Fine della riga
`^`	Primo carattere non vuoto

Comando	Azione
`gg`	Inizio dell'input
`G`	Fine dell'input
`f{char}`	Salta alla prossima occorrenza del carattere
`F{char}`	Salta alla precedente occorrenza del carattere
`t{char}`	Salta appena prima della prossima occorrenza del carattere
`T{char}`	Salta appena dopo la precedente occorrenza del carattere
`;`	Ripeti l'ultimo movimento f/F/t/T
`,`	Ripeti l'ultimo movimento f/F/t/T in ordine inverso

> **Note:**
>
> In modalità normale vim, se il cursore è all'inizio o alla fine dell'input e non può muoversi ulteriormente, i tasti freccia navigano nella cronologia dei comandi.

Modifica (modalità NORMAL)

Comando	Azione
`x`	Elimina carattere
`dd`	Elimina riga
`D`	Elimina fino alla fine della riga
`dw` / `de` / `db`	Elimina parola/fino alla fine/indietro
`cc`	Cambia riga
`C`	Cambia fino alla fine della riga
`cw` / `ce` / `cb`	Cambia parola/fino alla fine/indietro
`yy` / `Y`	Copia (yank) riga
`yw` / `ye` / `yb`	Copia parola/fino alla fine/indietro
`p`	Incolla dopo il cursore

Comando	Azione
`P`	Incolla prima del cursore
`>>`	Indenta riga
`<<`	Dedenta riga
`J`	Unisci righe
`.`	Ripeti l'ultima modifica

Oggetti di testo (modalità NORMAL)

Gli oggetti di testo funzionano con operatori come `d`, `c` e `y`:

Comando	Azione
`iw` / `aw`	Parola interna/intorno
`iW` / `aW`	PAROLA interna/intorno (delimitata da spazi bianchi)
`i"` / `a"`	Interno/intorno a virgolette doppie
`i'` / `a'`	Interno/intorno a virgolette singole
`i(` / `a(`	Interno/intorno a parentesi
`i[` / `a[`	Interno/intorno a parentesi quadre
`i{` / `a{`	Interno/intorno a parentesi graffe

Cronologia dei comandi

Claude Code mantiene la cronologia dei comandi per la sessione corrente:

- La cronologia degli input viene memorizzata per directory di lavoro
- La cronologia degli input si ripristina quando eseguite `/clear` per avviare una nuova sessione. La conversazione della sessione precedente viene preservata e può essere ripresa.
- Usate le frecce Su/Giù per navigare (vedere le scorciatoie da tastiera sopra)
- **Nota**: l'espansione della cronologia (`!`) è disabilitata per impostazione predefinita

Ricerca inversa con Ctrl+R

Premere `Ctrl+R` per cercare in modo interattivo nella vostra cronologia dei comandi:

1. **Avvia la ricerca**: premere `Ctrl+R` per attivare la ricerca inversa nella cronologia
2. **Digita la query**: inserire il testo da cercare nei comandi precedenti. Il termine di ricerca è evidenziato nei risultati corrispondenti
3. **Navigare i risultati**: premere `Ctrl+R` di nuovo per scorrere i risultati più vecchi
4. **Accetta il risultato**:
   - Premere `Tab` o `Esc` per accettare il risultato corrente e continuare a modificare
   - Premere `Enter` per accettare ed eseguire il comando immediatamente
5. **Annulla la ricerca**:
   - Premere `Ctrl+C` per annullare e ripristinare l'input originale
   - Premere `Backspace` su una ricerca vuota per annullare

La ricerca visualizza i comandi corrispondenti con il termine di ricerca evidenziato, in modo da poter trovare e riutilizzare gli input precedenti.

Comandi bash in background

Claude Code supporta l'esecuzione di comandi bash in background, consentendovi di continuare a lavorare mentre i processi a lunga esecuzione vengono eseguiti.

Come funziona l'esecuzione in background

Quando Claude Code esegue un comando in background, esegue il comando in modo asincrono e restituisce immediatamente un ID di attività in background. Claude Code può rispondere a nuovi prompt mentre il comando continua a essere eseguito in background.

Per eseguire i comandi in background, potete:

- Chiedere a Claude Code di eseguire un comando in background
- Premere Ctrl+B per spostare una normale invocazione dello strumento Bash in background. (Gli utenti Tmux devono premere Ctrl+B due volte a causa del tasto di prefisso di tmux.)

Caratteristiche principali:

- L'output viene memorizzato nel buffer e Claude può recuperarlo utilizzando lo strumento TaskOutput
- Le attività in background hanno ID univoci per il tracciamento e il recupero dell'output

- Le attività in background vengono pulite automaticamente quando Claude Code esce

Per disabilitare tutta la funzionalità di attività in background, impostare la variabile di ambiente `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` su `1`. Vedere Variabili di ambiente per i dettagli.

Comandi comunemente eseguiti in background:

- Strumenti di build (webpack, vite, make)
- Gestori di pacchetti (npm, yarn, pnpm)
- Test runner (jest, pytest)
- Server di sviluppo
- Processi a lunga esecuzione (docker, terraform)

Modalità Bash con prefisso `!`

Esegui i comandi bash direttamente senza passare per Claude aggiungendo il prefisso `!` al vostro input:

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

Modalità Bash:

- Aggiunge il comando e il suo output al contesto della conversazione
- Mostra il progresso e l'output in tempo reale
- Supporta lo stesso backgrounding `Ctrl+B` per i comandi a lunga esecuzione
- Non richiede a Claude di interpretare o approvare il comando
- Supporta l'autocompletamento basato sulla cronologia: digitate un comando parziale e premete **Tab** per completare dai comandi `!` precedenti nel progetto corrente
- Esci con `Escape`, `Backspace` o `Ctrl+U` su un prompt vuoto

Questo è utile per le operazioni shell rapide mantenendo il contesto della conversazione.

Suggerimenti di prompt

Quando aprite una sessione per la prima volta, un comando di esempio in grigio appare nell'input del prompt per aiutarvi a iniziare. Claude Code lo sceglie dalla cronologia git del vostro progetto, quindi riflette i file su cui avete lavorato di recente.

Dopo che Claude risponde, i suggerimenti continuano ad apparire in base alla vostra cronologia della conversazione, come un passaggio di follow-up da una richiesta in più parti o una continuazione naturale del vostro flusso di lavoro.

- Premete **Tab** per accettare il suggerimento, oppure premete **Enter** per accettare e inviare
- Iniziate a digitare per dismissarlo

Il suggerimento viene eseguito come una richiesta in background che riutilizza la cache del prompt della conversazione padre, quindi il costo aggiuntivo è minimo. Claude Code salta la generazione di suggerimenti quando la cache è fredda per evitare costi inutili.

I suggerimenti vengono automaticamente saltati dopo il primo turno di una conversazione, in modalità non interattiva e in plan mode.

Per disabilitare completamente i suggerimenti di prompt, impostare la variabile di ambiente o attivare l'impostazione in `/config`:

```
export CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION=false
```

Domande laterali con /btw

Usate `/btw` per fare una domanda rapida sul vostro lavoro corrente senza aggiungere alla cronologia della conversazione. Questo è utile quando desiderate una risposta veloce ma non volete ingombrare il contesto principale o far deviare Claude da un'attività a lunga esecuzione.

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

Le domande laterali hanno piena visibilità nella conversazione corrente, quindi potete chiedere informazioni sul codice che Claude ha già letto, sulle decisioni che ha preso in precedenza, o su qualsiasi altra cosa della sessione. La domanda e la risposta sono effimere: appaiono in un overlay dismissibile e non entrano mai nella cronologia della conversazione.

- **Disponibile mentre Claude sta lavorando**: potete eseguire `/btw` anche mentre Claude sta elaborando una risposta. La domanda laterale viene eseguita in modo indipendente e non interrompe il turno principale.

- **Nessun accesso agli strumenti**: le domande laterali rispondono solo da ciò che è già nel contesto. Claude non può leggere file, eseguire comandi o cercare quando risponde a una domanda laterale.
- **Risposta singola**: non ci sono turni di follow-up. Se avete bisogno di un dialogo, usate un prompt normale.
- **Costo basso**: la domanda laterale riutilizza la cache del prompt della conversazione padre, quindi il costo aggiuntivo è minimo.

Premete **Spazio**, **Enter** o **Escape** per dismissere la risposta e tornare al prompt.

`/btw` è l'inverso di un subagent: vede la vostra conversazione completa ma non ha strumenti, mentre un subagent ha strumenti completi ma inizia con un contesto vuoto. Usate `/btw` per chiedere informazioni su ciò che Claude già conosce da questa sessione; usate un subagent per scoprire qualcosa di nuovo.

Elenco delle attività

Quando si lavora su lavori complessi e multistep, Claude crea un elenco di attività per tracciare i progressi. Le attività appaiono nell'area di stato del vostro terminale con indicatori che mostrano cosa è in sospeso, in corso o completato.

- Premete `Ctrl+T` per attivare/disattivare la visualizzazione dell'elenco delle attività. La visualizzazione mostra fino a 10 attività alla volta
- Per visualizzare tutte le attività o cancellarle, chiedete direttamente a Claude: "mostrami tutte le attività" o "cancella tutte le attività"
- Le attività persistono attraverso le compattazioni del contesto, aiutando Claude a rimanere organizzato su progetti più grandi
- Per condividere un elenco di attività tra sessioni, impostare `CLAUDE_CODE_TASK_LIST_ID` per utilizzare una directory denominata in `~/.claude/tasks/` : `CLAUDE_CODE_TASK_LIST_ID=my-project claude`
- Per ripristinare l'elenco TODO precedente, impostare `CLAUDE_CODE_ENABLE_TASKS=false` .

Stato della revisione PR

Quando si lavora su un ramo con una pull request aperta, Claude Code visualizza un collegamento PR cliccabile nel footer (ad esempio, "PR #446"). Il collegamento ha un sottolineatura colorata che indica lo stato della revisione:

- Verde: approvato
- Giallo: revisione in sospeso

- Rosso: modifiche richieste
- Grigio: bozza
- Viola: unito

`Cmd+click` (Mac) o `Ctrl+click` (Windows/Linux) sul collegamento per aprire la pull request nel vostro browser. Lo stato si aggiorna automaticamente ogni 60 secondi.

> **Note:**
>
> Lo stato PR richiede che la CLI `gh` sia installata e autenticata (`gh auth login`).

Vedere anche

- Skills - Prompt e flussi di lavoro personalizzati
- Checkpointing - Riavvolgi le modifiche di Claude e ripristina gli stati precedenti
- Riferimento CLI - Flag e opzioni della riga di comando
- Impostazioni - Opzioni di configurazione
- Gestione della memoria - Gestione dei file CLAUDE.md

Best Practices per Claude Code

> *Suggerimenti e modelli per ottenere il massimo da Claude Code, dalla configurazione dell'ambiente al ridimensionamento tra sessioni parallele.*

Claude Code è un ambiente di codifica agenziale. A differenza di un chatbot che risponde alle domande e aspetta, Claude Code può leggere i vostri file, eseguire comandi, apportare modifiche e lavorare autonomamente attraverso i problemi mentre voi guardate, reindirizzate o vi allontanate completamente.

Questo cambia il modo in cui lavorate. Invece di scrivere il codice voi stessi e chiedere a Claude di rivederlo, descrivete quello che volete e Claude capisce come costruirlo. Claude esplora, pianifica e implementa.

Ma questa autonomia comporta comunque una curva di apprendimento. Claude lavora all'interno di determinati vincoli che dovete comprendere.

Questa guida copre i modelli che si sono dimostrati efficaci nei team interni di Anthropic e per gli ingegneri che utilizzano Claude Code in vari codebase, linguaggi e ambienti. Per sapere come funziona il ciclo agenziale sotto il cofano, consultate How Claude Code works.

La maggior parte delle best practice si basa su un vincolo: la finestra di contesto di Claude si riempie rapidamente e le prestazioni si degradano man mano che si riempie.

La finestra di contesto di Claude contiene l'intera conversazione, inclusi ogni messaggio, ogni file che Claude legge e ogni output di comando. Tuttavia, può riempirsi rapidamente. Una singola sessione di debug o esplorazione del codebase potrebbe generare e consumare decine di migliaia di token.

Questo è importante perché le prestazioni dell'LLM si degradano man mano che il contesto si riempie. Quando la finestra di contesto sta per riempirsi, Claude potrebbe iniziare a "dimenticare" le istruzioni precedenti o fare più errori. La finestra di contesto è la risorsa più importante da gestire. Monitorate continuamente l'utilizzo del contesto con una custom status line e consultate Reduce token usage per strategie su come ridurre l'utilizzo dei token.

Fornite a Claude un modo per verificare il suo lavoro

> **Tip:**
>
> Includete test, screenshot o output previsti in modo che Claude possa verificare se stesso. Questa è la cosa singola con il massimo effetto leva che potete fare.

Claude funziona drammaticamente meglio quando può verificare il suo lavoro, come eseguire test, confrontare screenshot e convalidare output.

Senza criteri di successo chiari, potrebbe produrre qualcosa che sembra giusto ma in realtà non funziona. Voi diventate l'unico ciclo di feedback e ogni errore richiede la vostra attenzione.

Strategia	Prima	Dopo
Fornire criteri di verifica	*"implementare una funzione che convalida gli indirizzi email"*	*"scrivere una funzione validateEmail. esempi di casi di test: user@example.com è true, invalid è false, user@.com è false. eseguire i test dopo l'implementazione"*
Verificare visivamente i cambiamenti dell'interfaccia utente	*"rendere il dashboard più bello"*	*"[incollare screenshot] implementare questo design. fare uno screenshot del risultato e confrontarlo con l'originale. elencare le differenze e correggerle"*
Affrontare le cause radici, non i sintomi	*"la build sta fallendo"*	*"la build fallisce con questo errore: [incollare errore]. correggerlo e verificare che la build abbia successo. affrontare la causa radice, non sopprimere l'errore"*

I cambiamenti dell'interfaccia utente possono essere verificati utilizzando l'estensione Claude in Chrome. Apre nuove schede nel vostro browser, testa l'interfaccia utente e itera fino a quando il codice non funziona.

La vostra verifica può anche essere una suite di test, un linter o un comando Bash che controlla l'output. Investite nel rendere la vostra verifica solida.

Esplorate prima, poi pianificate, poi codificate

> **Tip:**
>
> Separate la ricerca e la pianificazione dall'implementazione per evitare di risolvere il problema sbagliato.

Lasciare che Claude salti direttamente alla codifica può produrre codice che risolve il problema sbagliato. Utilizzate Plan Mode per separare l'esplorazione dall'esecuzione.

Il flusso di lavoro consigliato ha quattro fasi:

Step 1: Esplora

Entra in Plan Mode. Claude legge i file e risponde alle domande senza apportare modifiche.

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

Step 2: Pianifica

Chiedi a Claude di creare un piano di implementazione dettagliato.

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

Premi `Ctrl+G` per aprire il piano nel vostro editor di testo per la modifica diretta prima che Claude proceda.

Step 3: Implementa

Torna alla Normal Mode e lascia che Claude codifichi, verificando rispetto al suo piano.

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

Step 4: Commit

Chiedi a Claude di eseguire il commit con un messaggio descrittivo e creare una PR.

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

Plan Mode è utile, ma aggiunge anche overhead.

Per attività in cui l'ambito è chiaro e la correzione è piccola (come correggere un errore di battitura, aggiungere una riga di log o rinominare una variabile) chiedete a Claude di farlo direttamente.

La pianificazione è più utile quando siete incerti sull'approccio, quando il cambiamento modifica più file o quando non siete familiari con il codice da modificare. Se potete descrivere il diff in una frase, saltate il piano.

Fornite contesto specifico nei vostri prompt

> **Tip:**
>
> Più precise sono le vostre istruzioni, meno correzioni vi serviranno.

Claude può dedurre l'intento, ma non può leggere la mente. Fate riferimento a file specifici, menzionate i vincoli e indicate i modelli di esempio.

Strategia	Prima	Dopo
Definire l'ambito del compito. Specificate quale file, quale scenario e le preferenze di test.	*"aggiungere test per foo.py"*	*"scrivere un test per foo.py che copra il caso limite in cui l'utente è disconnesso. evitare i mock."*
Puntare alle fonti. Indirizzate Claude alla fonte che può rispondere a una domanda.	*"perché ExecutionFactory ha un'API così strana?"*	*"guardare la cronologia git di ExecutionFactory e riassumere come la sua API è arrivata a essere così"*

Strategia	Prima	Dopo
Fare riferimento ai modelli esistenti. Puntate Claude ai modelli nel vostro codebase.	*“aggiungere un widget calendario”*	*“guardare come i widget esistenti sono implementati nella home page per comprendere i modelli. HotDogWidget.php è un buon esempio. seguire il modello per implementare un nuovo widget calendario che consenta all’utente di selezionare un mese e paginare avanti/indietro per scegliere un anno. costruire da zero senza librerie diverse da quelle già utilizzate nel codebase.”*
Descrivere il sintomo. Fornite il sintomo, la probabile posizione e come appare “risolto”.	*“correggere il bug di login”*	*“gli utenti segnalano che il login fallisce dopo il timeout della sessione. controllare il flusso di autenticazione in src/auth/, in particolare l’aggiornamento del token. scrivere un test fallito che riproduce il problema, quindi correggerlo”*

I prompt vaghi possono essere utili quando state esplorando e potete permettervi di correggere il corso. Un prompt come `"cosa migliorereste in questo file?"` può evidenziare cose a cui non avreste pensato di chiedere.

Fornire contenuti ricchi

> **Tip:**
>
> Utilizzate `@` per fare riferimento ai file, incollate screenshot/immagini o inviate i dati direttamente.

Potete fornire dati ricchi a Claude in diversi modi:

- **Fare riferimento ai file con** `@` invece di descrivere dove vive il codice. Claude legge il file prima di rispondere.
- **Incollare le immagini direttamente**. Copiate/incollate o trascinate le immagini nel prompt.
- **Fornire URL** per la documentazione e i riferimenti API. Utilizzate `/permissions` per aggiungere alla whitelist i domini utilizzati di frequente.
- **Inviare i dati tramite pipe** eseguendo `cat error.log | claude` per inviare il contenuto del file direttamente.
- **Lasciare che Claude recuperi quello di cui ha bisogno**. Dite a Claude di estrarre il contesto stesso utilizzando comandi Bash, strumenti MCP o leggendo i file.

Configurate il vostro ambiente

Alcuni passaggi di configurazione rendono Claude Code significativamente più efficace in tutte le vostre sessioni. Per una panoramica completa delle funzionalità dell'estensione e di quando utilizzare ciascuna, consultate Extend Claude Code.

Scrivete un CLAUDE.md efficace

> **Tip:**
>
> Eseguite `/init` per generare un file CLAUDE.md iniziale basato sulla struttura del vostro progetto attuale, quindi affinate nel tempo.

CLAUDE.md è un file speciale che Claude legge all'inizio di ogni conversazione. Includete comandi Bash, stile del codice e regole del flusso di lavoro. Questo fornisce a Claude un contesto persistente che non può dedurre dal solo codice.

Il comando `/init` analizza il vostro codebase per rilevare sistemi di build, framework di test e modelli di codice, fornendovi una base solida da affinare.

Non esiste un formato obbligatorio per i file CLAUDE.md, ma manteneteli brevi e leggibili. Ad esempio:

```
## 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 viene caricato ogni sessione, quindi includete solo le cose che si applicano ampiamente. Per la conoscenza del dominio o i flussi di lavoro che sono rilevanti solo a volte, utilizzate skills invece. Claude li carica su richiesta senza gonfiare ogni conversazione.

Mantenetelo conciso. Per ogni riga, chiedetevi: *"Rimuovere questo causerebbe a Claude di fare errori?"* Se no, tagliatelo. I file CLAUDE.md gonfi causano a Claude di ignorare le vostre istruzioni effettive!

✅ Includere	❌ Escludere
Comandi Bash che Claude non può indovinare	Qualsiasi cosa Claude possa capire leggendo il codice
Regole di stile del codice che differiscono dai valori predefiniti	Convenzioni linguistiche standard che Claude già conosce
Istruzioni di test e runner di test preferiti	Documentazione API dettagliata (collegare alla documentazione invece)
Etichetta del repository (denominazione dei rami, convenzioni PR)	Informazioni che cambiano frequentemente
Decisioni architettoniche specifiche del vostro progetto	Spiegazioni lunghe o tutorial
Stranezze dell'ambiente di sviluppo (variabili env richieste)	Descrizioni file per file del codebase
Gotcha comuni o comportamenti non ovvi	Pratiche auto-evidenti come "scrivere codice pulito"

Se Claude continua a fare qualcosa che non volete nonostante abbia una regola contro di essa, il file è probabilmente troppo lungo e la regola si sta perdendo. Se Claude vi fa domande che sono risposte in CLAUDE.md, la formulazione potrebbe essere ambigua.

Trattate CLAUDE.md come codice: rivederlo quando le cose vanno male, potarlo regolarmente e testate i cambiamenti osservando se il comportamento di Claude effettivamente cambia.

Potete sintonizzare le istruzioni aggiungendo enfasi (ad es. “IMPORTANTE” o “DOVETE”) per migliorare l’aderenza. Controllate CLAUDE.md in git in modo che il vostro team possa contribuire. Il file aumenta di valore nel tempo.

I file CLAUDE.md possono importare file aggiuntivi utilizzando la sintassi `@path/to/import`:

```
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
```

Potete posizionare i file CLAUDE.md in diversi percorsi:

- **Cartella home (`~/.claude/CLAUDE.md`)**: si applica a tutte le sessioni Claude
- **Radice del progetto (`./CLAUDE.md`)**: controllare in git per condividere con il vostro team
- **Directory padre**: utile per monorepo dove sia `root/CLAUDE.md` che `root/foo/CLAUDE.md` vengono estratti automaticamente
- **Directory figlie**: Claude estrae i file CLAUDE.md figlio su richiesta quando lavora con i file in quelle directory

Configurate i permessi

> **Tip:**
>
> Utilizzate `/permissions` per aggiungere alla whitelist i comandi sicuri o `/sandbox` per l’isolamento a livello di sistema operativo. Questo riduce le interruzioni mantenendovi il controllo.

Per impostazione predefinita, Claude Code richiede il permesso per azioni che potrebbero modificare il vostro sistema: scritture di file, comandi Bash, strumenti MCP, ecc. Questo è sicuro ma tedioso. Dopo la decima approvazione non state davvero revisionando più, state solo cliccando. Ci sono due modi per ridurre queste interruzioni:

- **Whitelist di permessi**: consentire strumenti specifici che sapete essere sicuri (come `npm run lint` o `git commit`)

- **Sandboxing**: abilitare l'isolamento a livello di sistema operativo che limita l'accesso al filesystem e alla rete, consentendo a Claude di lavorare più liberamente all'interno di confini definiti

In alternativa, utilizzate `--dangerously-skip-permissions` per aggirare tutti i controlli di permesso per flussi di lavoro contenuti come correggere errori di lint o generare boilerplate.

> **Warning:**
>
> Lasciare che Claude esegua comandi arbitrari può causare perdita di dati, corruzione del sistema o esfiltrazione di dati tramite prompt injection. Utilizzate `--dangerously-skip-permissions` solo in una sandbox senza accesso a Internet.

Leggete di più su configuring permissions e enabling sandboxing.

Utilizzate gli strumenti CLI

> **Tip:**
>
> Dite a Claude Code di utilizzare strumenti CLI come `gh`, `aws`, `gcloud` e `sentry-cli` quando interagite con servizi esterni.

Gli strumenti CLI sono il modo più efficiente in termini di contesto per interagire con servizi esterni. Se utilizzate GitHub, installate la CLI `gh`. Claude sa come usarla per creare problemi, aprire pull request e leggere commenti. Senza `gh`, Claude può comunque utilizzare l'API GitHub, ma le richieste non autenticate spesso raggiungono i limiti di velocità.

Claude è anche efficace nell'imparare strumenti CLI che non conosce già. Provate prompt come `Use 'foo-cli-tool --help' to learn about foo tool, then use it to solve A, B, C.`

Connettete i server MCP

> **Tip:**
>
> Eseguite `claude mcp add` per connettere strumenti esterni come Notion, Figma o il vostro database.

Con i server MCP, potete chiedere a Claude di implementare funzionalità da tracker di problemi, interrogare database, analizzare dati di monitoraggio, integrare design da Figma e automatizzare flussi di lavoro.

Configurate gli hook

> **Tip:**
>
> Utilizzate gli hook per azioni che devono accadere ogni volta senza eccezioni.

Gli hook eseguono script automaticamente in punti specifici del flusso di lavoro di Claude. A differenza delle istruzioni CLAUDE.md che sono consultive, gli hook sono deterministici e garantiscono che l'azione accada.

Claude può scrivere hook per voi. Provate prompt come *"Write a hook that runs eslint after every file edit"* o *"Write a hook that blocks writes to the migrations folder."* Eseguite `/hooks` per la configurazione interattiva o modificate `.claude/settings.json` direttamente.

Createte skill

> **Tip:**
>
> Create file `SKILL.md` in `.claude/skills/` per fornire a Claude conoscenza del dominio e flussi di lavoro riutilizzabili.

Le skill estendono la conoscenza di Claude con informazioni specifiche del vostro progetto, team o dominio. Claude le applica automaticamente quando rilevanti, o potete invocarle direttamente con `/skill-name`.

Create una skill aggiungendo una directory con un `SKILL.md` a `.claude/skills/`:

```
---
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/)
```

Le skill possono anche definire flussi di lavoro ripetibili che invocate direttamente:

```
---
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
```

Eseguite `/fix-issue 1234` per invocarlo. Utilizzate `disable-model-invocation: true` per flussi di lavoro con effetti collaterali che volete attivare manualmente.

Create subagent personalizzati

> **Tip:**
>
> Definite assistenti specializzati in `.claude/agents/` che Claude può delegare per attività isolate.

I subagent vengono eseguiti nel loro contesto con il loro set di strumenti consentiti. Sono utili per attività che leggono molti file o necessitano di focus specializzato senza ingombrare la vostra conversazione principale.

```
---
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.
```

Dite a Claude di utilizzare i subagent esplicitamente: *"Use a subagent to review this code for security issues."*

Installate i plugin

> **Tip:**
>
> Eseguite `/plugin` per sfogliare il marketplace. I plugin aggiungono skill, strumenti e integrazioni senza configurazione.

I plugin raggruppano skill, hook, subagent e server MCP in una singola unità installabile dalla comunità e da Anthropic. Se lavorate con un linguaggio tipizzato, installate un plugin di code intelligence per fornire a Claude la navigazione precisa dei simboli e il rilevamento automatico degli errori dopo le modifiche.

Per una guida sulla scelta tra skill, subagent, hook e MCP, consultate Extend Claude Code.

Comunicate efficacemente

Il modo in cui comunicate con Claude Code ha un impatto significativo sulla qualità dei risultati.

Fate domande sul codebase

> **Tip:**
>
> Fate a Claude domande che fareste a un ingegnere senior.

Quando vi onboarding a un nuovo codebase, utilizzate Claude Code per l'apprendimento e l'esplorazione. Potete fare a Claude lo stesso tipo di domande che fareste a un altro ingegnere:

- Come funziona il logging?
- Come faccio a creare un nuovo endpoint API?
- Cosa fa `async move { ... }` sulla riga 134 di `foo.rs` ?
- Quali casi limite gestisce `CustomerOnboardingFlowImpl` ?
- Perché questo codice chiama `foo()` invece di `bar()` sulla riga 333?

Utilizzare Claude Code in questo modo è un flusso di lavoro di onboarding efficace, migliorando il tempo di ramp-up e riducendo il carico su altri ingegneri. Non è richiesto alcun prompt speciale: fate le domande direttamente.

Lasciate che Claude vi intervisti

> **Tip:**
>
> Per funzionalità più grandi, fate intervistare Claude prima. Iniziate con un prompt minimo e chiedete a Claude di intervistarvi utilizzando lo strumento `AskUserQuestion` .

Claude fa domande su cose che potreste non aver ancora considerato, inclusa l'implementazione tecnica, l'interfaccia utente/esperienza utente, i casi limite e i compromessi.

```
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.
```

Una volta completata la specifica, avviate una nuova sessione per eseguirla. La nuova sessione ha un contesto pulito focalizzato interamente sull'implementazione e avete una specifica scritta a cui fare riferimento.

Gestite la vostra sessione

Le conversazioni sono persistenti e reversibili. Utilizzate questo a vostro vantaggio!

Correggete il corso presto e spesso

> **Tip:**
>
> Correggete Claude non appena notate che sta andando fuori strada.

I migliori risultati provengono da cicli di feedback stretti. Sebbene Claude occasionalmente risolva i problemi perfettamente al primo tentativo, correggerlo rapidamente generalmente produce soluzioni migliori più velocemente.

- `Esc`: fermate Claude a metà azione con il tasto `Esc`. Il contesto viene preservato, quindi potete reindirizzare.
- `Esc + Esc` o `/rewind`: premete `Esc` due volte o eseguite `/rewind` per aprire il menu di rewind e ripristinare la conversazione e lo stato del codice precedenti, o riassumere da un messaggio selezionato.
- `"Undo that"`: fate in modo che Claude ripristini le sue modifiche.
- `/clear`: ripristinate il contesto tra attività non correlate. Le sessioni lunghe con contesto irrilevante possono ridurre le prestazioni.

Se avete corretto Claude più di due volte sullo stesso problema in una sessione, il contesto è ingombrato di approcci falliti. Eseguite `/clear` e iniziate di nuovo con un prompt più specifico che incorpori quello che avete imparato. Una sessione pulita con un prompt migliore quasi sempre supera una sessione lunga con correzioni accumulate.

Gestite il contesto aggressivamente

> **Tip:**
>
> Eseguite `/clear` tra attività non correlate per ripristinare il contesto.

Claude Code compatta automaticamente la cronologia della conversazione quando vi avvicinate ai limiti del contesto, il che preserva il codice e le decisioni importanti mentre libera spazio.

Durante le sessioni lunghe, la finestra di contesto di Claude può riempirsi di conversazioni irrilevanti, contenuti di file e comandi. Questo può ridurre le prestazioni e talvolta distrarre Claude.

- Utilizzate `/clear` frequentemente tra le attività per ripristinare completamente la finestra di contesto
- Quando la compattazione automatica si attiva, Claude riassume quello che conta di più, inclusi i modelli di codice, gli stati dei file e le decisioni chiave
- Per un maggiore controllo, eseguite `/compact <instructions>`, come `/compact Focus on the API changes`
- Per compattare solo parte della conversazione, utilizzate `Esc + Esc` o `/rewind`, selezionate un checkpoint di messaggio e scegliete **Summarize from here**. Questo condensa i messaggi da quel punto in poi mantenendo il contesto precedente intatto.
- Personalizzate il comportamento di compattazione in CLAUDE.md con istruzioni come `"When compacting, always preserve the full list of modified files and any test commands"` per assicurare che il contesto critico sopravviva alla riassunzione
- Per domande rapide che non devono rimanere nel contesto, utilizzate `/btw`. La risposta appare in un overlay dismissibile e non entra mai nella cronologia della conversazione, quindi potete controllare un dettaglio senza far crescere il contesto.

Utilizzate i subagent per l'investigazione

> **Tip:**
>
> Delegate la ricerca con `"use subagents to investigate X"`. Esplorano in un contesto separato, mantenendo la vostra conversazione principale pulita per l'implementazione.

Poiché il contesto è il vostro vincolo fondamentale, i subagent sono uno degli strumenti più potenti disponibili. Quando Claude ricerca un codebase legge molti file, tutti i quali consumano il vostro contesto. I subagent vengono eseguiti in finestre di contesto separate e riportano riassunti:

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

Il subagent esplora il codebase, legge i file rilevanti e riporta i risultati, il tutto senza ingombrare la vostra conversazione principale.

Potete anche utilizzare i subagent per la verifica dopo che Claude implementa qualcosa:

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

Riavvolgete con i checkpoint

> **Tip:**
>
> Ogni azione che Claude fa crea un checkpoint. Potete ripristinare la conversazione, il codice o entrambi a qualsiasi checkpoint precedente.

Claude crea automaticamente checkpoint prima delle modifiche. Premete il doppio tasto Escape o eseguite `/rewind` per aprire il menu di rewind. Potete ripristinare solo la conversazione, ripristinare solo il codice, ripristinare entrambi o riassumere da un messaggio selezionato. Consultate Checkpointing per i dettagli.

Invece di pianificare attentamente ogni mossa, potete dire a Claude di provare qualcosa di rischioso. Se non funziona, riavvolgete e provate un approccio diverso. I checkpoint persistono tra le sessioni, quindi potete chiudere il vostro terminale e comunque riavvolgere in seguito.

> **Warning:**
>
> I checkpoint tracciano solo le modifiche apportate *da Claude*, non i processi esterni. Questo non è un sostituto per git.

Riprendete le conversazioni

> **Tip:**
>
> Eseguite `claude --continue` per riprendere da dove avete lasciato, o `--resume` per scegliere tra le sessioni recenti.

Claude Code salva le conversazioni localmente. Quando un'attività si estende su più sessioni, non dovete rispiegare il contesto:

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

Utilizzate `/rename` per dare alle sessioni nomi descrittivi come `"oauth-migration"` o `"debugging-memory-leak"` in modo da poterle trovare in seguito. Trattate le sessioni come rami: diversi flussi di lavoro possono avere contesti separati e persistenti.

Automatizzate e ridimensionate

Una volta che siete efficaci con un Claude, moltiplicate il vostro output con sessioni parallele, modalità non interattiva e modelli di fan-out.

Tutto finora presuppone un umano, un Claude e una conversazione. Ma Claude Code si ridimensiona orizzontalmente. Le tecniche in questa sezione mostrano come potete fare di più.

Eseguite la modalità non interattiva

> **Tip:**
>
> Utilizzate `claude -p "prompt"` in CI, pre-commit hook o script. Aggiungete `--output-format stream-json` per l'output JSON in streaming.

Con `claude -p "your prompt"`, potete eseguire Claude in modo non interattivo, senza una sessione. La modalità non interattiva è il modo in cui integrate Claude nelle pipeline CI, pre-commit hook o qualsiasi flusso di lavoro automatizzato. I formati di output vi permettono di analizzare i risultati a livello di programmazione: testo semplice, JSON o JSON in streaming.

```
## 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
```

Eseguite più sessioni Claude

> **Tip:**
>
> Eseguite più sessioni Claude in parallelo per accelerare lo sviluppo, eseguire esperimenti isolati o avviare flussi di lavoro complessi.

Ci sono tre modi principali per eseguire sessioni parallele:

- App desktop Claude Code: Gestite più sessioni locali visivamente. Ogni sessione ottiene il suo worktree isolato.
- Claude Code sul web: Eseguite su infrastruttura cloud sicura di Anthropic in VM isolate.
- Team di agenti: Coordinamento automatizzato di più sessioni con attività condivise, messaggistica e un team lead.

Oltre a parallelizzare il lavoro, più sessioni consentono flussi di lavoro focalizzati sulla qualità. Un contesto fresco migliora la revisione del codice poiché Claude non sarà distorto verso il codice che ha appena scritto.

Ad esempio, utilizzate un modello Writer/Reviewer:

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

Potete fare qualcosa di simile con i test: fate scrivere i test a un Claude, poi a un altro scrivere il codice per passarli.

Fan out tra i file

> **Tip:**
>
> Eseguite un ciclo attraverso le attività chiamando `claude -p` per ciascuna. Utilizzate `--allowedTools` per limitare i permessi per le operazioni batch.

Per migrazioni o analisi su larga scala, potete distribuire il lavoro tra molte invocazioni parallele di Claude:

Step 1: Generare un elenco di attività

Fate in modo che Claude elenchi tutti i file che devono essere migrati (ad es. `list all 2,000 Python files that need migrating`)

Step 2: Scrivere uno script per eseguire un ciclo attraverso l'elenco

```
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: Testare su pochi file, quindi eseguire su larga scala

Affinate il vostro prompt in base a quello che va male con i primi 2-3 file, quindi eseguite sull'intero set. Il flag `--allowedTools` limita quello che Claude può fare, il che è importante quando state eseguendo senza supervisione.

Potete anche integrare Claude nelle pipeline di elaborazione/dati esistenti:

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

Utilizzate `--verbose` per il debug durante lo sviluppo e spegnetelo in produzione.

Evitate i modelli di fallimento comuni

Questi sono errori comuni. Riconoscerli presto fa risparmiare tempo:

- **La sessione del lavandino della cucina.** Iniziate con un'attività, poi chiedete a Claude qualcosa di non correlato, poi tornate alla prima attività. Il contesto è pieno di informazioni irrilevanti. > **Correzione**: `/clear` tra attività non correlate.

- **Correggere più volte.** Claude fa qualcosa di sbagliato, lo correggete, è ancora sbagliato, lo correggete di nuovo. Il contesto è inquinato da approcci falliti. > **Correzione**: Dopo due correzioni fallite, `/clear` e scrivete un prompt iniziale migliore che incorpori quello che avete imparato.
- **Il CLAUDE.md eccessivamente specificato.** Se il vostro CLAUDE.md è troppo lungo, Claude ignora metà di esso perché le regole importanti si perdono nel rumore. > **Correzione**: Potate spietatamente. Se Claude fa già qualcosa di corretto senza l'istruzione, eliminatelo o convertitelo in un hook.
- **Il divario tra fiducia e verifica.** Claude produce un'implementazione plausibile che non gestisce i casi limite. > **Correzione**: Fornite sempre la verifica (test, script, screenshot). Se non potete verificarlo, non lo spedite.
- **L'esplorazione infinita.** Chiedete a Claude di "investigare" qualcosa senza limitarlo. Claude legge centinaia di file, riempiendo il contesto. > **Correzione**: Limitate le investigazioni strettamente o utilizzate i subagent in modo che l'esplorazione non consumi il vostro contesto principale.

Sviluppate la vostra intuizione

I modelli in questa guida non sono scolpiti nella pietra. Sono punti di partenza che funzionano bene in generale, ma potrebbero non essere ottimali per ogni situazione.

A volte *dovreste* lasciare che il contesto si accumuli perché siete immersi in un problema complesso e la cronologia è preziosa. A volte dovreste saltare la pianificazione e lasciare che Claude lo capisca perché l'attività è esplorativa. A volte un prompt vago è esattamente giusto perché volete vedere come Claude interpreta il problema prima di limitarlo.

Prestate attenzione a quello che funziona. Quando Claude produce un output eccellente, notate quello che avete fatto: la struttura del prompt, il contesto che avete fornito, la modalità in cui siete stati. Quando Claude fatica, chiedetevi perché. Il contesto era troppo rumoroso? Il prompt troppo vago? L'attività troppo grande per un passaggio?

Nel tempo, svilupperete un'intuizione che nessuna guida può catturare. Saprete quando essere specifici e quando essere aperti, quando pianificare e quando esplorare, quando cancellare il contesto e quando lasciarlo accumulare.

Risorse correlate

- How Claude Code works: il ciclo agenziale, gli strumenti e la gestione del contesto
- Extend Claude Code: skill, hook, MCP, subagent e plugin
- Common workflows: ricette passo dopo passo per debug, test, PR e altro

- CLAUDE.md: archiviare le convenzioni del progetto e il contesto persistente

Flussi di lavoro comuni

> *Guide passo dopo passo per esplorare basi di codice, correggere bug, effettuare refactoring, testare e altri compiti quotidiani con Claude Code.*

Questa pagina copre flussi di lavoro pratici per lo sviluppo quotidiano: esplorare codice non familiare, eseguire il debug, effettuare refactoring, scrivere test, creare PR e gestire sessioni. Ogni sezione include prompt di esempio che puoi adattare ai tuoi progetti. Per modelli e suggerimenti di livello superiore, vedi Best practices.

Comprendere nuove basi di codice

Ottenere una rapida panoramica della base di codice

Supponiamo che tu abbia appena aderito a un nuovo progetto e debba comprendere rapidamente la sua struttura.

Step 1: Navigare alla directory radice del progetto

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

Step 2: Avviare Claude Code

```
claude
```

Step 3: Chiedere una panoramica di alto livello

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

Step 4: Approfondire componenti specifici

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

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

```
how is authentication handled?
```

> **Tip:**
>
> Suggerimenti:
>
> - Inizia con domande ampie, quindi restringi a aree specifiche
> - Chiedi informazioni sulle convenzioni di codifica e sui modelli utilizzati nel progetto
> - Richiedi un glossario di termini specifici del progetto

Trovare codice rilevante

Supponiamo che tu debba individuare il codice relativo a una funzionalità o funzione specifica.

Step 1: Chiedere a Claude di trovare file rilevanti

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

Step 2: Ottenere contesto su come i componenti interagiscono

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

Step 3: Comprendere il flusso di esecuzione

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

> **Tip:**
>
> Suggerimenti:
>
> - Sii specifico su ciò che stai cercando
> - Usa il linguaggio del dominio dal progetto
> - Installa un plugin di code intelligence per il tuo linguaggio per dare a Claude una navigazione precisa “go to definition” e “find references”

Correggere bug in modo efficiente

Supponiamo che tu abbia riscontrato un messaggio di errore e debba trovare e correggere la sua fonte.

Step 1: Condividere l'errore con Claude

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

Step 2: Chiedere raccomandazioni per la correzione

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

Step 3: Applicare la correzione

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

> **Tip:**
>
> Suggerimenti:
>
> - Comunica a Claude il comando per riprodurre il problema e ottenere una stack trace
> - Menziona eventuali passaggi per riprodurre l'errore
> - Fai sapere a Claude se l'errore è intermittente o coerente

---

Effettuare refactoring del codice

Supponiamo che tu debba aggiornare il codice precedente per utilizzare modelli e pratiche moderne.

Step 1: Identificare il codice legacy per il refactoring

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

Step 2: Ottenere raccomandazioni per il refactoring

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

Step 3: Applicare le modifiche in modo sicuro

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

Step 4: Verificare il refactoring

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

> **Tip:**
>
> Suggerimenti:
>
> - Chiedi a Claude di spiegare i vantaggi dell'approccio moderno
> - Richiedi che le modifiche mantengano la compatibilità all'indietro quando necessario
> - Effettua il refactoring in piccoli incrementi testabili

Utilizzare subagent specializzati

Supponiamo che tu voglia utilizzare subagent AI specializzati per gestire attività specifiche in modo più efficace.

Step 1: Visualizzare i subagent disponibili

```
/agents
```

Questo mostra tutti i subagent disponibili e ti consente di crearne di nuovi.

Step 2: Utilizzare i subagent automaticamente

Claude Code delega automaticamente le attività appropriate ai subagent specializzati:

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

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

Step 3: Richiedere esplicitamente subagent specifici

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

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

Step 4: Creare subagent personalizzati per il tuo flusso di lavoro

```
/agents
```

Quindi seleziona “Create New subagent” e segui i prompt per definire:

- Un identificatore univoco che descrive lo scopo del subagent (ad esempio, `code-reviewer`, `api-designer`).
- Quando Claude dovrebbe utilizzare questo agente
- Quali strumenti può accedere
- Un prompt di sistema che descrive il ruolo e il comportamento dell’agente

> **Tip:**
>
> Suggerimenti:
>
> - Crea subagent specifici del progetto in `.claude/agents/` per la condivisione del team
> - Usa campi `description` descrittivi per abilitare la delega automatica
> - Limita l’accesso agli strumenti a ciò di cui ogni subagent ha effettivamente bisogno
> - Controlla la documentazione dei subagent per esempi dettagliati

---

Utilizzare Plan Mode per l’analisi sicura del codice

Plan Mode istruisce Claude a creare un piano analizzando la base di codice con operazioni di sola lettura, perfetto per esplorare basi di codice, pianificare modifiche complesse o rivedere il codice in modo sicuro. In Plan Mode, Claude utilizza `AskUserQuestion` per raccogliere requisiti e chiarire i tuoi obiettivi prima di proporre un piano.

Quando utilizzare Plan Mode

- **Implementazione multi-step**: Quando la tua funzionalità richiede di apportare modifiche a molti file
- **Esplorazione del codice**: Quando desideri ricercare la base di codice a fondo prima di modificare qualsiasi cosa
- **Sviluppo interattivo**: Quando desideri iterare sulla direzione con Claude

Come utilizzare Plan Mode

Attivare Plan Mode durante una sessione

Puoi passare a Plan Mode durante una sessione utilizzando **Shift+Tab** per scorrere le modalità di autorizzazione.

Se sei in Normal Mode, **Shift+Tab** passa prima a Auto-Accept Mode, indicato da `▸▸ accept edits on` nella parte inferiore del terminale. Un successivo **Shift+Tab** passerà a Plan Mode, indicato da `⏸ plan mode on`.

Avviare una nuova sessione in Plan Mode

Per avviare una nuova sessione in Plan Mode, usa il flag `--permission-mode plan`:

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

Eseguire query "headless" in Plan Mode

Puoi anche eseguire una query in Plan Mode direttamente con `-p` (cioè in "headless mode"):

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

Esempio: Pianificazione di un refactoring complesso

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

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

Claude analizza l'implementazione attuale e crea un piano completo. Affina con follow-up:

```
What about backward compatibility?
```

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

> **Tip:**
>
> Premi `Ctrl+G` per aprire il piano nel tuo editor di testo predefinito, dove puoi modificarlo direttamente prima che Claude proceda.

Configurare Plan Mode come predefinito

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

Vedi la documentazione delle impostazioni per ulteriori opzioni di configurazione.

Lavorare con i test

Supponiamo che tu debba aggiungere test per il codice non coperto.

Step 1: Identificare il codice non testato

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

Step 2: Generare lo scaffolding dei test

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

Step 3: Aggiungere casi di test significativi

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

Step 4: Eseguire e verificare i test

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

Claude può generare test che seguono i modelli e le convenzioni esistenti del tuo progetto. Quando chiedi test, sii specifico sul comportamento che desideri verificare. Claude esamina i tuoi file di test esistenti per abbinare lo stile, i framework e i modelli di asserzione già in uso.

Per una copertura completa, chiedi a Claude di identificare i casi limite che potresti aver perso. Claude può analizzare i tuoi percorsi di codice e suggerire test per condizioni di errore, valori limite e input inaspettati che sono facili da trascurare.

Creare pull request

Puoi creare pull request chiedendo direttamente a Claude (“create a pr for my changes”), oppure guidare Claude attraverso i passaggi:

Step 1: Riassumere le tue modifiche

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

Step 2: Generare una pull request

```
create a pr
```

Step 3: Rivedere e affinare

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

Quando crei una PR utilizzando `gh pr create`, la sessione viene automaticamente collegata a quella PR. Puoi riprenderla in seguito con `claude --from-pr <number>`.

> **Tip:**

> Rivedi la PR generata da Claude prima di inviarla e chiedi a Claude di evidenziare i rischi potenziali o le considerazioni.

Gestire la documentazione

Supponiamo che tu debba aggiungere o aggiornare la documentazione per il tuo codice.

Step 1: Identificare il codice non documentato

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

Step 2: Generare la documentazione

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

Step 3: Rivedere e migliorare

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

Step 4: Verificare la documentazione

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

> **Tip:**
>
> Suggerimenti:
>
> - Specifica lo stile di documentazione che desideri (JSDoc, docstrings, ecc.)
> - Chiedi esempi nella documentazione
> - Richiedi documentazione per API pubbliche, interfacce e logica complessa

---

Lavorare con le immagini

Supponiamo che tu debba lavorare con immagini nella tua base di codice e desideri l'aiuto di Claude nell'analizzare il contenuto dell'immagine.

Step 1: Aggiungere un'immagine alla conversazione

Puoi utilizzare uno di questi metodi:

1. Trascina e rilascia un'immagine nella finestra di Claude Code
2. Copia un'immagine e incollala nella CLI con ctrl+v (Non usare cmd+v)
3. Fornisci un percorso di immagine a Claude. Ad esempio, "Analyze this image: /path/to/your/image.png"

Step 2: Chiedere a Claude di analizzare l'immagine

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

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

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

Step 3: Utilizzare le immagini per il contesto

```
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: Ottenere suggerimenti di codice dal contenuto visivo

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

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

Tip:

Suggerimenti:

- Usa le immagini quando le descrizioni di testo sarebbero poco chiare o ingombranti
- Includi screenshot di errori, design dell'interfaccia utente o diagrammi per un contesto migliore
- Puoi lavorare con più immagini in una conversazione

- L'analisi delle immagini funziona con diagrammi, screenshot, mockup e altro
- Quando Claude fa riferimento a immagini (ad esempio, `[Image #1]`), `Cmd+Click` (Mac) o `Ctrl+Click` (Windows/Linux) il collegamento per aprire l'immagine nel tuo visualizzatore predefinito

Fare riferimento a file e directory

Usa @ per includere rapidamente file o directory senza aspettare che Claude li legga.

Step 1: Fare riferimento a un singolo file

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

Questo include il contenuto completo del file nella conversazione.

Step 2: Fare riferimento a una directory

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

Questo fornisce un elenco di directory con informazioni sui file.

Step 3: Fare riferimento alle risorse MCP

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

Questo recupera i dati dai server MCP connessi utilizzando il formato @server:resource. Vedi risorse MCP per i dettagli.

Tip:

Suggerimenti:

- I percorsi dei file possono essere relativi o assoluti
- I riferimenti ai file @ aggiungono `CLAUDE.md` nella directory del file e nelle directory padre al contesto
- I riferimenti alle directory mostrano elenchi di file, non contenuti
- Puoi fare riferimento a più file in un singolo messaggio (ad esempio, "@file1.js and @file2.js")

Utilizzare il pensiero esteso (thinking mode)

Extended thinking è abilitato per impostazione predefinita, dando a Claude lo spazio per ragionare attraverso problemi complessi passo dopo passo prima di rispondere. Questo ragionamento è visibile in modalità verbose, che puoi attivare con `Ctrl+O`.

Inoltre, Opus 4.6 introduce il ragionamento adattivo: invece di un budget di token di pensiero fisso, il modello alloca dinamicamente il pensiero in base alla tua impostazione di livello di sforzo. Extended thinking e il ragionamento adattivo funzionano insieme per darti il controllo su quanto profondamente Claude ragiona prima di rispondere.

Extended thinking è particolarmente prezioso per decisioni architettoniche complesse, bug impegnativi, pianificazione dell'implementazione multi-step e valutazione dei compromessi tra diversi approcci.

> **Note:**
>
> Frasi come "think", "think hard" e "think more" sono interpretate come istruzioni di prompt regolari e non allocano token di pensiero.

Configurare la modalità di pensiero

Il pensiero è abilitato per impostazione predefinita, ma puoi regolarlo o disabilitarlo.

Ambito	Come configurare	Dettagli
Livello di sforzo	Regola in `/model` o imposta `CLAUDE_CODE_EFFORT_LEVEL`	Controlla la profondità del pensiero per Opus 4.6 e Sonnet 4.6: basso, medio, alto. Vedi Regola il livello di sforzo
Parola chiave `ultrathink`	Includi "ultrathink" in qualsiasi punto del tuo prompt	Imposta lo sforzo su alto per quel turno su Opus 4.6 e Sonnet 4.6. Utile per attività una tantum che richiedono un ragionamento profondo senza modificare permanentemente la tua impostazione di sforzo

Ambito	Come configurare	Dettagli
Scorciatoia di attivazione/disattivazione	Premi `Option+T` (macOS) o `Alt+T` (Windows/Linux)	Attiva/disattiva il pensiero per la sessione corrente (tutti i modelli). Potrebbe richiedere configurazione del terminale per abilitare le scorciatoie del tasto Option
Predefinito globale	Usa `/config` per attivare/disattivare la modalità di pensiero	Imposta il tuo predefinito in tutti i progetti (tutti i modelli). Salvato come `alwaysThinkingEnabled` in `~/.claude/settings.json`
Limitare il budget dei token	Imposta la variabile di ambiente `MAX_THINKING_TOKENS`	Limita il budget di pensiero a un numero specifico di token (ignorato su Opus 4.6 a meno che non sia impostato su 0). Esempio: `export MAX_THINKING_TOKENS=10000`

Per visualizzare il processo di pensiero di Claude, premi `Ctrl+O` per attivare la modalità verbose e vedi il ragionamento interno visualizzato come testo grigio in corsivo.

Come funziona il pensiero esteso

Extended thinking controlla quanto ragionamento interno Claude esegue prima di rispondere. Più pensiero fornisce più spazio per esplorare soluzioni, analizzare casi limite e autocorreggersi gli errori.

Con Opus 4.6, il pensiero utilizza il ragionamento adattivo: il modello alloca dinamicamente i token di pensiero in base al livello di sforzo che selezioni (basso, medio, alto). Questo è il modo consigliato per sintonizzare il compromesso tra velocità e profondità di ragionamento.

Con altri modelli, il pensiero utilizza un budget fisso di fino a 31.999 token dal tuo budget di output. Puoi limitare questo con la variabile di ambiente `MAX_THINKING_TOKENS`, o disabilitare completamente il pensiero tramite `/config` o l'attivazione/disattivazione `Option+T` / `Alt+T`.

`MAX_THINKING_TOKENS` è ignorato su Opus 4.6 e Sonnet 4.6, poiché il ragionamento adattivo controlla la profondità del pensiero. L'unica eccezione: impostare `MAX_THINKING_TOKENS=0` disabilita comunque completamente il pensiero su qualsiasi modello. Per disabilitare il pensiero adattivo e ripristinare il budget di pensiero fisso, imposta `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`. Vedi variabili di ambiente.

> **Warning:**
>
> Ti viene addebitato per tutti i token di pensiero utilizzati, anche se i modelli Claude 4 mostrano il pensiero riassunto

---

Riprendere conversazioni precedenti

Quando avvii Claude Code, puoi riprendere una sessione precedente:

- `claude --continue` continua la conversazione più recente nella directory corrente
- `claude --resume` apre un selettore di conversazione o riprende per nome
- `claude --from-pr 123` riprende le sessioni collegate a una pull request specifica

Da una sessione attiva, usa `/resume` per passare a una conversazione diversa.

Le sessioni sono archiviate per directory di progetto. Il selettore `/resume` mostra le sessioni dallo stesso repository git, inclusi i worktree.

Denominare le tue sessioni

Dai alle sessioni nomi descrittivi per trovarle in seguito. Questa è una best practice quando lavori su più attività o funzionalità.

Step 1: Denominare la sessione corrente

Usa `/rename` durante una sessione per darle un nome memorabile:

```
/rename auth-refactor
```

Puoi anche rinominare qualsiasi sessione dal selettore: esegui `/resume`, naviga a una sessione e premi `R`.

Step 2: Riprendere per nome in seguito

Dalla riga di comando:

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

O da una sessione attiva:

```
/resume auth-refactor
```

Utilizzare il selettore di sessione

Il comando `/resume` (o `claude --resume` senza argomenti) apre un selettore di sessione interattivo con queste funzionalità:

Scorciatoie da tastiera nel selettore:

Scorciatoia	Azione
`↑` / `↓`	Navigare tra le sessioni
`→` / `←`	Espandere o comprimere le sessioni raggruppate
`Enter`	Selezionare e riprendere la sessione evidenziata
`P`	Visualizzare l'anteprima del contenuto della sessione
`R`	Rinominare la sessione evidenziata
`/`	Cercare per filtrare le sessioni
`A`	Alternare tra la directory corrente e tutti i progetti
`B`	Filtrare le sessioni dal tuo ramo git corrente
`Esc`	Uscire dal selettore o dalla modalità di ricerca

Organizzazione della sessione:

Il selettore visualizza le sessioni con metadati utili:

- Nome della sessione o prompt iniziale
- Tempo trascorso dall'ultima attività
- Conteggio dei messaggi
- Ramo Git (se applicabile)

Le sessioni fork (create con `/rewind` o `--fork-session`) sono raggruppate insieme sotto la loro sessione radice, rendendo più facile trovare conversazioni correlate.

Tip:

Suggerimenti:

- **Denominare le sessioni in anticipo**: Usa `/rename` quando inizi a lavorare su un'attività distinta, è molto più facile trovare "payment-integration" che "explain this function" in seguito
- Usa `--continue` per un accesso rapido alla tua conversazione più recente nella directory corrente
- Usa `--resume session-name` quando sai quale sessione ti serve
- Usa `--resume` (senza un nome) quando hai bisogno di sfogliare e selezionare
- Per gli script, usa `claude --continue --print "prompt"` per riprendere in modalità non interattiva
- Premi `P` nel selettore per visualizzare l'anteprima di una sessione prima di riprenderla
- La conversazione ripresa inizia con lo stesso modello e configurazione dell'originale

Come funziona:

1. **Archiviazione della conversazione**: Tutte le conversazioni vengono salvate automaticamente localmente con la loro cronologia completa dei messaggi
2. **Deserializzazione dei messaggi**: Quando si riprende, l'intera cronologia dei messaggi viene ripristinata per mantenere il contesto
3. **Stato dello strumento**: L'utilizzo dello strumento e i risultati della conversazione precedente vengono preservati
4. **Ripristino del contesto**: La conversazione riprende con tutto il contesto precedente intatto

Eseguire sessioni parallele di Claude Code con Git worktrees

Quando lavori su più attività contemporaneamente, hai bisogno che ogni sessione di Claude abbia la sua copia della base di codice in modo che le modifiche non si scontrino. I worktree Git risolvono questo creando directory di lavoro separate che hanno ciascuna i propri file e ramo, mentre condividono la stessa cronologia del repository e le connessioni remote. Ciò significa che puoi avere Claude che lavora su una funzionalità in un worktree mentre corregge un bug in un altro, senza che nessuna sessione interferisca con l'altra.

Usa il flag `--worktree` (`-w`) per creare un worktree isolato e avviare Claude in esso. Il valore che passi diventa il nome della directory del worktree e il nome del ramo:

```
## Avviare Claude in un worktree denominato "feature-auth"
## Crea .claude/worktrees/feature-auth/ con un nuovo ramo
claude --worktree feature-auth

## Avviare un'altra sessione in un worktree separato
claude --worktree bugfix-123
```

Se ometti il nome, Claude ne genera uno casuale automaticamente:

```
## Genera automaticamente un nome come "bright-running-fox"
claude --worktree
```

I worktree vengono creati in `<repo>/.claude/worktrees/<name>` e si diramano dal ramo remoto predefinito. Il ramo del worktree è denominato `worktree-<name>`.

Puoi anche chiedere a Claude di “work in a worktree” o “start a worktree” durante una sessione, e creerà uno automaticamente.

Worktree dei subagent

I subagent possono anche utilizzare l’isolamento del worktree per lavorare in parallelo senza conflitti. Chiedi a Claude di “use worktrees for your agents” o configuralo in un subagent personalizzato aggiungendo `isolation: worktree` al frontmatter dell’agente. Ogni subagent ottiene il suo worktree che viene automaticamente pulito quando il subagent finisce senza modifiche.

Pulizia del worktree

Quando esci da una sessione di worktree, Claude gestisce la pulizia in base al fatto che tu abbia apportato modifiche:

- **Nessuna modifica**: il worktree e il suo ramo vengono rimossi automaticamente
- **Modifiche o commit esistenti**: Claude ti chiede se mantenere o rimuovere il worktree. Mantenere preserva la directory e il ramo in modo da poter tornare in seguito. Rimuovere elimina la directory del worktree e il suo ramo, scartando tutte le modifiche non sottoposte a commit e i commit

Per pulire i worktree al di fuori di una sessione di Claude, usa gestione manuale del worktree.

> **Tip:**
>
> Aggiungi `.claude/worktrees/` al tuo `.gitignore` per evitare che il contenuto del worktree appaia come file non tracciati nel tuo repository principale.

Gestire i worktree manualmente

Per un maggiore controllo sulla posizione del worktree e sulla configurazione del ramo, crea i worktree direttamente con Git. Questo è utile quando hai bisogno di controllare un ramo esistente specifico o posizionare il worktree al di fuori del repository.

```
## Creare un worktree con un nuovo ramo
git worktree add ../project-feature-a -b feature-a

## Creare un worktree con un ramo esistente
git worktree add ../project-bugfix bugfix-123

## Avviare Claude nel worktree
cd ../project-feature-a && claude

## Pulire al termine
git worktree list
git worktree remove ../project-feature-a
```

Scopri di più nella documentazione ufficiale di Git worktree.

> **Tip:**
>
> Ricorda di inizializzare il tuo ambiente di sviluppo in ogni nuovo worktree secondo la configurazione del tuo progetto. A seconda del tuo stack, questo potrebbe includere l'esecuzione dell'installazione delle dipendenze (`npm install` , `yarn`), la configurazione di ambienti virtuali o il seguire il processo di configurazione standard del tuo progetto.

Controllo della versione non git

L'isolamento del worktree funziona con git per impostazione predefinita. Per altri sistemi di controllo della versione come SVN, Perforce o Mercurial, configura hook WorktreeCreate e WorktreeRemove per fornire logica personalizzata di creazione e pulizia del worktree. Quando configurati, questi hook sostituiscono il comportamento git predefinito quando usi `--worktree` .

Per il coordinamento automatizzato di sessioni parallele con attività condivise e messaggistica, vedi team di agenti.

Ricevere notifiche quando Claude ha bisogno della tua attenzione

Quando avvii un'attività a lunga esecuzione e passi a un'altra finestra, puoi configurare notifiche desktop in modo da sapere quando Claude finisce o ha bisogno del tuo input. Questo utilizza l'evento `Notification` hook, che si attiva ogni volta che Claude è in attesa di autorizzazione, inattivo e pronto per un nuovo prompt, o completando l'autenticazione.

Step 1: Aprire il menu degli hook

Digita `/hooks` e seleziona `Notification` dall'elenco degli eventi.

Step 2: Configurare il matcher

Seleziona `+ Match all (no filter)` per attivarsi su tutti i tipi di notifica. Per notificare solo per eventi specifici, seleziona `+ Add new matcher…` e inserisci uno di questi valori:

Matcher	Si attiva quando
`permission_prompt`	Claude ha bisogno che tu approvi un utilizzo dello strumento
`idle_prompt`	Claude ha finito ed è in attesa del tuo prossimo prompt
`auth_success`	L'autenticazione si completa
`elicitation_dialog`	Claude ti sta facendo una domanda

Step 3: Aggiungere il tuo comando di notifica

Seleziona `+ Add new hook…` e inserisci il comando per il tuo sistema operativo:

macOS

Utilizza `osascript` per attivare una notifica macOS nativa tramite AppleScript:

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

Linux

Utilizza `notify-send`, che è preinstallato sulla maggior parte dei desktop Linux con un daemon di notifica:

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

Windows (PowerShell)

Utilizza PowerShell per mostrare una finestra di messaggio nativa tramite Windows Forms di .NET:

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

Step 4: Salvare nelle impostazioni utente

Seleziona `User settings` per applicare la notifica in tutti i tuoi progetti.

Per la procedura dettagliata completa con esempi di configurazione JSON, vedi Automatizzare i flussi di lavoro con gli hook. Per lo schema completo dell'evento e i tipi di notifica, vedi il riferimento Notification.

Utilizzare Claude come utilità di tipo unix

Aggiungere Claude al tuo processo di verifica

Supponiamo che tu voglia utilizzare Claude Code come linter o revisore del codice.

Aggiungere Claude al tuo script di build:

```
// 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:

Suggerimenti:

- Usa Claude per la revisione automatica del codice nella tua pipeline CI/CD
- Personalizza il prompt per verificare i problemi specifici rilevanti per il tuo progetto
- Considera di creare più script per diversi tipi di verifica

Pipe in, pipe out

Supponiamo che tu voglia inviare dati a Claude e ottenere dati in un formato strutturato.

Inviare dati attraverso Claude:

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

Tip:

Suggerimenti:

- Usa i pipe per integrare Claude negli script shell esistenti
- Combina con altri strumenti Unix per flussi di lavoro potenti
- Considera di utilizzare –output-format per output strutturato

Controllare il formato di output

Supponiamo che tu abbia bisogno dell'output di Claude in un formato specifico, specialmente quando integri Claude Code in script o altri strumenti.

Step 1: Utilizzare il formato testo (predefinito)

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

Questo restituisce solo la risposta di testo semplice di Claude (comportamento predefinito).

Step 2: Utilizzare il formato JSON

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

Questo restituisce un array JSON di messaggi con metadati inclusi costo e durata.

Step 3: Utilizzare il formato JSON in streaming

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

Questo restituisce una serie di oggetti JSON in tempo reale mentre Claude elabora la richiesta. Ogni messaggio è un oggetto JSON valido, ma l'intero output non è JSON valido se concatenato.

> **Tip:**
>
> Suggerimenti:
>
> - Usa `--output-format text` per integrazioni semplici dove hai solo bisogno della risposta di Claude
> - Usa `--output-format json` quando hai bisogno del registro completo della conversazione
> - Usa `--output-format stream-json` per l'output in tempo reale di ogni turno di conversazione

Chiedere a Claude le sue capacità

Claude ha accesso integrato alla sua documentazione e può rispondere a domande sulle sue stesse funzionalità e limitazioni.

Domande di esempio

```
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 fornisce risposte basate sulla documentazione a queste domande. Per esempi eseguibili e dimostrazioni pratiche, fai riferimento alle sezioni di flusso di lavoro specifiche sopra.

Tip:

Suggerimenti:

- Claude ha sempre accesso alla documentazione più recente di Claude Code, indipendentemente dalla versione che stai utilizzando
- Fai domande specifiche per ottenere risposte dettagliate
- Claude può spiegare funzionalità complesse come integrazione MCP, configurazioni aziendali e flussi di lavoro avanzati

Passaggi successivi

- Best practices Modelli per ottenere il massimo da Claude Code
- Come funziona Claude Code Comprendi il loop agentico e la gestione del contesto
- Estendere Claude Code Aggiungi skills, hooks, MCP, subagent e plugin
- Implementazione di riferimento Clona la nostra implementazione di riferimento del contenitore di sviluppo

Part 3: Commands & Reference

Riferimento CLI

> *Riferimento completo per l'interfaccia da riga di comando di Claude Code, inclusi comandi e flag.*

Comandi CLI

Puoi avviare sessioni, inviare contenuti tramite pipe, riprendere conversazioni e gestire gli aggiornamenti con questi comandi:

Comando	Descrizione	Esempio
`claude`	Avvia sessione interattiva	`claude`
`claude "query"`	Avvia sessione interattiva con prompt iniziale	`claude "explain this project"`
`claude -p "query"`	Esegui query tramite SDK, quindi esci	`claude -p "explain this function"`
`cat file | claude -p "query"`	Elabora contenuto inviato tramite pipe	`cat logs.txt | claude -p "explain"`
`claude -c`	Continua la conversazione più recente nella directory corrente	`claude -c`
`claude -c -p "query"`	Continua tramite SDK	`claude -c -p "Check for type errors"`
`claude -r "<session>" "query"`	Riprendi sessione per ID o nome	`claude -r "auth-refactor" "Finish this PR"`
`claude update`	Aggiorna alla versione più recente	`claude update`
`claude auth login`	Accedi al tuo account Anthropic. Usa `--email` per pre-compilare il tuo indirizzo email e `--sso` per forzare l'autenticazione SSO	`claude auth login --email user@example.com --sso`

Comando	Descrizione	Esempio
`claude auth logout`	Esci dal tuo account Anthropic	`claude auth logout`
`claude auth status`	Mostra lo stato di autenticazione come JSON. Usa `--text` per output leggibile dall'uomo. Esce con codice 0 se connesso, 1 se no	`claude auth status`
`claude agents`	Elenca tutti i subagents configurati, raggruppati per fonte	`claude agents`
`claude mcp`	Configura i server Model Context Protocol (MCP)	Vedi la documentazione Claude Code MCP.
`claude remote-control`	Avvia una sessione Remote Control per controllare Claude Code da Claude.ai o dall'app Claude mentre è in esecuzione localmente. Vedi Remote Control per i flag	`claude remote-control`

Flag CLI

Personalizza il comportamento di Claude Code con questi flag da riga di comando:

Flag	Descrizione	Esempio
`--add-dir`	Aggiungi directory di lavoro aggiuntive per l'accesso di Claude (convalida che ogni percorso esista come directory)	`claude --add-dir ../apps ../lib`
`--agent`	Specifica un agente per la sessione corrente (sostituisce l'impostazione `agent`)	`claude --agent my-custom-agent`
`--agents`	Definisci subagents personalizzati dinamicamente tramite JSON (vedi sotto per il formato)	`claude --agents '{"reviewer":{"description":"Reviews code","prompt":"You are a code reviewer"}}'`

Flag	Descrizione	Esempio
`--allow-dangerously-skip-permissions`	Abilita il bypass dei permessi come opzione senza attivarlo immediatamente. Consente di comporre con `--permission-mode` (usa con cautela)	`claude --permission-mode plan --allow-dangerously-skip-permissions`
`--allowedTools`	Strumenti che si eseguono senza richiedere il permesso. Vedi sintassi della regola di permesso per la corrispondenza dei pattern. Per limitare quali strumenti sono disponibili, usa `--tools` invece	`"Bash(git log *)" "Bash(git diff *)" "Read"`
`--append-system-prompt`	Aggiungi testo personalizzato alla fine del prompt di sistema predefinito	`claude --append-system-prompt "Always use TypeScript"`
`--append-system-prompt-file`	Carica testo di prompt di sistema aggiuntivo da un file e aggiungi al prompt predefinito	`claude --append-system-prompt-file ./extra-rules.txt`
`--betas`	Intestazioni beta da includere nelle richieste API (solo utenti con chiave API)	`claude --betas interleaved-thinking`
`--chrome`	Abilita integrazione del browser Chrome per l'automazione web e i test	`claude --chrome`
`--continue`, `-c`	Carica la conversazione più recente nella directory corrente	`claude --continue`
`--dangerously-skip-permissions`	Salta tutti i prompt di permesso (usa con cautela)	`claude --dangerously-skip-permissions`
`--debug`	Abilita la modalità debug con filtro di categoria opzionale (ad esempio, `"api,hooks"` o `"!statsig,!file"`)	`claude --debug "api,mcp"`
`--disable-slash-commands`	Disabilita tutti gli skills e i comandi per questa sessione	`claude --disable-slash-commands`

Flag	Descrizione	Esempio
`--disallowedTools`	Strumenti che vengono rimossi dal contesto del modello e non possono essere utilizzati	`"Bash(git log *)" "Bash(git diff *)" "Edit"`
`--fallback-model`	Abilita il fallback automatico al modello specificato quando il modello predefinito è sovraccarico (solo modalità print)	`claude -p --fallback-model sonnet "query"`
`--fork-session`	Quando riprendi, crea un nuovo ID di sessione invece di riutilizzare l'originale (usa con `--resume` o `--continue`)	`claude --resume abc123 --fork-session`
`--from-pr`	Riprendi sessioni collegate a una PR GitHub specifica. Accetta un numero di PR o un URL. Le sessioni vengono collegate automaticamente quando create tramite `gh pr create`	`claude --from-pr 123`
`--ide`	Connettiti automaticamente all'IDE all'avvio se esattamente un IDE valido è disponibile	`claude --ide`
`--init`	Esegui gli hook di inizializzazione e avvia la modalità interattiva	`claude --init`
`--init-only`	Esegui gli hook di inizializzazione e esci (nessuna sessione interattiva)	`claude --init-only`
`--include-partial-messages`	Includi gli eventi di streaming parziale nell'output (richiede `--print` e `--output-format=stream-json`)	`claude -p --output-format stream-json --include-partial-messages "query"`
`--input-format`	Specifica il formato di input per la modalità print (opzioni: `text`, `stream-json`)	`claude -p --output-format json --input-format stream-json`

Flag	Descrizione	Esempio
`--json-schema`	Ottieni output JSON convalidato corrispondente a uno JSON Schema dopo che l'agente completa il suo flusso di lavoro (solo modalità print, vedi structured outputs)	`claude -p --json-schema '{"type":"object","properties":{...}}' "query"`
`--maintenance`	Esegui gli hook di manutenzione e esci	`claude --maintenance`
`--max-budget-usd`	Importo massimo in dollari da spendere nelle chiamate API prima di fermarsi (solo modalità print)	`claude -p --max-budget-usd 5.00 "query"`
`--max-turns`	Limita il numero di turni agentici (solo modalità print). Esce con un errore quando il limite viene raggiunto. Nessun limite per impostazione predefinita	`claude -p --max-turns 3 "query"`
`--mcp-config`	Carica i server MCP da file JSON o stringhe (separati da spazi)	`claude --mcp-config ./mcp.json`
`--model`	Imposta il modello per la sessione corrente con un alias per il modello più recente (`sonnet` o `opus`) o il nome completo di un modello	`claude --model claude-sonnet-4-6`
`--no-chrome`	Disabilita integrazione del browser Chrome per questa sessione	`claude --no-chrome`
`--no-session-persistence`	Disabilita la persistenza della sessione in modo che le sessioni non vengano salvate su disco e non possano essere riprese (solo modalità print)	`claude -p --no-session-persistence "query"`
`--output-format`	Specifica il formato di output per la modalità print (opzioni: `text` , `json` , `stream-json`)	`claude -p "query" --output-format json`
`--permission-mode`	Inizia in una modalità di permesso specificata	`claude --permission-mode plan`

Flag	Descrizione	Esempio
`--permission-prompt-tool`	Specifica uno strumento MCP per gestire i prompt di permesso in modalità non interattiva	`claude -p --permission-prompt-tool mcp_auth_tool "query"`
`--plugin-dir`	Carica i plugin da directory per questa sessione solo (ripetibile)	`claude --plugin-dir ./my-plugins`
`--print`, `-p`	Stampa la risposta senza modalità interattiva (vedi documentazione Agent SDK per i dettagli di utilizzo programmatico)	`claude -p "query"`
`--remote`	Crea una nuova sessione web su claude.ai con la descrizione dell'attività fornita	`claude --remote "Fix the login bug"`
`--resume`, `-r`	Riprendi una sessione specifica per ID o nome, o mostra un selettore interattivo per scegliere una sessione	`claude --resume auth-refactor`
`--session-id`	Usa un ID di sessione specifico per la conversazione (deve essere un UUID valido)	`claude --session-id "550e8400-e29b-41d4-a716-446655440000"`
`--setting-sources`	Elenco separato da virgole delle fonti di impostazioni da caricare (`user`, `project`, `local`)	`claude --setting-sources user,project`
`--settings`	Percorso di un file JSON di impostazioni o una stringa JSON da cui caricare impostazioni aggiuntive	`claude --settings ./settings.json`
`--strict-mcp-config`	Usa solo i server MCP da `--mcp-config`, ignorando tutte le altre configurazioni MCP	`claude --strict-mcp-config --mcp-config ./mcp.json`
`--system-prompt`	Sostituisci l'intero prompt di sistema con testo personalizzato	`claude --system-prompt "You are a Python expert"`
`--system-prompt-file`	Carica il prompt di sistema da un file, sostituendo il prompt predefinito	`claude --system-prompt-file ./custom-prompt.txt`

Flag	Descrizione	Esempio
`--teleport`	Riprendi una sessione web nel tuo terminale locale	`claude --teleport`
`--teammate-mode`	Imposta come i compagni di squadra dell'agent team vengono visualizzati: `auto` (predefinito), `in-process`, o `tmux`. Vedi configurare agent teams	`claude --teammate-mode in-process`
`--tools`	Limita quali strumenti integrati Claude può utilizzare. Usa `""` per disabilitare tutti, `"default"` per tutti, o nomi di strumenti come `"Bash,Edit,Read"`	`claude --tools "Bash,Edit,Read"`
`--verbose`	Abilita la registrazione dettagliata, mostra l'output completo turno per turno	`claude --verbose`
`--version`, `-v`	Restituisce il numero di versione	`claude -v`
`--worktree`, `-w`	Avvia Claude in un git worktree isolato in `<repo>/.claude/worktrees/<name>`. Se non viene fornito alcun nome, uno viene generato automaticamente	`claude -w feature-auth`

> **Tip:**
>
> Il flag `--output-format json` è particolarmente utile per lo scripting e l'automazione, consentendoti di analizzare le risposte di Claude a livello di programmazione.

Formato del flag agents

Il flag `--agents` accetta un oggetto JSON che definisce uno o più subagents personalizzati. Ogni subagent richiede un nome univoco (come chiave) e un oggetto di definizione con i seguenti campi:

Campo	Obbligatorio	Descrizione
`description`	Sì	Descrizione in linguaggio naturale di quando il subagent dovrebbe essere invocato
`prompt`	Sì	Il prompt di sistema che guida il comportamento del subagent
`tools`	No	Array di strumenti specifici che il subagent può utilizzare, ad esempio `["Read", "Edit", "Bash"]`. Se omesso, eredita tutti gli strumenti. Supporta la sintassi `Agent(agent_type)`
`disallowedTools`	No	Array di nomi di strumenti da negare esplicitamente per questo subagent
`model`	No	Alias del modello da utilizzare: `sonnet`, `opus`, `haiku`, o `inherit`. Se omesso, per impostazione predefinita è `inherit`
`skills`	No	Array di nomi di skill da precaricare nel contesto del subagent
`mcpServers`	No	Array di MCP servers per questo subagent. Ogni voce è una stringa del nome del server o un oggetto `{name: config}`
`maxTurns`	No	Numero massimo di turni agentici prima che il subagent si fermi

Esempio:

```
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."
  }
}'
```

Per ulteriori dettagli sulla creazione e l'utilizzo di subagents, vedi la documentazione dei subagents.

Flag del prompt di sistema

Claude Code fornisce quattro flag per personalizzare il prompt di sistema. Tutti e quattro funzionano sia in modalità interattiva che non interattiva.

Flag	Comportamento	Caso d'uso
`--system-prompt`	**Sostituisce** l'intero prompt predefinito	Controllo completo sul comportamento e le istruzioni di Claude
`--system-prompt-file`	**Sostituisce** con il contenuto del file	Carica i prompt dai file per la riproducibilità e il controllo della versione
`--append-system-prompt`	**Aggiunge** al prompt predefinito	Aggiungi istruzioni specifiche mantenendo il comportamento predefinito di Claude Code
`--append-system-prompt-file`	**Aggiunge** il contenuto del file al prompt predefinito	Carica istruzioni aggiuntive dai file mantenendo i valori predefiniti

Quando usare ciascuno:

- `--system-prompt` : usa quando hai bisogno di un controllo completo sul prompt di sistema di Claude. Questo rimuove tutte le istruzioni predefinite di Claude Code, dandoti una lavagna pulita.

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

- `--system-prompt-file` : usa quando vuoi caricare un prompt personalizzato da un file, utile per la coerenza del team o per i modelli di prompt controllati dalla versione.

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

- `--append-system-prompt` : usa quando vuoi aggiungere istruzioni specifiche mantenendo intatte le capacità predefinite di Claude Code. Questa è l'opzione più sicura per la maggior parte dei casi d'uso.

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

- `--append-system-prompt-file` : usa quando vuoi aggiungere istruzioni da un file mantenendo i valori predefiniti di Claude Code. Utile per le aggiunte controllate dalla versione.

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

`--system-prompt` e `--system-prompt-file` si escludono a vicenda. I flag di aggiunta possono essere utilizzati insieme a uno dei flag di sostituzione.

Per la maggior parte dei casi d'uso, `--append-system-prompt` o `--append-system-prompt-file` è consigliato poiché preservano le capacità integrate di Claude Code mentre aggiungono i tuoi requisiti personalizzati. Usa `--system-prompt` o `--system-prompt-file` solo quando hai bisogno di un controllo completo sul prompt di sistema.

Vedi anche

- Estensione Chrome - Automazione del browser e test web
- Modalità interattiva - Scorciatoie, modalità di input e funzionalità interattive

- Guida di avvio rapido - Introduzione a Claude Code
- Flussi di lavoro comuni - Flussi di lavoro e pattern avanzati
- Impostazioni - Opzioni di configurazione
- Documentazione Agent SDK - Utilizzo programmatico e integrazioni

Personalizzare le scorciatoie da tastiera

> *Personalizzare le scorciatoie da tastiera in Claude Code con un file di configurazione keybindings.*

> **Note:**
>
> Le scorciatoie da tastiera personalizzabili richiedono Claude Code v2.1.18 o versioni successive. Controllare la versione con `claude --version`.

Claude Code supporta scorciatoie da tastiera personalizzabili. Eseguire `/keybindings` per creare o aprire il file di configurazione in `~/.claude/keybindings.json`.

File di configurazione

Il file di configurazione delle scorciatoie da tastiera è un oggetto con un array `bindings`. Ogni blocco specifica un contesto e una mappa di sequenze di tasti per le azioni.

> **Note:**
>
> Le modifiche al file keybindings vengono rilevate automaticamente e applicate senza riavviare Claude Code.

Campo	Descrizione
`$schema`	URL dello schema JSON opzionale per l'autocompletamento dell'editor
`$docs`	URL della documentazione opzionale
`bindings`	Array di blocchi di binding per contesto

Questo esempio associa `Ctrl+E` per aprire un editor esterno nel contesto della chat e annulla l'associazione di `Ctrl+U`:

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

Contesti

Ogni blocco di binding specifica un **contesto** dove si applicano i binding:

Contesto	Descrizione
`Global`	Si applica ovunque nell'app
`Chat`	Area di input della chat principale
`Autocomplete`	Menu di autocompletamento è aperto
`Settings`	Menu delle impostazioni (chiusura solo con escape)
`Confirmation`	Dialoghi di permesso e conferma
`Tabs`	Componenti di navigazione delle schede
`Help`	Menu della guida è visibile
`Transcript`	Visualizzatore di trascrizione
`HistorySearch`	Modalità di ricerca nella cronologia (Ctrl+R)
`Task`	Attività in background in esecuzione
`ThemePicker`	Dialogo di selezione del tema
`Attachments`	Navigazione della barra immagini/allegati

Contesto	Descrizione
`Footer`	Navigazione dell'indicatore di piè di pagina (attività, team, diff)
`MessageSelector`	Selezione dei messaggi nella finestra di dialogo di riavvolgimento e riepilogo
`DiffDialog`	Navigazione del visualizzatore diff
`ModelPicker`	Livello di sforzo del selezionatore di modelli
`Select`	Componenti generici di selezione/elenco
`Plugin`	Dialogo dei plugin (sfoglia, scopri, gestisci)

Azioni disponibili

Le azioni seguono un formato `namespace:action`, come `chat:submit` per inviare un messaggio o `app:toggleTodos` per mostrare l'elenco delle attività. Ogni contesto ha azioni specifiche disponibili.

Azioni dell'app

Azioni disponibili nel contesto `Global`:

Azione	Predefinito	Descrizione
`app:interrupt`	Ctrl+C	Annulla l'operazione corrente
`app:exit`	Ctrl+D	Esci da Claude Code
`app:toggleTodos`	Ctrl+T	Attiva/disattiva la visibilità dell'elenco delle attività
`app:toggleTranscript`	Ctrl+O	Attiva/disattiva la trascrizione dettagliata

Azioni della cronologia

Azioni per navigare nella cronologia dei comandi:

Azione	Predefinito	Descrizione
`history:search`	Ctrl+R	Apri ricerca nella cronologia

Azione	Predefinito	Descrizione
`history:previous`	Su	Elemento della cronologia precedente
`history:next`	Giù	Elemento della cronologia successivo

Azioni della chat

Azioni disponibili nel contesto `Chat`:

Azione	Predefinito	Descrizione
`chat:cancel`	Escape	Annulla l'input corrente
`chat:cycleMode`	Shift+Tab*	Cicla le modalità di permesso
`chat:modelPicker`	Cmd+P / Meta+P	Apri il selezionatore di modelli
`chat:thinkingToggle`	Cmd+T / Meta+T	Attiva/disattiva il pensiero esteso
`chat:submit`	Invio	Invia il messaggio
`chat:undo`	Ctrl+_	Annulla l'ultima azione
`chat:externalEditor`	Ctrl+G	Apri nell'editor esterno
`chat:stash`	Ctrl+S	Nascondi il prompt corrente
`chat:imagePaste`	Ctrl+V (Alt+V su Windows)	Incolla immagine

*Su Windows senza modalità VT (Node <24.2.0/<22.17.0, Bun <1.2.23), il valore predefinito è Meta+M.

Azioni di autocompletamento

Azioni disponibili nel contesto `Autocomplete`:

Azione	Predefinito	Descrizione
`autocomplete:accept`	Tab	Accetta il suggerimento

Azione	Predefinito	Descrizione
`autocomplete:dismiss`	Escape	Chiudi il menu
`autocomplete:previous`	Su	Suggerimento precedente
`autocomplete:next`	Giù	Suggerimento successivo

Azioni di conferma

Azioni disponibili nel contesto `Confirmation`:

Azione	Predefinito	Descrizione
`confirm:yes`	Y, Invio	Conferma l'azione
`confirm:no`	N, Escape	Rifiuta l'azione
`confirm:previous`	Su	Opzione precedente
`confirm:next`	Giù	Opzione successiva
`confirm:nextField`	Tab	Campo successivo
`confirm:previousField`	(non associato)	Campo precedente
`confirm:cycleMode`	Shift+Tab	Cicla le modalità di permesso
`confirm:toggleExplanation`	Ctrl+E	Attiva/disattiva la spiegazione del permesso

Azioni di permesso

Azioni disponibili nel contesto `Confirmation` per i dialoghi di permesso:

Azione	Predefinito	Descrizione
`permission:toggleDebug`	Ctrl+D	Attiva/disattiva le informazioni di debug del permesso

Azioni di trascrizione

Azioni disponibili nel contesto `Transcript`:

Azione	Predefinito	Descrizione
`transcript:toggleShowAll`	Ctrl+E	Attiva/disattiva la visualizzazione di tutto il contenuto
`transcript:exit`	Ctrl+C, Escape	Esci dalla visualizzazione della trascrizione

Azioni di ricerca nella cronologia

Azioni disponibili nel contesto `HistorySearch`:

Azione	Predefinito	Descrizione
`historySearch:next`	Ctrl+R	Corrispondenza successiva
`historySearch:accept`	Escape, Tab	Accetta la selezione
`historySearch:cancel`	Ctrl+C	Annulla la ricerca
`historySearch:execute`	Invio	Esegui il comando selezionato

Azioni delle attività

Azioni disponibili nel contesto `Task`:

Azione	Predefinito	Descrizione
`task:background`	Ctrl+B	Attività in background corrente

Azioni del tema

Azioni disponibili nel contesto `ThemePicker`:

Azione	Predefinito	Descrizione
`theme:toggleSyntaxHighlighting`	Ctrl+T	Attiva/disattiva l'evidenziazione della sintassi

Azioni della guida

Azioni disponibili nel contesto `Help`:

Azione	Predefinito	Descrizione
`help:dismiss`	Escape	Chiudi il menu della guida

Azioni delle schede

Azioni disponibili nel contesto `Tabs`:

Azione	Predefinito	Descrizione
`tabs:next`	Tab, Destra	Scheda successiva
`tabs:previous`	Shift+Tab, Sinistra	Scheda precedente

Azioni degli allegati

Azioni disponibili nel contesto `Attachments`:

Azione	Predefinito	Descrizione
`attachments:next`	Destra	Allegato successivo
`attachments:previous`	Sinistra	Allegato precedente
`attachments:remove`	Backspace, Canc	Rimuovi l'allegato selezionato
`attachments:exit`	Giù, Escape	Esci dalla barra degli allegati

Azioni del piè di pagina

Azioni disponibili nel contesto `Footer`:

Azione	Predefinito	Descrizione
`footer:next`	Destra	Elemento del piè di pagina successivo
`footer:previous`	Sinistra	Elemento del piè di pagina precedente
`footer:openSelected`	Invio	Apri l'elemento del piè di pagina selezionato
`footer:clearSelection`	Escape	Cancella la selezione del piè di pagina

Azioni del selezionatore di messaggi

Azioni disponibili nel contesto `MessageSelector`:

Azione	Predefinito	Descrizione
`messageSelector:up`	Su, K, Ctrl+P	Sposta verso l'alto nell'elenco
`messageSelector:down`	Giù, J, Ctrl+N	Sposta verso il basso nell'elenco
`messageSelector:top`	Ctrl+Su, Shift+Su, Meta+Su, Shift+K	Salta all'inizio
`messageSelector:bottom`	Ctrl+Giù, Shift+Giù, Meta+Giù, Shift+J	Salta alla fine
`messageSelector:select`	Invio	Seleziona il messaggio

Azioni diff

Azioni disponibili nel contesto `DiffDialog`:

Azione	Predefinito	Descrizione
`diff:dismiss`	Escape	Chiudi il visualizzatore diff
`diff:previousSource`	Sinistra	Sorgente diff precedente
`diff:nextSource`	Destra	Sorgente diff successiva
`diff:previousFile`	Su	File precedente nel diff
`diff:nextFile`	Giù	File successivo nel diff
`diff:viewDetails`	Invio	Visualizza i dettagli del diff
`diff:back`	(specifico del contesto)	Torna indietro nel visualizzatore diff

Azioni del selezionatore di modelli

Azioni disponibili nel contesto `ModelPicker`:

Azione	Predefinito	Descrizione
`modelPicker:decreaseEffort`	Sinistra	Diminuisci il livello di sforzo
`modelPicker:increaseEffort`	Destra	Aumenta il livello di sforzo

Azioni di selezione

Azioni disponibili nel contesto `Select`:

Azione	Predefinito	Descrizione
`select:next`	Giù, J, Ctrl+N	Opzione successiva
`select:previous`	Su, K, Ctrl+P	Opzione precedente
`select:accept`	Invio	Accetta la selezione
`select:cancel`	Escape	Annulla la selezione

Azioni dei plugin

Azioni disponibili nel contesto `Plugin`:

Azione	Predefinito	Descrizione
`plugin:toggle`	Spazio	Attiva/disattiva la selezione del plugin
`plugin:install`	I	Installa i plugin selezionati

Azioni delle impostazioni

Azioni disponibili nel contesto `Settings`:

Azione	Predefinito	Descrizione
`settings:search`	/	Entra in modalità di ricerca
`settings:retry`	R	Riprova a caricare i dati di utilizzo (in caso di errore)

Sintassi delle sequenze di tasti

Modificatori

Utilizzare i tasti modificatori con il separatore `+`:

- `ctrl` o `control` - Tasto Control
- `alt`, `opt`, o `option` - Tasto Alt/Option
- `shift` - Tasto Shift
- `meta`, `cmd`, o `command` - Tasto Meta/Command

Ad esempio:

```
ctrl+k          Singolo tasto con modificatore
shift+tab       Shift + Tab
meta+p          Command/Meta + P
ctrl+shift+c    Più modificatori
```

Lettere maiuscole

Una lettera maiuscola autonoma implica Shift. Ad esempio, `K` è equivalente a `shift+k`. Questo è utile per i binding in stile vim dove i tasti maiuscoli e minuscoli hanno significati diversi.

Le lettere maiuscole con modificatori (ad es. `ctrl+K`) sono trattate come stilistiche e **non** implicano Shift — `ctrl+K` è lo stesso di `ctrl+k`.

Accordi

Gli accordi sono sequenze di tasti separate da spazi:

```
ctrl+k ctrl+s   Premi Ctrl+K, rilascia, quindi Ctrl+S
```

Tasti speciali

- `escape` o `esc` - Tasto Escape
- `enter` o `return` - Tasto Invio
- `tab` - Tasto Tab
- `space` - Barra spaziatrice
- `up`, `down`, `left`, `right` - Tasti freccia
- `backspace`, `delete` - Tasti Canc

Annulla l'associazione delle scorciatoie predefinite

Impostare un'azione su `null` per annullare l'associazione di una scorciatoia predefinita:

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

Scorciatoie riservate

Queste scorciatoie non possono essere riassociate:

Scorciatoia	Motivo
Ctrl+C	Interrupt/annullamento hardcoded
Ctrl+D	Uscita hardcoded

Conflitti del terminale

Alcune scorciatoie potrebbero entrare in conflitto con i multiplexer di terminale:

Scorciatoia	Conflitto
Ctrl+B	Prefisso tmux (premere due volte per inviare)
Ctrl+A	Prefisso GNU screen
Ctrl+Z	Sospensione del processo Unix (SIGTSTP)

Interazione con la modalità Vim

Quando la modalità vim è abilitata (`/vim`), i keybindings e la modalità vim operano indipendentemente:

- **Modalità Vim** gestisce l'input a livello di input di testo (movimento del cursore, modalità, movimenti)
- **Keybindings** gestisce le azioni a livello di componente (attiva/disattiva attività, invia, ecc.)
- Il tasto Escape in modalità vim passa da INSERT a NORMAL; non attiva `chat:cancel`
- La maggior parte delle scorciatoie Ctrl+tasto passano attraverso la modalità vim al sistema di keybinding
- In modalità vim NORMAL, `?` mostra il menu della guida (comportamento vim)

Convalida

Claude Code convalida i tuoi keybindings e mostra avvisi per:

- Errori di analisi (JSON non valido o struttura non valida)
- Nomi di contesto non validi
- Conflitti di scorciatoie riservate
- Conflitti di multiplexer di terminale
- Binding duplicati nello stesso contesto

Eseguire `/doctor` per visualizzare eventuali avvisi di keybinding.

Part 4: Configuration

Configurare le autorizzazioni

> *Controlla cosa Claude Code può accedere e fare con regole di autorizzazione granulari, modalità e criteri gestiti.*

Claude Code supporta autorizzazioni granulari in modo che tu possa specificare esattamente cosa l'agente è autorizzato a fare e cosa non può fare. Le impostazioni di autorizzazione possono essere archiviate nel controllo della versione e distribuite a tutti gli sviluppatori della tua organizzazione, nonché personalizzate dai singoli sviluppatori.

Sistema di autorizzazione

Claude Code utilizza un sistema di autorizzazione a livelli per bilanciare potenza e sicurezza:

Tipo di strumento	Esempio	Approvazione richiesta	Comportamento "Sì, non chiedere più"
Sola lettura	Letture di file, Grep	No	N/A
Comandi Bash	Esecuzione shell	Sì	Permanentemente per directory di progetto e comando
Modifica di file	Modifica/scrittura di file	Sì	Fino alla fine della sessione

Gestire le autorizzazioni

Puoi visualizzare e gestire le autorizzazioni degli strumenti di Claude Code con `/permissions`. Questa interfaccia utente elenca tutte le regole di autorizzazione e il file settings.json da cui provengono.

- Le regole **Allow** consentono a Claude Code di utilizzare lo strumento specificato senza approvazione manuale.

- Le regole **Ask** richiedono una conferma ogni volta che Claude Code tenta di utilizzare lo strumento specificato.
- Le regole **Deny** impediscono a Claude Code di utilizzare lo strumento specificato.

Le regole vengono valutate in ordine: **deny -> ask -> allow**. La prima regola corrispondente vince, quindi le regole deny hanno sempre la precedenza.

Modalità di autorizzazione

Claude Code supporta diverse modalità di autorizzazione che controllano come gli strumenti vengono approvati. Imposta `defaultMode` nei tuoi file di impostazioni:

Modalità	Descrizione
`default`	Comportamento standard: richiede l'autorizzazione al primo utilizzo di ogni strumento
`acceptEdits`	Accetta automaticamente le autorizzazioni di modifica dei file per la sessione
`plan`	Plan Mode: Claude può analizzare ma non modificare file o eseguire comandi
`dontAsk`	Nega automaticamente gli strumenti a meno che non siano pre-approvati tramite `/permissions` o regole `permissions.allow`
`bypassPermissions`	Salta tutti i prompt di autorizzazione (richiede un ambiente sicuro, vedi avviso di seguito)

> **Warning:**
>
> La modalità `bypassPermissions` disabilita tutti i controlli di autorizzazione. Utilizzala solo in ambienti isolati come contenitori o macchine virtuali dove Claude Code non può causare danni. Gli amministratori possono impedire questa modalità impostando `disableBypassPermissionsMode` su `"disable"` nelle impostazioni gestite.

Sintassi delle regole di autorizzazione

Le regole di autorizzazione seguono il formato `Tool` o `Tool(specifier)`.

Corrispondere a tutti gli utilizzi di uno strumento

Per corrispondere a tutti gli utilizzi di uno strumento, utilizza solo il nome dello strumento senza parentesi:

Regola	Effetto
`Bash`	Corrisponde a tutti i comandi Bash
`WebFetch`	Corrisponde a tutte le richieste di web fetch
`Read`	Corrisponde a tutte le letture di file

`Bash(*)` è equivalente a `Bash` e corrisponde a tutti i comandi Bash.

Utilizzare gli specificatori per il controllo granulare

Aggiungi uno specificatore tra parentesi per corrispondere a utilizzi specifici dello strumento:

Regola	Effetto
`Bash(npm run build)`	Corrisponde al comando esatto `npm run build`
`Read(./.env)`	Corrisponde alla lettura del file `.env` nella directory corrente
`WebFetch(domain:example.com)`	Corrisponde alle richieste di fetch a example.com

Modelli con caratteri jolly

Le regole Bash supportano modelli glob con `*`. I caratteri jolly possono apparire in qualsiasi posizione nel comando. Questa configurazione consente comandi npm e git commit mentre blocca git push:

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

Lo spazio prima di `*` è importante: `Bash(ls *)` corrisponde a `ls -la` ma non a `lsof`, mentre `Bash(ls*)` corrisponde a entrambi. La sintassi legacy con suffisso `:*` è equivalente a `*` ma è deprecata.

Regole di autorizzazione specifiche dello strumento

Bash

Le regole di autorizzazione Bash supportano la corrispondenza con caratteri jolly con `*`. I caratteri jolly possono apparire in qualsiasi posizione nel comando, incluso all'inizio, nel mezzo o alla fine:

- `Bash(npm run build)` corrisponde al comando Bash esatto `npm run build`
- `Bash(npm run test *)` corrisponde ai comandi Bash che iniziano con `npm run test`
- `Bash(npm *)` corrisponde a qualsiasi comando che inizia con `npm`
- `Bash(* install)` corrisponde a qualsiasi comando che termina con `install`
- `Bash(git * main)` corrisponde a comandi come `git checkout main`, `git merge main`

Quando `*` appare alla fine con uno spazio prima (come `Bash(ls *)`), applica un confine di parola, richiedendo che il prefisso sia seguito da uno spazio o dalla fine della stringa. Ad esempio, `Bash(ls *)` corrisponde a `ls -la` ma non a `lsof`. Al contrario, `Bash(ls*)` senza spazio corrisponde sia a `ls -la` che a `lsof` perché non c'è un vincolo di confine di parola.

Tip:

Claude Code è consapevole degli operatori shell (come `&&`) quindi una regola di corrispondenza del prefisso come `Bash(safe-cmd *)` non gli darà il permesso di eseguire il comando `safe-cmd && other-cmd`.

Warning:

I modelli di autorizzazione Bash che tentano di vincolare gli argomenti del comando sono fragili. Ad esempio, `Bash(curl http://github.com/ *)` intende limitare curl agli URL di GitHub, ma non corrisponderà a variazioni come:

- Opzioni prima dell'URL: `curl -X GET http://github.com/...`
- Protocollo diverso: `curl https://github.com/...`
- Reindirizzamenti: `curl -L http://bit.ly/xyz` (reindirizza a github)
- Variabili: `URL=http://github.com && curl $URL`
- Spazi extra: `curl http://github.com`

Per un filtraggio URL più affidabile, considera:

- **Limitare gli strumenti di rete Bash**: utilizza regole deny per bloccare `curl`, `wget` e comandi simili, quindi utilizza lo strumento WebFetch con l'autorizzazione `WebFetch(domain:github.com)` per i domini consentiti
- **Utilizzare hook PreToolUse**: implementa un hook che convalida gli URL nei comandi Bash e blocca i domini non consentiti
- Istruire Claude Code sui tuoi modelli curl consentiti tramite CLAUDE.md

Nota che l'utilizzo di WebFetch da solo non impedisce l'accesso alla rete. Se Bash è consentito, Claude può comunque utilizzare `curl`, `wget` o altri strumenti per raggiungere qualsiasi URL.

Read e Edit

Le regole `Edit` si applicano a tutti gli strumenti integrati che modificano i file. Claude fa un tentativo migliore per applicare le regole `Read` a tutti gli strumenti integrati che leggono file come Grep e Glob.

Le regole Read e Edit seguono entrambe la specifica gitignore con quattro tipi di modello distinti:

Modello	Significato	Esempio	Corrisponde
`//path`	Percorso **assoluto** dalla radice del filesystem	`Read(//Users/alice/secrets/**)`	`/Users/alice/secrets/**`

Modello	Significato	Esempio	Corrisponde
`~/path`	Percorso dalla directory **home**	`Read(~/Documents/*.pdf)`	`/Users/alice/Documents/*.pdf`
`/path`	Percorso **relativo alla radice del progetto**	`Edit(/src/**/*.ts)`	`<project root>/src/**/*.ts`
`path` o `./path`	Percorso **relativo alla directory corrente**	`Read(*.env)`	`<cwd>/*.env`

> **Warning:**
>
> Un modello come `/Users/alice/file` NON è un percorso assoluto. È relativo alla radice del progetto. Utilizza `//Users/alice/file` per i percorsi assoluti.

Esempi:

- `Edit(/docs/**)` : modifica in `<project>/docs/` (NON `/docs/` e NON `<project>/.claude/docs/`)
- `Read(~/.zshrc)` : legge il `.zshrc` della tua directory home
- `Edit(//tmp/scratch.txt)` : modifica il percorso assoluto `/tmp/scratch.txt`
- `Read(src/**)` : legge da `<current-directory>/src/`

> **Note:**
>
> Nei modelli gitignore, `*` corrisponde ai file in una singola directory mentre `**` corrisponde ricorsivamente tra le directory. Per consentire l'accesso a tutti i file, utilizza solo il nome dello strumento senza parentesi: `Read` , `Edit` o `Write` .

WebFetch

- `WebFetch(domain:example.com)` corrisponde alle richieste di fetch a example.com

MCP

- `mcp__puppeteer` corrisponde a qualsiasi strumento fornito dal server `puppeteer` (nome configurato in Claude Code)
- `mcp__puppeteer__*` sintassi con caratteri jolly che corrisponde anche a tutti gli strumenti dal server `puppeteer`
- `mcp__puppeteer__puppeteer_navigate` corrisponde allo strumento `puppeteer_navigate` fornito dal server `puppeteer`

Agent (subagents)

Utilizza le regole `Agent(AgentName)` per controllare quali subagents Claude può utilizzare:

- `Agent(Explore)` corrisponde al subagent Explore
- `Agent(Plan)` corrisponde al subagent Plan
- `Agent(my-custom-agent)` corrisponde a un subagent personalizzato denominato `my-custom-agent`

Aggiungi queste regole all'array `deny` nelle tue impostazioni o utilizza il flag CLI `--disallowedTools` per disabilitare agenti specifici. Per disabilitare l'agente Explore:

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

Estendere le autorizzazioni con hook

Gli hook di Claude Code forniscono un modo per registrare comandi shell personalizzati per eseguire la valutazione delle autorizzazioni in fase di esecuzione. Quando Claude Code effettua una chiamata di strumento, gli hook PreToolUse vengono eseguiti prima del sistema di autorizzazione e l'output dell'hook può determinare se approvare o negare la chiamata dello strumento al posto del sistema di autorizzazione.

Directory di lavoro

Per impostazione predefinita, Claude ha accesso ai file nella directory in cui è stato avviato. Puoi estendere questo accesso:

- **Durante l'avvio**: utilizza l'argomento CLI `--add-dir <path>`
- **Durante la sessione**: utilizza il comando `/add-dir`
- **Configurazione persistente**: aggiungi a `additionalDirectories` nei file di impostazioni

I file nelle directory aggiuntive seguono le stesse regole di autorizzazione della directory di lavoro originale: diventano leggibili senza prompt e le autorizzazioni di modifica dei file seguono la modalità di autorizzazione corrente.

Come le autorizzazioni interagiscono con il sandboxing

Le autorizzazioni e il sandboxing sono livelli di sicurezza complementari:

- **Autorizzazioni** controllano quali strumenti Claude Code può utilizzare e quali file o domini può accedere. Si applicano a tutti gli strumenti (Bash, Read, Edit, WebFetch, MCP e altri).
- **Sandboxing** fornisce l'applicazione a livello del sistema operativo che limita l'accesso al filesystem e alla rete dello strumento Bash. Si applica solo ai comandi Bash e ai loro processi figlio.

Utilizza entrambi per la difesa in profondità:

- Le regole deny di autorizzazione impediscono a Claude di tentare anche di accedere alle risorse limitate
- Le restrizioni sandbox impediscono ai comandi Bash di raggiungere risorse al di fuori dei confini definiti, anche se un'iniezione di prompt bypassa il processo decisionale di Claude
- Le restrizioni del filesystem nella sandbox utilizzano le regole deny di Read e Edit, non una configurazione sandbox separata
- Le restrizioni di rete combinano le regole di autorizzazione WebFetch con l'elenco `allowedDomains` della sandbox

Impostazioni gestite

Per le organizzazioni che necessitano di un controllo centralizzato sulla configurazione di Claude Code, gli amministratori possono distribuire impostazioni gestite che non possono essere ignorate dalle impostazioni utente o di progetto. Queste impostazioni di criteri seguono lo stesso formato dei file di impostazioni regolari e possono essere fornite tramite criteri MDM/a livello del sistema operativo, file di impostazioni gestite o impostazioni gestite dal server. Vedi file di impostazioni per i meccanismi di consegna e i percorsi dei file.

Impostazioni solo gestite

Alcune impostazioni sono efficaci solo nelle impostazioni gestite:

Impostazione	Descrizione
`disableBypassPermissionsMode`	Imposta su `"disable"` per impedire la modalità `bypassPermissions` e il flag `--dangerously-skip-permissions`

Impostazione	Descrizione
`allowManagedPermissionRulesOnly`	Quando `true`, impedisce alle impostazioni utente e di progetto di definire regole di autorizzazione `allow`, `ask` o `deny`. Si applicano solo le regole nelle impostazioni gestite
`allowManagedHooksOnly`	Quando `true`, impedisce il caricamento di hook utente, progetto e plugin. Sono consentiti solo hook gestiti e hook SDK
`allowManagedMcpServersOnly`	Quando `true`, solo `allowedMcpServers` dalle impostazioni gestite sono rispettati. `deniedMcpServers` si unisce comunque da tutte le fonti. Vedi Configurazione MCP gestita
`blockedMarketplaces`	Elenco di blocco delle fonti del marketplace. Le fonti bloccate vengono controllate prima del download, quindi non toccano mai il filesystem. Vedi restrizioni marketplace gestite
`sandbox.network.allowManagedDomainsOnly`	Quando `true`, solo `allowedDomains` e le regole allow `WebFetch(domain:...)` dalle impostazioni gestite sono rispettate. I domini non consentiti vengono bloccati automaticamente senza richiedere all'utente. I domini negati si uniscono comunque da tutte le fonti
`strictKnownMarketplaces`	Controlla quali marketplace di plugin gli utenti possono aggiungere. Vedi restrizioni marketplace gestite
`allow_remote_sessions`	Quando `true`, consente agli utenti di avviare Remote Control e sessioni web. Predefinito su `true`. Imposta su `false` per impedire l'accesso alle sessioni remote

Precedenza delle impostazioni

Le regole di autorizzazione seguono la stessa precedenza delle impostazioni di tutte le altre impostazioni di Claude Code:

1. **Impostazioni gestite**: non possono essere ignorate da nessun altro livello, inclusi gli argomenti della riga di comando
2. **Argomenti della riga di comando**: override temporanei della sessione
3. **Impostazioni di progetto locale** (`.claude/settings.local.json`)
4. **Impostazioni di progetto condivise** (`.claude/settings.json`)
5. **Impostazioni utente** (`~/.claude/settings.json`)

Se uno strumento viene negato a qualsiasi livello, nessun altro livello può consentirlo. Ad esempio, un deny delle impostazioni gestite non può essere ignorato da `--allowedTools` e `--disallowedTools` può aggiungere restrizioni oltre a quelle definite dalle impostazioni gestite.

Se un'autorizzazione è consentita nelle impostazioni utente ma negata nelle impostazioni di progetto, l'impostazione di progetto ha la precedenza e l'autorizzazione viene bloccata.

Configurazioni di esempio

Questo repository include configurazioni di impostazioni iniziali per scenari di distribuzione comuni. Utilizzale come punti di partenza e adattale alle tue esigenze.

Vedi anche

- Settings: riferimento di configurazione completo inclusa la tabella delle impostazioni di autorizzazione
- Sandboxing: isolamento del filesystem e della rete a livello del sistema operativo per i comandi Bash
- Authentication: configura l'accesso utente a Claude Code
- Security: salvaguardie di sicurezza e best practice
- Hooks: automatizza i flussi di lavoro ed estendi la valutazione delle autorizzazioni

Come Claude ricorda il vostro progetto

> *Fornite a Claude istruzioni persistenti con file CLAUDE.md e lasciate che Claude accumuli apprendimenti automaticamente con la memoria automatica.*

Ogni sessione di Claude Code inizia con una finestra di contesto nuova. Due meccanismi trasportano la conoscenza tra le sessioni:

- **File CLAUDE.md**: istruzioni che scrivete per dare a Claude un contesto persistente
- **Memoria automatica**: note che Claude scrive da solo in base alle vostre correzioni e preferenze

Questa pagina spiega come:

- Scrivere e organizzare file CLAUDE.md
- Limitare le regole a tipi di file specifici con `.claude/rules/`
- Configurare la memoria automatica in modo che Claude prenda note automaticamente
- Risolvere i problemi quando le istruzioni non vengono seguite

CLAUDE.md vs memoria automatica

Claude Code ha due sistemi di memoria complementari. Entrambi vengono caricati all'inizio di ogni conversazione. Claude li tratta come contesto, non come configurazione forzata. Più specifiche e concise sono le vostre istruzioni, più coerentemente Claude le segue.

	File CLAUDE.md	Memoria automatica
Chi lo scrive	Voi	Claude
Cosa contiene	Istruzioni e regole	Apprendimenti e modelli
Ambito	Progetto, utente o organizzazione	Per worktree
Caricato in	Ogni sessione	Ogni sessione (prime 200 righe)
Usare per	Standard di codifica, flussi di lavoro, architettura del progetto	Comandi di compilazione, approfondimenti sul debug, preferenze che Claude scopre

Usate i file CLAUDE.md quando volete guidare il comportamento di Claude. La memoria automatica permette a Claude di imparare dalle vostre correzioni senza sforzo manuale.

I subagents possono anche mantenere la loro propria memoria automatica. Consultate la configurazione dei subagent per i dettagli.

File CLAUDE.md

I file CLAUDE.md sono file markdown che forniscono a Claude istruzioni persistenti per un progetto, il vostro flusso di lavoro personale o l'intera organizzazione. Scrivete questi file in testo semplice; Claude li legge all'inizio di ogni sessione.

Scegliere dove mettere i file CLAUDE.md

I file CLAUDE.md possono trovarsi in diversi percorsi, ognuno con un ambito diverso. I percorsi più specifici hanno la precedenza su quelli più ampi.

Ambito	Percorso	Scopo	Esempi di casi d'uso	Condiviso con
Politica gestita	• macOS: `/Library/Application Support/ClaudeCode/CLAUDE.md` • Linux e WSL: `/etc/claude-code/CLAUDE.md` • Windows: `C:\Program Files\ClaudeCode\CLAUDE.md`	Istruzioni a livello organizzativo gestite da IT/DevOps	Standard di codifica aziendale, politiche di sicurezza, requisiti di conformità	Tutti gli utenti dell'organizzazione
Istruzioni del progetto	`./CLAUDE.md` o `./.claude/CLAUDE.md`	Istruzioni condivise dal team per il progetto	Architettura del progetto, standard di codifica, flussi di lavoro comuni	Membri del team tramite controllo del codice sorgente
Istruzioni dell'utente	`~/.claude/CLAUDE.md`	Preferenze personali per tutti i progetti	Preferenze di stile del codice, scorciatoie di strumenti personali	Solo voi (tutti i progetti)

I file CLAUDE.md nella gerarchia di directory sopra la directory di lavoro vengono caricati completamente all'avvio. I file CLAUDE.md nelle sottodirectory vengono caricati su richiesta quando Claude legge i file in quelle directory. Consultate Come vengono caricati i file CLAUDE.md per l'ordine di risoluzione completo.

Per i progetti di grandi dimensioni, potete suddividere le istruzioni in file specifici per argomento utilizzando regole di progetto. Le regole vi permettono di limitare le istruzioni a tipi di file specifici o sottodirectory.

Configurare un CLAUDE.md di progetto

Un CLAUDE.md di progetto può essere archiviato in `./CLAUDE.md` o `./.claude/CLAUDE.md`. Create questo file e aggiungete istruzioni che si applicano a chiunque lavori sul progetto: comandi di compilazione e test, standard di codifica, decisioni architettoniche, convenzioni di denominazione e flussi di lavoro comuni. Queste istruzioni vengono condivise con il vostro team tramite controllo del codice sorgente, quindi concentratevi su standard a livello di progetto piuttosto che su preferenze personali.

> **Tip:**
>
> Eseguite `/init` per generare automaticamente un CLAUDE.md iniziale. Claude analizza la vostra base di codice e crea un file con comandi di compilazione, istruzioni di test e convenzioni di progetto che scopre. Se un CLAUDE.md esiste già, `/init` suggerisce miglioramenti piuttosto che sovrascriverlo. Perfezionate da lì con istruzioni che Claude non scoprirebbe da solo.

Scrivere istruzioni efficaci

I file CLAUDE.md vengono caricati nella finestra di contesto all'inizio di ogni sessione, consumando token insieme alla vostra conversazione. Poiché sono contesto e non configurazione forzata, il modo in cui scrivete le istruzioni influisce su quanto affidabilmente Claude le segue. Le istruzioni specifiche, concise e ben strutturate funzionano meglio.

Dimensione: mirate a meno di 200 righe per file CLAUDE.md. I file più lunghi consumano più contesto e riducono l'aderenza. Se le vostre istruzioni stanno crescendo molto, dividetele utilizzando importazioni o file `.claude/rules/`.

Struttura: usate intestazioni markdown e punti elenco per raggruppare le istruzioni correlate. Claude scansiona la struttura nello stesso modo in cui i lettori lo fanno: le sezioni organizzate sono più facili da seguire rispetto ai paragrafi densi.

Specificità: scrivete istruzioni abbastanza concrete da poter verificare. Ad esempio:

- "Usate l'indentazione a 2 spazi" invece di "Formattate il codice correttamente"

- “Eseguite `npm test` prima di eseguire il commit” invece di “Testate le vostre modifiche”
- “I gestori API si trovano in `src/api/handlers/` ” invece di “Mantenete i file organizzati”

Coerenza: se due regole si contraddicono a vicenda, Claude potrebbe sceglierne una arbitrariamente. Esaminate periodicamente i vostri file CLAUDE.md, i file CLAUDE.md annidati nelle sottodirectory e i file `.claude/rules/` per rimuovere istruzioni obsolete o conflittuali. Nei monorepo, usate `claudeMdExcludes` per saltare i file CLAUDE.md di altri team che non sono rilevanti per il vostro lavoro.

Importare file aggiuntivi

I file CLAUDE.md possono importare file aggiuntivi utilizzando la sintassi `@path/to/import` . I file importati vengono espansi e caricati nel contesto all’avvio insieme al CLAUDE.md che li riferisce.

Sono consentiti sia i percorsi relativi che assoluti. I percorsi relativi si risolvono rispetto al file che contiene l’importazione, non alla directory di lavoro. I file importati possono importare ricorsivamente altri file, con una profondità massima di cinque hop.

Per includere un README, package.json e una guida al flusso di lavoro, fate riferimento ad essi con la sintassi `@` in qualsiasi punto del vostro CLAUDE.md:

```
Consultate @README per una panoramica del progetto e @package.json per i comandi
npm disponibili per questo progetto.

## Istruzioni aggiuntive
- flusso di lavoro git @docs/git-instructions.md
```

Per le preferenze personali che non volete archiviare, importate un file dalla vostra home directory. L’importazione va nel CLAUDE.md condiviso, ma il file a cui punta rimane sulla vostra macchina:

```
## Preferenze individuali
- @~/.claude/my-project-instructions.md
```

> **Warning:**

> La prima volta che Claude Code incontra importazioni esterne in un progetto, mostra una finestra di dialogo di approvazione che elenca i file. Se rifiutate, le importazioni rimangono disabilitate e la finestra di dialogo non appare di nuovo.

Per un approccio più strutturato all'organizzazione delle istruzioni, consultate `.claude/rules/`.

Come vengono caricati i file CLAUDE.md

Claude Code legge i file CLAUDE.md camminando verso l'alto nell'albero delle directory dalla vostra directory di lavoro corrente, controllando ogni directory lungo il percorso. Ciò significa che se eseguite Claude Code in `foo/bar/`, carica le istruzioni sia da `foo/bar/CLAUDE.md` che da `foo/CLAUDE.md`.

Claude scopre anche i file CLAUDE.md nelle sottodirectory sotto la vostra directory di lavoro corrente. Invece di caricarli all'avvio, vengono inclusi quando Claude legge i file in quelle sottodirectory.

Se lavorate in un grande monorepo dove i file CLAUDE.md di altri team vengono raccolti, usate `claudeMdExcludes` per saltarli.

Caricare da directory aggiuntive

Il flag `--add-dir` dà a Claude accesso a directory aggiuntive al di fuori della vostra directory di lavoro principale. Per impostazione predefinita, i file CLAUDE.md da queste directory non vengono caricati.

Per caricare anche i file CLAUDE.md da directory aggiuntive, inclusi `CLAUDE.md`, `.claude/CLAUDE.md` e `.claude/rules/*.md`, impostate la variabile di ambiente `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD`:

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

Organizzare le regole con `.claude/rules/`

Per i progetti più grandi, potete organizzare le istruzioni in più file utilizzando la directory `.claude/rules/`. Ciò mantiene le istruzioni modulari e più facili da mantenere per i team. Le regole possono anche essere limitate a percorsi di file specifici, quindi vengono caricate nel contesto solo quando Claude lavora con file corrispondenti, riducendo il rumore e risparmiando spazio di contesto.

> **Note:**

> Le regole vengono caricate nel contesto ogni sessione o quando vengono aperti file corrispondenti. Per le istruzioni specifiche di un'attività che non devono essere nel contesto tutto il tempo, usate skills invece, che vengono caricate solo quando le richiamate o quando Claude determina che sono rilevanti per il vostro prompt.

Configurare le regole

Posizionate i file markdown nella directory `.claude/rules/` del vostro progetto. Ogni file dovrebbe coprire un argomento, con un nome file descrittivo come `testing.md` o `api-design.md`. Tutti i file `.md` vengono scoperti ricorsivamente, quindi potete organizzare le regole in sottodirectory come `frontend/` o `backend/`:

```
your-project/
├── .claude/
│   ├── CLAUDE.md           # Istruzioni principali del progetto
│   └── rules/
│       ├── code-style.md   # Linee guida di stile del codice
│       ├── testing.md      # Convenzioni di test
│       └── security.md     # Requisiti di sicurezza
```

Le regole senza frontmatter `paths` vengono caricate all'avvio con la stessa priorità di `.claude/CLAUDE.md`.

Regole specifiche del percorso

Le regole possono essere limitate a file specifici utilizzando il frontmatter YAML con il campo `paths`. Queste regole condizionali si applicano solo quando Claude lavora con file che corrispondono ai modelli specificati.

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

## Regole di sviluppo API

- Tutti gli endpoint API devono includere la convalida dell'input
- Usate il formato di risposta di errore standard
- Includete commenti di documentazione OpenAPI
```

Le regole senza un campo `paths` vengono caricate incondizionatamente e si applicano a tutti i file. Le regole con ambito di percorso si attivano quando Claude legge file che corrispondono al modello, non ad ogni utilizzo dello strumento.

Usate i modelli glob nel campo `paths` per abbinare i file per estensione, directory o qualsiasi combinazione:

Modello	Corrisponde a
`**/*.ts`	Tutti i file TypeScript in qualsiasi directory
`src/**/*`	Tutti i file sotto la directory `src/`
`*.md`	File Markdown nella radice del progetto
`src/components/*.tsx`	Componenti React in una directory specifica

Potete specificare più modelli e usare l'espansione tra parentesi graffe per abbinare più estensioni in un modello:

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

Condividere le regole tra i progetti con symlink

La directory `.claude/rules/` supporta i symlink, quindi potete mantenere un set di regole condivise e collegarle a più progetti. I symlink vengono risolti e caricati normalmente, e i symlink circolari vengono rilevati e gestiti correttamente.

Questo esempio collega sia una directory condivisa che un file individuale:

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

Regole a livello di utente

Le regole personali in `~/.claude/rules/` si applicano a ogni progetto sulla vostra macchina. Usatele per le preferenze che non sono specifiche del progetto:

```
~/.claude/rules/
├── preferences.md    # Le vostre preferenze di codifica personali
└── workflows.md      # I vostri flussi di lavoro preferiti
```

Le regole a livello di utente vengono caricate prima delle regole di progetto, dando alle regole di progetto una priorità più alta.

Gestire CLAUDE.md per team di grandi dimensioni

Per le organizzazioni che distribuiscono Claude Code tra i team, potete centralizzare le istruzioni e controllare quali file CLAUDE.md vengono caricati.

Distribuire CLAUDE.md a livello organizzativo

Le organizzazioni possono distribuire un CLAUDE.md gestito centralmente che si applica a tutti gli utenti su una macchina. Questo file non può essere escluso dalle impostazioni individuali.

Step 1: Creare il file nel percorso della politica gestita

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

Step 2: Distribuire con il vostro sistema di gestione della configurazione

Usate MDM, Group Policy, Ansible o strumenti simili per distribuire il file tra le macchine degli sviluppatori. Consultate impostazioni gestite per altre opzioni di configurazione a livello organizzativo.

Escludere file CLAUDE.md specifici

Nei grandi monorepo, i file CLAUDE.md antenati possono contenere istruzioni che non sono rilevanti per il vostro lavoro. L'impostazione `claudeMdExcludes` vi permette di saltare file specifici per percorso o modello glob.

Questo esempio esclude un CLAUDE.md di primo livello e una directory di regole da una cartella padre. Aggiungetelo a `.claude/settings.local.json` in modo che l'esclusione rimanga locale alla vostra macchina:

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

I modelli vengono abbinati ai percorsi di file assoluti utilizzando la sintassi glob. Potete configurare `claudeMdExcludes` in qualsiasi livello di impostazioni: utente, progetto, locale o politica gestita. Gli array si uniscono tra i livelli.

I file CLAUDE.md della politica gestita non possono essere esclusi. Ciò garantisce che le istruzioni a livello organizzativo si applichino sempre indipendentemente dalle impostazioni individuali.

Memoria automatica

La memoria automatica permette a Claude di accumulare conoscenze tra le sessioni senza che scriviate nulla. Claude salva note per se stesso mentre lavora: comandi di compilazione, approfondimenti sul debug, note sull'architettura, preferenze di stile del codice e abitudini di flusso di lavoro. Claude non salva qualcosa ad ogni sessione. Decide cosa vale la pena ricordare in base al fatto che l'informazione sarebbe utile in una conversazione futura.

> **Note:**
>
> La memoria automatica richiede Claude Code v2.1.59 o successivo. Controllate la vostra versione con `claude --version`.

Abilitare o disabilitare la memoria automatica

La memoria automatica è attivata per impostazione predefinita. Per attivarla/disattivarla, aprite `/memory` in una sessione e usate l'interruttore di memoria automatica, oppure impostate `autoMemoryEnabled` nelle impostazioni del vostro progetto:

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

Per disabilitare la memoria automatica tramite variabile di ambiente, impostate `CLAUDE_CODE_DISABLE_AUTO_MEMORY=1`.

Percorso di archiviazione

Ogni progetto ottiene la sua propria directory di memoria in `~/.claude/projects/<project>/memory/`. Il percorso `<project>` è derivato dal repository git, quindi tutti i worktree e le sottodirectory all'interno dello stesso repo condividono una directory di memoria automatica. Al di fuori di un repository git, viene utilizzata la radice del progetto.

Per archiviare la memoria automatica in una posizione diversa, impostate `autoMemoryDirectory` nelle vostre impostazioni utente o locali:

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

Questa impostazione è accettata dalle impostazioni di politica, locali e utente. Non è accettata dalle impostazioni di progetto (`.claude/settings.json`) per evitare che un progetto condiviso reindirizza le scritture di memoria automatica a posizioni sensibili.

La directory contiene un punto di ingresso `MEMORY.md` e file di argomento opzionali:

```
~/.claude/projects/<project>/memory/
├── MEMORY.md          # Indice conciso, caricato in ogni sessione
├── debugging.md       # Note dettagliate sui modelli di debug
├── api-conventions.md # Decisioni di progettazione API
└── ...                # Qualsiasi altro file di argomento che Claude crea
```

`MEMORY.md` funge da indice della directory di memoria. Claude legge e scrive file in questa directory durante la vostra sessione, utilizzando `MEMORY.md` per tenere traccia di ciò che è archiviato dove.

La memoria automatica è locale alla macchina. Tutti i worktree e le sottodirectory all'interno dello stesso repository git condividono una directory di memoria automatica. I file non vengono condivisi tra macchine o ambienti cloud.

Come funziona

Le prime 200 righe di `MEMORY.md` vengono caricate all'inizio di ogni conversazione. Il contenuto oltre la riga 200 non viene caricato all'inizio della sessione. Claude mantiene `MEMORY.md` conciso spostando le note dettagliate in file di argomento separati.

Questo limite di 200 righe si applica solo a `MEMORY.md`. I file CLAUDE.md vengono caricati completamente indipendentemente dalla lunghezza, anche se i file più brevi producono una migliore aderenza.

I file di argomento come `debugging.md` o `patterns.md` non vengono caricati all'avvio. Claude li legge su richiesta utilizzando i suoi strumenti di file standard quando ha bisogno delle informazioni.

Claude legge e scrive file di memoria durante la vostra sessione. Quando vedete "Writing memory" o "Recalled memory" nell'interfaccia di Claude Code, Claude sta attivamente aggiornando o leggendo da `~/.claude/projects/<project>/memory/`.

Controllare e modificare la vostra memoria

I file di memoria automatica sono markdown semplice che potete modificare o eliminare in qualsiasi momento. Eseguite `/memory` per sfogliare e aprire i file di memoria da una sessione.

Visualizzare e modificare con `/memory`

Il comando `/memory` elenca tutti i file CLAUDE.md e rules caricati nella vostra sessione corrente, vi permette di attivare o disattivare la memoria automatica e fornisce un collegamento per aprire la cartella di memoria automatica. Selezionate qualsiasi file per aprirlo nel vostro editor.

Quando chiedete a Claude di ricordare qualcosa, come "usa sempre pnpm, non npm" o "ricorda che i test API richiedono un'istanza Redis locale", Claude lo salva nella memoria automatica. Per aggiungere istruzioni a CLAUDE.md, chiedete direttamente a Claude, come "aggiungi questo a CLAUDE.md", oppure modificate il file voi stessi tramite `/memory`.

Risolvere i problemi di memoria

Questi sono i problemi più comuni con CLAUDE.md e la memoria automatica, insieme ai passaggi per risolverli.

Claude non sta seguendo il mio CLAUDE.md

CLAUDE.md è contesto, non applicazione. Claude lo legge e cerca di seguirlo, ma non c'è garanzia di conformità rigorosa, specialmente per istruzioni vaghe o conflittuali.

Per risolvere i problemi:

- Eseguite `/memory` per verificare che i vostri file CLAUDE.md vengono caricati. Se un file non è elencato, Claude non può vederlo.

- Controllate che il CLAUDE.md rilevante si trovi in una posizione che viene caricata per la vostra sessione (consultate Scegliere dove mettere i file CLAUDE.md).
- Rendete le istruzioni più specifiche. “Usate l’indentazione a 2 spazi” funziona meglio di “formattate il codice bene”.
- Cercate istruzioni conflittuali tra i file CLAUDE.md. Se due file danno una guida diversa per lo stesso comportamento, Claude potrebbe sceglierne una arbitrariamente.

> **Tip:**
>
> Usate l’hook `InstructionsLoaded` per registrare esattamente quali file di istruzioni vengono caricati, quando vengono caricati e perché. Questo è utile per il debug delle regole specifiche del percorso o dei file caricati pigriamente nelle sottodirectory.

Non so cosa ha salvato la memoria automatica

Eseguite `/memory` e selezionate la cartella di memoria automatica per sfogliare ciò che Claude ha salvato. Tutto è markdown semplice che potete leggere, modificare o eliminare.

Il mio CLAUDE.md è troppo grande

I file con più di 200 righe consumano più contesto e possono ridurre l’aderenza. Spostate il contenuto dettagliato in file separati a cui si fa riferimento con importazioni `@path` (consultate Importare file aggiuntivi), oppure dividete le vostre istruzioni tra file `.claude/rules/`.

Le istruzioni sembrano perse dopo `/compact`

CLAUDE.md sopravvive completamente alla compattazione. Dopo `/compact`, Claude rilegge il vostro CLAUDE.md dal disco e lo reinietta fresco nella sessione. Se un’istruzione è scomparsa dopo la compattazione, è stata data solo nella conversazione, non scritta in CLAUDE.md. Aggiungetela a CLAUDE.md per farla persistere tra le sessioni.

Consultate Scrivere istruzioni efficaci per una guida su dimensione, struttura e specificità.

Risorse correlate

- Skills: pacchetto di flussi di lavoro ripetibili che vengono caricati su richiesta
- Impostazioni: configurare il comportamento di Claude Code con file di impostazioni
- Gestire le sessioni: gestire il contesto, riprendere le conversazioni ed eseguire sessioni parallele
- Memoria dei subagent: permettere ai subagent di mantenere la loro propria memoria automatica

Impostazioni di Claude Code

> *Configura Claude Code con impostazioni globali e a livello di progetto, e variabili di ambiente.*

Claude Code offre una varietà di impostazioni per configurare il suo comportamento in base alle tue esigenze. Puoi configurare Claude Code eseguendo il comando `/config` quando utilizzi il REPL interattivo, che apre un'interfaccia Impostazioni con schede dove puoi visualizzare informazioni di stato e modificare le opzioni di configurazione.

Ambiti di configurazione

Claude Code utilizza un **sistema di ambiti** per determinare dove si applicano le configurazioni e con chi vengono condivise. Comprendere gli ambiti ti aiuta a decidere come configurare Claude Code per uso personale, collaborazione di team o distribuzione aziendale.

Ambiti disponibili

Ambito	Posizione	Chi interessa	Condiviso con il team?
Managed	Impostazioni gestite dal server, plist / registro, o `managed-settings.json` a livello di sistema	Tutti gli utenti sulla macchina	Sì (distribuito da IT)
User	Directory `~/.claude/`	Tu, in tutti i progetti	No
Project	`.claude/` nel repository	Tutti i collaboratori su questo repository	Sì (committato su git)
Local	`.claude/settings.local.json`	Tu, solo in questo repository	No (gitignored)

Quando utilizzare ogni ambito

L'ambito **Managed** è per:

- Politiche di sicurezza che devono essere applicate a livello organizzativo
- Requisiti di conformità che non possono essere ignorati

- Configurazioni standardizzate distribuite da IT/DevOps

L'ambito **User** è migliore per:

- Preferenze personali che desideri ovunque (temi, impostazioni dell'editor)
- Strumenti e plugin che utilizzi in tutti i progetti
- Chiavi API e autenticazione (archiviate in modo sicuro)

L'ambito **Project** è migliore per:

- Impostazioni condivise dal team (permessi, hooks, MCP servers)
- Plugin che l'intero team dovrebbe avere
- Standardizzazione degli strumenti tra i collaboratori

L'ambito **Local** è migliore per:

- Override personali per un progetto specifico
- Test delle configurazioni prima di condividerle con il team
- Impostazioni specifiche della macchina che non funzioneranno per altri

Come interagiscono gli ambiti

Quando la stessa impostazione è configurata in più ambiti, gli ambiti più specifici hanno la precedenza:

1. **Managed** (più alta) - non può essere ignorata da nulla
2. **Argomenti della riga di comando** - override temporanei della sessione
3. **Local** - ignora le impostazioni di progetto e utente
4. **Project** - ignora le impostazioni utente
5. **User** (più bassa) - si applica quando nient'altro specifica l'impostazione

Ad esempio, se un permesso è consentito nelle impostazioni utente ma negato nelle impostazioni di progetto, l'impostazione di progetto ha la precedenza e il permesso è bloccato.

Cosa utilizza gli ambiti

Gli ambiti si applicano a molte funzionalità di Claude Code:

Funzionalità	Posizione utente	Posizione progetto	Posizione locale
Settings	`~/.claude/settings.json`	`.claude/settings.json`	`.claude/settings.local.json`

Funzionalità	Posizione utente	Posizione progetto	Posizione locale
Subagents	`~/.claude/agents/`	`.claude/agents/`	—
MCP servers	`~/.claude.json`	`.mcp.json`	`~/.claude.json` (per-progetto)
Plugins	`~/.claude/settings.json`	`.claude/settings.json`	`.claude/settings.local.json`
CLAUDE.md	`~/.claude/CLAUDE.md`	`CLAUDE.md` o `.claude/CLAUDE.md`	—

File di impostazioni

Il file `settings.json` è il nostro meccanismo ufficiale per configurare Claude Code attraverso impostazioni gerarchiche:

- **Le impostazioni utente** sono definite in `~/.claude/settings.json` e si applicano a tutti i progetti.
- **Le impostazioni di progetto** vengono salvate nella directory del tuo progetto:
  - `.claude/settings.json` per le impostazioni che vengono controllate nel controllo del codice sorgente e condivise con il tuo team
  - `.claude/settings.local.json` per le impostazioni che non vengono controllate, utili per preferenze personali e sperimentazione. Claude Code configurerà git per ignorare `.claude/settings.local.json` quando viene creato.
- **Impostazioni gestite**: Per le organizzazioni che necessitano di controllo centralizzato, Claude Code supporta più meccanismi di distribuzione per le impostazioni gestite. Tutti utilizzano lo stesso formato JSON e non possono essere ignorati dalle impostazioni utente o di progetto:
  - **Impostazioni gestite dal server**: consegnate dai server di Anthropic tramite la console di amministrazione di Claude.ai. Vedi impostazioni gestite dal server.
  - **Politiche MDM/a livello di sistema operativo**: consegnate tramite la gestione nativa dei dispositivi su macOS e Windows:

- macOS: dominio delle preferenze gestite `com.anthropic.claudecode` (distribuito tramite profili di configurazione in Jamf, Kandji o altri strumenti MDM)
- Windows: chiave di registro `HKLM\SOFTWARE\Policies\ClaudeCode` con un valore `Settings` (REG_SZ o REG_EXPAND_SZ) contenente JSON (distribuito tramite Criteri di gruppo o Intune)
- Windows (a livello utente): `HKCU\SOFTWARE\Policies\ClaudeCode` (priorità di politica più bassa, utilizzata solo quando non esiste alcuna fonte a livello di amministratore)
- **Basate su file**: `managed-settings.json` e `managed-mcp.json` distribuite alle directory di sistema:
- macOS: `/Library/Application Support/ClaudeCode/`
- Linux e WSL: `/etc/claude-code/`
- Windows: `C:\Program Files\ClaudeCode\`

Vedi impostazioni gestite e Configurazione MCP gestita per i dettagli.

> **Note:**
>
> Le distribuzioni gestite possono anche limitare le **aggiunte del marketplace dei plugin** utilizzando `strictKnownMarketplaces`. Per ulteriori informazioni, vedi Restrizioni del marketplace gestito.

- **Altre configurazioni** vengono archiviate in `~/.claude.json`. Questo file contiene le tue preferenze (tema, impostazioni di notifica, modalità editor), sessione OAuth, configurazioni dei server MCP per gli ambiti utente e locale, stato per-progetto (strumenti consentiti, impostazioni di fiducia) e varie cache. I server MCP con ambito di progetto vengono archiviati separatamente in `.mcp.json`.

> **Note:**
>
> Claude Code crea automaticamente backup con timestamp dei file di configurazione e conserva i cinque backup più recenti per prevenire la perdita di dati.

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "allow": [
      "Bash(npm run lint)",
      "Bash(npm run test *)",
      "Read(~/.zshrc)"
    ],
    "deny": [
      "Bash(curl *)",
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(./secrets/**)"
    ]
  },
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_METRICS_EXPORTER": "otlp"
  },
  "companyAnnouncements": [
    "Welcome to Acme Corp! Review our code guidelines at docs.acme.com",
    "Reminder: Code reviews required for all PRs",
    "New security policy in effect"
  ]
}
```

La riga `$schema` nell'esempio sopra punta allo schema JSON ufficiale per le impostazioni di Claude Code. Aggiungerlo al tuo `settings.json` abilita l'autocompletamento e la convalida inline in VS Code, Cursor e qualsiasi altro editor che supporta la convalida dello schema JSON.

Impostazioni disponibili

`settings.json` supporta un numero di opzioni:

Chiave	Descrizione	Esempio
`apiKeyHelper`	Script personalizzato, da eseguire in `/bin/sh`, per generare un valore di autenticazione. Questo valore verrà inviato come intestazioni `X-Api-Key` e `Authorization: Bearer` per le richieste del modello	`/bin/generate_temp_api_key.sh`
`cleanupPeriodDays`	Le sessioni inattive per più tempo di questo periodo vengono eliminate all'avvio. L'impostazione a `0` elimina immediatamente tutte le sessioni. (predefinito: 30 giorni)	`20`
`companyAnnouncements`	Annuncio da visualizzare agli utenti all'avvio. Se vengono forniti più annunci, verranno alternati casualmente.	`["Welcome to Acme Corp! Review our code guidelines at docs.acme.com"]`
`env`	Variabili di ambiente che verranno applicate a ogni sessione	`{"FOO": "bar"}`
`attribution`	Personalizza l'attribuzione per i commit git e le pull request. Vedi Impostazioni di attribuzione	`{"commit": "🤖 Generated with Claude Code", "pr": ""}`
`includeCoAuthoredBy`	**Deprecato**: Usa `attribution` invece. Se includere la riga `co-authored-by Claude` nei commit git e nelle pull request (predefinito: `true`)	`false`
`includeGitInstructions`	Includi le istruzioni integrate del flusso di lavoro di commit e PR nel prompt di sistema di Claude (predefinito: `true`). Impostare su `false` per rimuovere queste istruzioni, ad esempio quando si utilizzano le proprie skill di flusso di lavoro git. La variabile di ambiente `CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS` ha la precedenza su questa impostazione quando impostata	`false`

Chiave	Descrizione	Esempio
`permissions`	Vedi la tabella sottostante per la struttura dei permessi.	
`hooks`	Configura comandi personalizzati da eseguire agli eventi del ciclo di vita. Vedi documentazione hooks per il formato	Vedi hooks
`disableAllHooks`	Disabilita tutti gli hooks e qualsiasi riga di stato personalizzata	`true`
`allowManagedHooksOnly`	(Solo impostazioni gestite) Impedisci il caricamento degli hook utente, progetto e plugin. Consenti solo gli hook gestiti e gli hook SDK. Vedi Configurazione hook	`true`
`allowedHttpHookUrls`	Elenco di autorizzazione dei modelli di URL che gli hook HTTP possono indirizzare. Supporta `*` come carattere jolly. Quando impostato, gli hook con URL non corrispondenti vengono bloccati. Non definito = nessuna restrizione, array vuoto = blocca tutti gli hook HTTP. Gli array si uniscono tra le fonti di impostazioni. Vedi Configurazione hook	`["https://hooks.example.com/*"]`
`httpHookAllowedEnvVars`	Elenco di autorizzazione dei nomi delle variabili di ambiente che gli hook HTTP possono interpolare nelle intestazioni. Quando impostato, l'`allowedEnvVars` effettivo di ogni hook è l'intersezione con questo elenco. Non definito = nessuna restrizione. Gli array si uniscono tra le fonti di impostazioni. Vedi Configurazione hook	`["MY_TOKEN", "HOOK_SECRET"]`

Chiave	Descrizione	Esempio
`allowManagedPermissionRulesOnly`	(Solo impostazioni gestite) Impedisci alle impostazioni utente e progetto di definire regole di permesso `allow`, `ask` o `deny`. Si applicano solo le regole nelle impostazioni gestite. Vedi Impostazioni solo gestite	`true`
`allowManagedMcpServersOnly`	(Solo impostazioni gestite) Solo `allowedMcpServers` dalle impostazioni gestite vengono rispettati. `deniedMcpServers` si unisce comunque da tutte le fonti. Gli utenti possono comunque aggiungere server MCP, ma si applica solo l'elenco di autorizzazione definito dall'amministratore. Vedi Configurazione MCP gestita	`true`
`model`	Ignora il modello predefinito da utilizzare per Claude Code	`"claude-sonnet-4-6"`
`availableModels`	Limita quali modelli gli utenti possono selezionare tramite `/model`, `--model`, strumento Config o `ANTHROPIC_MODEL`. Non influisce sull'opzione Predefinito. Vedi Limita la selezione del modello	`["sonnet", "haiku"]`
`modelOverrides`	Mappa gli ID dei modelli Anthropic agli ID dei modelli specifici del provider come gli ARN del profilo di inferenza Bedrock. Ogni voce del selettore di modelli utilizza il suo valore mappato quando chiama l'API del provider. Vedi Ignora gli ID dei modelli per versione	`{"claude-opus-4-6": "arn:aws:bedrock:..."}`
`otelHeadersHelper`	Script per generare intestazioni OpenTelemetry dinamiche. Viene eseguito all'avvio e periodicamente (vedi Intestazioni dinamiche)	`/bin/generate_otel_headers.sh`

Chiave	Descrizione	Esempio
`statusLine`	Configura una riga di stato personalizzata per visualizzare il contesto. Vedi documentazione `statusLine`	`{"type": "command", "command": "~/.claude/statusline.sh"}`
`fileSuggestion`	Configura uno script personalizzato per l'autocompletamento dei file `@`. Vedi Impostazioni di suggerimento file	`{"type": "command", "command": "~/.claude/file-suggestion.sh"}`
`respectGitignore`	Controlla se il selettore di file `@` rispetta i modelli `.gitignore`. Quando `true` (predefinito), i file che corrispondono ai modelli `.gitignore` vengono esclusi dai suggerimenti	`false`
`outputStyle`	Configura uno stile di output per regolare il prompt di sistema. Vedi documentazione degli stili di output	`"Explanatory"`
`forceLoginMethod`	Usa `claudeai` per limitare l'accesso agli account Claude.ai, `console` per limitare l'accesso agli account Claude Console (fatturazione dell'utilizzo dell'API)	`claudeai`
`forceLoginOrgUUID`	Specifica l'UUID di un'organizzazione per selezionarla automaticamente durante l'accesso, ignorando il passaggio di selezione dell'organizzazione. Richiede che `forceLoginMethod` sia impostato	`"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"`
`enableAllProjectMcpServers`	Approva automaticamente tutti i server MCP definiti nei file `.mcp.json` del progetto	`true`
`enabledMcpjsonServers`	Elenco di server MCP specifici dai file `.mcp.json` da approvare	`["memory", "github"]`
`disabledMcpjsonServers`	Elenco di server MCP specifici dai file `.mcp.json` da rifiutare	`["filesystem"]`

Chiave	Descrizione	Esempio
`allowedMcpServers`	Quando impostato in managed-settings.json, elenco di autorizzazione dei server MCP che gli utenti possono configurare. Non definito = nessuna restrizione, array vuoto = blocco. Si applica a tutti gli ambiti. L'elenco di negazione ha la precedenza. Vedi Configurazione MCP gestita	`[{ "serverName": "github" }]`
`deniedMcpServers`	Quando impostato in managed-settings.json, elenco di negazione dei server MCP che sono esplicitamente bloccati. Si applica a tutti gli ambiti inclusi i server gestiti. L'elenco di negazione ha la precedenza sull'elenco di autorizzazione. Vedi Configurazione MCP gestita	`[{ "serverName": "filesystem" }]`
`strictKnownMarketplaces`	Quando impostato in managed-settings.json, elenco di autorizzazione dei marketplace dei plugin che gli utenti possono aggiungere. Non definito = nessuna restrizione, array vuoto = blocco. Si applica solo alle aggiunte del marketplace. Vedi Restrizioni del marketplace gestito	`[{ "source": "github", "repo": "acme-corp/plugins" }]`
`blockedMarketplaces`	(Solo impostazioni gestite) Elenco di negazione delle fonti del marketplace. Le fonti bloccate vengono controllate prima del download, quindi non toccano mai il filesystem. Vedi Restrizioni del marketplace gestito	`[{ "source": "github", "repo": "untrusted/plugins" }]`
`pluginTrustMessage`	(Solo impostazioni gestite) Messaggio personalizzato aggiunto all'avviso di fiducia del plugin mostrato prima dell'installazione. Usa questo per aggiungere contesto specifico dell'organizzazione, ad esempio per confermare che i plugin dal tuo marketplace interno sono controllati.	`"All plugins from our marketplace are approved by IT"`

Chiave	Descrizione	Esempio
`awsAuthRefresh`	Script personalizzato che modifica la directory `.aws` (vedi configurazione avanzata delle credenziali)	`aws sso login --profile myprofile`
`awsCredentialExport`	Script personalizzato che restituisce JSON con le credenziali AWS (vedi configurazione avanzata delle credenziali)	`/bin/generate_aws_grant.sh`
`alwaysThinkingEnabled`	Abilita il pensiero esteso per impostazione predefinita per tutte le sessioni. Tipicamente configurato tramite il comando `/config` piuttosto che modificato direttamente	`true`
`plansDirectory`	Personalizza dove vengono archiviati i file di piano. Il percorso è relativo alla radice del progetto. Predefinito: `~/.claude/plans`	`"./plans"`
`showTurnDuration`	Mostra i messaggi di durata del turno dopo le risposte (ad es. “Cooked for 1m 6s”). Impostare su `false` per nascondere questi messaggi	`true`
`spinnerVerbs`	Personalizza i verbi di azione mostrati nello spinner e nei messaggi di durata del turno. Impostare `mode` su `"replace"` per utilizzare solo i tuoi verbi, o `"append"` per aggiungerli ai predefiniti	`{"mode": "append", "verbs": ["Pondering", "Crafting"]}`
`language`	Configura la lingua di risposta preferita di Claude (ad es. `"japanese"`, `"spanish"`, `"french"`). Claude risponderà in questa lingua per impostazione predefinita	`"japanese"`

Chiave	Descrizione	Esempio
`autoUpdatesChannel`	Canale di rilascio da seguire per gli aggiornamenti. Usa `"stable"` per una versione che è tipicamente circa una settimana vecchia e salta le versioni con regressioni importanti, o `"latest"` (predefinito) per il rilascio più recente	`"stable"`
`spinnerTipsEnabled`	Mostra suggerimenti nello spinner mentre Claude sta lavorando. Impostare su `false` per disabilitare i suggerimenti (predefinito: `true`)	`false`
`spinnerTipsOverride`	Ignora i suggerimenti dello spinner con stringhe personalizzate. `tips`: array di stringhe di suggerimento. `excludeDefault`: se `true`, mostra solo suggerimenti personalizzati; se `false` o assente, i suggerimenti personalizzati vengono uniti ai suggerimenti incorporati	`{ "excludeDefault": true, "tips": ["Use our internal tool X"] }`
`terminalProgressBarEnabled`	Abilita la barra di avanzamento del terminale che mostra l'avanzamento nei terminali supportati come Windows Terminal e iTerm2 (predefinito: `true`)	`false`
`prefersReducedMotion`	Riduci o disabilita le animazioni dell'interfaccia utente (spinner, shimmer, effetti flash) per l'accessibilità	`true`
`fastModePerSessionOptIn`	Quando `true`, la modalità veloce non persiste tra le sessioni. Ogni sessione inizia con la modalità veloce disattivata, richiedendo agli utenti di abilitarla con `/fast`. La preferenza della modalità veloce dell'utente viene comunque salvata. Vedi Richiedi opt-in per sessione	`true`

Chiave	Descrizione	Esempio
`teammateMode`	Come i compagni di squadra del team di agenti vengono visualizzati: `auto` (sceglie riquadri divisi in tmux o iTerm2, in-process altrimenti), `in-process` o `tmux`. Vedi configura i team di agenti	`"in-process"`

Impostazioni di permesso

Chiavi	Descrizione	Esempio
`allow`	Array di regole di permesso per consentire l'uso dello strumento. Vedi Sintassi della regola di permesso sottostante per i dettagli della corrispondenza dei modelli	`[ "Bash(git diff *)" ]`
`ask`	Array di regole di permesso per chiedere conferma all'uso dello strumento. Vedi Sintassi della regola di permesso sottostante	`[ "Bash(git push *)" ]`
`deny`	Array di regole di permesso per negare l'uso dello strumento. Usa questo per escludere file sensibili dall'accesso di Claude Code. Vedi Sintassi della regola di permesso e Limitazioni dei permessi Bash	`[ "WebFetch", "Bash(curl *)", "Read(./.env)", "Read(./secrets/**)" ]`
`additionalDirectories`	Directory di lavoro aggiuntive a cui Claude ha accesso	`[ "../docs/" ]`
`defaultMode`	Modalità di permesso predefinita all'apertura di Claude Code	`"acceptEdits"`
`disableBypassPermissionsMode`	Impostare su `"disable"` per impedire l'attivazione della modalità `bypassPermissions`. Questo disabilita il flag della riga di comando `--dangerously-skip-permissions`. Vedi impostazioni gestite	`"disable"`

Sintassi della regola di permesso

Le regole di permesso seguono il formato `Tool` o `Tool(specifier)`. Le regole vengono valutate in ordine: prima le regole di negazione, poi di richiesta, poi di autorizzazione. La prima regola corrispondente vince.

Esempi rapidi:

Regola	Effetto
`Bash`	Corrisponde a tutti i comandi Bash
`Bash(npm run *)`	Corrisponde ai comandi che iniziano con `npm run`
`Read(./.env)`	Corrisponde alla lettura del file `.env`
`WebFetch(domain:example.com)`	Corrisponde alle richieste di recupero a example.com

Per il riferimento completo della sintassi delle regole, incluso il comportamento dei caratteri jolly, i modelli specifici dello strumento per Read, Edit, WebFetch, MCP e Agent, e le limitazioni di sicurezza dei modelli Bash, vedi Sintassi della regola di permesso.

Impostazioni sandbox

Configura il comportamento avanzato del sandboxing. Il sandboxing isola i comandi bash dal tuo filesystem e dalla rete. Vedi Sandboxing per i dettagli.

Chiavi	Descrizione	Esempio
`enabled`	Abilita il sandboxing bash (macOS, Linux e WSL2). Predefinito: false	`true`
`autoAllowBashIfSandboxed`	Approva automaticamente i comandi bash quando sandboxed. Predefinito: true	`true`
`excludedCommands`	Comandi che dovrebbero essere eseguiti al di fuori della sandbox	`["git", "docker"]`

Chiavi	Descrizione	Esempio
`allowUnsandboxedCommands`	Consenti ai comandi di essere eseguiti al di fuori della sandbox tramite il parametro `dangerouslyDisableSandbox`. Quando impostato su `false`, la scappatoia `dangerouslyDisableSandbox` è completamente disabilitata e tutti i comandi devono essere sandboxed (o essere in `excludedCommands`). Utile per le politiche aziendali che richiedono un sandboxing rigoroso. Predefinito: true	`false`
`filesystem.allowWrite`	Percorsi aggiuntivi dove i comandi sandboxed possono scrivere. Gli array vengono uniti in tutti gli ambiti di impostazioni: i percorsi utente, progetto e gestiti vengono combinati, non sostituiti. Anche uniti con i percorsi dalle regole di permesso `Edit(...)` allow. Vedi prefissi di percorso sottostante.	`["//tmp/build", "~/.kube"]`
`filesystem.denyWrite`	Percorsi dove i comandi sandboxed non possono scrivere. Gli array vengono uniti in tutti gli ambiti di impostazioni. Anche uniti con i percorsi dalle regole di permesso `Edit(...)` deny.	`["//etc", "//usr/local/bin"]`
`filesystem.denyRead`	Percorsi dove i comandi sandboxed non possono leggere. Gli array vengono uniti in tutti gli ambiti di impostazioni. Anche uniti con i percorsi dalle regole di permesso `Read(...)` deny.	`["~/.aws/credentials"]`
`network.allowUnixSockets`	Percorsi dei socket Unix accessibili nella sandbox (per agenti SSH, ecc.)	`["~/.ssh/agent-socket"]`

Chiavi	Descrizione	Esempio
`network.allowAllUnixSockets`	Consenti tutte le connessioni ai socket Unix nella sandbox. Predefinito: false	`true`
`network.allowLocalBinding`	Consenti il binding alle porte localhost (solo macOS). Predefinito: false	`true`
`network.allowedDomains`	Array di domini da consentire per il traffico di rete in uscita. Supporta i caratteri jolly (ad es. `*.example.com`).	`["github.com", "*.npmjs.org"]`
`network.allowManagedDomainsOnly`	(Solo impostazioni gestite) Solo `allowedDomains` e le regole di autorizzazione `WebFetch(domain:...)` dalle impostazioni gestite vengono rispettate. I domini dalle impostazioni utente, progetto e locale vengono ignorati. I domini non consentiti vengono bloccati automaticamente senza richiedere all'utente. I domini negati vengono comunque rispettati da tutte le fonti. Predefinito: false	`true`
`network.httpProxyPort`	Porta del proxy HTTP utilizzata se desideri portare il tuo proxy. Se non specificato, Claude eseguirà il suo proxy.	`8080`
`network.socksProxyPort`	Porta del proxy SOCKS5 utilizzata se desideri portare il tuo proxy. Se non specificato, Claude eseguirà il suo proxy.	`8081`
`enableWeakerNestedSandbox`	Abilita una sandbox più debole per ambienti Docker senza privilegi (solo Linux e WSL2). **Riduce la sicurezza.** Predefinito: false	`true`

Chiavi	Descrizione	Esempio
`enableWeakerNetworkIsolation`	(Solo macOS) Consenti l'accesso al servizio di fiducia TLS del sistema (`com.apple.trustd.agent`) nella sandbox. Richiesto affinché gli strumenti basati su Go come `gh` , `gcloud` e `terraform` verifichino i certificati TLS quando si utilizza `httpProxyPort` con un proxy MITM e una CA personalizzata. **Riduce la sicurezza** aprendo un potenziale percorso di esfiltrazione dei dati. Predefinito: false	`true`

Prefissi di percorso sandbox

I percorsi in `filesystem.allowWrite` , `filesystem.denyWrite` e `filesystem.denyRead` supportano questi prefissi:

Prefisso	Significato	Esempio
`//`	Percorso assoluto dalla radice del filesystem	`//tmp/build` diventa `/tmp/build`
`~/`	Relativo alla directory home	`~/.kube` diventa `$HOME/.kube`
`/`	Relativo alla directory del file di impostazioni	`/build` diventa `$SETTINGS_DIR/build`
`./` o nessun prefisso	Percorso relativo (risolto dal runtime della sandbox)	`./output`

Esempio di configurazione:

```
{
  "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
    }
  }
}
```

Le **restrizioni del filesystem e della rete** possono essere configurate in due modi che vengono uniti insieme:

- **Impostazioni** `sandbox.filesystem` (mostrate sopra): Controllano i percorsi al confine della sandbox a livello di sistema operativo. Queste restrizioni si applicano a tutti i comandi dei sottoprocessi (ad es. `kubectl`, `terraform`, `npm`), non solo agli strumenti di file di Claude.
- **Regole di permesso**: Usa le regole di autorizzazione/negazione `Edit` per controllare l'accesso allo strumento di file di Claude, le regole di negazione `Read` per bloccare le letture e le regole di autorizzazione/negazione `WebFetch` per controllare i domini di rete. I percorsi da queste regole vengono anche uniti nella configurazione della sandbox.

Impostazioni di attribuzione

Claude Code aggiunge attribuzione ai commit git e alle pull request. Questi vengono configurati separatamente:

- I commit utilizzano i trailer git (come `Co-Authored-By`) per impostazione predefinita, che possono essere personalizzati o disabilitati
- Le descrizioni delle pull request sono testo semplice

Chiavi	Descrizione
`commit`	Attribuzione per i commit git, inclusi eventuali trailer. La stringa vuota nasconde l'attribuzione del commit
`pr`	Attribuzione per le descrizioni delle pull request. La stringa vuota nasconde l'attribuzione della pull request

Attribuzione predefinita del commit:

```
🤖 Generated with [Claude Code](https://claude.com/claude-code)

   Co-Authored-By: Claude Sonnet 4.6 <noreply@anthropic.com>
```

Attribuzione predefinita della pull request:

```
🤖 Generated with [Claude Code](https://claude.com/claude-code)
```

Esempio:

```
{
  "attribution": {
    "commit": "Generated with AI\n\nCo-Authored-By: AI <ai@example.com>",
    "pr": ""
  }
}
```

> **Note:**
>
> L'impostazione `attribution` ha la precedenza sull'impostazione deprecata `includeCoAuthoredBy`. Per nascondere tutta l'attribuzione, impostare `commit` e `pr` su stringhe vuote.

Impostazioni di suggerimento file

Configura un comando personalizzato per l'autocompletamento del percorso del file `@`. Il suggerimento di file incorporato utilizza l'attraversamento veloce del filesystem, ma i monorepo di grandi dimensioni potrebbero beneficiare dell'indicizzazione specifica del progetto come un indice di file pre-costruito o strumenti personalizzati.

```
{
  "fileSuggestion": {
    "type": "command",
    "command": "~/.claude/file-suggestion.sh"
  }
}
```

Il comando viene eseguito con le stesse variabili di ambiente degli hooks, incluso `CLAUDE_PROJECT_DIR`. Riceve JSON tramite stdin con un campo `query`:

```
{"query": "src/comp"}
```

Restituisci i percorsi dei file separati da newline a stdout (attualmente limitati a 15):

```
src/components/Button.tsx
src/components/Modal.tsx
src/components/Form.tsx
```

Esempio:

```
#!/bin/bash
query=$(cat | jq -r '.query')
your-repo-file-index --query "$query" | head -20
```

Configurazione hook

Queste impostazioni controllano quali hook possono essere eseguiti e a cosa possono accedere gli hook HTTP. L'impostazione `allowManagedHooksOnly` può essere configurata solo nelle impostazioni gestite. Gli elenchi di autorizzazione degli URL e delle variabili di ambiente possono essere impostati a qualsiasi livello di impostazioni e si uniscono tra le fonti.

Comportamento quando `allowManagedHooksOnly` è `true`:

- Gli hook gestiti e gli hook SDK vengono caricati
- Gli hook utente, progetto e plugin vengono bloccati

Limita gli URL degli hook HTTP:

Limita quali URL gli hook HTTP possono indirizzare. Supporta `*` come carattere jolly per la corrispondenza. Quando l'array è definito, gli hook HTTP che indirizzano URL non corrispondenti vengono silenziosamente bloccati.

```
{
  "allowedHttpHookUrls": ["https://hooks.example.com/*", "http://localhost:*"]
}
```

Limita le variabili di ambiente degli hook HTTP:

Limita quali nomi di variabili di ambiente gli hook HTTP possono interpolare nei valori delle intestazioni. L' `allowedEnvVars` effettivo di ogni hook è l'intersezione del suo elenco e di questa impostazione.

```
{
  "httpHookAllowedEnvVars": ["MY_TOKEN", "HOOK_SECRET"]
}
```

Precedenza delle impostazioni

Le impostazioni si applicano in ordine di precedenza. Dal più alto al più basso:

1. **Impostazioni gestite** (gestite dal server, politiche MDM/a livello di sistema operativo o impostazioni gestite)
   - Politiche distribuite da IT tramite consegna dal server, profili di configurazione MDM, politiche di registro o file di impostazioni gestite
   - Non possono essere ignorate da nessun altro livello, inclusi gli argomenti della riga di comando
   - All'interno del livello gestito, la precedenza è: gestite dal server > politiche MDM/a livello di sistema operativo > `managed-settings.json` > registro HKCU (solo Windows). Viene utilizzata una sola fonte gestita; le fonti non si uniscono.
2. **Argomenti della riga di comando**
   - Override temporanei per una sessione specifica
3. **Impostazioni di progetto locale** (`.claude/settings.local.json`)
   - Impostazioni personali specifiche del progetto
4. **Impostazioni di progetto condivise** (`.claude/settings.json`)
   - Impostazioni di progetto condivise dal team nel controllo del codice sorgente

5. **Impostazioni utente** (`~/.claude/settings.json`)
    - Impostazioni globali personali

Questa gerarchia garantisce che le politiche organizzative siano sempre applicate mentre consente comunque ai team e agli individui di personalizzare la loro esperienza.

Ad esempio, se le tue impostazioni utente consentono `Bash(npm run *)` ma le impostazioni condivise di un progetto lo negano, l'impostazione di progetto ha la precedenza e il comando viene bloccato.

> **Note:**
>
> **Gli array di impostazioni si uniscono tra gli ambiti.** Quando la stessa impostazione con valore array (come `sandbox.filesystem.allowWrite` o `permissions.allow`) appare in più ambiti, gli array vengono **concatenati e deduplicati**, non sostituiti. Ciò significa che gli ambiti con priorità inferiore possono aggiungere voci senza ignorare quelle impostate da ambiti con priorità più alta, e viceversa. Ad esempio, se le impostazioni gestite impostano `allowWrite` su `["//opt/company-tools"]` e un utente aggiunge `["~/.kube"]` , entrambi i percorsi sono inclusi nella configurazione finale.

Verifica le impostazioni attive

Esegui `/status` all'interno di Claude Code per vedere quali fonti di impostazioni sono attive e da dove provengono. L'output mostra ogni livello di configurazione (gestito, utente, progetto) insieme alla sua origine, come `Enterprise managed settings (remote)` , `Enterprise managed settings (plist)` , `Enterprise managed settings (HKLM)` o `Enterprise managed settings (file)` . Se un file di impostazioni contiene errori, `/status` segnala il problema in modo che tu possa risolverlo.

Punti chiave sul sistema di configurazione

- **File di memoria (`CLAUDE.md`)**: Contengono istruzioni e contesto che Claude carica all'avvio
- **File di impostazioni (JSON)**: Configurano permessi, variabili di ambiente e comportamento dello strumento
- **Skills**: Prompt personalizzati che possono essere invocati con `/skill-name` o caricati automaticamente da Claude
- **Server MCP**: Estendono Claude Code con strumenti e integrazioni aggiuntivi
- **Precedenza**: Le configurazioni di livello superiore (Managed) ignorano quelle di livello inferiore (User/Project)
- **Ereditarietà**: Le impostazioni vengono unite, con impostazioni più specifiche che si aggiungono o ignorano quelle più ampie

Prompt di sistema

Il prompt di sistema interno di Claude Code non è pubblicato. Per aggiungere istruzioni personalizzate, usa i file `CLAUDE.md` o il flag `--append-system-prompt`.

Esclusione di file sensibili

Per impedire a Claude Code di accedere a file contenenti informazioni sensibili come chiavi API, segreti e file di ambiente, usa l'impostazione `permissions.deny` nel tuo file `.claude/settings.json`:

```
{
  "permissions": {
    "deny": [
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(./secrets/**)",
      "Read(./config/credentials.json)",
      "Read(./build)"
    ]
  }
}
```

Questo sostituisce la configurazione deprecata `ignorePatterns`. I file che corrispondono a questi modelli vengono esclusi dalla scoperta dei file e dai risultati della ricerca, e le operazioni di lettura su questi file vengono negate.

Configurazione subagent

Claude Code supporta subagent AI personalizzati che possono essere configurati sia a livello utente che di progetto. Questi subagent vengono archiviati come file Markdown con frontmatter YAML:

- **Subagent utente**: `~/.claude/agents/` - Disponibili in tutti i tuoi progetti
- **Subagent di progetto**: `.claude/agents/` - Specifici del tuo progetto e possono essere condivisi con il tuo team

I file subagent definiscono assistenti AI specializzati con prompt personalizzati e permessi degli strumenti. Scopri di più sulla creazione e l'utilizzo dei subagent nella documentazione dei subagent.

Configurazione dei plugin

Claude Code supporta un sistema di plugin che ti consente di estendere la funzionalità con skill, agenti, hook e server MCP. I plugin vengono distribuiti tramite marketplace e possono essere configurati sia a livello utente che di repository.

Impostazioni dei plugin

Impostazioni relative ai plugin in `settings.json`:

```
{
  "enabledPlugins": {
    "formatter@acme-tools": true,
    "deployer@acme-tools": true,
    "analyzer@security-plugins": false
  },
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": "github",
      "repo": "acme-corp/claude-plugins"
    }
  }
}
```

`enabledPlugins`

Controlla quali plugin sono abilitati. Formato: `"plugin-name@marketplace-name": true/false`

Ambiti:

- **Impostazioni utente** (`~/.claude/settings.json`): Preferenze personali dei plugin
- **Impostazioni di progetto** (`.claude/settings.json`): Plugin specifici del progetto condivisi con il team
- **Impostazioni locali** (`.claude/settings.local.json`): Override per macchina (non committati)

Esempio:

```
{
  "enabledPlugins": {
    "code-formatter@team-tools": true,
    "deployment-tools@team-tools": true,
    "experimental-features@personal": false
  }
}
```

`extraKnownMarketplaces`

Definisce marketplace aggiuntivi che dovrebbero essere resi disponibili per il repository. Tipicamente utilizzato nelle impostazioni a livello di repository per garantire che i membri del team abbiano accesso alle fonti di plugin richieste.

Quando un repository include `extraKnownMarketplaces`:

1. I membri del team vengono invitati a installare il marketplace quando fidano della cartella
2. I membri del team vengono quindi invitati a installare i plugin da quel marketplace
3. Gli utenti possono saltare i marketplace o i plugin indesiderati (archiviati nelle impostazioni utente)
4. L'installazione rispetta i confini di fiducia e richiede il consenso esplicito

Esempio:

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/claude-plugins"
      }
    },
    "security-plugins": {
      "source": {
        "source": "git",
        "url": "https://git.example.com/security/plugins.git"
      }
    }
  }
}
```

Tipi di fonte del marketplace:

- `github`: Repository GitHub (utilizza `repo`)
- `git`: Qualsiasi URL git (utilizza `url`)
- `directory`: Percorso del filesystem locale (utilizza `path`, solo per lo sviluppo)
- `hostPattern`: Modello regex per abbinare gli host del marketplace (utilizza `hostPattern`)

`strictKnownMarketplaces`

Solo impostazioni gestite: Controlla quali marketplace dei plugin gli utenti possono aggiungere. Questa impostazione può essere configurata solo nelle impostazioni gestite e fornisce agli amministratori un controllo rigoroso sulle fonti del marketplace.

Posizioni dei file di impostazioni gestite:

- **macOS**: `/Library/Application Support/ClaudeCode/managed-settings.json`
- **Linux e WSL**: `/etc/claude-code/managed-settings.json`
- **Windows**: `C:\Program Files\ClaudeCode\managed-settings.json`

Caratteristiche chiave:

- Disponibile solo nelle impostazioni gestite (`managed-settings.json`)
- Non può essere ignorato dalle impostazioni utente o progetto (precedenza più alta)

- Applicato PRIMA delle operazioni di rete/filesystem (le fonti bloccate non vengono mai eseguite)
- Utilizza la corrispondenza esatta per le specifiche di origine (incluso `ref`, `path` per le fonti git), tranne `hostPattern`, che utilizza la corrispondenza regex

Comportamento dell'elenco di autorizzazione:

- `undefined` (predefinito): Nessuna restrizione - gli utenti possono aggiungere qualsiasi marketplace
- Array vuoto `[]`: Blocco completo - gli utenti non possono aggiungere nuovi marketplace
- Elenco di fonti: Gli utenti possono aggiungere solo i marketplace che corrispondono esattamente

Tutti i tipi di fonte supportati:

L'elenco di autorizzazione supporta sette tipi di fonte del marketplace. La maggior parte delle fonti utilizza la corrispondenza esatta, mentre `hostPattern` utilizza la corrispondenza regex rispetto all'host del marketplace.

1. **Repository GitHub**:

```
{ "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" }
```

Campi: `repo` (obbligatorio), `ref` (facoltativo: ramo/tag/SHA), `path` (facoltativo: sottodirectory)

1. **Repository Git**:

```
{ "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" }
```

Campi: `url` (obbligatorio), `ref` (facoltativo: ramo/tag/SHA), `path` (facoltativo: sottodirectory)

1. **Marketplace basati su URL**:

```
{ "source": "url", "url": "https://plugins.example.com/marketplace.json" }
{ "source": "url", "url": "https://cdn.example.com/marketplace.json", "headers":
{ "Authorization": "Bearer ${TOKEN}" } }
```

Campi: `url` (obbligatorio), `headers` (facoltativo: intestazioni HTTP per l'accesso autenticato)

> **Note:**
>
> I marketplace basati su URL scaricano solo il file `marketplace.json` . Non scaricano i file dei plugin dal server. I plugin nei marketplace basati su URL devono utilizzare fonti esterne (URL GitHub, npm o git) piuttosto che percorsi relativi. Per i plugin con percorsi relativi, utilizza un marketplace basato su Git. Vedi Troubleshooting per i dettagli.

1. **Pacchetti NPM**:

```
{ "source": "npm", "package": "@acme-corp/claude-plugins" }
{ "source": "npm", "package": "@acme-corp/approved-marketplace" }
```

Campi: `package` (obbligatorio, supporta pacchetti con scope)

1. **Percorsi di file**:

```
{ "source": "file", "path": "/usr/local/share/claude/acme-marketplace.json" }
{ "source": "file", "path": "/opt/acme-corp/plugins/marketplace.json" }
```

Campi: `path` (obbligatorio: percorso assoluto al file marketplace.json)

1. **Percorsi di directory**:

```
{ "source": "directory", "path": "/usr/local/share/claude/acme-plugins" }
{ "source": "directory", "path": "/opt/acme-corp/approved-marketplaces" }
```

Campi: `path` (obbligatorio: percorso assoluto alla directory contenente `.claude-plugin/marketplace.json`)

1. **Corrispondenza del modello di host**:

```
{ "source": "hostPattern", "hostPattern": "^github\\.example\\.com$" }
{ "source": "hostPattern", "hostPattern": "^gitlab\\.internal\\.example\\.com$" }
```

Campi: `hostPattern` (obbligatorio: modello regex per abbinare l'host del marketplace)

Utilizza la corrispondenza del modello di host quando desideri consentire tutti i marketplace da un host specifico senza enumerare ogni repository individualmente. Questo è utile per le organizzazioni con server GitHub Enterprise o GitLab interni dove gli sviluppatori creano i propri marketplace.

Estrazione dell'host per tipo di fonte:

- `github`: corrisponde sempre a `github.com`
- `git`: estrae il nome host dall'URL (supporta sia i formati HTTPS che SSH)
- `url`: estrae il nome host dall'URL
- `npm`, `file`, `directory`: non supportati per la corrispondenza del modello di host

Esempi di configurazione:

Esempio: consenti solo marketplace specifici:

```
{
  "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"
    }
  ]
}
```

Esempio - Disabilita tutte le aggiunte del marketplace:

```
{
  "strictKnownMarketplaces": []
}
```

Esempio: consenti tutti i marketplace da un server git interno:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

Requisiti di corrispondenza esatta:

Le fonti del marketplace devono corrispondere **esattamente** affinché l'aggiunta di un utente sia consentita. Per le fonti basate su git (`github` e `git`), questo include tutti i campi facoltativi:

- Il `repo` o `url` deve corrispondere esattamente
- Il campo `ref` deve corrispondere esattamente (o entrambi non essere definiti)
- Il campo `path` deve corrispondere esattamente (o entrambi non essere definiti)

Esempi di fonti che **NON corrispondono**:

```
// Queste sono DIVERSE fonti:
{ "source": "github", "repo": "acme-corp/plugins" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main" }

// Anche queste sono DIVERSE:
{ "source": "github", "repo": "acme-corp/plugins", "path": "marketplace" }
{ "source": "github", "repo": "acme-corp/plugins" }
```

Confronto con `extraKnownMarketplaces`:

Aspetto	`strictKnownMarketplaces`	`extraKnownMarketplaces`
Scopo	Applicazione della politica organizzativa	Comodità del team
File di impostazioni	Solo `managed-settings.json`	Qualsiasi file di impostazioni

Aspetto	`strictKnownMarketplaces`	`extraKnownMarketplaces`
Comportamento	Blocca le aggiunte non autorizzate	Installa automaticamente i marketplace mancanti
Quando applicato	Prima delle operazioni di rete/filesystem	Dopo il prompt di fiducia dell'utente
Può essere ignorato	No (precedenza più alta)	Sì (dalle impostazioni di precedenza più alta)
Formato della fonte	Oggetto di fonte diretto	Marketplace denominato con fonte nidificata
Caso d'uso	Conformità, restrizioni di sicurezza	Onboarding, standardizzazione

Differenza di formato:

`strictKnownMarketplaces` utilizza oggetti di fonte diretti:

```
{
  "strictKnownMarketplaces": [
    { "source": "github", "repo": "acme-corp/plugins" }
  ]
}
```

`extraKnownMarketplaces` richiede marketplace denominati:

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": { "source": "github", "repo": "acme-corp/plugins" }
    }
  }
}
```

Note importanti:

- Le restrizioni vengono controllate PRIMA di qualsiasi richiesta di rete o operazione del filesystem

- Quando bloccato, gli utenti vedono messaggi di errore chiari che indicano che la fonte è bloccata dalla politica gestita
- La restrizione si applica solo all'aggiunta di NUOVI marketplace; i marketplace precedentemente installati rimangono accessibili
- Le impostazioni gestite hanno la precedenza più alta e non possono essere ignorate

Vedi Restrizioni del marketplace gestito per la documentazione rivolta agli utenti.

Gestione dei plugin

Usa il comando `/plugin` per gestire i plugin in modo interattivo:

- Sfoglia i plugin disponibili dai marketplace
- Installa/disinstalla plugin
- Abilita/disabilita plugin
- Visualizza i dettagli del plugin (comandi, agenti, hook forniti)
- Aggiungi/rimuovi marketplace

Scopri di più sul sistema di plugin nella documentazione dei plugin.

Variabili di ambiente

Claude Code supporta le seguenti variabili di ambiente per controllare il suo comportamento:

> **Note:**
>
> Tutte le variabili di ambiente possono anche essere configurate in `settings.json`. Questo è utile come modo per impostare automaticamente le variabili di ambiente per ogni sessione, o per distribuire un insieme di variabili di ambiente per l'intero team o organizzazione.

Variabile	Scopo	
`ANTHROPIC_API_KEY`	Chiave API inviata come intestazione `X-Api-Key`, tipicamente per l'SDK Claude (per l'utilizzo interattivo, esegui `/login`)	
`ANTHROPIC_AUTH_TOKEN`	Valore personalizzato per l'intestazione `Authorization` (il valore che imposti qui sarà preceduto da `Bearer`)	

Variabile	Scopo	
`ANTHROPIC_CUSTOM_HEADERS`	Intestazioni personalizzate da aggiungere alle richieste (formato `Name: Value`, separato da newline per più intestazioni)	
`ANTHROPIC_DEFAULT_HAIKU_MODEL`	Vedi Configurazione del modello	
`ANTHROPIC_DEFAULT_OPUS_MODEL`	Vedi Configurazione del modello	
`ANTHROPIC_DEFAULT_SONNET_MODEL`	Vedi Configurazione del modello	
`ANTHROPIC_FOUNDRY_API_KEY`	Chiave API per l'autenticazione Microsoft Foundry (vedi Microsoft Foundry)	
`ANTHROPIC_FOUNDRY_BASE_URL`	URL di base completo per la risorsa Foundry (ad esempio, `https://my-resource.services.ai.azure.com/anthropic`). Alternativa a `ANTHROPIC_FOUNDRY_RESOURCE` (vedi Microsoft Foundry)	
`ANTHROPIC_FOUNDRY_RESOURCE`	Nome della risorsa Foundry (ad esempio, `my-resource`). Obbligatorio se `ANTHROPIC_FOUNDRY_BASE_URL` non è impostato (vedi Microsoft Foundry)	
`ANTHROPIC_MODEL`	Nome dell'impostazione del modello da utilizzare (vedi Configurazione del modello)	
`ANTHROPIC_SMALL_FAST_MODEL`	[DEPRECATO] Nome del modello di classe Haiku per attività in background	
`ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION`	Ignora la regione AWS per il modello di classe Haiku quando si utilizza Bedrock	

Variabile	Scopo	
`AWS_BEARER_TOKEN_BEDROCK`	Chiave API Bedrock per l'autenticazione (vedi Chiavi API Bedrock)	
`BASH_DEFAULT_TIMEOUT_MS`	Timeout predefinito per i comandi bash a lunga esecuzione	
`BASH_MAX_OUTPUT_LENGTH`	Numero massimo di caratteri negli output bash prima che vengano troncati al centro	
`BASH_MAX_TIMEOUT_MS`	Timeout massimo che il modello può impostare per i comandi bash a lunga esecuzione	
`CLAUDE_AUTOCOMPACT_PCT_OVERRIDE`	Imposta la percentuale della capacità del contesto (1-100) a cui viene attivata la compattazione automatica. Per impostazione predefinita, la compattazione automatica viene attivata a circa il 95% della capacità. Usa valori inferiori come `50` per compattare prima. I valori superiori alla soglia predefinita non hanno effetto. Si applica sia alle conversazioni principali che ai subagent. Questa percentuale si allinea con il campo `context_window.used_percentage` disponibile nella riga di stato	
`CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR`	Ritorna alla directory di lavoro originale dopo ogni comando Bash	
`CLAUDE_CODE_ACCOUNT_UUID`	UUID dell'account per l'utente autenticato. Utilizzato dai chiamanti dell'SDK per fornire informazioni sull'account in modo sincrono, evitando una condizione di gara in cui gli eventi di telemetria iniziali mancano di metadati dell'account. Richiede che `CLAUDE_CODE_USER_EMAIL` e `CLAUDE_CODE_ORGANIZATION_UUID` siano anche impostati	

Variabile	Scopo	
`CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD`	Impostare su `1` per caricare i file CLAUDE.md dalle directory specificate con `--add-dir`. Per impostazione predefinita, le directory aggiuntive non caricano file di memoria	`1`
`CLAUDE_CODE_API_KEY_HELPER_TTL_MS`	Intervallo in millisecondi a cui le credenziali dovrebbero essere aggiornate (quando si utilizza `apiKeyHelper`)	
`CLAUDE_CODE_CLIENT_CERT`	Percorso al file del certificato client per l'autenticazione mTLS	
`CLAUDE_CODE_CLIENT_KEY`	Percorso al file della chiave privata del client per l'autenticazione mTLS	
`CLAUDE_CODE_CLIENT_KEY_PASSPHRASE`	Passphrase per `CLAUDE_CODE_CLIENT_KEY` crittografato (facoltativo)	
`CLAUDE_CODE_DISABLE_1M_CONTEXT`	Impostare su `1` per disabilitare il supporto della finestra di contesto 1M. Quando impostato, le varianti del modello 1M non sono disponibili nel selettore di modelli. Utile per gli ambienti aziendali con requisiti di conformità	
`CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING`	Impostare su `1` per disabilitare il ragionamento adattivo per Opus 4.6 e Sonnet 4.6. Quando disabilitato, questi modelli tornano al budget di pensiero fisso controllato da `MAX_THINKING_TOKENS`	
`CLAUDE_CODE_DISABLE_AUTO_MEMORY`	Impostare su `1` per disabilitare la memoria automatica. Impostare su `0` per forzare la memoria automatica durante il rollout graduale. Quando disabilitato, Claude non crea o carica file di memoria automatica	

Variabile	Scopo	
`CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS`	Impostare su `1` per rimuovere le istruzioni integrate del flusso di lavoro di commit e PR dal prompt di sistema di Claude. Utile quando si utilizzano le proprie skill di flusso di lavoro git. Ha la precedenza sull'impostazione `includeGitInstructions` quando impostato	
`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS`	Impostare su `1` per disabilitare tutta la funzionalità di attività in background, incluso il parametro `run_in_background` su strumenti Bash e subagent, auto-backgrounding e la scorciatoia Ctrl+B	
`CLAUDE_CODE_DISABLE_CRON`	Impostare su `1` per disabilitare le attività pianificate. La skill `/loop` e gli strumenti cron diventano non disponibili e qualsiasi attività già pianificata smette di attivarsi, incluse le attività che sono già in esecuzione a metà sessione	
`CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS`	Impostare su `1` per disabilitare le intestazioni `anthropic-beta` specifiche dell'API Anthropic. Usa questo se riscontri problemi come "Unexpected value(s) for the `anthropic-beta` header" quando utilizzi un gateway LLM con provider di terze parti	
`CLAUDE_CODE_DISABLE_FAST_MODE`	Impostare su `1` per disabilitare la modalità veloce	

Variabile	Scopo	
`CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY`	Impostare su `1` per disabilitare i sondaggi sulla qualità della sessione "How is Claude doing?". Anche disabilitato quando si utilizzano provider di terze parti o quando la telemetria è disabilitata. Vedi Sondaggi sulla qualità della sessione	
`CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC`	Equivalente all'impostazione di `DISABLE_AUTOUPDATER`, `DISABLE_BUG_COMMAND`, `DISABLE_ERROR_REPORTING` e `DISABLE_TELEMETRY`	
`CLAUDE_CODE_DISABLE_TERMINAL_TITLE`	Impostare su `1` per disabilitare gli aggiornamenti automatici del titolo del terminale in base al contesto della conversazione	
`CLAUDE_CODE_EFFORT_LEVEL`	Imposta il livello di sforzo per i modelli supportati. Valori: `low`, `medium`, `high`. Lo sforzo inferiore è più veloce e più economico, lo sforzo superiore fornisce un ragionamento più profondo. Supportato su Opus 4.6 e Sonnet 4.6. Vedi Regola il livello di sforzo	
`CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION`	Impostare su `false` per disabilitare i suggerimenti di prompt (l'interruttore "Prompt suggestions" in `/config`). Queste sono le previsioni grigie che appaiono nel tuo input di prompt dopo che Claude risponde. Vedi Suggerimenti di prompt	
`CLAUDE_CODE_ENABLE_TASKS`	Impostare su `false` per tornare temporaneamente all'elenco TODO precedente invece del sistema di tracciamento delle attività. Predefinito: `true`. Vedi Elenco attività	

Variabile	Scopo	
`CLAUDE_CODE_ENABLE_TELEMETRY`	Impostare su `1` per abilitare la raccolta di dati OpenTelemetry per metriche e registrazione. Obbligatorio prima di configurare gli esportatori OTel. Vedi Monitoraggio	
`CLAUDE_CODE_EXIT_AFTER_STOP_DELAY`	Tempo in millisecondi da attendere dopo che il ciclo di query diventa inattivo prima di uscire automaticamente. Utile per flussi di lavoro automatizzati e script che utilizzano la modalità SDK	
`CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS`	Impostare su `1` per abilitare i team di agenti. I team di agenti sono sperimentali e disabilitati per impostazione predefinita	
`CLAUDE_CODE_FILE_READ_MAX_OUTPUT_TOKENS`	Ignora il limite di token predefinito per le letture di file. Utile quando è necessario leggere file più grandi per intero	
`CLAUDE_CODE_HIDE_ACCOUNT_INFO`	Impostare su `1` per nascondere il tuo indirizzo email e il nome dell'organizzazione dall'interfaccia utente di Claude Code. Utile quando si trasmette in streaming o si registra	
`CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL`	Salta l'installazione automatica delle estensioni IDE	
`CLAUDE_CODE_MAX_OUTPUT_TOKENS`	Imposta il numero massimo di token di output per la maggior parte delle richieste. Predefinito: 32.000. Massimo: 64.000. L'aumento di questo valore riduce la finestra di contesto effettiva disponibile prima che venga attivata la compattazione automatica.	

Variabile	Scopo	
`CLAUDE_CODE_ORGANIZATION_UUID`	UUID dell'organizzazione per l'utente autenticato. Utilizzato dai chiamanti dell'SDK per fornire informazioni sull'account in modo sincrono. Richiede che `CLAUDE_CODE_ACCOUNT_UUID` e `CLAUDE_CODE_USER_EMAIL` siano anche impostati	
`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS`	Intervallo per l'aggiornamento delle intestazioni OpenTelemetry dinamiche in millisecondi (predefinito: 1740000 / 29 minuti). Vedi Intestazioni dinamiche	
`CLAUDE_CODE_PLAN_MODE_REQUIRED`	Impostato automaticamente su `true` sui compagni di squadra del team di agenti che richiedono l'approvazione del piano. Sola lettura: impostato da Claude Code quando genera i compagni di squadra. Vedi richiedi approvazione del piano	
`CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS`	Timeout in millisecondi per le operazioni git durante l'installazione o l'aggiornamento dei plugin (predefinito: 120000). Aumenta questo valore per repository di grandi dimensioni o connessioni di rete lente. Vedi Le operazioni Git scadono	
`CLAUDE_CODE_PROXY_RESOLVES_HOSTS`	Impostare su `true` per consentire al proxy di eseguire la risoluzione DNS invece del chiamante. Opt-in per gli ambienti in cui il proxy dovrebbe gestire la risoluzione del nome host	
`CLAUDE_CODE_SHELL`	Ignora il rilevamento automatico della shell. Utile quando la tua shell di accesso differisce dalla tua shell di lavoro preferita (ad esempio, `bash` vs `zsh`)	

Variabile	Scopo	
`CLAUDE_CODE_SHELL_PREFIX`	Prefisso del comando per avvolgere tutti i comandi bash (ad esempio, per la registrazione o il controllo). Esempio: `/path/to/logger.sh` eseguirà `/path/to/logger.sh <command>`	
`CLAUDE_CODE_SIMPLE`	Impostare su `1` per eseguire con un prompt di sistema minimo e solo gli strumenti Bash, lettura di file e modifica di file. Disabilita gli strumenti MCP, gli allegati, gli hook e i file CLAUDE.md	
`CLAUDE_CODE_SKIP_BEDROCK_AUTH`	Salta l'autenticazione AWS per Bedrock (ad esempio, quando si utilizza un gateway LLM)	
`CLAUDE_CODE_SKIP_FOUNDRY_AUTH`	Salta l'autenticazione Azure per Microsoft Foundry (ad esempio, quando si utilizza un gateway LLM)	
`CLAUDE_CODE_SKIP_VERTEX_AUTH`	Salta l'autenticazione Google per Vertex (ad esempio, quando si utilizza un gateway LLM)	
`CLAUDE_CODE_SUBAGENT_MODEL`	Vedi Configurazione del modello	
`CLAUDE_CODE_TASK_LIST_ID`	Condividi un elenco di attività tra le sessioni. Imposta lo stesso ID in più istanze di Claude Code per coordinare un elenco di attività condiviso. Vedi Elenco attività	
`CLAUDE_CODE_TEAM_NAME`	Nome del team di agenti a cui appartiene questo compagno di squadra. Impostato automaticamente sui membri del team di agenti	

Variabile	Scopo	
`CLAUDE_CODE_TMPDIR`	Ignora la directory temporanea utilizzata per i file temporanei interni. Claude Code aggiunge `/claude/` a questo percorso. Predefinito: `/tmp` su Unix/macOS, `os.tmpdir()` su Windows	
`CLAUDE_CODE_USER_EMAIL`	Indirizzo email per l'utente autenticato. Utilizzato dai chiamanti dell'SDK per fornire informazioni sull'account in modo sincrono. Richiede che `CLAUDE_CODE_ACCOUNT_UUID` e `CLAUDE_CODE_ORGANIZATION_UUID` siano anche impostati	
`CLAUDE_CODE_USE_BEDROCK`	Usa Bedrock	
`CLAUDE_CODE_USE_FOUNDRY`	Usa Microsoft Foundry	
`CLAUDE_CODE_USE_VERTEX`	Usa Vertex	
`CLAUDE_CONFIG_DIR`	Personalizza dove Claude Code archivia i suoi file di configurazione e dati	
`DISABLE_AUTOUPDATER`	Impostare su `1` per disabilitare gli aggiornamenti automatici.	
`DISABLE_BUG_COMMAND`	Impostare su `1` per disabilitare il comando `/bug`	
`DISABLE_COST_WARNINGS`	Impostare su `1` per disabilitare i messaggi di avviso sui costi	
`DISABLE_ERROR_REPORTING`	Impostare su `1` per rinunciare alla segnalazione degli errori di Sentry	

Variabile	Scopo	
`DISABLE_INSTALLATION_CHECKS`	Impostare su `1` per disabilitare gli avvisi di installazione. Usa solo quando gestisci manualmente la posizione di installazione, poiché questo può mascherare i problemi con le installazioni standard	
`DISABLE_NON_ESSENTIAL_MODEL_CALLS`	Impostare su `1` per disabilitare le chiamate del modello per percorsi non critici come il testo di sapore	
`DISABLE_PROMPT_CACHING`	Impostare su `1` per disabilitare il caching dei prompt per tutti i modelli (ha la precedenza sulle impostazioni per modello)	
`DISABLE_PROMPT_CACHING_HAIKU`	Impostare su `1` per disabilitare il caching dei prompt per i modelli Haiku	
`DISABLE_PROMPT_CACHING_OPUS`	Impostare su `1` per disabilitare il caching dei prompt per i modelli Opus	
`DISABLE_PROMPT_CACHING_SONNET`	Impostare su `1` per disabilitare il caching dei prompt per i modelli Sonnet	
`DISABLE_TELEMETRY`	Impostare su `1` per rinunciare alla telemetria Statsig (nota che gli eventi Statsig non includono dati utente come codice, percorsi di file o comandi bash)	
`ENABLE_CLAUDEAI_MCP_SERVERS`	Impostare su `false` per disabilitare i server MCP di claude.ai in Claude Code. Abilitato per impostazione predefinita per gli utenti connessi	
`ENABLE_TOOL_SEARCH`	Controlla la ricerca degli strumenti MCP. Valori: `auto` (predefinito, abilita al 10% del contesto), `auto:N` (soglia personalizzata, ad es. `auto:5` per il 5%), `true` (sempre attivo), `false` (disabilitato)	

Variabile	Scopo	
`FORCE_AUTOUPDATE_PLUGINS`	Impostare su `true` per forzare gli aggiornamenti automatici dei plugin anche quando l'auto-updater principale è disabilitato tramite `DISABLE_AUTOUPDATER`	
`HTTP_PROXY`	Specifica il server proxy HTTP per le connessioni di rete	
`HTTPS_PROXY`	Specifica il server proxy HTTPS per le connessioni di rete	
`IS_DEMO`	Impostare su `true` per abilitare la modalità demo: nasconde l'email e l'organizzazione dall'interfaccia utente, salta l'onboarding e nasconde i comandi interni. Utile per la trasmissione in streaming o la registrazione di sessioni	
`MAX_MCP_OUTPUT_TOKENS`	Numero massimo di token consentiti nelle risposte dello strumento MCP. Claude Code visualizza un avviso quando l'output supera 10.000 token (predefinito: 25000)	
`MAX_THINKING_TOKENS`	Ignora il budget del pensiero esteso. Il pensiero è abilitato al budget massimo (31.999 token) per impostazione predefinita. Usa questo per limitare il budget (ad esempio, `MAX_THINKING_TOKENS=10000`) o disabilitare completamente il pensiero (`MAX_THINKING_TOKENS=0`). Per Opus 4.6, la profondità del pensiero è controllata dal livello di sforzo invece, e questa variabile viene ignorata a meno che non sia impostata su `0` per disabilitare il pensiero.	

Variabile	Scopo	
`MCP_CLIENT_SECRET`	Segreto del client OAuth per i server MCP che richiedono credenziali pre-configurate. Evita il prompt interattivo quando si aggiunge un server con `--client-secret`	
`MCP_OAUTH_CALLBACK_PORT`	Porta fissa per il callback di reindirizzamento OAuth, come alternativa a `--callback-port` quando si aggiunge un server MCP con credenziali pre-configurate	
`MCP_TIMEOUT`	Timeout in millisecondi per l'avvio del server MCP	
`MCP_TOOL_TIMEOUT`	Timeout in millisecondi per l'esecuzione dello strumento MCP	
`NO_PROXY`	Elenco di domini e IP a cui le richieste verranno emesse direttamente, ignorando il proxy	
`SLASH_COMMAND_TOOL_CHAR_BUDGET`	Ignora il budget dei caratteri per i metadati della skill mostrati allo strumento Skill. Il budget si ridimensiona dinamicamente al 2% della finestra di contesto, con un fallback di 16.000 caratteri. Nome legacy mantenuto per la compatibilità all'indietro	
`USE_BUILTIN_RIPGREP`	Impostare su `0` per utilizzare il `rg` installato dal sistema invece del `rg` incluso con Claude Code	
`VERTEX_REGION_CLAUDE_3_5_HAIKU`	Ignora la regione per Claude 3.5 Haiku quando si utilizza Vertex AI	
`VERTEX_REGION_CLAUDE_3_7_SONNET`	Ignora la regione per Claude 3.7 Sonnet quando si utilizza Vertex AI	
`VERTEX_REGION_CLAUDE_4_0_OPUS`	Ignora la regione per Claude 4.0 Opus quando si utilizza Vertex AI	

Variabile	Scopo	
`VERTEX_REGION_CLAUDE_4_0_SONNET`	Ignora la regione per Claude 4.0 Sonnet quando si utilizza Vertex AI	
`VERTEX_REGION_CLAUDE_4_1_OPUS`	Ignora la regione per Claude 4.1 Opus quando si utilizza Vertex AI	

Strumenti disponibili per Claude

Claude Code ha accesso a un insieme di strumenti potenti che lo aiutano a comprendere e modificare la tua base di codice:

Strumento	Descrizione	Permesso richiesto
Agent	Genera un subagent con la sua finestra di contesto per gestire un'attività	No
AskUserQuestion	Pone domande a scelta multipla per raccogliere requisiti o chiarire l'ambiguità	No
Bash	Esegue comandi shell nel tuo ambiente. Vedi Comportamento dello strumento Bash	Sì
CronCreate	Pianifica un prompt ricorrente o una tantum all'interno della sessione corrente (scompare quando Claude esce). Vedi attività pianificate	No
CronDelete	Annulla un'attività pianificata per ID	No
CronList	Elenca tutte le attività pianificate nella sessione	No
Edit	Effettua modifiche mirate a file specifici	Sì
EnterPlanMode	Passa alla modalità piano per progettare un approccio prima di codificare	No
EnterWorktree	Crea un git worktree isolato e vi entra	No

Strumento	Descrizione	Permesso richiesto
ExitPlanMode	Presenta un piano per l'approvazione ed esce dalla modalità piano	Sì
ExitWorktree	Esce da una sessione worktree e ritorna alla directory originale	No
Glob	Trova file in base alla corrispondenza dei modelli	No
Grep	Cerca modelli nei contenuti dei file	No
ListMcpResourcesTool	Elenca le risorse esposte dai server MCP connessi	No
LSP	Intelligenza del codice tramite server di linguaggio. Segnala automaticamente errori di tipo e avvisi dopo le modifiche ai file. Supporta anche operazioni di navigazione: salta alle definizioni, trova riferimenti, ottieni informazioni sul tipo, elenca i simboli, trova implementazioni, traccia gerarchie di chiamate. Richiede un plugin di intelligenza del codice e il suo binario del server di linguaggio	No
NotebookEdit	Modifica le celle del notebook Jupyter	Sì
Read	Legge il contenuto dei file	No
ReadMcpResourceTool	Legge una risorsa MCP specifica per URI	No
Skill	Esegue una skill all'interno della conversazione principale	Sì
TaskCreate	Crea una nuova attività nell'elenco delle attività	No
TaskGet	Recupera i dettagli completi per un'attività specifica	No

Strumento	Descrizione	Permesso richiesto
TaskList	Elenca tutte le attività con il loro stato attuale	No
TaskOutput	Recupera l'output da un'attività in background	No
TaskStop	Uccide un'attività in esecuzione per ID	No
TaskUpdate	Aggiorna lo stato dell'attività, le dipendenze, i dettagli o elimina le attività	No
TodoWrite	Gestisce l'elenco di controllo delle attività della sessione. Disponibile in modalità non interattiva e nell'SDK Agent; le sessioni interattive utilizzano TaskCreate, TaskGet, TaskList e TaskUpdate	No
ToolSearch	Cerca e carica gli strumenti differiti quando la ricerca degli strumenti è abilitata	No
WebFetch	Recupera il contenuto da un URL specificato	Sì
WebSearch	Esegue ricerche web	Sì
Write	Crea o sovrascrivi file	Sì

Le regole di permesso possono essere configurate utilizzando `/allowed-tools` o nelle impostazioni di permesso. Vedi anche Regole di permesso specifiche dello strumento.

Comportamento dello strumento Bash

Lo strumento Bash esegue comandi shell con il seguente comportamento di persistenza:

- **La directory di lavoro persiste**: Quando Claude cambia la directory di lavoro (ad esempio, `cd /path/to/dir`), i successivi comandi Bash verranno eseguiti in quella directory. Puoi usare `CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR=1` per ripristinare la directory del progetto dopo ogni comando.

- **Le variabili di ambiente NON persistono**: Le variabili di ambiente impostate in un comando Bash (ad esempio, `export MY_VAR=value`) **non** sono disponibili nei successivi comandi Bash. Ogni comando Bash viene eseguito in un ambiente shell fresco.

Per rendere disponibili le variabili di ambiente nei comandi Bash, hai **tre opzioni**:

Opzione 1: Attiva l'ambiente prima di avviare Claude Code (approccio più semplice)

Attiva il tuo ambiente virtuale nel tuo terminale prima di avviare Claude Code:

```
conda activate myenv
## o: source /path/to/venv/bin/activate
claude
```

Questo funziona per gli ambienti shell ma le variabili di ambiente impostate all'interno dei comandi Bash di Claude non persisteranno tra i comandi.

Opzione 2: Imposta CLAUDE_ENV_FILE prima di avviare Claude Code (configurazione dell'ambiente persistente)

Esporta il percorso a uno script shell contenente la configurazione del tuo ambiente:

```
export CLAUDE_ENV_FILE=/path/to/env-setup.sh
claude
```

Dove `/path/to/env-setup.sh` contiene:

```
conda activate myenv
## o: source /path/to/venv/bin/activate
## o: export MY_VAR=value
```

Claude Code fornirà questo file prima di ogni comando Bash, rendendo l'ambiente persistente in tutti i comandi.

Opzione 3: Usa un hook SessionStart (configurazione specifica del progetto)

Configura in `.claude/settings.json`:

```
{
  "hooks": {
    "SessionStart": [{
      "matcher": "startup",
      "hooks": [{
        "type": "command",
        "command": "echo 'conda activate myenv' >> \"$CLAUDE_ENV_FILE\""
      }]
    }]
  }
}
```

L'hook scrive in `$CLAUDE_ENV_FILE`, che viene quindi fornito prima di ogni comando Bash. Questo è ideale per le configurazioni di progetto condivise dal team.

Vedi Hook SessionStart per ulteriori dettagli sull'Opzione 3.

Estensione degli strumenti con hook

Puoi eseguire comandi personalizzati prima o dopo l'esecuzione di qualsiasi strumento utilizzando gli hook di Claude Code.

Ad esempio, potresti eseguire automaticamente un formattatore Python dopo che Claude modifica i file Python, o impedire le modifiche ai file di configurazione di produzione bloccando le operazioni Write su determinati percorsi.

Vedi anche

- Permessi: sistema di permessi, sintassi delle regole, modelli specifici dello strumento e politiche gestite
- Autenticazione: configura l'accesso degli utenti a Claude Code
- Troubleshooting: soluzioni per i problemi di configurazione comuni

Configurazione del modello

> *Scopri la configurazione del modello Claude Code, inclusi gli alias dei modelli come* `opusplan`

Modelli disponibili

Per l'impostazione `model` in Claude Code, è possibile configurare:

- Un **alias del modello**
- Un **nome del modello**
  - API Anthropic: un **nome del modello** completo
  - Bedrock: un ARN del profilo di inferenza
  - Foundry: un nome di distribuzione
  - Vertex: un nome di versione

Alias dei modelli

Gli alias dei modelli forniscono un modo conveniente per selezionare le impostazioni del modello senza dover ricordare i numeri di versione esatti:

Alias del modello	Comportamento
`default`	Impostazione del modello consigliata, a seconda del tipo di account
`sonnet`	Utilizza l'ultimo modello Sonnet (attualmente Sonnet 4.6) per le attività di codifica quotidiane
`opus`	Utilizza l'ultimo modello Opus (attualmente Opus 4.6) per attività di ragionamento complesso
`haiku`	Utilizza il modello Haiku veloce ed efficiente per attività semplici
`sonnet[1m]`	Utilizza Sonnet con una finestra di contesto di 1 milione di token per sessioni lunghe
`opusplan`	Modalità speciale che utilizza `opus` durante la modalità piano, quindi passa a `sonnet` per l'esecuzione

Gli alias puntano sempre alla versione più recente. Per fissare una versione specifica, utilizzare il nome completo del modello (ad esempio, `claude-opus-4-6`) o impostare la variabile di ambiente corrispondente come `ANTHROPIC_DEFAULT_OPUS_MODEL`.

Impostazione del modello

È possibile configurare il modello in diversi modi, elencati in ordine di priorità:

1. **Durante la sessione** - Utilizzare `/model <alias|name>` per cambiare modello durante la sessione
2. **All'avvio** - Avviare con `claude --model <alias|name>`
3. **Variabile di ambiente** - Impostare `ANTHROPIC_MODEL=<alias|name>`
4. **Impostazioni** - Configurare in modo permanente nel file delle impostazioni utilizzando il campo `model`.

Esempio di utilizzo:

```
## Avvia con Opus
claude --model opus

## Passa a Sonnet durante la sessione
/model sonnet
```

File delle impostazioni di esempio:

```
{
    "permissions": {
        ...
    },
    "model": "opus"
}
```

Limitare la selezione del modello

Gli amministratori aziendali possono utilizzare `availableModels` nelle impostazioni gestite o di policy per limitare quali modelli gli utenti possono selezionare.

Quando `availableModels` è impostato, gli utenti non possono passare a modelli non presenti nell'elenco tramite `/model`, il flag `--model`, lo strumento Config o la variabile di ambiente `ANTHROPIC_MODEL`.

```
{
  "availableModels": ["sonnet", "haiku"]
}
```

Comportamento del modello predefinito

L'opzione Predefinito nel selettore di modelli non è interessata da `availableModels`. Rimane sempre disponibile e rappresenta il valore predefinito di runtime del sistema basato sul livello di sottoscrizione dell'utente.

Anche con `availableModels: []`, gli utenti possono comunque utilizzare Claude Code con il modello Predefinito per il loro livello.

Controllare il modello su cui gli utenti eseguono

Per controllare completamente l'esperienza del modello, utilizzare `availableModels` insieme all'impostazione `model`:

- **availableModels**: limita a cosa gli utenti possono passare
- **model**: imposta l'override esplicito del modello, che ha la precedenza sul Predefinito

Questo esempio garantisce che tutti gli utenti eseguano Sonnet 4.6 e possono scegliere solo tra Sonnet e Haiku:

```
{
  "model": "sonnet",
  "availableModels": ["sonnet", "haiku"]
}
```

Comportamento di unione

Quando `availableModels` è impostato a più livelli, come le impostazioni utente e le impostazioni del progetto, gli array vengono uniti e deduplicati. Per applicare un elenco di autorizzazione rigoroso, impostare `availableModels` nelle impostazioni gestite o di policy che hanno la priorità più alta.

Comportamento speciale del modello

Impostazione del modello `default`

Il comportamento di `default` dipende dal tipo di account:

- **Max e Team Premium**: per impostazione predefinita Opus 4.6

- **Pro e Team Standard**: per impostazione predefinita Sonnet 4.6
- **Enterprise**: Opus 4.6 è disponibile ma non è il valore predefinito

Claude Code potrebbe eseguire automaticamente il fallback a Sonnet se si raggiunge una soglia di utilizzo con Opus.

Impostazione del modello `opusplan`

L'alias del modello `opusplan` fornisce un approccio ibrido automatizzato:

- **In modalità piano** - Utilizza `opus` per il ragionamento complesso e le decisioni architettoniche
- **In modalità esecuzione** - Passa automaticamente a `sonnet` per la generazione di codice e l'implementazione

Questo ti dà il meglio di entrambi i mondi: il ragionamento superiore di Opus per la pianificazione e l'efficienza di Sonnet per l'esecuzione.

Regolare il livello di sforzo

I livelli di sforzo controllano il ragionamento adattivo, che alloca dinamicamente il pensiero in base alla complessità dell'attività. Lo sforzo inferiore è più veloce ed economico per attività semplici, mentre lo sforzo superiore fornisce un ragionamento più profondo per problemi complessi.

Sono disponibili tre livelli: **low**, **medium** e **high**. Opus 4.6 per impostazione predefinita utilizza uno sforzo medio per gli abbonati Max e Team.

Impostazione dello sforzo:

- **In** `/model` : utilizzare i tasti freccia sinistra/destra per regolare il cursore dello sforzo quando si seleziona un modello
- **Variabile di ambiente**: impostare `CLAUDE_CODE_EFFORT_LEVEL=low|medium|high`
- **Impostazioni**: impostare `effortLevel` nel file delle impostazioni

Lo sforzo è supportato su Opus 4.6 e Sonnet 4.6. Il cursore dello sforzo appare in `/model` quando è selezionato un modello supportato. Il livello di sforzo corrente viene visualizzato anche accanto al logo e al spinner (ad esempio, "with low effort"), in modo da poter confermare quale impostazione è attiva senza aprire `/model` .

Per disabilitare il ragionamento adattivo su Opus 4.6 e Sonnet 4.6 e ripristinare il budget di pensiero fisso precedente, impostare `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1` . Se disabilitato, questi modelli utilizzano il budget fisso controllato da `MAX_THINKING_TOKENS` . Vedere variabili di ambiente.

Contesto esteso

Opus 4.6 e Sonnet 4.6 supportano una finestra di contesto di 1 milione di token per sessioni lunghe con basi di codice di grandi dimensioni.

> **Note:**
>
> La finestra di contesto 1M è attualmente in beta. Le funzionalità, i prezzi e la disponibilità potrebbero cambiare.

Il contesto esteso è disponibile per:

- **Utenti API e pay-as-you-go**: accesso completo al contesto 1M
- **Abbonati Pro, Max, Teams ed Enterprise**: disponibile con utilizzo extra abilitato

Per disabilitare completamente il contesto 1M, impostare `CLAUDE_CODE_DISABLE_1M_CONTEXT=1`. Questo rimuove le varianti del modello 1M dal selettore di modelli. Vedere variabili di ambiente.

La selezione di un modello 1M non cambia immediatamente la fatturazione. La sessione utilizza tariffe standard fino a quando non supera 200K token di contesto. Oltre 200K token, le richieste vengono addebitate ai prezzi del contesto lungo con limiti di velocità dedicati. Per gli abbonati, i token oltre 200K vengono fatturati come utilizzo extra anziché tramite l'abbonamento.

Se il tuo account supporta il contesto 1M, l'opzione appare nel selettore di modelli (`/model`) nelle versioni più recenti di Claude Code. Se non la vedi, prova a riavviare la sessione.

Puoi anche utilizzare il suffisso `[1m]` con alias di modelli o nomi di modelli completi:

```
## Utilizza l'alias sonnet[1m]
/model sonnet[1m]

## Oppure aggiungi [1m] a un nome di modello completo
/model claude-sonnet-4-6[1m]
```

Verifica del modello corrente

Puoi vedere quale modello stai utilizzando attualmente in diversi modi:

1. Nella riga di stato (se configurata)

2. In `/status`, che visualizza anche le informazioni del tuo account.

Variabili di ambiente

Puoi utilizzare le seguenti variabili di ambiente, che devono essere **nomi di modelli** completi (o equivalenti per il tuo provider API), per controllare i nomi dei modelli a cui gli alias si mappano.

Variabile di ambiente	Descrizione
`ANTHROPIC_DEFAULT_OPUS_MODEL`	Il modello da utilizzare per `opus`, o per `opusplan` quando Plan Mode è attivo.
`ANTHROPIC_DEFAULT_SONNET_MODEL`	Il modello da utilizzare per `sonnet`, o per `opusplan` quando Plan Mode non è attivo.
`ANTHROPIC_DEFAULT_HAIKU_MODEL`	Il modello da utilizzare per `haiku`, o funzionalità in background
`CLAUDE_CODE_SUBAGENT_MODEL`	Il modello da utilizzare per subagents

Nota: `ANTHROPIC_SMALL_FAST_MODEL` è deprecato a favore di `ANTHROPIC_DEFAULT_HAIKU_MODEL`.

Fissare i modelli per distribuzioni di terze parti

Quando si distribuisce Claude Code tramite Bedrock, Vertex AI o Foundry, fissare le versioni dei modelli prima di distribuire agli utenti.

Senza fissare, Claude Code utilizza alias di modelli (`sonnet`, `opus`, `haiku`) che si risolvono nella versione più recente. Quando Anthropic rilascia un nuovo modello, gli utenti i cui account non hanno la nuova versione abilitata si interromperanno silenziosamente.

> **Warning:**
>
> Imposta tutte e tre le variabili di ambiente del modello su ID di versione specifici come parte della configurazione iniziale. Saltare questo passaggio significa che un aggiornamento di Claude Code può interrompere gli utenti senza alcuna azione da parte tua.

Utilizza le seguenti variabili di ambiente con ID di modello specifici della versione per il tuo provider:

Provider	Esempio
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'`

Applica lo stesso modello per `ANTHROPIC_DEFAULT_SONNET_MODEL` e `ANTHROPIC_DEFAULT_HAIKU_MODEL`. Per gli ID dei modelli attuali e legacy su tutti i provider, vedere Panoramica dei modelli. Per aggiornare gli utenti a una nuova versione del modello, aggiorna queste variabili di ambiente e ridistribuisci.

> **Note:**
>
> L'elenco di autorizzazione `settings.availableModels` si applica comunque quando si utilizzano provider di terze parti. Il filtro corrisponde all'alias del modello (`opus` , `sonnet` , `haiku`), non all'ID del modello specifico del provider.

Eseguire l'override degli ID dei modelli per versione

Le variabili di ambiente a livello di famiglia sopra configurano un ID di modello per alias di famiglia. Se è necessario mappare più versioni all'interno della stessa famiglia a ID di provider distinti, utilizzare invece l'impostazione `modelOverrides`.

`modelOverrides` mappa i singoli ID di modelli Anthropic alle stringhe specifiche del provider che Claude Code invia all'API del tuo provider. Quando un utente seleziona un modello mappato nel selettore `/model`, Claude Code utilizza il valore configurato invece del valore predefinito incorporato.

Questo consente agli amministratori aziendali di instradare ogni versione del modello a un ARN del profilo di inferenza Bedrock specifico, a un nome di versione Vertex AI o a un nome di distribuzione Foundry per la governance, l'allocazione dei costi o l'instradamento regionale.

Imposta `modelOverrides` nel tuo file delle impostazioni:

```
{
  "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"
  }
}
```

Le chiavi devono essere ID di modelli Anthropic come elencati nella Panoramica dei modelli. Per gli ID dei modelli datati, includere il suffisso della data esattamente come appare lì. Le chiavi sconosciute vengono ignorate.

Gli override sostituiscono gli ID dei modelli incorporati che supportano ogni voce nel selettore `/model`. Su Bedrock, gli override hanno la precedenza su qualsiasi profilo di inferenza che Claude Code scopre automaticamente all'avvio. I valori forniti direttamente tramite `ANTHROPIC_MODEL`, `--model` o le variabili di ambiente `ANTHROPIC_DEFAULT_*_MODEL` vengono passati al provider così come sono e non vengono trasformati da `modelOverrides`.

`modelOverrides` funziona insieme a `availableModels`. L'elenco di autorizzazione viene valutato rispetto all'ID del modello Anthropic, non al valore di override, quindi una voce come `"opus"` in `availableModels` continua a corrispondere anche quando le versioni di Opus vengono mappate agli ARN.

Configurazione del prompt caching

Claude Code utilizza automaticamente il prompt caching per ottimizzare le prestazioni e ridurre i costi. Puoi disabilitare il prompt caching globalmente o per livelli di modello specifici:

Variabile di ambiente	Descrizione
`DISABLE_PROMPT_CACHING`	Impostare su `1` per disabilitare il prompt caching per tutti i modelli (ha la precedenza sulle impostazioni per modello)
`DISABLE_PROMPT_CACHING_HAIKU`	Impostare su `1` per disabilitare il prompt caching solo per i modelli Haiku

Variabile di ambiente	Descrizione
`DISABLE_PROMPT_CACHING_SONNET`	Impostare su `1` per disabilitare il prompt caching solo per i modelli Sonnet
`DISABLE_PROMPT_CACHING_OPUS`	Impostare su `1` per disabilitare il prompt caching solo per i modelli Opus

Queste variabili di ambiente ti danno un controllo granulare sul comportamento del prompt caching. L'impostazione globale `DISABLE_PROMPT_CACHING` ha la precedenza sulle impostazioni specifiche del modello, consentendoti di disabilitare rapidamente tutto il caching quando necessario. Le impostazioni per modello sono utili per il controllo selettivo, ad esempio quando si esegue il debug di modelli specifici o si lavora con provider cloud che potrebbero avere implementazioni di caching diverse.

Output styles

> *Adattare Claude Code per usi oltre l'ingegneria del software*

Output styles consente di utilizzare Claude Code come qualsiasi tipo di agente mantenendo le sue capacità principali, come l'esecuzione di script locali, la lettura/scrittura di file e il tracciamento dei TODO.

Output styles integrati

Lo **Default** output style di Claude Code è il prompt di sistema esistente, progettato per aiutarvi a completare i compiti di ingegneria del software in modo efficiente.

Ci sono due output styles integrati aggiuntivi focalizzati sull'insegnamento del codebase e su come Claude opera:

- **Explanatory**: Fornisce "Insights" educativi tra l'aiuto nel completamento dei compiti di ingegneria del software. Aiuta a comprendere le scelte di implementazione e i pattern del codebase.
- **Learning**: Modalità collaborativa di apprendimento pratico in cui Claude non solo condividerà "Insights" durante la codifica, ma vi chiederà anche di contribuire con piccoli, strategici pezzi di codice voi stessi. Claude Code aggiungerà marcatori `TODO(human)` nel vostro codice per voi da implementare.

Come funzionano gli output styles

Gli output styles modificano direttamente il prompt di sistema di Claude Code.

- Tutti gli output styles escludono istruzioni per un output efficiente (come rispondere in modo conciso).
- Gli output styles personalizzati escludono istruzioni per la codifica (come la verifica del codice con i test), a meno che `keep-coding-instructions` non sia true.
- Tutti gli output styles hanno le loro istruzioni personalizzate aggiunte alla fine del prompt di sistema.
- Tutti gli output styles attivano promemoria affinché Claude aderisca alle istruzioni dell'output style durante la conversazione.

Cambiare il vostro output style

Eseguite `/config` e selezionate **Output style** per scegliere uno stile da un menu. La vostra selezione viene salvata in `.claude/settings.local.json` al livello del progetto locale.

Per impostare uno stile senza il menu, modificate direttamente il campo `outputStyle` in un file di impostazioni:

```
{
  "outputStyle": "Explanatory"
}
```

Poiché l'output style è impostato nel prompt di sistema all'avvio della sessione, le modifiche hanno effetto la prossima volta che avviate una nuova sessione. Questo mantiene il prompt di sistema stabile durante una conversazione in modo che il prompt caching possa ridurre la latenza e il costo.

Creare un output style personalizzato

Gli output styles personalizzati sono file Markdown con frontmatter e il testo che verrà aggiunto al prompt di sistema:

```
---
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...]
```

Potete salvare questi file a livello utente (`~/.claude/output-styles`) o a livello di progetto (`.claude/output-styles`).

Frontmatter

I file di output style supportano frontmatter per specificare i metadati:

Frontmatter	Scopo	Predefinito
`name`	Nome dell'output style, se non il nome del file	Eredita dal nome del file
`description`	Descrizione dell'output style, mostrata nel picker `/config`	Nessuno
`keep-coding-instructions`	Se mantenere le parti del prompt di sistema di Claude Code relative alla codifica.	false

Confronti con funzionalità correlate

Output Styles vs. CLAUDE.md vs. –append-system-prompt

Gli output styles "disattivano" completamente le parti del prompt di sistema predefinito di Claude Code specifiche per l'ingegneria del software. Né CLAUDE.md né `--append-system-prompt` modificano il prompt di sistema predefinito di Claude Code. CLAUDE.md aggiunge i contenuti come messaggio utente *seguendo* il prompt di sistema predefinito di Claude Code. `--append-system-prompt` aggiunge il contenuto al prompt di sistema.

Output Styles vs. Agents

Gli output styles influenzano direttamente il loop dell'agente principale e influenzano solo il prompt di sistema. Gli agenti vengono invocati per gestire compiti specifici e possono includere impostazioni aggiuntive come il modello da utilizzare, gli strumenti disponibili e un contesto su quando utilizzare l'agente.

Output Styles vs. Skills

Gli output styles modificano il modo in cui Claude risponde (formattazione, tono, struttura) e sono sempre attivi una volta selezionati. Skills sono prompt specifici per compiti che invocate con `/skill-name` o che Claude carica automaticamente quando rilevante. Utilizzate gli output styles per preferenze di formattazione coerenti; utilizzate skills per flussi di lavoro e compiti riutilizzabili.

Accelera le risposte con la modalità veloce

> *Ottieni risposte più veloci di Opus 4.6 in Claude Code attivando la modalità veloce.*

Note:

La modalità veloce è in anteprima di ricerca. La funzione, i prezzi e la disponibilità potrebbero cambiare in base al feedback.

La modalità veloce è una configurazione ad alta velocità per Claude Opus 4.6, che rende il modello 2,5 volte più veloce a un costo per token più elevato. Attivala con `/fast` quando hai bisogno di velocità per il lavoro interattivo come l'iterazione rapida o il debug in tempo reale, e disattivala quando il costo è più importante della latenza.

La modalità veloce non è un modello diverso. Utilizza lo stesso Opus 4.6 con una configurazione API diversa che dà priorità alla velocità rispetto all'efficienza dei costi. Ottieni la stessa qualità e capacità, solo risposte più veloci.

Note:

La modalità veloce richiede Claude Code v2.1.36 o successivo. Controlla la tua versione con `claude --version`.

Cosa sapere:

- Usa `/fast` per attivare/disattivare la modalità veloce in Claude Code CLI. Disponibile anche tramite `/fast` nell'estensione Claude Code VS Code.
- I prezzi della modalità veloce per Opus 4.6 iniziano da $30/150 MTok. La modalità veloce è disponibile con uno sconto del 50% per tutti i piani fino alle 23:59 PT del 16 febbraio.
- Disponibile per tutti gli utenti di Claude Code sui piani di abbonamento (Pro/Max/Team/Enterprise) e Claude Console.
- Per gli utenti di Claude Code sui piani di abbonamento (Pro/Max/Team/Enterprise), la modalità veloce è disponibile solo tramite utilizzo aggiuntivo e non è inclusa nei limiti di velocità dell'abbonamento.

Questa pagina copre come attivare/disattivare la modalità veloce, il suo compromesso di costo, quando usarla, requisiti, opt-in per sessione, e comportamento dei limiti di velocità.

Attiva/disattiva la modalità veloce

Attiva/disattiva la modalità veloce in uno di questi modi:

- Digita `/fast` e premi Tab per attivare o disattivare
- Imposta `"fastMode": true` nel tuo file di impostazioni utente

Per impostazione predefinita, la modalità veloce persiste tra le sessioni. Gli amministratori possono configurare la modalità veloce per ripristinarsi ogni sessione. Vedi richiedi opt-in per sessione per i dettagli.

Per la migliore efficienza dei costi, abilita la modalità veloce all'inizio di una sessione piuttosto che passare a metà conversazione. Vedi comprendi il compromesso di costo per i dettagli.

Quando abiliti la modalità veloce:

- Se sei su un modello diverso, Claude Code passa automaticamente a Opus 4.6
- Vedrai un messaggio di conferma: "Fast mode ON"
- Un piccolo icona `↯` appare accanto al prompt mentre la modalità veloce è attiva
- Esegui `/fast` di nuovo in qualsiasi momento per verificare se la modalità veloce è attiva o disattiva

Quando disabiliti la modalità veloce con `/fast` di nuovo, rimani su Opus 4.6. Il modello non torna al tuo modello precedente. Per passare a un modello diverso, usa `/model`.

Comprendi il compromesso di costo

La modalità veloce ha un prezzo per token più elevato rispetto a Opus 4.6 standard:

Modalità	Input (MTok)	Output (MTok)
Modalità veloce su Opus 4.6 (<200K)	$30	$150
Modalità veloce su Opus 4.6 (>200K)	$60	$225

La modalità veloce è compatibile con la finestra di contesto estesa di 1M token.

Quando passi alla modalità veloce a metà conversazione, paghi il prezzo completo del token di input non memorizzato nella cache della modalità veloce per l'intero contesto della conversazione. Questo costa più di quanto avresti pagato se avessi abilitato la modalità veloce dall'inizio.

Decidi quando usare la modalità veloce

La modalità veloce è migliore per il lavoro interattivo dove la latenza della risposta è più importante del costo:

- Iterazione rapida su modifiche del codice
- Sessioni di debug in tempo reale
- Lavoro sensibile al tempo con scadenze strette

La modalità standard è migliore per:

- Attività autonome lunghe dove la velocità è meno importante
- Elaborazione batch o pipeline CI/CD
- Carichi di lavoro sensibili ai costi

Modalità veloce rispetto al livello di sforzo

La modalità veloce e il livello di sforzo influenzano entrambi la velocità di risposta, ma in modo diverso:

Impostazione	Effetto
Modalità veloce	Stessa qualità del modello, latenza inferiore, costo più elevato
Livello di sforzo inferiore	Meno tempo di riflessione, risposte più veloci, potenzialmente qualità inferiore su attività complesse

Puoi combinare entrambi: usa la modalità veloce con un livello di sforzo inferiore per la massima velocità su attività semplici.

Requisiti

La modalità veloce richiede tutti i seguenti elementi:

- **Non disponibile su provider cloud di terze parti**: la modalità veloce non è disponibile su Amazon Bedrock, Google Vertex AI o Microsoft Azure Foundry. La modalità veloce è disponibile tramite l'API Anthropic Console e per i piani di abbonamento Claude utilizzando l'utilizzo aggiuntivo.

- **Utilizzo aggiuntivo abilitato**: il tuo account deve avere l'utilizzo aggiuntivo abilitato, che consente la fatturazione oltre l'utilizzo incluso nel tuo piano. Per gli account individuali, abilita questo nelle tue impostazioni di fatturazione della Console. Per Teams e Enterprise, un amministratore deve abilitare l'utilizzo aggiuntivo per l'organizzazione.

> **Note:**
>
> L'utilizzo della modalità veloce viene fatturato direttamente all'utilizzo aggiuntivo, anche se hai un utilizzo rimanente nel tuo piano. Ciò significa che i token della modalità veloce non contano rispetto all'utilizzo incluso nel tuo piano e vengono addebitati alla tariffa della modalità veloce dal primo token.

- **Abilitazione dell'amministratore per Teams e Enterprise**: la modalità veloce è disabilitata per impostazione predefinita per le organizzazioni Teams e Enterprise. Un amministratore deve esplicitamente abilitare la modalità veloce prima che gli utenti possano accedervi.

> **Note:**
>
> Se il tuo amministratore non ha abilitato la modalità veloce per la tua organizzazione, il comando `/fast` mostrerà "Fast mode has been disabled by your organization."

Abilita la modalità veloce per la tua organizzazione

Gli amministratori possono abilitare la modalità veloce in:

- **Console** (clienti API): Preferenze Claude Code
- **Claude AI** (Teams e Enterprise): Admin Settings > Claude Code

Un'altra opzione per disabilitare completamente la modalità veloce è impostare `CLAUDE_CODE_DISABLE_FAST_MODE=1`. Vedi Variabili di ambiente.

Richiedi opt-in per sessione

Per impostazione predefinita, la modalità veloce persiste tra le sessioni: se un utente abilita la modalità veloce, rimane attiva nelle sessioni future. Gli amministratori sui piani Teams o Enterprise possono impedire questo impostando `fastModePerSessionOptIn` a `true` nelle impostazioni gestite o impostazioni gestite dal server. Ciò fa sì che ogni sessione inizi con la modalità veloce disattivata, richiedendo agli utenti di abilitarla esplicitamente con `/fast`.

```
{
  "fastModePerSessionOptIn": true
}
```

Questo è utile per controllare i costi nelle organizzazioni in cui gli utenti eseguono più sessioni simultanee. Gli utenti possono comunque abilitare la modalità veloce con `/fast` quando hanno bisogno di velocità, ma si ripristina all'inizio di ogni nuova sessione. La preferenza della modalità veloce dell'utente è ancora salvata, quindi rimuovere questa impostazione ripristina il comportamento persistente predefinito.

Gestisci i limiti di velocità

La modalità veloce ha limiti di velocità separati da Opus 4.6 standard. Quando raggiungi il limite di velocità della modalità veloce o esaurisci i crediti di utilizzo aggiuntivo:

1. La modalità veloce torna automaticamente a Opus 4.6 standard
2. L'icona `↯` diventa grigia per indicare il raffreddamento
3. Continui a lavorare a velocità e prezzi standard
4. Quando il raffreddamento scade, la modalità veloce si riabilita automaticamente

Per disabilitare manualmente la modalità veloce invece di aspettare il raffreddamento, esegui `/fast` di nuovo.

Anteprima di ricerca

La modalità veloce è una funzione di anteprima di ricerca. Ciò significa:

- La funzione potrebbe cambiare in base al feedback
- La disponibilità e i prezzi sono soggetti a modifiche
- La configurazione API sottostante potrebbe evolversi

Segnala problemi o feedback tramite i tuoi soliti canali di supporto Anthropic.

Vedi anche

- Configurazione del modello: cambia modelli e regola i livelli di sforzo
- Gestisci i costi in modo efficace: traccia l'utilizzo dei token e riduci i costi
- Configurazione della riga di stato: visualizza le informazioni del modello e del contesto

Gestisci i costi in modo efficace

> *Traccia l'utilizzo dei token, imposta i limiti di spesa del team e riduci i costi di Claude Code con la gestione del contesto, la selezione del modello, le impostazioni del pensiero esteso e gli hook di pre-elaborazione.*

Claude Code consuma token per ogni interazione. I costi variano in base alle dimensioni della codebase, alla complessità della query e alla lunghezza della conversazione. Il costo medio è di $6 per sviluppatore al giorno, con costi giornalieri che rimangono al di sotto di $12 per il 90% degli utenti.

Per l'utilizzo del team, Claude Code addebita il consumo di token API. In media, Claude Code costa circa $100-200 per sviluppatore al mese con Sonnet 4.6, anche se c'è una grande varianza a seconda di quante istanze gli utenti stanno eseguendo e se la stanno utilizzando nell'automazione.

Questa pagina spiega come tracciare i tuoi costi, gestire i costi per i team e ridurre l'utilizzo dei token.

Traccia i tuoi costi

Utilizzo del comando `/cost`

> **Note:**
>
> Il comando `/cost` mostra l'utilizzo dei token API ed è destinato agli utenti API. I sottoscrittori di Claude Max e Pro hanno l'utilizzo incluso nel loro abbonamento, quindi i dati di `/cost` non sono rilevanti per scopi di fatturazione. I sottoscrittori possono utilizzare `/stats` per visualizzare i modelli di utilizzo.

Il comando `/cost` fornisce statistiche dettagliate sull'utilizzo dei token per la tua sessione attuale:

```
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
```

Gestione dei costi per i team

Quando utilizzi Claude API, puoi impostare i limiti di spesa dell'area di lavoro sulla spesa totale dell'area di lavoro di Claude Code. Gli amministratori possono visualizzare i rapporti di costo e utilizzo nella Console.

> **Note:**
>
> Quando autentichi per la prima volta Claude Code con il tuo account Claude Console, viene creata automaticamente un'area di lavoro chiamata "Claude Code". Questa area di lavoro fornisce il tracciamento e la gestione centralizzati dei costi per tutto l'utilizzo di Claude Code nella tua organizzazione. Non puoi creare chiavi API per questa area di lavoro; è esclusivamente per l'autenticazione e l'utilizzo di Claude Code.

Su Bedrock, Vertex e Foundry, Claude Code non invia metriche dal tuo cloud. Per ottenere metriche di costo, diversi grandi enterprise hanno riferito di utilizzare LiteLLM, uno strumento open-source che aiuta le aziende a tracciare la spesa per chiave. Questo progetto non è affiliato ad Anthropic e non è stato sottoposto a audit di sicurezza.

Raccomandazioni sui limiti di velocità

Quando configuri Claude Code per i team, considera queste raccomandazioni Token Per Minuto (TPM) e Richieste Per Minuto (RPM) per utente in base alle dimensioni della tua organizzazione:

Dimensione del team	TPM per utente	RPM per utente
1-5 utenti	200k-300k	5-7
5-20 utenti	100k-150k	2.5-3.5
20-50 utenti	50k-75k	1.25-1.75
50-100 utenti	25k-35k	0.62-0.87
100-500 utenti	15k-20k	0.37-0.47
500+ utenti	10k-15k	0.25-0.35

Ad esempio, se hai 200 utenti, potresti richiedere 20k TPM per ogni utente, o 4 milioni di TPM totali (200*20.000 = 4 milioni).

Il TPM per utente diminuisce man mano che le dimensioni del team crescono perché meno utenti tendono a utilizzare Claude Code contemporaneamente nelle organizzazioni più grandi. Questi limiti di velocità si applicano a livello organizzativo, non per singolo utente, il che significa che i singoli utenti possono temporaneamente consumare più della loro quota calcolata quando altri non stanno utilizzando attivamente il servizio.

> **Note:**
>
> Se prevedi scenari con utilizzo concorrente insolitamente elevato (come sessioni di formazione dal vivo con grandi gruppi), potresti aver bisogno di allocazioni TPM più elevate per utente.

Costi dei token del team di agenti

I team di agenti generano più istanze di Claude Code, ognuna con la propria finestra di contesto. L'utilizzo dei token si ridimensiona con il numero di compagni di squadra attivi e per quanto tempo ognuno viene eseguito.

Per mantenere i costi del team di agenti gestibili:

- Utilizza Sonnet per i compagni di squadra. Bilancia la capacità e il costo per i compiti di coordinamento.
- Mantieni i team piccoli. Ogni compagno di squadra esegue la propria finestra di contesto, quindi l'utilizzo dei token è approssimativamente proporzionale alle dimensioni del team.
- Mantieni i prompt di generazione focalizzati. I compagni di squadra caricano automaticamente CLAUDE.md, i server MCP e le skills, ma tutto nel prompt di generazione si aggiunge al loro contesto dall'inizio.
- Pulisci i team quando il lavoro è terminato. I compagni di squadra attivi continuano a consumare token anche se inattivi.
- I team di agenti sono disabilitati per impostazione predefinita. Imposta `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` nel tuo settings.json o nell'ambiente per abilitarli. Vedi abilita i team di agenti.

Riduci l'utilizzo dei token

I costi dei token si ridimensionano con la dimensione del contesto: più contesto Claude elabora, più token utilizzi. Claude Code ottimizza automaticamente i costi attraverso il prompt caching (che riduce i costi per il contenuto ripetuto come i prompt di sistema) e l'auto-compaction (che riassume la cronologia della conversazione quando ci si avvicina ai limiti del contesto).

Le seguenti strategie ti aiutano a mantenere il contesto piccolo e ridurre i costi per messaggio.

Gestisci il contesto in modo proattivo

Utilizza `/cost` per controllare l'utilizzo attuale dei token, o configura la tua linea di stato per visualizzarla continuamente.

- **Cancella tra i compiti**: Utilizza `/clear` per ricominciare da capo quando passi a lavori non correlati. Il contesto obsoleto spreca token su ogni messaggio successivo. Utilizza `/rename` prima di cancellare in modo da poter trovare facilmente la sessione in seguito, quindi `/resume` per tornare ad essa.
- **Aggiungi istruzioni di compaction personalizzate**: `/compact Focus on code samples and API usage` dice a Claude cosa preservare durante la sintesi.

Puoi anche personalizzare il comportamento della compaction nel tuo CLAUDE.md:

```
## Compact instructions

When you are using compact, please focus on test output and code changes
```

Scegli il modello giusto

Sonnet gestisce bene la maggior parte dei compiti di codifica e costa meno di Opus. Riserva Opus per decisioni architettoniche complesse o ragionamento multi-step. Utilizza `/model` per cambiare modello a metà sessione, o imposta un valore predefinito in `/config`. Per semplici compiti subagent, specifica `model: haiku` nella tua configurazione subagent.

Riduci l'overhead del server MCP

Ogni server MCP aggiunge definizioni di strumenti al tuo contesto, anche quando inattivo. Esegui `/context` per vedere cosa sta consumando spazio.

- **Preferisci gli strumenti CLI quando disponibili**: Strumenti come `gh`, `aws`, `gcloud` e `sentry-cli` sono più efficienti dal punto di vista del contesto rispetto ai server MCP perché non aggiungono definizioni di strumenti persistenti. Claude può eseguire comandi CLI direttamente senza l'overhead.
- **Disabilita i server inutilizzati**: Esegui `/mcp` per vedere i server configurati e disabilita quelli che non stai utilizzando attivamente.
- **La ricerca degli strumenti è automatica**: Quando le descrizioni degli strumenti MCP superano il 10% della tua finestra di contesto, Claude Code li rinvia

automaticamente e carica gli strumenti su richiesta tramite ricerca degli strumenti. Poiché gli strumenti rinviati entrano nel contesto solo quando effettivamente utilizzati, una soglia più bassa significa meno definizioni di strumenti inattivi che consumano spazio. Imposta una soglia più bassa con `ENABLE_TOOL_SEARCH=auto:<N>` (ad esempio, `auto:5` si attiva quando gli strumenti superano il 5% della tua finestra di contesto).

Installa plugin di intelligenza del codice per i linguaggi tipizzati

I plugin di intelligenza del codice danno a Claude una navigazione precisa dei simboli invece della ricerca basata su testo, riducendo le letture di file non necessarie quando si esplora codice sconosciuto. Una singola chiamata "vai alla definizione" sostituisce quello che altrimenti potrebbe essere un grep seguito dalla lettura di più file candidati. I server di linguaggio installati segnalano anche gli errori di tipo automaticamente dopo le modifiche, quindi Claude cattura gli errori senza eseguire un compilatore.

Offload dell'elaborazione agli hook e alle skills

Gli hook personalizzati possono pre-elaborare i dati prima che Claude li veda. Invece di Claude che legge un file di log di 10.000 righe per trovare errori, un hook può cercare `ERROR` e restituire solo le righe corrispondenti, riducendo il contesto da decine di migliaia di token a centinaia.

Una skill può dare a Claude la conoscenza del dominio in modo che non debba esplorare. Ad esempio, una skill "codebase-overview" potrebbe descrivere l'architettura del tuo progetto, le directory chiave e le convenzioni di denominazione. Quando Claude invoca la skill, ottiene questo contesto immediatamente invece di spendere token leggendo più file per comprendere la struttura.

Ad esempio, questo hook PreToolUse filtra l'output del test per mostrare solo i fallimenti:

settings.json

Aggiungi questo al tuo settings.json per eseguire l'hook prima di ogni comando Bash:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/.claude/hooks/filter-test-output.sh"
          }
        ]
      }
    ]
  }
}
```

filter-test-output.sh

L'hook chiama questo script, che controlla se il comando è un test runner e lo modifica per mostrare solo i fallimenti:

```
#!/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
```

Sposta le istruzioni da CLAUDE.md alle skills

Il tuo file CLAUDE.md viene caricato nel contesto all'inizio della sessione. Se contiene istruzioni dettagliate per flussi di lavoro specifici (come revisioni PR o migrazioni di database), quei token sono presenti anche quando stai facendo lavori non correlati. Le

skills si caricano su richiesta solo quando invocate, quindi spostare le istruzioni specializzate nelle skills mantiene il tuo contesto di base più piccolo. Mira a mantenere CLAUDE.md sotto circa 500 righe includendo solo gli elementi essenziali.

Regola il pensiero esteso

Il pensiero esteso è abilitato per impostazione predefinita con un budget di 31.999 token perché migliora significativamente le prestazioni su compiti complessi di pianificazione e ragionamento. Tuttavia, i token di pensiero vengono fatturati come token di output, quindi per compiti più semplici dove il ragionamento profondo non è necessario, puoi ridurre i costi abbassando il livello di sforzo in `/model` per Opus 4.6, disabilitando il pensiero in `/config`, o abbassando il budget (ad esempio, `MAX_THINKING_TOKENS=8000`).

Delega le operazioni dettagliate ai subagent

L'esecuzione di test, il recupero della documentazione o l'elaborazione di file di log possono consumare un contesto significativo. Delega questi ai subagent in modo che l'output dettagliato rimanga nel contesto del subagent mentre solo un riassunto ritorna alla tua conversazione principale.

Gestisci i costi del team di agenti

I team di agenti utilizzano approssimativamente 7 volte più token rispetto alle sessioni standard quando i compagni di squadra vengono eseguiti in plan mode, perché ogni compagno di squadra mantiene la propria finestra di contesto ed esegue come un'istanza Claude separata. Mantieni i compiti del team piccoli e autonomi per limitare l'utilizzo dei token per compagno di squadra. Vedi team di agenti per i dettagli.

Scrivi prompt specifici

Richieste vaghe come "migliora questa codebase" attivano una scansione ampia. Richieste specifiche come "aggiungi la convalida dell'input alla funzione di accesso in auth.ts" permettono a Claude di lavorare in modo efficiente con letture di file minime.

Lavora in modo efficiente su compiti complessi

Per lavori più lunghi o complessi, queste abitudini aiutano a evitare token sprecati andando nella direzione sbagliata:

- **Utilizza plan mode per compiti complessi**: Premi Shift+Tab per entrare in plan mode prima dell'implementazione. Claude esplora la codebase e propone un approccio per la tua approvazione, prevenendo il rielaborazione costosa quando la direzione iniziale è sbagliata.

- **Correggi la rotta presto**: Se Claude inizia a andare nella direzione sbagliata, premi Escape per fermarti immediatamente. Utilizza `/rewind` o doppio tocco Escape per ripristinare la conversazione e il codice a un checkpoint precedente.
- **Fornisci target di verifica**: Includi casi di test, incolla screenshot o definisci l'output previsto nel tuo prompt. Quando Claude può verificare il suo lavoro, cattura i problemi prima che tu debba richiedere correzioni.
- **Testa in modo incrementale**: Scrivi un file, testalo, quindi continua. Questo cattura i problemi presto quando sono economici da risolvere.

Utilizzo dei token in background

Claude Code utilizza token per alcune funzionalità in background anche quando inattivo:

- **Sintesi della conversazione**: Processi in background che riassumono le conversazioni precedenti per la funzione `claude --resume`
- **Elaborazione dei comandi**: Alcuni comandi come `/cost` possono generare richieste per controllare lo stato

Questi processi in background consumano una piccola quantità di token (in genere meno di $0,04 per sessione) anche senza interazione attiva.

Comprensione dei cambiamenti nel comportamento di Claude Code

Claude Code riceve regolarmente aggiornamenti che possono cambiare il funzionamento delle funzionalità, inclusa la segnalazione dei costi. Esegui `claude --version` per controllare la tua versione attuale. Per domande specifiche sulla fatturazione, contatta il supporto di Anthropic tramite il tuo account Console. Per distribuzioni di team, inizia con un piccolo gruppo pilota per stabilire i modelli di utilizzo prima di un rollout più ampio.

Part 5: Extensibility

Riferimento dei hooks

> *Riferimento per gli eventi dei hook di Claude Code, schema di configurazione, formati JSON di input/output, codici di uscita, hook asincroni, hook HTTP, hook di prompt e hook degli strumenti MCP.*

> **Tip:**
>
> Per una guida di avvio rapido con esempi, vedere Automatizzare i flussi di lavoro con i hook.

Gli hook sono comandi shell definiti dall'utente, endpoint HTTP o prompt LLM che si eseguono automaticamente in punti specifici del ciclo di vita di Claude Code. Utilizzare questo riferimento per cercare schemi di eventi, opzioni di configurazione, formati JSON di input/output e funzionalità avanzate come hook asincroni, hook HTTP e hook degli strumenti MCP. Se si stanno configurando i hook per la prima volta, iniziare con la guida.

Ciclo di vita dei hook

Gli hook si attivano in punti specifici durante una sessione di Claude Code. Quando un evento si attiva e un matcher corrisponde, Claude Code passa il contesto JSON dell'evento al gestore del hook. Per i hook di comando, l'input arriva su stdin. Per i hook HTTP, arriva come corpo della richiesta POST. Il gestore può quindi ispezionare l'input, intraprendere un'azione e facoltativamente restituire una decisione. Alcuni eventi si attivano una volta per sessione, mentre altri si attivano ripetutamente all'interno del ciclo agentico:

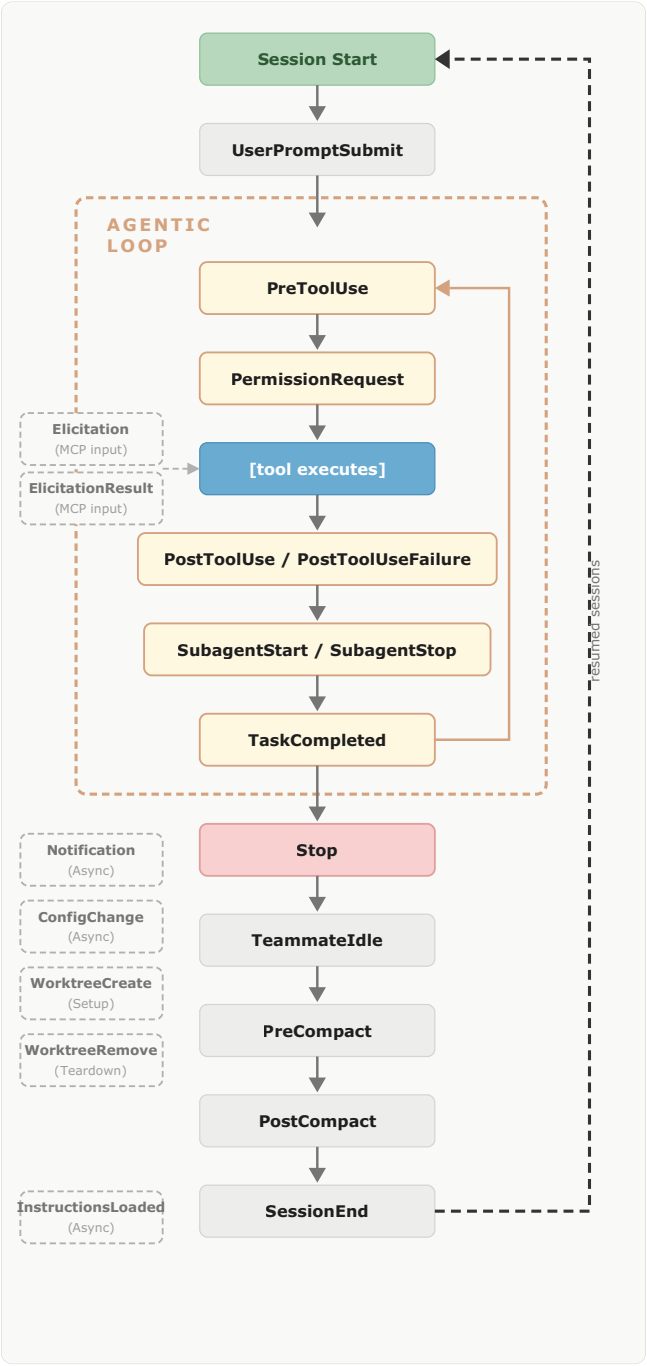

Session Start
UserPromptSubmit
AGENTIC LOOP
PreToolUse
PermissionRequest
Elicitation
(MCP input)
[tool executes]
ElicitationResult
(MCP input)
PostToolUse / PostToolUseFailure
SubagentStart / SubagentStop
TaskCompleted
Stop
Notification
(Async)
ConfigChange
(Async)
TeammateIdle
WorktreeCreate
(Setup)
PreCompact
WorktreeRemove
(Teardown)
PostCompact
InstructionsLoaded
(Async)
SessionEnd
resumed sessions

Diagramma del ciclo di vita dei hook che mostra la sequenza dei hook da SessionStart attraverso il ciclo agentico a SessionEnd, con WorktreeCreate, WorktreeRemove e InstructionsLoaded come eventi asincroni autonomi

La tabella seguente riassume quando si attiva ogni evento. La sezione Hook events documenta lo schema di input completo e le opzioni di controllo della decisione per ognuno.

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

Come si risolve un hook

Per vedere come questi elementi si combinano, considerare questo hook `PreToolUse` che blocca i comandi shell distruttivi. L'hook esegue `block-rm.sh` prima di ogni chiamata dello strumento Bash:

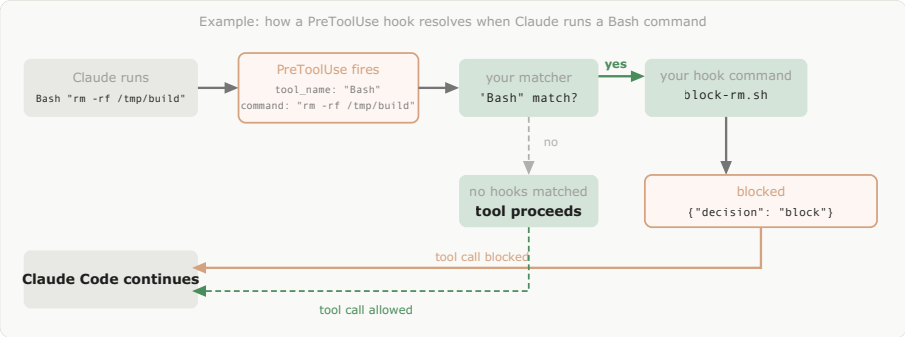
```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": ".claude/hooks/block-rm.sh"
          }
        ]
      }
    ]
  }
}
```

Lo script legge l'input JSON da stdin, estrae il comando e restituisce una `permissionDecision` di `"deny"` se contiene `rm -rf`:

```
#!/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
```

Ora supponiamo che Claude Code decida di eseguire `Bash "rm -rf /tmp/build"`. Ecco cosa accade:



Flusso di risoluzione del hook: l'evento PreToolUse si attiva, il matcher controlla la corrispondenza di Bash, il gestore del hook viene eseguito, il risultato ritorna a Claude Code

Step 1: L'evento si attiva

L'evento `PreToolUse` si attiva. Claude Code invia l'input dello strumento come JSON su stdin al hook:

```
{ "tool_name": "Bash", "tool_input": { "command": "rm -rf /tmp/build" }, ... }
```

Step 2: Il matcher controlla

Il matcher `"Bash"` corrisponde al nome dello strumento, quindi `block-rm.sh` viene eseguito. Se si omette il matcher o si utilizza `"*"`, l'hook viene eseguito su ogni occorrenza dell'evento. Gli hook vengono saltati solo quando un matcher è definito e non corrisponde.

Step 3: Il gestore del hook viene eseguito

Lo script estrae `"rm -rf /tmp/build"` dall'input e trova `rm -rf`, quindi stampa una decisione su stdout:

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Destructive command blocked by hook"
  }
}
```

Se il comando fosse stato sicuro (come `npm test`), lo script avrebbe raggiunto `exit 0`, che dice a Claude Code di consentire la chiamata dello strumento senza ulteriori azioni.

Step 4: Claude Code agisce sul risultato

Claude Code legge la decisione JSON, blocca la chiamata dello strumento e mostra a Claude il motivo.

La sezione Configuration seguente documenta lo schema completo e ogni sezione hook event documenta quale input riceve il comando e quale output può restituire.

Configuration

Gli hook sono definiti in file di impostazioni JSON. La configurazione ha tre livelli di annidamento:

1. Scegliere un hook event a cui rispondere, come `PreToolUse` o `Stop`
2. Aggiungere un matcher group per filtrare quando si attiva, come "solo per lo strumento Bash"
3. Definire uno o più hook handlers da eseguire quando corrisponde

Vedere Come si risolve un hook sopra per una procedura dettagliata completa con un esempio annotato.

> **Note:**
>
> Questa pagina utilizza termini specifici per ogni livello: **hook event** per il punto del ciclo di vita, **matcher group** per il filtro e **hook handler** per il comando shell, endpoint HTTP, prompt o agente che viene eseguito. "Hook" da solo si riferisce alla funzionalità generale.

Posizioni dei hook

Il luogo in cui si definisce un hook determina il suo ambito:

Posizione	Ambito	Condivisibile
`~/.claude/settings.json`	Tutti i vostri progetti	No, locale sulla vostra macchina
`.claude/settings.json`	Progetto singolo	Sì, può essere committato nel repository
`.claude/settings.local.json`	Progetto singolo	No, gitignored
Impostazioni della politica gestita	Organizzazione intera	Sì, controllato dall'amministratore
Plugin `hooks/hooks.json`	Quando il plugin è abilitato	Sì, raggruppato con il plugin
Skill o agent frontmatter	Mentre il componente è attivo	Sì, definito nel file del componente

Per i dettagli sulla risoluzione del file di impostazioni, vedere settings. Gli amministratori aziendali possono utilizzare `allowManagedHooksOnly` per bloccare i hook dell'utente, del progetto e del plugin. Vedere Hook configuration.

Modelli di matcher

Il campo `matcher` è una stringa regex che filtra quando si attivano i hook. Utilizzare `"*"`, `""` o omettere completamente `matcher` per corrispondere a tutte le occorrenze. Ogni tipo di evento corrisponde a un campo diverso:

Evento	Su cosa filtra il matcher	Valori di matcher di esempio
`PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `PermissionRequest`	nome dello strumento	`Bash`, `Edit|Write`, `mcp__.*`
`SessionStart`	come è iniziata la sessione	`startup`, `resume`, `clear`, `compact`
`SessionEnd`	perché è terminata la sessione	`clear`, `logout`, `prompt_input_exit`, `bypass_permissions_disabled`, `other`
`Notification`	tipo di notifica	`permission_prompt`, `idle_prompt`, `auth_success`, `elicitation_dialog`
`SubagentStart`	tipo di agente	`Bash`, `Explore`, `Plan` o nomi di agenti personalizzati
`PreCompact`	cosa ha attivato la compattazione	`manual`, `auto`
`SubagentStop`	tipo di agente	stessi valori di `SubagentStart`
`ConfigChange`	fonte di configurazione	`user_settings`, `project_settings`, `local_settings`, `policy_settings`, `skills`
`UserPromptSubmit`, `Stop`, `TeammateIdle`, `TaskCompleted`, `WorktreeCreate`, `WorktreeRemove`, `InstructionsLoaded`	nessun supporto matcher	si attiva sempre su ogni occorrenza

Il matcher è una regex, quindi `Edit|Write` corrisponde a entrambi gli strumenti e `Notebook.*` corrisponde a qualsiasi strumento che inizia con Notebook. Il matcher viene eseguito su un campo dall'input JSON che Claude Code invia al vostro hook su stdin. Per gli eventi degli strumenti, quel campo è `tool_name`. Ogni sezione hook event elenca l'insieme completo di valori di matcher e lo schema di input per quell'evento.

Questo esempio esegue uno script di linting solo quando Claude scrive o modifica un file:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/lint-check.sh"
          }
        ]
      }
    ]
  }
}
```

`UserPromptSubmit`, `Stop`, `TeammateIdle`, `TaskCompleted`, `WorktreeCreate`, `WorktreeRemove` e `InstructionsLoaded` non supportano i matcher e si attivano sempre su ogni occorrenza. Se si aggiunge un campo `matcher` a questi eventi, viene silenziosamente ignorato.

Corrispondere ai strumenti MCP

Gli strumenti del server MCP appaiono come strumenti regolari negli eventi degli strumenti (`PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `PermissionRequest`), quindi potete farvi corrispondere allo stesso modo di qualsiasi altro nome di strumento.

Gli strumenti MCP seguono il modello di denominazione `mcp__<server>__<tool>`, ad esempio:

- `mcp__memory__create_entities`: strumento create entities del server Memory
- `mcp__filesystem__read_file`: strumento read file del server Filesystem
- `mcp__github__search_repositories`: strumento search del server GitHub

Utilizzare modelli regex per indirizzare strumenti MCP specifici o gruppi di strumenti:

- `mcp__memory__.*` corrisponde a tutti gli strumenti dal server `memory`
- `mcp__.*__write.*` corrisponde a qualsiasi strumento contenente “write” da qualsiasi server

Questo esempio registra tutte le operazioni del server memory e convalida le operazioni di scrittura da qualsiasi server MCP:

```
{
  "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"
          }
        ]
      }
    ]
  }
}
```

Campi del gestore del hook

Ogni oggetto nell'array `hooks` interno è un gestore del hook: il comando shell, endpoint HTTP, prompt LLM o agente che viene eseguito quando il matcher corrisponde. Ci sono quattro tipi:

- **Command hooks** (`type: "command"`): eseguono un comando shell. Lo script riceve l'input JSON dell'evento su stdin e comunica i risultati attraverso codici di uscita e stdout.
- **HTTP hooks** (`type: "http"`): inviano l'input JSON dell'evento come richiesta HTTP POST a un URL. L'endpoint comunica i risultati attraverso il corpo della risposta utilizzando lo stesso formato JSON di output dei command hook.
- **Prompt hooks** (`type: "prompt"`): inviano un prompt a un modello Claude per la valutazione a turno singolo. Il modello restituisce una decisione sì/no come JSON. Vedere Prompt-based hooks.
- **Agent hooks** (`type: "agent"`): generano un subagent che può utilizzare strumenti come Read, Grep e Glob per verificare le condizioni prima di restituire una decisione. Vedere Agent-based hooks.

Campi comuni

Questi campi si applicano a tutti i tipi di hook:

Campo	Obbligatorio	Descrizione
`type`	sì	`"command"` , `"http"` , `"prompt"` o `"agent"`
`timeout`	no	Secondi prima dell'annullamento. Impostazioni predefinite: 600 per command, 30 per prompt, 60 per agent
`statusMessage`	no	Messaggio spinner personalizzato visualizzato mentre il hook viene eseguito
`once`	no	Se `true` , viene eseguito una sola volta per sessione e poi rimosso. Solo skill, non agenti. Vedere Hooks in skills and agents

Campi del command hook

Oltre ai campi comuni, i command hook accettano questi campi:

Campo	Obbligatorio	Descrizione
`command`	sì	Comando shell da eseguire
`async`	no	Se `true`, viene eseguito in background senza bloccare. Vedere Run hooks in the background

Campi dell'HTTP hook

Oltre ai campi comuni, gli HTTP hook accettano questi campi:

Campo	Obbligatorio	Descrizione
`url`	sì	URL a cui inviare la richiesta POST
`headers`	no	Intestazioni HTTP aggiuntive come coppie chiave-valore. I valori supportano l'interpolazione delle variabili di ambiente utilizzando la sintassi `$VAR_NAME` o `${VAR_NAME}`. Solo le variabili elencate in `allowedEnvVars` vengono risolte
`allowedEnvVars`	no	Elenco dei nomi delle variabili di ambiente che possono essere interpolate nei valori dell'intestazione. I riferimenti alle variabili non elencate vengono sostituiti con stringhe vuote. Obbligatorio per qualsiasi interpolazione di variabili di ambiente

Claude Code invia l'input JSON del hook come corpo della richiesta POST con `Content-Type: application/json`. Il corpo della risposta utilizza lo stesso formato JSON di output dei command hook.

La gestione degli errori differisce dai command hook: le risposte non-2xx, i guasti di connessione e i timeout producono tutti errori non bloccanti che consentono l'esecuzione di continuare. Per bloccare una chiamata dello strumento o negare un'autorizzazione, restituire una risposta 2xx con un corpo JSON contenente `decision: "block"` o un `hookSpecificOutput` con `permissionDecision: "deny"`.

Questo esempio invia gli eventi `PreToolUse` a un servizio di convalida locale, autenticandosi con un token dalla variabile di ambiente `MY_TOKEN`:

```
{
  "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:

Gli HTTP hook devono essere configurati modificando direttamente il JSON delle impostazioni. Il menu interattivo `/hooks` supporta solo l'aggiunta di command hook.

Campi dei prompt hook e agent hook

Oltre ai campi comuni, i prompt hook e agent hook accettano questi campi:

Campo	Obbligatorio	Descrizione
`prompt`	sì	Testo del prompt da inviare al modello. Utilizzare `$ARGUMENTS` come segnaposto per l'input JSON del hook
`model`	no	Modello da utilizzare per la valutazione. Impostazione predefinita su un modello veloce

Tutti i hook corrispondenti vengono eseguiti in parallelo e i gestori identici vengono automaticamente deduplicati. I command hook vengono deduplicati per stringa di comando e gli HTTP hook vengono deduplicati per URL. I gestori vengono eseguiti nella directory corrente con l'ambiente di Claude Code. La variabile di ambiente `$CLAUDE_CODE_REMOTE` è impostata su `"true"` negli ambienti web remoti e non è impostata nella CLI locale.

Fare riferimento agli script per percorso

Utilizzare le variabili di ambiente per fare riferimento agli script del hook relativi alla radice del progetto o del plugin, indipendentemente dalla directory di lavoro quando il hook viene eseguito:

- `$CLAUDE_PROJECT_DIR` : la radice del progetto. Racchiudere tra virgolette per gestire i percorsi con spazi.
- `${CLAUDE_PLUGIN_ROOT}` : la directory radice del plugin, per gli script raggruppati con un plugin.

Script del progetto

Questo esempio utilizza `$CLAUDE_PROJECT_DIR` per eseguire un controllo dello stile dalla directory `.claude/hooks/` del progetto dopo qualsiasi chiamata dello strumento `Write` o `Edit` :

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/check-style.sh"
          }
        ]
      }
    ]
  }
}
```

Script del plugin

Definire i hook del plugin in `hooks/hooks.json` con un campo `description` facoltativo di livello superiore. Quando un plugin è abilitato, i suoi hook si uniscono ai vostri hook utente e progetto.

Questo esempio esegue uno script di formattazione raggruppato con il plugin:

```
{
  "description": "Automatic code formatting",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format.sh",
            "timeout": 30
          }
        ]
      }
    ]
  }
}
```

Vedere il plugin components reference per i dettagli sulla creazione di hook del plugin.

Hook in skill e agenti

Oltre ai file di impostazioni e ai plugin, i hook possono essere definiti direttamente in skill e subagenti utilizzando il frontmatter. Questi hook sono limitati al ciclo di vita del componente e vengono eseguiti solo quando quel componente è attivo.

Tutti gli hook event sono supportati. Per i subagenti, i hook `Stop` vengono automaticamente convertiti in `SubagentStop` poiché questo è l'evento che si attiva quando un subagent termina.

Gli hook utilizzano lo stesso formato di configurazione dei hook basati su impostazioni ma sono limitati alla durata del componente e vengono puliti quando termina.

Questa skill definisce un hook `PreToolUse` che esegue uno script di convalida della sicurezza prima di ogni comando `Bash`:

```
---
name: secure-operations
description: Perform operations with security checks
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/security-check.sh"
---
```

Gli agenti utilizzano lo stesso formato nel loro frontmatter YAML.

Il menu `/hooks`

Digitare `/hooks` in Claude Code per aprire il gestore interattivo dei hook, dove potete visualizzare, aggiungere ed eliminare i hook senza modificare direttamente i file di impostazioni. Per una procedura dettagliata, vedere Set up your first hook nella guida.

Ogni hook nel menu è etichettato con un prefisso tra parentesi che indica la sua fonte:

- `[User]` : da `~/.claude/settings.json`
- `[Project]` : da `.claude/settings.json`
- `[Local]` : da `.claude/settings.local.json`
- `[Plugin]` : da `hooks/hooks.json` di un plugin, sola lettura

Disabilitare o rimuovere i hook

Per rimuovere un hook, eliminare la sua voce dal file di impostazioni JSON o utilizzare il menu `/hooks` e selezionare l'hook da eliminare.

Per disabilitare temporaneamente tutti i hook senza rimuoverli, impostare `"disableAllHooks": true` nel file di impostazioni o utilizzare l'interruttore nel menu `/hooks`. Non c'è modo di disabilitare un singolo hook mantenendolo nella configurazione.

L'impostazione `disableAllHooks` rispetta la gerarchia delle impostazioni gestite. Se un amministratore ha configurato i hook attraverso le impostazioni della politica gestita, `disableAllHooks` impostato nelle impostazioni utente, progetto o locale non può disabilitare quei hook gestiti. Solo `disableAllHooks` impostato a livello di impostazioni gestite può disabilitare i hook gestiti.

Le modifiche dirette ai hook nei file di impostazioni non hanno effetto immediato. Claude Code acquisisce uno snapshot dei hook all'avvio e lo utilizza durante tutta la sessione. Questo impedisce che le modifiche ai hook malintenzionate o accidentali abbiano effetto a metà sessione senza la vostra revisione. Se i hook vengono modificati esternamente, Claude Code vi avverte e richiede una revisione nel menu `/hooks` prima che le modifiche abbiano effetto.

Input e output dei hook

I command hook ricevono dati JSON via stdin e comunicano i risultati attraverso codici di uscita, stdout e stderr. Gli HTTP hook ricevono lo stesso JSON come corpo della richiesta POST e comunicano i risultati attraverso il corpo della risposta HTTP. Questa sezione copre i campi e il comportamento comuni a tutti gli eventi. Ogni sezione dell'evento sotto Hook events include il suo schema di input specifico e le opzioni di controllo della decisione.

Campi di input comuni

Tutti gli hook event ricevono questi campi come JSON, oltre ai campi specifici dell'evento documentati in ogni sezione hook event. Per i command hook, questo JSON arriva via stdin. Per gli HTTP hook, arriva come corpo della richiesta POST.

Campo	Descrizione
`session_id`	Identificatore della sessione corrente
`transcript_path`	Percorso al JSON della conversazione
`cwd`	Directory di lavoro corrente quando il hook viene invocato
`permission_mode`	Modalità di autorizzazione corrente: `"default"`, `"plan"`, `"acceptEdits"`, `"dontAsk"` o `"bypassPermissions"`
`hook_event_name`	Nome dell'evento che si è attivato

Quando si esegue con `--agent` o all'interno di un subagent, vengono inclusi due campi aggiuntivi:

Campo	Descrizione
`agent_id`	Identificatore univoco per il subagent. Presente solo quando il hook si attiva all'interno di una chiamata di subagent. Utilizzare questo per distinguere le chiamate del hook del subagent dalle chiamate del thread principale.

Campo	Descrizione
`agent_type`	Nome dell'agente (ad esempio, `"Explore"` o `"security-reviewer"`). Presente quando la sessione utilizza `--agent` o il hook si attiva all'interno di un subagent. Per i subagenti, il tipo del subagent ha la precedenza sul valore `--agent` della sessione.

Ad esempio, un hook `PreToolUse` per un comando Bash riceve questo su stdin:

```
{
  "session_id": "abc123",
  "transcript_path": "/home/user/.claude/projects/.../transcript.jsonl",
  "cwd": "/home/user/my-project",
  "permission_mode": "default",
  "hook_event_name": "PreToolUse",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test"
  }
}
```

I campi `tool_name` e `tool_input` sono specifici dell'evento. Ogni sezione hook event documenta i campi aggiuntivi per quell'evento.

Output del codice di uscita

Il codice di uscita dal comando del hook dice a Claude Code se l'azione deve procedere, essere bloccata o essere ignorata.

Exit 0 significa successo. Claude Code analizza stdout per i campi di output JSON. L'output JSON viene elaborato solo su exit 0. Per la maggior parte degli eventi, stdout viene mostrato solo in modalità verbose (`Ctrl+O`). Le eccezioni sono `UserPromptSubmit` e `SessionStart`, dove stdout viene aggiunto come contesto che Claude può vedere e su cui agire.

Exit 2 significa un errore bloccante. Claude Code ignora stdout e qualsiasi JSON in esso. Invece, il testo stderr viene restituito a Claude come messaggio di errore. L'effetto dipende dall'evento: `PreToolUse` blocca la chiamata dello strumento, `UserPromptSubmit` rifiuta il prompt e così via. Vedere exit code 2 behavior per l'elenco completo.

Qualsiasi altro codice di uscita è un errore non bloccante. stderr viene mostrato in modalità verbose (`Ctrl+O`) e l'esecuzione continua.

Ad esempio, uno script di comando hook che blocca i comandi Bash pericolosi:

```
#!/bin/bash
## Legge l'input JSON da stdin, controlla il comando
command=$(jq -r '.tool_input.command' < /dev/stdin)

if [[ "$command" == rm* ]]; then
  echo "Blocked: rm commands are not allowed" >&2
  exit 2  # Errore bloccante: la chiamata dello strumento viene impedita
fi

exit 0  # Successo: la chiamata dello strumento procede
```

Comportamento del codice di uscita 2 per evento

Il codice di uscita 2 è il modo in cui un hook segnala "fermarsi, non farlo". L'effetto dipende dall'evento, perché alcuni eventi rappresentano azioni che possono essere bloccate (come una chiamata dello strumento che non è ancora accaduta) e altri rappresentano cose che sono già accadute o non possono essere prevenute.

Hook event	Può bloccare?	Cosa accade su exit 2
`PreToolUse`	Sì	Blocca la chiamata dello strumento
`PermissionRequest`	Sì	Nega l'autorizzazione
`UserPromptSubmit`	Sì	Blocca l'elaborazione del prompt e cancella il prompt
`Stop`	Sì	Impedisce a Claude di fermarsi, continua la conversazione
`SubagentStop`	Sì	Impedisce al subagent di fermarsi
`TeammateIdle`	Sì	Impedisce al compagno di squadra di andare inattivo (il compagno di squadra continua a lavorare)

Hook event	Può bloccare?	Cosa accade su exit 2
`TaskCompleted`	Sì	Impedisce che l'attività sia contrassegnata come completata
`ConfigChange`	Sì	Blocca la modifica della configurazione dall'avere effetto (eccetto `policy_settings`)
`PostToolUse`	No	Mostra stderr a Claude (lo strumento è già stato eseguito)
`PostToolUseFailure`	No	Mostra stderr a Claude (lo strumento è già fallito)
`Notification`	No	Mostra stderr solo all'utente
`SubagentStart`	No	Mostra stderr solo all'utente
`SessionStart`	No	Mostra stderr solo all'utente
`SessionEnd`	No	Mostra stderr solo all'utente
`PreCompact`	No	Mostra stderr solo all'utente
`WorktreeCreate`	Sì	Qualsiasi codice di uscita diverso da zero causa il fallimento della creazione del worktree
`WorktreeRemove`	No	I guasti vengono registrati solo in modalità debug
`InstructionsLoaded`	No	Il codice di uscita viene ignorato

Gestione della risposta HTTP

Gli HTTP hook utilizzano i codici di stato HTTP e i corpi della risposta invece dei codici di uscita e stdout:

- **2xx con corpo vuoto**: successo, equivalente al codice di uscita 0 senza output
- **2xx con corpo di testo semplice**: successo, il testo viene aggiunto come contesto
- **2xx con corpo JSON**: successo, analizzato utilizzando lo stesso schema JSON output dei command hook
- **Stato non-2xx**: errore non bloccante, l'esecuzione continua
- **Guasto di connessione o timeout**: errore non bloccante, l'esecuzione continua

A differenza dei command hook, gli HTTP hook non possono segnalare un errore bloccante solo attraverso i codici di stato. Per bloccare una chiamata dello strumento o negare un'autorizzazione, restituire una risposta 2xx con un corpo JSON contenente i campi di decisione appropriati.

Output JSON

I codici di uscita consentono di consentire o bloccare, ma l'output JSON offre un controllo più granulare. Invece di uscire con il codice 2 per bloccare, uscire 0 e stampare un oggetto JSON su stdout. Claude Code legge campi specifici da quel JSON per controllare il comportamento, incluso il decision control per bloccare, consentire o escalare all'utente.

> **Note:**
>
> Dovete scegliere un approccio per hook, non entrambi: utilizzate i codici di uscita da soli per la segnalazione oppure uscite 0 e stampate JSON per il controllo strutturato. Claude Code elabora JSON solo su exit 0. Se uscite 2, qualsiasi JSON viene ignorato.

Lo stdout del vostro hook deve contenere solo l'oggetto JSON. Se il vostro profilo shell stampa testo all'avvio, può interferire con l'analisi JSON. Vedere JSON validation failed nella guida alla risoluzione dei problemi.

L'oggetto JSON supporta tre tipi di campi:

- **Campi universali** come `continue` funzionano su tutti gli eventi. Questi sono elencati nella tabella seguente.
- `decision` **e** `reason` **di livello superiore** vengono utilizzati da alcuni eventi per bloccare o fornire feedback.

- `hookSpecificOutput` è un oggetto annidato per gli eventi che necessitano di un controllo più ricco. Richiede un campo `hookEventName` impostato sul nome dell'evento.

Campo	Impostazione predefinita	Descrizione
`continue`	`true`	Se `false`, Claude interrompe completamente l'elaborazione dopo l'esecuzione del hook. Ha la precedenza su qualsiasi campo di decisione specifico dell'evento
`stopReason`	nessuno	Messaggio mostrato all'utente quando `continue` è `false`. Non mostrato a Claude
`suppressOutput`	`false`	Se `true`, nasconde stdout dall'output della modalità verbose
`systemMessage`	nessuno	Messaggio di avviso mostrato all'utente

Per fermare Claude completamente indipendentemente dal tipo di evento:

```
{ "continue": false, "stopReason": "Build failed, fix errors before continuing" }
```

Controllo della decisione

Non ogni evento supporta il blocco o il controllo del comportamento attraverso JSON. Gli eventi che lo fanno utilizzano ciascuno un insieme diverso di campi per esprimere quella decisione. Utilizzare questa tabella come riferimento rapido prima di scrivere un hook:

Eventi	Modello di decisione	Campi chiave
UserPromptSubmit, PostToolUse, PostToolUseFailure, Stop, SubagentStop, ConfigChange	`decision` di livello superiore	`decision: "block"`, `reason`

Eventi	Modello di decisione	Campi chiave
TeammateIdle, TaskCompleted	Codice di uscita o `continue: false`	Il codice di uscita 2 blocca l'azione con feedback stderr. JSON `{"continue": false, "stopReason": "..."}` interrompe anche completamente il compagno di squadra, corrispondendo al comportamento del hook `Stop`
PreToolUse	`hookSpecificOutput`	`permissionDecision` (allow/deny/ask), `permissionDecisionReason`
PermissionRequest	`hookSpecificOutput`	`decision.behavior` (allow/deny)
WorktreeCreate	percorso stdout	L'hook stampa il percorso assoluto del worktree creato. L'uscita diversa da zero non riesce nella creazione
WorktreeRemove, Notification, SessionEnd, PreCompact, InstructionsLoaded	Nessuno	Nessun controllo della decisione. Utilizzato per effetti collaterali come la registrazione o la pulizia

Ecco esempi di ogni modello in azione:

Decisione di livello superiore

Utilizzato da `UserPromptSubmit`, `PostToolUse`, `PostToolUseFailure`, `Stop`, `SubagentStop` e `ConfigChange`. L'unico valore è `"block"`. Per consentire all'azione di procedere, omettere `decision` dal vostro JSON o uscire 0 senza alcun JSON:

```
{
  "decision": "block",
  "reason": "Test suite must pass before proceeding"
}
```

PreToolUse

Utilizza `hookSpecificOutput` per un controllo più ricco: consentire, negare o escalare all'utente. Potete anche modificare l'input dello strumento prima che venga eseguito o iniettare contesto aggiuntivo per Claude. Vedere PreToolUse decision control per l'insieme completo di opzioni.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Database writes are not allowed"
  }
}
```

PermissionRequest

Utilizza `hookSpecificOutput` per consentire o negare una richiesta di autorizzazione per conto dell'utente. Quando si consente, potete anche modificare l'input dello strumento o applicare regole di autorizzazione in modo che l'utente non venga richiesto di nuovo. Vedere PermissionRequest decision control per l'insieme completo di opzioni.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

Per esempi estesi inclusa la convalida dei comandi Bash, il filtraggio dei prompt e gli script di approvazione automatica, vedere What you can automate nella guida e l'implementazione di riferimento del validatore di comandi Bash.

Hook events

Ogni evento corrisponde a un punto nel ciclo di vita di Claude Code in cui i hook possono essere eseguiti. Le sezioni seguenti sono ordinate per corrispondere al ciclo di vita: dalla configurazione della sessione attraverso il ciclo agentico alla fine della sessione. Ogni sezione descrive quando l'evento si attiva, quali matcher supporta, l'input JSON che riceve e come controllare il comportamento attraverso l'output.

SessionStart

Viene eseguito quando Claude Code avvia una nuova sessione o riprende una sessione esistente. Utile per caricare il contesto di sviluppo come problemi esistenti o modifiche recenti al vostro codebase, o per configurare le variabili di ambiente. Per il contesto statico che non richiede uno script, utilizzare CLAUDE.md invece.

SessionStart viene eseguito su ogni sessione, quindi mantenete questi hook veloci. Solo i hook `type: "command"` sono supportati.

Il valore del matcher corrisponde a come è stata avviata la sessione:

Matcher	Quando si attiva
`startup`	Nuova sessione
`resume`	`--resume`, `--continue` o `/resume`
`clear`	`/clear`
`compact`	Compattazione automatica o manuale

Input di SessionStart

Oltre ai campi di input comuni, i hook SessionStart ricevono `source`, `model` e facoltativamente `agent_type`. Il campo `source` indica come è iniziata la sessione: `"startup"` per le nuove sessioni, `"resume"` per le sessioni riprese, `"clear"` dopo `/clear` o `"compact"` dopo la compattazione. Il campo `model` contiene l'identificatore del modello. Se avviate Claude Code con `claude --agent <name>`, un campo `agent_type` contiene il nome dell'agente.

```
{
  "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"
}
```

Controllo della decisione di SessionStart

Qualsiasi testo che il vostro script hook stampa su stdout viene aggiunto come contesto per Claude. Oltre ai campi di output JSON disponibili per tutti i hook, potete restituire questi campi specifici dell'evento:

Campo	Descrizione
`additionalContext`	Stringa aggiunta al contesto di Claude. I valori di più hook vengono concatenati

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "My additional context here"
  }
}
```

Persistere le variabili di ambiente

I hook SessionStart hanno accesso alla variabile di ambiente `CLAUDE_ENV_FILE`, che fornisce un percorso di file in cui potete persistere le variabili di ambiente per i successivi comandi Bash.

Per impostare le singole variabili di ambiente, scrivere le istruzioni `export` su `CLAUDE_ENV_FILE`. Utilizzare l'append (`>>`) per preservare le variabili impostate da altri hook:

```
#!/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
```

Per acquisire tutte le modifiche all'ambiente dai comandi di configurazione, confrontare le variabili esportate prima e dopo:

```
#!/bin/bash

ENV_BEFORE=$(export -p | sort)

## Eseguire i comandi di configurazione che modificano l'ambiente
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
```

Qualsiasi variabile scritta in questo file sarà disponibile in tutti i successivi comandi Bash che Claude Code esegue durante la sessione.

> **Note:**
>
> `CLAUDE_ENV_FILE` è disponibile per i hook SessionStart. Gli altri tipi di hook non hanno accesso a questa variabile.

InstructionsLoaded

Si attiva quando un file `CLAUDE.md` o `.claude/rules/*.md` viene caricato nel contesto. Questo evento si attiva all'avvio della sessione per i file caricati con entusiasmo e di nuovo in seguito quando i file vengono caricati in modo pigro, ad esempio quando Claude accede a una sottodirectory che contiene un `CLAUDE.md` annidato o quando le regole condizionali con frontmatter `paths:` corrispondono. L'hook non supporta il blocco o il controllo della decisione. Viene eseguito in modo asincrono per scopi di osservabilità.

InstructionsLoaded non supporta i matcher e si attiva su ogni occorrenza di caricamento.

Input di InstructionsLoaded

Oltre ai campi di input comuni, i hook InstructionsLoaded ricevono questi campi:

Campo	Descrizione
`file_path`	Percorso assoluto al file di istruzioni che è stato caricato
`memory_type`	Ambito del file: `"User"`, `"Project"`, `"Local"` o `"Managed"`
`load_reason`	Perché il file è stato caricato: `"session_start"`, `"nested_traversal"`, `"path_glob_match"` o `"include"`
`globs`	Modelli glob del percorso dal frontmatter `paths:` del file, se presenti. Presente solo per i caricamenti `path_glob_match`
`trigger_file_path`	Percorso al file il cui accesso ha attivato questo caricamento, per i caricamenti pigri
`parent_file_path`	Percorso al file di istruzioni padre che ha incluso questo, per i caricamenti `include`

```
{
  "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"
}
```

Controllo della decisione di InstructionsLoaded

I hook InstructionsLoaded non hanno controllo della decisione. Non possono bloccare o modificare il caricamento delle istruzioni. Utilizzare questo evento per la registrazione di audit, il tracciamento della conformità o l'osservabilità.

UserPromptSubmit

Viene eseguito quando l'utente invia un prompt, prima che Claude lo elabori. Questo consente di aggiungere contesto aggiuntivo in base al prompt/conversazione, convalidare i prompt o bloccare determinati tipi di prompt.

Input di UserPromptSubmit

Oltre ai campi di input comuni, i hook UserPromptSubmit ricevono il campo `prompt` contenente il testo che l'utente ha inviato.

```
{
  "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"
}
```

Controllo della decisione di UserPromptSubmit

I hook `UserPromptSubmit` possono controllare se un prompt dell'utente viene elaborato e aggiungere contesto. Tutti i campi di output JSON sono disponibili.

Ci sono due modi per aggiungere contesto alla conversazione su exit code 0:

- **Stdout di testo semplice**: qualsiasi testo non-JSON scritto su stdout viene aggiunto come contesto
- **JSON con `additionalContext`**: utilizzare il formato JSON seguente per un controllo più granulare. Il campo `additionalContext` viene aggiunto come contesto

Lo stdout semplice viene mostrato come output del hook nella trascrizione. Il campo `additionalContext` viene aggiunto più discretamente.

Per bloccare un prompt, restituire un oggetto JSON con `decision` impostato su `"block"`:

Campo	Descrizione
`decision`	`"block"` impedisce l'elaborazione del prompt e lo cancella dal contesto. Omettere per consentire al prompt di procedere
`reason`	Mostrato all'utente quando `decision` è `"block"`. Non aggiunto al contesto
`additionalContext`	Stringa aggiunta al contesto di Claude

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "My additional context here"
  }
}
```

> **Note:**
>
> Il formato JSON non è obbligatorio per i casi semplici. Per aggiungere contesto, potete stampare testo semplice su stdout con exit code 0. Utilizzare JSON quando dovete bloccare i prompt o desiderate un controllo più strutturato.

PreToolUse

Viene eseguito dopo che Claude crea i parametri dello strumento e prima dell'elaborazione della chiamata dello strumento. Corrisponde al nome dello strumento: `Bash`, `Edit`, `Write`, `Read`, `Glob`, `Grep`, `Agent`, `WebFetch`, `WebSearch` e qualsiasi nome di strumento MCP.

Utilizzare il PreToolUse decision control per consentire, negare o chiedere l'autorizzazione per utilizzare lo strumento.

Input di PreToolUse

Oltre ai campi di input comuni, i hook PreToolUse ricevono `tool_name`, `tool_input` e `tool_use_id`. I campi `tool_input` dipendono dallo strumento:

Bash

Esegue comandi shell.

Campo	Tipo	Esempio	Descrizione
`command`	string	`"npm test"`	Il comando shell da eseguire
`description`	string	`"Run test suite"`	Descrizione facoltativa di cosa fa il comando
`timeout`	number	`120000`	Timeout facoltativo in millisecondi
`run_in_background`	boolean	`false`	Se eseguire il comando in background

Write

Crea o sovrascrive un file.

Campo	Tipo	Esempio	Descrizione
`file_path`	string	`"/path/to/file.txt"`	Percorso assoluto al file da scrivere
`content`	string	`"file content"`	Contenuto da scrivere nel file

Edit

Sostituisce una stringa in un file esistente.

Campo	Tipo	Esempio	Descrizione
`file_path`	string	`"/path/to/file.txt"`	Percorso assoluto al file da modificare
`old_string`	string	`"original text"`	Testo da trovare e sostituire
`new_string`	string	`"replacement text"`	Testo di sostituzione
`replace_all`	boolean	`false`	Se sostituire tutte le occorrenze

Read

Legge il contenuto del file.

Campo	Tipo	Esempio	Descrizione
`file_path`	string	`"/path/to/file.txt"`	Percorso assoluto al file da leggere
`offset`	number	`10`	Numero di riga facoltativo da cui iniziare la lettura
`limit`	number	`50`	Numero facoltativo di righe da leggere

Glob

Trova file che corrispondono a un modello glob.

Campo	Tipo	Esempio	Descrizione
`pattern`	string	`"**/*.ts"`	Modello glob per corrispondere ai file
`path`	string	`"/path/to/dir"`	Directory facoltativa in cui cercare. Impostazione predefinita sulla directory di lavoro corrente

Grep

Cerca il contenuto dei file con espressioni regolari.

Campo	Tipo	Esempio	Descrizione
`pattern`	string	`"TODO.*fix"`	Modello di espressione regolare da cercare
`path`	string	`"/path/to/dir"`	File o directory facoltativa in cui cercare
`glob`	string	`"*.ts"`	Modello glob facoltativo per filtrare i file

Campo	Tipo	Esempio	Descrizione
`output_mode`	string	`"content"`	`"content"`, `"files_with_matches"` o `"count"`. Impostazione predefinita su `"files_with_matches"`
`-i`	boolean	`true`	Ricerca senza distinzione tra maiuscole e minuscole
`multiline`	boolean	`false`	Abilita la corrispondenza multilinea

WebFetch

Recupera ed elabora il contenuto web.

Campo	Tipo	Esempio	Descrizione
`url`	string	`"https://example.com/api"`	URL da cui recuperare il contenuto
`prompt`	string	`"Extract the API endpoints"`	Prompt da eseguire sul contenuto recuperato

WebSearch

Cerca il web.

Campo	Tipo	Esempio	Descrizione
`query`	string	`"react hooks best practices"`	Query di ricerca
`allowed_domains`	array	`["docs.example.com"]`	Facoltativo: includere solo i risultati da questi domini
`blocked_domains`	array	`["spam.example.com"]`	Facoltativo: escludere i risultati da questi domini

Agent

Genera un subagent.

Campo	Tipo	Esempio	Descrizione
`prompt`	string	`"Find all API endpoints"`	L'attività per l'agente da eseguire
`description`	string	`"Find API endpoints"`	Breve descrizione dell'attività
`subagent_type`	string	`"Explore"`	Tipo di agente specializzato da utilizzare
`model`	string	`"sonnet"`	Alias del modello facoltativo per sovrascrivere l'impostazione predefinita

Controllo della decisione di PreToolUse

I hook `PreToolUse` possono controllare se una chiamata dello strumento procede. A differenza di altri hook che utilizzano un campo `decision` di livello superiore, PreToolUse restituisce la sua decisione all'interno di un oggetto `hookSpecificOutput`. Questo offre un controllo più ricco: tre risultati (consentire, negare o chiedere) più la capacità di modificare l'input dello strumento prima dell'esecuzione.

Campo	Descrizione
`permissionDecision`	`"allow"` bypassa il sistema di autorizzazione, `"deny"` impedisce la chiamata dello strumento, `"ask"` chiede all'utente di confermare
`permissionDecisionReason`	Per `"allow"` e `"ask"`, mostrato all'utente ma non a Claude. Per `"deny"`, mostrato a Claude
`updatedInput`	Modifica i parametri di input dello strumento prima dell'esecuzione. Combinare con `"allow"` per l'approvazione automatica o `"ask"` per mostrare l'input modificato all'utente
`additionalContext`	Stringa aggiunta al contesto di Claude prima dell'esecuzione dello strumento

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow",
    "permissionDecisionReason": "My reason here",
    "updatedInput": {
      "field_to_modify": "new value"
    },
    "additionalContext": "Current environment: production. Proceed with caution."
  }
}
```

Note:

PreToolUse in precedenza utilizzava i campi `decision` e `reason` di livello superiore, ma questi sono deprecati per questo evento. Utilizzare `hookSpecificOutput.permissionDecision` e `hookSpecificOutput.permissionDecisionReason`. I valori deprecati `"approve"` e `"block"` si mappano a `"allow"` e `"deny"` rispettivamente. Gli altri eventi come PostToolUse e Stop continuano a utilizzare `decision` e `reason` di livello superiore come formato corrente.

PermissionRequest

Viene eseguito quando all'utente viene mostrata una finestra di dialogo di autorizzazione. Utilizzare il PermissionRequest decision control per consentire o negare per conto dell'utente.

Corrisponde al nome dello strumento, stessi valori di PreToolUse.

Input di PermissionRequest

I hook PermissionRequest ricevono i campi `tool_name` e `tool_input` come i hook PreToolUse, ma senza `tool_use_id`. Un array facoltativo `permission_suggestions` contiene le opzioni "consenti sempre" che l'utente normalmente vedrebbe nella finestra di dialogo di autorizzazione. La differenza è quando il hook si attiva: i hook PermissionRequest vengono eseguiti quando una finestra di dialogo di autorizzazione sta per essere mostrata all'utente, mentre i hook PreToolUse vengono eseguiti prima dell'esecuzione dello strumento indipendentemente dallo stato di autorizzazione.

```
{
  "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" }
  ]
}
```

Controllo della decisione di PermissionRequest

I hook `PermissionRequest` possono consentire o negare le richieste di autorizzazione. Oltre ai campi di output JSON disponibili per tutti i hook, il vostro script hook può restituire un oggetto `decision` con questi campi specifici dell'evento:

Campo	Descrizione
`behavior`	`"allow"` concede l'autorizzazione, `"deny"` la nega
`updatedInput`	Solo per `"allow"` : modifica i parametri di input dello strumento prima dell'esecuzione
`updatedPermissions`	Solo per `"allow"` : applica gli aggiornamenti delle regole di autorizzazione, equivalente all'utente che seleziona un'opzione "consenti sempre"
`message`	Solo per `"deny"` : dice a Claude perché l'autorizzazione è stata negata
`interrupt`	Solo per `"deny"` : se `true` , interrompe Claude

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

PostToolUse

Viene eseguito immediatamente dopo il completamento riuscito di uno strumento.

Corrisponde al nome dello strumento, stessi valori di PreToolUse.

Input di PostToolUse

I hook `PostToolUse` si attivano dopo che uno strumento è già stato eseguito con successo. L'input include sia `tool_input`, gli argomenti inviati allo strumento, che `tool_response`, il risultato che ha restituito. Lo schema esatto per entrambi dipende dallo strumento.

```
{
  "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..."
}
```

Controllo della decisione di PostToolUse

I hook `PostToolUse` possono fornire feedback a Claude dopo l'esecuzione dello strumento. Oltre ai campi di output JSON disponibili per tutti i hook, il vostro script hook può restituire questi campi specifici dell'evento:

Campo	Descrizione
`decision`	`"block"` chiede a Claude con il `reason`. Omettere per consentire all'azione di procedere
`reason`	Spiegazione mostrata a Claude quando `decision` è `"block"`
`additionalContext`	Contesto aggiuntivo per Claude da considerare
`updatedMCPToolOutput`	Solo per strumenti MCP: sostituisce l'output dello strumento con il valore fornito

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "additionalContext": "Additional information for Claude"
  }
}
```

PostToolUseFailure

Viene eseguito quando l'esecuzione di uno strumento non riesce. Questo evento si attiva per le chiamate dello strumento che generano errori o restituiscono risultati di fallimento. Utilizzare questo per registrare i fallimenti, inviare avvisi o fornire feedback correttivo a Claude.

Corrisponde al nome dello strumento, stessi valori di PreToolUse.

Input di PostToolUseFailure

I hook PostToolUseFailure ricevono gli stessi campi `tool_name` e `tool_input` di PostToolUse, insieme alle informazioni di errore come campi di livello superiore:

```
{
  "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
}
```

Campo	Descrizione
`error`	Stringa che descrive cosa è andato male
`is_interrupt`	Booleano facoltativo che indica se il fallimento è stato causato dall'interruzione dell'utente

Controllo della decisione di PostToolUseFailure

I hook `PostToolUseFailure` possono fornire contesto a Claude dopo un fallimento dello strumento. Oltre ai campi di output JSON disponibili per tutti i hook, il vostro script hook può restituire questi campi specifici dell'evento:

Campo	Descrizione
`additionalContext`	Contesto aggiuntivo per Claude da considerare insieme all'errore

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUseFailure",
    "additionalContext": "Additional information about the failure for Claude"
  }
}
```

Notification

Viene eseguito quando Claude Code invia notifiche. Corrisponde al tipo di notifica: `permission_prompt`, `idle_prompt`, `auth_success`, `elicitation_dialog`. Omettere il matcher per eseguire i hook per tutti i tipi di notifica.

Utilizzare matcher separati per eseguire diversi gestori a seconda del tipo di notifica. Questa configurazione attiva uno script di avviso specifico per l'autorizzazione quando Claude ha bisogno dell'approvazione dell'autorizzazione e una notifica diversa quando Claude è stato inattivo:

```
{
  "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"
          }
        ]
      }
    ]
  }
}
```

Input di Notification

Oltre ai campi di input comuni, i hook Notification ricevono `message` con il testo della notifica, un `title` facoltativo e `notification_type` che indica quale tipo si è attivato.

```
{
  "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"
}
```

I hook Notification non possono bloccare o modificare le notifiche. Oltre ai campi di output JSON disponibili per tutti i hook, potete restituire `additionalContext` per aggiungere contesto alla conversazione:

Campo	Descrizione
`additionalContext`	Stringa aggiunta al contesto di Claude

SubagentStart

Viene eseguito quando un subagent di Claude Code viene generato tramite lo strumento Agent. Supporta i matcher per filtrare per nome del tipo di agente (agenti incorporati come `Bash`, `Explore`, `Plan` o nomi di agenti personalizzati da `.claude/agents/`).

Input di SubagentStart

Oltre ai campi di input comuni, i hook SubagentStart ricevono `agent_id` con l'identificatore univoco per il subagent e `agent_type` con il nome dell'agente (agenti incorporati come `"Bash"`, `"Explore"`, `"Plan"` o nomi di agenti personalizzati).

```
{
  "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"
}
```

I hook SubagentStart non possono bloccare la creazione del subagent, ma possono iniettare contesto nel subagent. Oltre ai campi di output JSON disponibili per tutti i hook, potete restituire:

Campo	Descrizione
`additionalContext`	Stringa aggiunta al contesto del subagent

```
{
  "hookSpecificOutput": {
    "hookEventName": "SubagentStart",
    "additionalContext": "Follow security guidelines for this task"
  }
}
```

SubagentStop

Viene eseguito quando un subagent di Claude Code ha finito di rispondere. Corrisponde al tipo di agente, stessi valori di SubagentStart.

Input di SubagentStop

Oltre ai campi di input comuni, i hook SubagentStop ricevono `stop_hook_active`, `agent_id`, `agent_type`, `agent_transcript_path` e `last_assistant_message`. Il campo `agent_type` è il valore utilizzato per il filtraggio del matcher. Il `transcript_path` è la trascrizione della sessione principale, mentre `agent_transcript_path` è la trascrizione pro-

pria del subagent memorizzata in una cartella `subagents/` annidato. Il campo `last_assistant_message` contiene il contenuto del testo della risposta finale del subagent, quindi i hook possono accedervi senza analizzare il file della trascrizione.

```
{
  "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..."
}
```

I hook SubagentStop utilizzano lo stesso formato di controllo della decisione dei hook Stop.

Stop

Viene eseguito quando l'agente principale di Claude Code ha finito di rispondere. Non viene eseguito se l'arresto si è verificato a causa di un'interruzione dell'utente.

Input di Stop

Oltre ai campi di input comuni, i hook Stop ricevono `stop_hook_active` e `last_assistant_message`. Il campo `stop_hook_active` è `true` quando Claude Code sta già continuando a causa di un hook di arresto. Controllare questo valore o elaborare la trascrizione per impedire a Claude Code di eseguire indefinitamente. Il campo `last_assistant_message` contiene il contenuto del testo della risposta finale di Claude, quindi i hook possono accedervi senza analizzare il file della trascrizione.

```
{
  "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..."
}
```

Controllo della decisione di Stop

I hook `Stop` e `SubagentStop` possono controllare se Claude continua. Oltre ai campi di output JSON disponibili per tutti i hook, il vostro script hook può restituire questi campi specifici dell'evento:

Campo	Descrizione
`decision`	`"block"` impedisce a Claude di fermarsi. Omettere per consentire a Claude di fermarsi
`reason`	Obbligatorio quando `decision` è `"block"`. Dice a Claude perché dovrebbe continuare

```
{
  "decision": "block",
  "reason": "Must be provided when Claude is blocked from stopping"
}
```

TeammateIdle

Viene eseguito quando un compagno di squadra di un agent team sta per andare inattivo dopo aver finito il suo turno. Utilizzare questo per applicare gate di qualità prima che un compagno di squadra smetta di lavorare, come richiedere il passaggio dei controlli di linting o verificare che i file di output esistano.

Quando un hook `TeammateIdle` esce con il codice 2, il compagno di squadra riceve il messaggio stderr come feedback e continua a lavorare invece di andare inattivo. Per interrompere completamente il compagno di squadra invece di rieseguirlo, restituire JSON con `{"continue": false, "stopReason": "..."}`. I hook TeammateIdle non supportano i matcher e si attivano su ogni occorrenza.

Input di TeammateIdle

Oltre ai campi di input comuni, i hook TeammateIdle ricevono `teammate_name` e `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"
}
```

Campo	Descrizione
`teammate_name`	Nome del compagno di squadra che sta per andare inattivo
`team_name`	Nome del team

Controllo della decisione di TeammateIdle

I hook TeammateIdle supportano due modi per controllare il comportamento del compagno di squadra:

- **Exit code 2**: il compagno di squadra riceve il messaggio stderr come feedback e continua a lavorare invece di andare inattivo.
- **JSON** `{"continue": false, "stopReason": "..."}`: interrompe completamente il compagno di squadra, corrispondendo al comportamento del hook `Stop`. Il `stopReason` viene mostrato all'utente.

Questo esempio controlla che un artefatto di build esista prima di consentire a un compagno di squadra di andare inattivo:

```
#!/bin/bash

if [ ! -f "./dist/output.js" ]; then
  echo "Build artifact missing. Run the build before stopping." >&2
  exit 2
fi

exit 0
```

TaskCompleted

Viene eseguito quando un'attività sta per essere contrassegnata come completata. Questo si attiva in due situazioni: quando qualsiasi agente contrassegna esplicitamente un'attività come completata attraverso lo strumento TaskUpdate, o quando un compagno di squadra di un agent team finisce il suo turno con attività in corso. Utilizzare questo per applicare criteri di completamento come il passaggio dei test o dei controlli di linting prima che un'attività possa chiudersi.

Quando un hook `TaskCompleted` esce con il codice 2, l'attività non viene contrassegnata come completata e il messaggio stderr viene restituito al modello come feedback. Per interrompere completamente il compagno di squadra invece di rieseguirlo, restituire JSON con `{"continue": false, "stopReason": "..."}`. I hook TaskCompleted non supportano i matcher e si attivano su ogni occorrenza.

Input di TaskCompleted

Oltre ai campi di input comuni, i hook TaskCompleted ricevono `task_id`, `task_subject` e facoltativamente `task_description`, `teammate_name` e `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"
}
```

Campo	Descrizione
`task_id`	Identificatore dell'attività in corso di completamento
`task_subject`	Titolo dell'attività
`task_description`	Descrizione dettagliata dell'attività. Può essere assente
`teammate_name`	Nome del compagno di squadra che completa l'attività. Può essere assente
`team_name`	Nome del team. Può essere assente

Controllo della decisione di TaskCompleted

I hook TaskCompleted supportano due modi per controllare il completamento dell'attività:

- **Exit code 2**: l'attività non viene contrassegnata come completata e il messaggio stderr viene restituito al modello come feedback.
- **JSON** `{"continue": false, "stopReason": "..."}` : interrompe completamente il compagno di squadra, corrispondendo al comportamento del hook `Stop` . Il `stopReason` viene mostrato all'utente.

Questo esempio esegue i test e blocca il completamento dell'attività se non riescono:

```
#!/bin/bash
INPUT=$(cat)
TASK_SUBJECT=$(echo "$INPUT" | jq -r '.task_subject')

## Eseguire la suite di test
if ! npm test 2>&1; then
  echo "Tests not passing. Fix failing tests before completing: $TASK_SUBJECT" >&2
  exit 2
fi

exit 0
```

ConfigChange

Viene eseguito quando un file di configurazione cambia durante una sessione. Utilizzare questo per controllare le modifiche alle impostazioni, applicare le politiche di sicurezza o bloccare le modifiche non autorizzate ai file di configurazione.

I hook ConfigChange si attivano per le modifiche ai file di impostazioni, alle impostazioni della politica gestita e ai file di skill. Il campo `source` nell'input vi dice quale tipo di configurazione è cambiato e il campo facoltativo `file_path` fornisce il percorso al file modificato.

Il matcher filtra sulla fonte di configurazione:

Matcher	Quando si attiva
`user_settings`	`~/.claude/settings.json` cambia
`project_settings`	`.claude/settings.json` cambia
`local_settings`	`.claude/settings.local.json` cambia
`policy_settings`	Le impostazioni della politica gestita cambiano
`skills`	Un file di skill in `.claude/skills/` cambia

Questo esempio registra tutte le modifiche di configurazione per il controllo di sicurezza:

```
{
  "hooks": {
    "ConfigChange": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/audit-config-
change.sh"
          }
        ]
      }
    ]
  }
}
```

Input di ConfigChange

Oltre ai campi di input comuni, i hook ConfigChange ricevono `source` e facoltativamente `file_path`. Il campo `source` indica quale tipo di configurazione è cambiato e `file_path` fornisce il percorso al file specifico che è stato modificato.

```
{
  "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"
}
```

Controllo della decisione di ConfigChange

I hook ConfigChange possono bloccare le modifiche di configurazione dall'avere effetto. Utilizzare il codice di uscita 2 o un JSON `decision` per impedire il cambiamento. Quando bloccato, le nuove impostazioni non vengono applicate alla sessione in esecuzione.

Campo	Descrizione
`decision`	`"block"` impedisce l'applicazione della modifica di configurazione. Omettere per consentire il cambiamento
`reason`	Spiegazione mostrata all'utente quando `decision` è `"block"`

```
{
  "decision": "block",
  "reason": "Configuration changes to project settings require admin approval"
}
```

Le modifiche a `policy_settings` non possono essere bloccate. I hook si attivano ancora per le fonti `policy_settings`, quindi potete utilizzarli per la registrazione di audit, ma qualsiasi decisione di blocco viene ignorata. Questo garantisce che le impostazioni gestite dall'azienda abbiano sempre effetto.

WorktreeCreate

Quando eseguite `claude --worktree` o un subagent utilizza `isolation: "worktree"`, Claude Code crea una copia di lavoro isolata utilizzando `git worktree`. Se configurate un hook WorktreeCreate, sostituisce il comportamento git predefinito, consentendovi di utilizzare un sistema di controllo della versione diverso come SVN, Perforce o Mercurial.

L'hook deve stampare il percorso assoluto della directory del worktree creato su stdout. Claude Code utilizza questo percorso come directory di lavoro per la sessione isolata.

Questo esempio crea una copia di lavoro SVN e stampa il percorso per Claude Code da utilizzare. Sostituire l'URL del repository con il vostro:

```
{
  "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\"'"
          }
        ]
      }
    ]
  }
}
```

L'hook legge il `name` del worktree dall'input JSON su stdin, controlla una copia fresca in una nuova directory e stampa il percorso della directory. L'`echo` sull'ultima riga è quello che Claude Code legge come percorso del worktree. Reindirizzare qualsiasi altro output a stderr in modo che non interferisca con il percorso.

Input di WorktreeCreate

Oltre ai campi di input comuni, i hook WorktreeCreate ricevono il campo `name`. Questo è un identificatore slug per il nuovo worktree, specificato dall'utente o generato automaticamente (ad esempio, `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"
}
```

Output di WorktreeCreate

L'hook deve stampare il percorso assoluto della directory del worktree creato su stdout. Se l'hook non riesce o non produce output, la creazione del worktree non riesce con un errore.

I hook WorktreeCreate non utilizzano il modello di decisione di blocco/consentimento standard. Invece, il successo o il fallimento dell'hook determina il risultato. Solo i hook `type: "command"` sono supportati.

WorktreeRemove

La controparte di pulizia di WorktreeCreate. Questo hook si attiva quando un worktree viene rimosso, sia quando uscite da una sessione `--worktree` e scegliete di rimuoverla, sia quando un subagent con `isolation: "worktree"` termina. Per i worktree basati su git, Claude gestisce la pulizia automaticamente con `git worktree remove`. Se avete configurato un hook WorktreeCreate per un sistema di controllo della versione non-git, abbinarlo con un hook WorktreeRemove per gestire la pulizia. Senza uno, la directory del worktree rimane su disco.

Claude Code passa il percorso che WorktreeCreate ha stampato su stdout come `worktree_path` nell'input del hook. Questo esempio legge quel percorso e rimuove la directory:

```
{
  "hooks": {
    "WorktreeRemove": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'jq -r .worktree_path | xargs rm -rf'"
          }
        ]
      }
    ]
  }
}
```

Input di WorktreeRemove

Oltre ai campi di input comuni, i hook WorktreeRemove ricevono il campo `worktree_path`, che è il percorso assoluto al worktree in corso di rimozione.

```
{
  "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"
}
```

I hook WorktreeRemove non hanno controllo della decisione. Non possono bloccare la rimozione del worktree ma possono eseguire attività di pulizia come la rimozione dello stato del controllo della versione o l'archiviazione delle modifiche. I fallimenti del hook vengono registrati solo in modalità debug. Solo i hook `type: "command"` sono supportati.

PreCompact

Viene eseguito prima che Claude Code stia per eseguire un'operazione di compattazione.

Il valore del matcher indica se la compattazione è stata attivata manualmente o automaticamente:

Matcher	Quando si attiva
`manual`	`/compact`
`auto`	Compattazione automatica quando la finestra di contesto è piena

Input di PreCompact

Oltre ai campi di input comuni, i hook PreCompact ricevono `trigger` e `custom_instructions`. Per `manual`, `custom_instructions` contiene quello che l'utente passa in `/compact`. Per `auto`, `custom_instructions` è vuoto.

```
{
  "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

Viene eseguito quando una sessione di Claude Code termina. Utile per le attività di pulizia, la registrazione delle statistiche della sessione o il salvataggio dello stato della sessione. Supporta i matcher per filtrare per motivo di uscita.

Il campo `reason` nell'input del hook indica perché la sessione è terminata:

Motivo	Descrizione
`clear`	Sessione cancellata con il comando `/clear`
`logout`	L'utente ha effettuato il logout
`prompt_input_exit`	L'utente è uscito mentre l'input del prompt era visibile
`bypass_permissions_disabled`	La modalità bypass delle autorizzazioni è stata disabilitata
`other`	Altri motivi di uscita

Input di SessionEnd

Oltre ai campi di input comuni, i hook SessionEnd ricevono un campo `reason` che indica perché la sessione è terminata. Vedere la tabella dei motivi sopra per tutti i valori.

```
{
  "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"
}
```

I hook SessionEnd non hanno controllo della decisione. Non possono bloccare la terminazione della sessione ma possono eseguire attività di pulizia.

Hook basati su prompt

Oltre ai command hook e agli HTTP hook, Claude Code supporta i hook basati su prompt (`type: "prompt"`) che utilizzano un LLM per valutare se consentire o bloccare un'azione, e gli hook basati su agenti (`type: "agent"`) che generano un verificatore agentico con accesso agli strumenti. Non tutti gli eventi supportano ogni tipo di hook.

Gli eventi che supportano tutti e quattro i tipi di hook (`command` , `http` , `prompt` e `agent`):

- `PermissionRequest`
- `PostToolUse`
- `PostToolUseFailure`
- `PreToolUse`
- `Stop`
- `SubagentStop`
- `TaskCompleted`
- `UserPromptSubmit`

Gli eventi che supportano solo i hook `type: "command"` :

- `ConfigChange`
- `InstructionsLoaded`
- `Notification`
- `PreCompact`
- `SessionEnd`

- `SessionStart`
- `SubagentStart`
- `TeammateIdle`
- `WorktreeCreate`
- `WorktreeRemove`

Come funzionano i hook basati su prompt

Invece di eseguire un comando Bash, i hook basati su prompt:

1. Inviano l'input del hook e il vostro prompt a un modello Claude, Haiku per impostazione predefinita
2. L'LLM risponde con JSON strutturato contenente una decisione
3. Claude Code elabora automaticamente la decisione

Configurazione del prompt hook

Impostare `type` su `"prompt"` e fornire una stringa `prompt` invece di un `command`. Utilizzare il segnaposto `$ARGUMENTS` per iniettare i dati di input JSON del hook nel testo del prompt. Claude Code invia il prompt combinato e l'input a un modello Claude veloce, che restituisce una decisione JSON.

Questo hook `Stop` chiede all'LLM di valutare se tutti i compiti sono completi prima di consentire a Claude di terminare:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if Claude should stop: $ARGUMENTS. Check if all
tasks are complete."
          }
        ]
      }
    ]
  }
}
```

Campo	Obbligatorio	Descrizione
`type`	sì	Deve essere `"prompt"`
`prompt`	sì	Il testo del prompt da inviare all'LLM. Utilizzare `$ARGUMENTS` come segnaposto per l'input JSON del hook. Se `$ARGUMENTS` non è presente, l'input JSON viene aggiunto al prompt
`model`	no	Modello da utilizzare per la valutazione. Impostazione predefinita su un modello veloce
`timeout`	no	Timeout in secondi. Impostazione predefinita: 30

Schema di risposta

L'LLM deve rispondere con JSON contenente:

```
{
  "ok": true | false,
  "reason": "Explanation for the decision"
}
```

Campo	Descrizione
`ok`	`true` consente l'azione, `false` la impedisce
`reason`	Obbligatorio quando `ok` è `false`. Spiegazione mostrata a Claude

Esempio: Hook Stop multi-criterio

Questo hook `Stop` utilizza un prompt dettagliato per controllare tre condizioni prima di consentire a Claude di fermarsi. Se `"ok"` è `false`, Claude continua a lavorare con la spiegazione fornita come sua prossima istruzione. I hook `SubagentStop` utilizzano lo stesso formato per valutare se un subagent dovrebbe fermarsi:

```
{
  "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
          }
        ]
      }
    ]
  }
}
```

Hook basati su agenti

Gli hook basati su agenti (`type: "agent"`) sono come i hook basati su prompt ma con accesso agli strumenti multi-turno. Invece di una singola chiamata LLM, un hook agente genera un subagent che può leggere file, cercare codice e ispezionare il codebase per verificare le condizioni. Gli hook agente supportano gli stessi eventi dei hook basati su prompt.

Come funzionano gli hook basati su agenti

Quando un hook agente si attiva:

1. Claude Code genera un subagent con il vostro prompt e l'input JSON del hook
2. Il subagent può utilizzare strumenti come Read, Grep e Glob per investigare
3. Dopo fino a 50 turni, il subagent restituisce una decisione strutturata `{ "ok": true/false }`
4. Claude Code elabora la decisione allo stesso modo di un hook di prompt

Gli hook agente sono utili quando la verifica richiede l'ispezione di file effettivi o output di test, non solo la valutazione dei dati di input del hook da soli.

Configurazione dell'hook agente

Impostare `type` su `"agent"` e fornire una stringa `prompt`. I campi di configurazione sono gli stessi dei prompt hook, con un timeout predefinito più lungo:

Campo	Obbligatorio	Descrizione
`type`	sì	Deve essere `"agent"`
`prompt`	sì	Prompt che descrive cosa verificare. Utilizzare `$ARGUMENTS` come segnaposto per l'input JSON del hook
`model`	no	Modello da utilizzare. Impostazione predefinita su un modello veloce
`timeout`	no	Timeout in secondi. Impostazione predefinita: 60

Lo schema di risposta è lo stesso dei prompt hook: `{ "ok": true }` per consentire o `{ "ok": false, "reason": "..." }` per bloccare.

Questo hook `Stop` verifica che tutti i test unitari passino prima di consentire a Claude di terminare:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

Eseguire i hook in background

Per impostazione predefinita, i hook bloccano l'esecuzione di Claude fino al completamento. Per le attività a lunga esecuzione come distribuzioni, suite di test o chiamate API esterne, impostare `"async": true` per eseguire il hook in background mentre Claude continua a lavorare. Gli hook asincroni non possono bloccare o controllare il comportamento di Claude: i campi di risposta come `decision`, `permissionDecision` e `continue` non hanno effetto, perché l'azione che avrebbero controllato è già stata completata.

Configurare un hook asincrono

Aggiungere `"async": true` alla configurazione di un command hook per eseguirlo in background senza bloccare Claude. Questo campo è disponibile solo sui hook `type: "command"`.

Questo hook esegue uno script di test dopo ogni chiamata dello strumento `Write`. Claude continua a lavorare immediatamente mentre `run-tests.sh` viene eseguito per un massimo di 120 secondi. Quando lo script termina, il suo output viene consegnato al turno di conversazione successivo:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/run-tests.sh",
            "async": true,
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

Il campo `timeout` imposta il tempo massimo in secondi per il processo in background. Se non specificato, gli hook asincroni utilizzano lo stesso valore predefinito di 10 minuti dei hook sincroni.

Come si eseguono gli hook asincroni

Quando un hook asincrono si attiva, Claude Code avvia il processo del hook e continua immediatamente senza aspettare il completamento. L'hook riceve lo stesso input JSON via stdin di un hook sincrono.

Dopo che il processo in background esce, se l'hook ha prodotto una risposta JSON con un campo `systemMessage` o `additionalContext`, quel contenuto viene consegnato a Claude come contesto al turno di conversazione successivo.

Esempio: eseguire i test dopo le modifiche ai file

Questo hook avvia una suite di test in background ogni volta che Claude scrive un file, quindi segnala i risultati a Claude quando i test terminano. Salvare questo script in `.claude/hooks/run-tests-async.sh` nel vostro progetto e renderlo eseguibile con `chmod +x`:

```
#!/bin/bash
## run-tests-async.sh

## Legge l'input del hook da stdin
INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

## Eseguire i test solo per i file sorgente
if [[ "$FILE_PATH" =~ *.ts && "$FILE_PATH" =~ *.js ]]; then
  exit 0
fi

## Eseguire i test e segnalare i risultati tramite systemMessage
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
```

Quindi aggiungere questa configurazione a `.claude/settings.json` nella radice del vostro progetto. Il flag `async: true` consente a Claude di continuare a lavorare mentre i test vengono eseguiti:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/run-tests-async.sh",
            "async": true,
            "timeout": 300
          }
        ]
      }
    ]
  }
}
```

Limitazioni

Gli hook asincroni hanno diversi vincoli rispetto ai hook sincroni:

- Solo i hook `type: "command"` supportano `async`. I hook basati su prompt non possono essere eseguiti in modo asincrono.
- Gli hook asincroni non possono bloccare le chiamate dello strumento o restituire decisioni. Nel momento in cui l'hook si completa, l'azione che lo ha attivato è già proceduta.
- L'output del hook viene consegnato al turno di conversazione successivo. Se la sessione è inattiva, la risposta attende fino alla prossima interazione dell'utente.
- Ogni esecuzione crea un processo in background separato. Non c'è deduplicazione tra più esecuzioni dello stesso hook asincrono.

Considerazioni sulla sicurezza

Disclaimer

I command hook vengono eseguiti con i permessi completi dell'utente del sistema.

> **Warning:**

> I command hook eseguono comandi shell con i vostri permessi utente completi. Possono modificare, eliminare o accedere a qualsiasi file a cui il vostro account utente può accedere. Rivedere e testare tutti i comandi del hook prima di aggiungerli alla vostra configurazione.

Migliori pratiche di sicurezza

Tenere a mente queste pratiche quando si scrivono i hook:

- **Convalidare e bonificare gli input**: non fidarsi mai ciecamente dei dati di input
- **Citare sempre le variabili shell**: utilizzare `"$VAR"` non `$VAR`
- **Bloccare l'attraversamento del percorso**: controllare `..` nei percorsi dei file
- **Utilizzare percorsi assoluti**: specificare percorsi completi per gli script, utilizzando `"$CLAUDE_PROJECT_DIR"` per la radice del progetto
- **Saltare i file sensibili**: evitare `.env`, `.git/`, chiavi, ecc.

Debug dei hook

Eseguire `claude --debug` per vedere i dettagli dell'esecuzione del hook, inclusi quali hook hanno corrisposto, i loro codici di uscita e l'output. Attivare la modalità verbose con `Ctrl+O` per vedere l'avanzamento del hook nella trascrizione.

```
[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>
```

Per la risoluzione dei problemi comuni come i hook che non si attivano, i cicli infiniti di Stop hook o gli errori di configurazione, vedere Limitations and troubleshooting nella guida.

Automatizzare i flussi di lavoro con hooks

> *Esegui comandi shell automaticamente quando Claude Code modifica file, completa attività o ha bisogno di input. Formatta il codice, invia notifiche, convalida comandi e applica le regole del progetto.*

Gli hooks sono comandi shell definiti dall'utente che si eseguono in punti specifici del ciclo di vita di Claude Code. Forniscono un controllo deterministico sul comportamento di Claude Code, garantendo che determinate azioni avvengano sempre piuttosto che affidarsi al modello linguistico per scegliere di eseguirle. Utilizza gli hooks per applicare le regole del progetto, automatizzare attività ripetitive e integrare Claude Code con i tuoi strumenti esistenti.

Per decisioni che richiedono giudizio piuttosto che regole deterministiche, puoi anche utilizzare hooks basati su prompt o hooks basati su agenti che utilizzano un modello Claude per valutare le condizioni.

Per altri modi di estendere Claude Code, vedi skills per fornire a Claude istruzioni aggiuntive e comandi eseguibili, subagents per eseguire attività in contesti isolati, e plugins per pacchettizzare estensioni da condividere tra i progetti.

> **Tip:**
>
> Questa guida copre i casi d'uso comuni e come iniziare. Per schemi di eventi completi, formati di input/output JSON e funzionalità avanzate come hooks asincroni e hooks di strumenti MCP, vedi il riferimento Hooks.

Configura il tuo primo hook

Il modo più veloce per creare un hook è attraverso il menu interattivo `/hooks` in Claude Code. Questa procedura crea un hook di notifica desktop, in modo da ricevere un avviso ogni volta che Claude sta aspettando il tuo input invece di guardare il terminale.

Step 1: Apri il menu degli hooks

Digita `/hooks` nella CLI di Claude Code. Vedrai un elenco di tutti gli eventi hook disponibili, più un'opzione per disabilitare tutti gli hooks. Ogni evento corrisponde a un punto nel ciclo di vita di Claude dove puoi eseguire codice personalizzato. Seleziona `Notification` per creare un hook che si attiva quando Claude ha bisogno della tua attenzione.

Step 2: Configura il matcher

Il menu mostra un elenco di matcher, che filtrano quando l'hook si attiva. Imposta il matcher su `*` per attivarsi su tutti i tipi di notifica. Puoi restringerlo in seguito cambiando il matcher a un valore specifico come `permission_prompt` o `idle_prompt`.

Step 3: Aggiungi il tuo comando

Seleziona `+ Add new hook…`. Il menu ti chiede un comando shell da eseguire quando l'evento si attiva. Gli hooks eseguono qualsiasi comando shell tu fornisca, quindi puoi utilizzare lo strumento di notifica integrato della tua piattaforma. Copia il comando per il tuo sistema operativo:

macOS

Utilizza `osascript` per attivare una notifica macOS nativa tramite AppleScript:

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

Linux

Utilizza `notify-send`, che è preinstallato sulla maggior parte dei desktop Linux con un daemon di notifica:

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

Windows (PowerShell)

Utilizza PowerShell per mostrare una finestra di messaggio nativa tramite Windows Forms di .NET:

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

Step 4: Scegli una posizione di archiviazione

Il menu ti chiede dove salvare la configurazione dell'hook. Seleziona `User settings` per archiviarla in `~/.claude/settings.json`, che applica l'hook a tutti i tuoi progetti. Potresti anche scegliere `Project settings` per limitarlo al progetto corrente. Vedi Configura la posizione dell'hook per tutti gli ambiti disponibili.

Step 5: Testa l'hook

Premi `Esc` per tornare alla CLI. Chiedi a Claude di fare qualcosa che richieda autorizzazione, quindi allontanati dal terminale. Dovresti ricevere una notifica desktop.

Cosa puoi automatizzare

Gli hooks ti permettono di eseguire codice in punti chiave del ciclo di vita di Claude Code: formattare file dopo le modifiche, bloccare comandi prima che si eseguano, inviare notifiche quando Claude ha bisogno di input, iniettare contesto all'inizio della sessione, e altro ancora. Per l'elenco completo degli eventi hook, vedi il riferimento Hooks.

Ogni esempio include un blocco di configurazione pronto all'uso che aggiungi a un file di impostazioni. I modelli più comuni:

- Ricevi una notifica quando Claude ha bisogno di input
- Formatta automaticamente il codice dopo le modifiche
- Blocca le modifiche ai file protetti
- Reinietta il contesto dopo la compattazione
- Controlla le modifiche di configurazione

Ricevi una notifica quando Claude ha bisogno di input

Ricevi una notifica desktop ogni volta che Claude finisce di lavorare e ha bisogno del tuo input, in modo da poter passare ad altri compiti senza controllare il terminale.

Questo hook utilizza l'evento `Notification`, che si attiva quando Claude sta aspettando input o autorizzazione. Ogni scheda di seguito utilizza il comando di notifica nativo della piattaforma. Aggiungi questo a `~/.claude/settings.json`, o utilizza la procedura interattiva sopra per configurarlo con `/hooks`:

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]::Lo
adWithPartialName('System.Windows.Forms');
[System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention',
'Claude Code')\""
          }
        ]
      }
    ]
  }
}
```

Formatta automaticamente il codice dopo le modifiche

Esegui automaticamente Prettier su ogni file che Claude modifica, in modo che la formattazione rimanga coerente senza intervento manuale.

Questo hook utilizza l'evento `PostToolUse` con un matcher `Edit|Write`, quindi si esegue solo dopo gli strumenti di modifica dei file. Il comando estrae il percorso del file modificato con `jq` e lo passa a Prettier. Aggiungi questo a `.claude/settings.json` nella radice del tuo progetto:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --
write"
          }
        ]
      }
    ]
  }
}
```

Note:

Gli esempi Bash in questa pagina utilizzano `jq` per l'analisi JSON. Installalo con `brew install jq` (macOS), `apt-get install jq` (Debian/Ubuntu), o vedi download di `jq`.

Blocca le modifiche ai file protetti

Impedisci a Claude di modificare file sensibili come `.env`, `package-lock.json`, o qualsiasi cosa in `.git/`. Claude riceve un feedback che spiega perché la modifica è stata bloccata, in modo da poter adattare il suo approccio.

Questo esempio utilizza un file di script separato che l'hook chiama. Lo script controlla il percorso del file di destinazione rispetto a un elenco di modelli protetti ed esce con il codice 2 per bloccare la modifica.

Step 1: Crea lo script dell'hook

Salva questo in `.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: Rendi lo script eseguibile (macOS/Linux)

Gli script degli hook devono essere eseguibili affinché Claude Code li esegua:

```
chmod +x .claude/hooks/protect-files.sh
```

Step 3: Registra l'hook

Aggiungi un hook `PreToolUse` a `.claude/settings.json` che esegue lo script prima di qualsiasi chiamata dello strumento `Edit` o `Write`:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/protect-files.sh"
          }
        ]
      }
    ]
  }
}
```

Reinietta il contesto dopo la compattazione

Quando la finestra di contesto di Claude si riempie, la compattazione riassume la conversazione per liberare spazio. Questo può perdere dettagli importanti. Utilizza un hook `SessionStart` con un matcher `compact` per reiniettare il contesto critico dopo ogni compattazione.

Qualsiasi testo che il tuo comando scrive su stdout viene aggiunto al contesto di Claude. Questo esempio ricorda a Claude le convenzioni del progetto e il lavoro recente. Aggiungi questo a `.claude/settings.json` nella radice del tuo progetto:

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Reminder: use Bun, not npm. Run bun test before
committing. Current sprint: auth refactor.'"
          }
        ]
      }
    ]
  }
}
```

Puoi sostituire l' `echo` con qualsiasi comando che produce output dinamico, come `git log --oneline -5` per mostrare i commit recenti. Per iniettare contesto all'inizio di ogni sessione, considera invece di utilizzare CLAUDE.md. Per le variabili di ambiente, vedi `CLAUDE_ENV_FILE` nel riferimento.

Controlla le modifiche di configurazione

Traccia quando i file di impostazioni o skills cambiano durante una sessione. L'evento `ConfigChange` si attiva quando un processo esterno o un editor modifica un file di configurazione, in modo da poter registrare le modifiche per la conformità o bloccare le modifiche non autorizzate.

Questo esempio aggiunge ogni modifica a un registro di controllo. Aggiungi questo a `~/.claude/settings.json` :

```
{
  "hooks": {
    "ConfigChange": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "jq -c '{timestamp: now | todate, source: .source,
file: .file_path}' >> ~/claude-config-audit.log"
          }
        ]
      }
    ]
  }
}
```

Il matcher filtra per tipo di configurazione: `user_settings`, `project_settings`, `local_settings`, `policy_settings`, o `skills`. Per bloccare una modifica dall'avere effetto, esci con il codice 2 o restituisci `{"decision": "block"}`. Vedi il riferimento Config-Change per lo schema di input completo.

Come funzionano gli hooks

Gli eventi degli hooks si attivano in punti specifici del ciclo di vita di Claude Code. Quando un evento si attiva, tutti gli hooks corrispondenti si eseguono in parallelo, e i comandi degli hooks identici vengono automaticamente deduplicati. La tabella di seguito mostra ogni evento e quando si attiva:

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

Event	When it fires
`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
`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

Ogni hook ha un `type` che determina come si esegue. La maggior parte degli hooks utilizza `"type": "command"`, che esegue un comando shell. Sono disponibili altri tre tipi:

- `"type": "http"` : POST dei dati dell'evento a un URL. Vedi HTTP hooks.
- `"type": "prompt"` : valutazione LLM a turno singolo. Vedi Prompt-based hooks.
- `"type": "agent"` : verifica multi-turno con accesso agli strumenti. Vedi Agent-based hooks.

Leggi l'input e restituisci l'output

Gli hooks comunicano con Claude Code attraverso stdin, stdout, stderr e codici di uscita. Quando un evento si attiva, Claude Code passa i dati specifici dell'evento come JSON allo stdin del tuo script. Il tuo script legge quei dati, fa il suo lavoro, e dice a Claude Code cosa fare dopo tramite il codice di uscita.

Input dell'hook

Ogni evento include campi comuni come `session_id` e `cwd`, ma ogni tipo di evento aggiunge dati diversi. Ad esempio, quando Claude esegue un comando Bash, un hook `PreToolUse` riceve qualcosa di simile a questo su stdin:

```
{
  "session_id": "abc123",           // unique ID for this session
  "cwd": "/Users/sarah/myproject", // working directory when the event fired
  "hook_event_name": "PreToolUse", // which event triggered this hook
  "tool_name": "Bash",             // the tool Claude is about to use
  "tool_input": {                  // the arguments Claude passed to the tool
    "command": "npm test"          // for Bash, this is the shell command
  }
}
```

Il tuo script può analizzare quel JSON e agire su qualsiasi di quei campi. Gli hooks `UserPromptSubmit` ottengono il testo `prompt` invece, gli hook `SessionStart` ottengono la `source` (startup, resume, clear, compact), e così via. Vedi Campi di input comuni nel riferimento per i campi condivisi, e la sezione di ogni evento per gli schemi specifici dell'evento.

Output dell'hook

Il tuo script dice a Claude Code cosa fare dopo scrivendo su stdout o stderr e uscendo con un codice specifico. Ad esempio, un hook `PreToolUse` che vuole bloccare un comando:

```
#!/bin/bash
INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command')

if echo "$COMMAND" | grep -q "drop table"; then
  echo "Blocked: dropping tables is not allowed" >&2  # stderr becomes Claude's
feedback
  exit 2 # exit 2 = block the action
fi

exit 0  # exit 0 = let it proceed
```

Il codice di uscita determina cosa succede dopo:

- **Exit 0**: l'azione procede. Per gli hook `UserPromptSubmit` e `SessionStart`, qualsiasi cosa tu scriva su stdout viene aggiunta al contesto di Claude.
- **Exit 2**: l'azione è bloccata. Scrivi un motivo su stderr, e Claude lo riceve come feedback in modo da poter adattarsi.
- **Qualsiasi altro codice di uscita**: l'azione procede. Stderr viene registrato ma non mostrato a Claude. Attiva la modalità verbose con `Ctrl+O` per vedere questi messaggi nella trascrizione.

Output JSON strutturato

I codici di uscita ti danno due opzioni: consentire o bloccare. Per un controllo maggiore, esci con 0 e stampa un oggetto JSON su stdout invece.

> **Note:**
>
> Utilizza exit 2 per bloccare con un messaggio stderr, o exit 0 con JSON per un controllo strutturato. Non mischiarli: Claude Code ignora JSON quando esci con 2.

Ad esempio, un hook `PreToolUse` può negare una chiamata di strumento e dire a Claude perché, o escalare al utente per l'approvazione:

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Use rg instead of grep for better performance"
  }
}
```

Claude Code legge `permissionDecision` e annulla la chiamata dello strumento, quindi alimenta `permissionDecisionReason` di nuovo a Claude come feedback. Queste tre opzioni sono specifiche per `PreToolUse`:

- `"allow"`: procedi senza mostrare un prompt di autorizzazione
- `"deny"`: annulla la chiamata dello strumento e invia il motivo a Claude
- `"ask"`: mostra il prompt di autorizzazione all'utente come al solito

Altri eventi utilizzano diversi modelli di decisione. Ad esempio, gli hook `PostToolUse` e `Stop` utilizzano un campo `decision: "block"` di livello superiore, mentre `PermissionRequest` utilizza `hookSpecificOutput.decision.behavior`. Vedi la tabella di riepilogo nel riferimento per una suddivisione completa per evento.

Per gli hook `UserPromptSubmit`, utilizza `additionalContext` invece per iniettare testo nel contesto di Claude. Gli hooks basati su prompt (`type: "prompt"`) gestiscono l'output diversamente: vedi Prompt-based hooks.

Filtra gli hooks con i matcher

Senza un matcher, un hook si attiva su ogni occorrenza del suo evento. I matcher ti permettono di restringere questo. Ad esempio, se vuoi eseguire un formattatore solo dopo le modifiche ai file (non dopo ogni chiamata di strumento), aggiungi un matcher al tuo hook `PostToolUse`:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "prettier --write ..." }
        ]
      }
    ]
  }
}
```

Il matcher `"Edit|Write"` è un modello regex che corrisponde al nome dello strumento. L'hook si attiva solo quando Claude utilizza lo strumento `Edit` o `Write`, non quando utilizza `Bash`, `Read`, o qualsiasi altro strumento.

Ogni tipo di evento corrisponde a un campo specifico. I matcher supportano stringhe esatte e modelli regex:

Evento	Cosa filtra il matcher	Valori matcher di esempio
`PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `PermissionRequest`	nome dello strumento	`Bash`, `Edit|Write`, `mcp__.*`
`SessionStart`	come è iniziata la sessione	`startup`, `resume`, `clear`, `compact`
`SessionEnd`	perché è terminata la sessione	`clear`, `logout`, `prompt_input_exit`, `bypass_permissions_disabled`, `other`
`Notification`	tipo di notifica	`permission_prompt`, `idle_prompt`, `auth_success`, `elicitation_dialog`

Evento	Cosa filtra il matcher	Valori matcher di esempio
`SubagentStart`	tipo di agente	`Bash`, `Explore`, `Plan`, o nomi di agenti personalizzati
`PreCompact`	cosa ha attivato la compattazione	`manual`, `auto`
`SubagentStop`	tipo di agente	stessi valori di `SubagentStart`
`ConfigChange`	fonte di configurazione	`user_settings`, `project_settings`, `local_settings`, `policy_settings`, `skills`
`UserPromptSubmit`, `Stop`, `TeammateIdle`, `TaskCompleted`, `WorktreeCreate`, `WorktreeRemove`	nessun supporto matcher	si attiva sempre su ogni occorrenza

Alcuni altri esempi che mostrano i matcher su diversi tipi di evento:

Registra ogni comando Bash

Corrisponde solo alle chiamate dello strumento `Bash` e registra ogni comando in un file. L'evento `PostToolUse` si attiva dopo che il comando è completato, quindi `tool_input.command` contiene quello che è stato eseguito. L'hook riceve i dati dell'evento come JSON su stdin, e `jq -r '.tool_input.command'` estrae solo la stringa del comando, che `>>` aggiunge al file di registro:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.command' >> ~/.claude/command-log.txt"
          }
        ]
      }
    ]
  }
}
```

Corrisponde agli strumenti MCP

Gli strumenti MCP utilizzano una convenzione di denominazione diversa rispetto agli strumenti integrati: `mcp__<server>__<tool>`, dove `<server>` è il nome del server MCP e `<tool>` è lo strumento che fornisce. Ad esempio, `mcp__github__search_repositories` o `mcp__filesystem__read_file`. Utilizza un matcher regex per indirizzare tutti gli strumenti da un server specifico, o corrispondere tra i server con un modello come `mcp__.*__write.*`. Vedi Corrisponde agli strumenti MCP nel riferimento per l'elenco completo degli esempi.

Il comando di seguito estrae il nome dello strumento dall'input JSON dell'hook con `jq` e lo scrive su stderr, dove appare in modalità verbose (`Ctrl+O`):

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__github__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"GitHub tool called: $(jq -r '.tool_name')\" >&2"
          }
        ]
      }
    ]
  }
}
```

Pulisci alla fine della sessione

L'evento `SessionEnd` supporta i matcher sul motivo per cui la sessione è terminata. Questo hook si attiva solo su `clear` (quando esegui `/clear`), non su uscite normali:

```
{
  "hooks": {
    "SessionEnd": [
      {
        "matcher": "clear",
        "hooks": [
          {
            "type": "command",
            "command": "rm -f /tmp/claude-scratch-*.txt"
          }
        ]
      }
    ]
  }
}
```

Per la sintassi completa del matcher, vedi il riferimento Hooks.

Configura la posizione dell'hook

Dove aggiungi un hook determina il suo ambito:

Posizione	Ambito	Condivisibile
`~/.claude/settings.json`	Tutti i tuoi progetti	No, locale alla tua macchina
`.claude/settings.json`	Singolo progetto	Sì, può essere committato nel repo
`.claude/settings.local.json`	Singolo progetto	No, gitignored
Impostazioni di policy gestite	Organizzazione intera	Sì, controllato dall'amministratore
Plugin `hooks/hooks.json`	Quando il plugin è abilitato	Sì, raggruppato con il plugin
Skill o agente frontmatter	Mentre la skill o l'agente è attivo	Sì, definito nel file del componente

Puoi anche utilizzare il menu `/hooks` in Claude Code per aggiungere, eliminare e visualizzare gli hooks in modo interattivo. Per disabilitare tutti gli hooks contemporaneamente, utilizza l'interruttore in fondo al menu `/hooks` o imposta `"disableAllHooks": true` nel tuo file di impostazioni.

Gli hooks aggiunti tramite il menu `/hooks` hanno effetto immediato. Se modifichi i file di impostazioni direttamente mentre Claude Code è in esecuzione, le modifiche non avranno effetto fino a quando non le esamini nel menu `/hooks` o non riavvii la tua sessione.

Prompt-based hooks

Per decisioni che richiedono giudizio piuttosto che regole deterministiche, utilizza gli hook `type: "prompt"`. Invece di eseguire un comando shell, Claude Code invia il tuo prompt e i dati di input dell'hook a un modello Claude (Haiku per impostazione predefinita) per prendere la decisione. Puoi specificare un modello diverso con il campo `model` se hai bisogno di più capacità.

L'unico lavoro del modello è restituire una decisione sì/no come JSON:

- `"ok": true`: l'azione procede

- `"ok": false`: l'azione è bloccata. Il `"reason"` del modello viene alimentato di nuovo a Claude in modo da poter adattarsi.

Questo esempio utilizza un hook `Stop` per chiedere al modello se tutti i compiti richiesti sono completi. Se il modello restituisce `"ok": false`, Claude continua a lavorare e utilizza il `reason` come sua prossima istruzione:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Check if all tasks are complete. If not, respond with
{\"ok\": false, \"reason\": \"what remains to be done\"}."
          }
        ]
      }
    ]
  }
}
```

Per le opzioni di configurazione complete, vedi Prompt-based hooks nel riferimento.

Agent-based hooks

Quando la verifica richiede l'ispezione di file o l'esecuzione di comandi, utilizza gli hook `type: "agent"`. A differenza degli hook di prompt che effettuano una singola chiamata LLM, gli hook di agente generano un subagent che può leggere file, cercare codice e utilizzare altri strumenti per verificare le condizioni prima di restituire una decisione.

Gli hook di agente utilizzano lo stesso formato di risposta `"ok"` / `"reason"` degli hook di prompt, ma con un timeout predefinito più lungo di 60 secondi e fino a 50 turni di utilizzo dello strumento.

Questo esempio verifica che i test passino prima di consentire a Claude di fermarsi:

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

Utilizza gli hook di prompt quando i dati di input dell'hook da soli sono sufficienti per prendere una decisione. Utilizza gli hook di agente quando hai bisogno di verificare qualcosa rispetto allo stato effettivo della base di codice.

Per le opzioni di configurazione complete, vedi Agent-based hooks nel riferimento.

HTTP hooks

Utilizza gli hook `type: "http"` per POST dei dati dell'evento a un endpoint HTTP invece di eseguire un comando shell. L'endpoint riceve lo stesso JSON che un hook di comando riceverebbe su stdin, e restituisce i risultati attraverso il corpo della risposta HTTP utilizzando lo stesso formato JSON.

Gli HTTP hooks sono utili quando vuoi che un server web, una funzione cloud o un servizio esterno gestisca la logica dell'hook: ad esempio, un servizio di controllo condiviso che registra gli eventi di utilizzo dello strumento in un team.

Questo esempio pubblica ogni utilizzo dello strumento a un servizio di registrazione locale:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/tool-use",
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}
```

L'endpoint dovrebbe restituire un corpo di risposta JSON utilizzando lo stesso formato di output degli hook di comando. Per bloccare una chiamata di strumento, restituisci una risposta 2xx con i campi `hookSpecificOutput` appropriati. I codici di stato HTTP da soli non possono bloccare le azioni.

I valori dell'intestazione supportano l'interpolazione delle variabili di ambiente utilizzando la sintassi `$VAR_NAME` o `${VAR_NAME}`. Solo le variabili elencate nell'array `allowedEnvVars` vengono risolte; tutti gli altri riferimenti `$VAR` rimangono vuoti.

> **Note:**
>
> Gli HTTP hooks devono essere configurati modificando direttamente il tuo JSON di impostazioni. Il menu interattivo `/hooks` supporta solo l'aggiunta di hook di comando.

Per le opzioni di configurazione complete e la gestione delle risposte, vedi HTTP hooks nel riferimento.

Limitazioni e risoluzione dei problemi

Limitazioni

- Gli hook di comando comunicano solo attraverso stdout, stderr e codici di uscita. Non possono attivare comandi o chiamate di strumenti direttamente. Gli HTTP hooks comunicano attraverso il corpo della risposta invece.
- Il timeout dell'hook è di 10 minuti per impostazione predefinita, configurabile per hook con il campo `timeout` (in secondi).
- Gli hook `PostToolUse` non possono annullare le azioni poiché lo strumento è già stato eseguito.
- Gli hook `PermissionRequest` non si attivano in modalità non interattiva (`-p`). Utilizza gli hook `PreToolUse` per le decisioni di autorizzazione automatizzate.
- Gli hook `Stop` si attivano ogni volta che Claude finisce di rispondere, non solo al completamento dell'attività. Non si attivano su interruzioni dell'utente.

Hook non si attiva

L'hook è configurato ma non si esegue mai.

- Esegui `/hooks` e conferma che l'hook appare sotto l'evento corretto
- Controlla che il modello del matcher corrisponda al nome dello strumento esattamente (i matcher sono sensibili alle maiuscole)
- Verifica che stai attivando il tipo di evento corretto (ad esempio, `PreToolUse` si attiva prima dell'esecuzione dello strumento, `PostToolUse` si attiva dopo)
- Se utilizzi gli hook `PermissionRequest` in modalità non interattiva (`-p`), passa agli hook `PreToolUse` invece

Errore dell'hook nell'output

Vedi un messaggio come "PreToolUse hook error: ..." nella trascrizione.

- Il tuo script è uscito con un codice diverso da zero inaspettatamente. Testalo manualmente inviando JSON di esempio:

```
echo '{"tool_name":"Bash","tool_input":{"command":"ls"}}' | ./my-hook.sh
echo $?  # Check the exit code
```

- Se vedi "command not found", utilizza percorsi assoluti o `$CLAUDE_PROJECT_DIR` per fare riferimento agli script

- Se vedi “jq: command not found”, installa `jq` o utilizza Python/Node.js per l’analisi JSON
- Se lo script non si esegue affatto, rendilo eseguibile: `chmod +x ./my-hook.sh`

`/hooks` non mostra hook configurati

Hai modificato un file di impostazioni ma gli hook non appaiono nel menu.

- Riavvia la tua sessione o apri `/hooks` per ricaricare. Gli hook aggiunti tramite il menu `/hooks` hanno effetto immediato, ma le modifiche manuali ai file richiedono un ricaricamento.
- Verifica che il tuo JSON sia valido (le virgole finali e i commenti non sono consentiti)
- Conferma che il file di impostazioni è nella posizione corretta: `.claude/settings.json` per gli hook del progetto, `~/.claude/settings.json` per gli hook globali

L’hook Stop si esegue per sempre

Claude continua a lavorare in un ciclo infinito invece di fermarsi.

Il tuo script dell’hook Stop deve controllare se ha già attivato una continuazione. Analizza il campo `stop_hook_active` dall’input JSON e esci presto se è `true`:

```
#!/bin/bash
INPUT=$(cat)
if [ "$(echo "$INPUT" | jq -r '.stop_hook_active')" = "true" ]; then
  exit 0  # Allow Claude to stop
fi
## ... rest of your hook logic
```

Convalida JSON non riuscita

Claude Code mostra un errore di analisi JSON anche se il tuo script dell’hook produce JSON valido.

Quando Claude Code esegue un hook, genera una shell che fornisce il tuo profilo (`~/.zshrc` o `~/.bashrc`). Se il tuo profilo contiene istruzioni `echo` incondizionate, quell’output viene anteposto al JSON del tuo hook:

```
Shell ready on arm64
{"decision": "block", "reason": "Not allowed"}
```

Claude Code tenta di analizzare questo come JSON e fallisce. Per risolvere questo, avvolgi le istruzioni echo nel tuo profilo shell in modo che si eseguano solo in shell interattive:

```
## In ~/.zshrc or ~/.bashrc
if [[ $- == *i* ]]; then
  echo "Shell ready"
fi
```

La variabile `$-` contiene i flag della shell, e `i` significa interattivo. Gli hook si eseguono in shell non interattive, quindi l'echo viene saltato.

Tecniche di debug

Attiva la modalità verbose con `Ctrl+O` per vedere l'output dell'hook nella trascrizione, o esegui `claude --debug` per i dettagli di esecuzione completi incluso quali hook hanno corrisposto e i loro codici di uscita.

Scopri di più

- Riferimento Hooks: schemi di eventi completi, formato di output JSON, hook asincroni e hook di strumenti MCP
- Considerazioni sulla sicurezza: esamina prima di distribuire gli hook in ambienti condivisi o di produzione
- Esempio di validatore di comandi Bash: implementazione di riferimento completa

Connetti Claude Code ai tuoi strumenti tramite MCP

> *Scopri come connettere Claude Code ai tuoi strumenti con il Model Context Protocol.*

Claude Code può connettersi a centinaia di strumenti e fonti di dati esterni attraverso il Model Context Protocol (MCP), uno standard open source per le integrazioni AI-tool. I server MCP danno a Claude Code accesso ai tuoi strumenti, database e API.

Cosa puoi fare con MCP

Con i server MCP connessi, puoi chiedere a Claude Code di:

- **Implementare funzionalità da issue tracker**: "Aggiungi la funzionalità descritta nel ticket JIRA ENG-4521 e crea una PR su GitHub."
- **Analizzare dati di monitoraggio**: "Controlla Sentry e Statsig per verificare l'utilizzo della funzionalità descritta in ENG-4521."
- **Interrogare database**: "Trova gli indirizzi email di 10 utenti casuali che hanno utilizzato la funzionalità ENG-4521, in base al nostro database PostgreSQL."
- **Integrare design**: "Aggiorna il nostro modello di email standard in base ai nuovi design Figma che sono stati pubblicati su Slack"
- **Automatizzare flussi di lavoro**: "Crea bozze Gmail invitando questi 10 utenti a una sessione di feedback sulla nuova funzionalità."

Server MCP popolari

Ecco alcuni server MCP comunemente utilizzati che puoi connettere a Claude Code:

> **Warning:**
>
> Utilizza server MCP di terze parti a tuo rischio - Anthropic non ha verificato la correttezza o la sicurezza di tutti questi server. Assicurati di fidarti dei server MCP che stai installando. Fai particolare attenzione quando utilizzi server MCP che potrebbero recuperare contenuti non attendibili, poiché questi possono esporti al rischio di prompt injection.

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`
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`

Server	Description	Command
Granola	The AI notepad for meetings	`claude mcp add --transport http granola https://mcp.granola.ai/mcp`
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`

Server	Description	Command
Microsoft Learn	Search trusted Microsoft docs to power your development	`claude mcp add --transport http microsoft-learn https://learn.microsoft.com/api/mcp`
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`

Server	Description	Command
GoDaddy	Search domains and check availability	`claude mcp add --transport http godaddy https://api.godaddy.com/v1/domains/mcp`
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`

Server	Description	Command
Jam	Record screen and collect automatic context for issues	`claude mcp add --transport http jam https://mcp.jam.dev/mcp`
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:

Hai bisogno di un'integrazione specifica? Trova centinaia di altri server MCP su GitHub, oppure crea il tuo utilizzando l'MCP SDK.

Installazione dei server MCP

I server MCP possono essere configurati in tre modi diversi a seconda delle tue esigenze:

Opzione 1: Aggiungi un server HTTP remoto

I server HTTP sono l'opzione consigliata per connettersi ai server MCP remoti. Questo è il trasporto più ampiamente supportato per i servizi basati su cloud.

```
## Sintassi di base
claude mcp add --transport http <name> <url>

## Esempio reale: Connessione a Notion
claude mcp add --transport http notion https://mcp.notion.com/mcp

## Esempio con token Bearer
claude mcp add --transport http secure-api https://api.example.com/mcp \
  --header "Authorization: Bearer your-token"
```

Opzione 2: Aggiungi un server SSE remoto

> **Warning:**
>
> Il trasporto SSE (Server-Sent Events) è deprecato. Utilizza server HTTP invece, dove disponibili.

```
## Sintassi di base
claude mcp add --transport sse <name> <url>

## Esempio reale: Connessione ad Asana
claude mcp add --transport sse asana https://mcp.asana.com/sse

## Esempio con header di autenticazione
claude mcp add --transport sse private-api https://api.company.com/sse \
  --header "X-API-Key: your-key-here"
```

Opzione 3: Aggiungi un server stdio locale

I server stdio vengono eseguiti come processi locali sulla tua macchina. Sono ideali per strumenti che necessitano di accesso diretto al sistema o script personalizzati.

```
## Sintassi di base
claude mcp add [options] <name> -- <command> [args...]

## Esempio reale: Aggiungi server Airtable
claude mcp add --transport stdio --env AIRTABLE_API_KEY=YOUR_KEY airtable \
  -- npx -y airtable-mcp-server
```

Note:

Importante: Ordine delle opzioni

Tutte le opzioni (`--transport` , `--env` , `--scope` , `--header`) devono venire **prima** del nome del server. Il `--` (doppio trattino) separa quindi il nome del server dal comando e dagli argomenti che vengono passati al server MCP.

Per esempio:

- `claude mcp add --transport stdio myserver -- npx server` → esegue `npx server`
- `claude mcp add --transport stdio --env KEY=value myserver -- python server.py --port 8080` → esegue `python server.py --port 8080` con `KEY=value` nell'ambiente

Questo previene conflitti tra i flag di Claude e i flag del server.

Gestione dei tuoi server

Una volta configurati, puoi gestire i tuoi server MCP con questi comandi:

```
## Elenca tutti i server configurati
claude mcp list

## Ottieni dettagli per un server specifico
claude mcp get github

## Rimuovi un server
claude mcp remove github

## (all'interno di Claude Code) Controlla lo stato del server
/mcp
```

Aggiornamenti dinamici degli strumenti

Claude Code supporta le notifiche `list_changed` di MCP, consentendo ai server MCP di aggiornare dinamicamente i loro strumenti, prompt e risorse disponibili senza richiedere di disconnettersi e riconnettersi. Quando un server MCP invia una notifica `list_changed`, Claude Code aggiorna automaticamente le capacità disponibili da quel server.

Tip:

Suggerimenti:

- Utilizza il flag `--scope` per specificare dove viene archiviata la configurazione:
- `local` (predefinito): Disponibile solo per te nel progetto corrente (era chiamato `project` nelle versioni precedenti)
- `project`: Condiviso con tutti nel progetto tramite il file `.mcp.json`
- `user`: Disponibile per te in tutti i progetti (era chiamato `global` nelle versioni precedenti)
- Imposta le variabili di ambiente con i flag `--env` (per esempio, `--env KEY=value`)
- Configura il timeout di avvio del server MCP utilizzando la variabile di ambiente MCP_TIMEOUT (per esempio, `MCP_TIMEOUT=10000 claude` imposta un timeout di 10 secondi)
- Claude Code visualizzerà un avviso quando l'output dello strumento MCP supera 10.000 token. Per aumentare questo limite, imposta la variabile di ambiente `MAX_MCP_OUTPUT_TOKENS` (per esempio, `MAX_MCP_OUTPUT_TOKENS=50000`)
- Utilizza `/mcp` per autenticarti con server remoti che richiedono l'autenticazione OAuth 2.0

Warning:

Utenti Windows: Su Windows nativo (non WSL), i server MCP locali che utilizzano `npx` richiedono il wrapper `cmd /c` per garantire l'esecuzione corretta.

```
## Questo crea command="cmd" che Windows può eseguire
claude mcp add --transport stdio my-server -- cmd /c npx -y @some/package
```

Senza il wrapper `cmd /c`, incontrerai errori "Connection closed" perché Windows non può eseguire direttamente `npx`. (Vedi la nota sopra per una spiegazione del parametro `--`.)

Server MCP forniti da plugin

I plugin possono raggruppare server MCP, fornendo automaticamente strumenti e integrazioni quando il plugin è abilitato. I server MCP dei plugin funzionano in modo identico ai server configurati dall'utente.

Come funzionano i server MCP dei plugin:

- I plugin definiscono i server MCP in `.mcp.json` nella radice del plugin o inline in `plugin.json`
- Quando un plugin è abilitato, i suoi server MCP si avviano automaticamente
- Gli strumenti MCP del plugin appaiono insieme agli strumenti MCP configurati manualmente
- I server dei plugin vengono gestiti tramite l'installazione del plugin (non tramite comandi `/mcp`)

Esempio di configurazione MCP del plugin:

In `.mcp.json` nella radice del plugin:

```
{
  "database-tools": {
    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
    "env": {
      "DB_URL": "${DB_URL}"
    }
  }
}
```

O inline in `plugin.json`:

```
{
  "name": "my-plugin",
  "mcpServers": {
    "plugin-api": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/api-server",
      "args": ["--port", "8080"]
    }
  }
}
```

Funzionalità MCP del plugin:

- **Ciclo di vita automatico**: I server si avviano quando il plugin si abilita, ma devi riavviare Claude Code per applicare le modifiche del server MCP (abilitazione o disabilitazione)
- **Variabili di ambiente**: Utilizza `${CLAUDE_PLUGIN_ROOT}` per i percorsi relativi al plugin
- **Accesso alle variabili di ambiente dell'utente**: Accesso alle stesse variabili di ambiente dei server configurati manualmente
- **Tipi di trasporto multipli**: Supporto per trasporti stdio, SSE e HTTP (il supporto del trasporto può variare a seconda del server)

Visualizzazione dei server MCP del plugin:

```
## All'interno di Claude Code, vedi tutti i server MCP inclusi quelli dei plugin
/mcp
```

I server dei plugin appaiono nell'elenco con indicatori che mostrano che provengono dai plugin.

Vantaggi dei server MCP dei plugin:

- **Distribuzione raggruppata**: Strumenti e server confezionati insieme
- **Configurazione automatica**: Nessuna configurazione MCP manuale necessaria
- **Coerenza del team**: Tutti ottengono gli stessi strumenti quando il plugin è installato

Vedi il riferimento dei componenti del plugin per i dettagli su come raggruppare i server MCP con i plugin.

Ambiti di installazione MCP

I server MCP possono essere configurati a tre diversi livelli di ambito, ognuno dei quali serve scopi distinti per gestire l'accessibilità e la condivisione dei server. Comprendere questi ambiti ti aiuta a determinare il modo migliore per configurare i server per le tue esigenze specifiche.

Ambito locale

I server con ambito locale rappresentano il livello di configurazione predefinito e vengono archiviati in `~/.claude.json` nel percorso del tuo progetto. Questi server rimangono privati per te e sono accessibili solo quando lavori all'interno della directory del progetto corrente. Questo ambito è ideale per server di sviluppo personali, configurazioni sperimentali o server contenenti credenziali sensibili che non dovrebbero essere condivise.

> **Note:**
>
> Il termine "ambito locale" per i server MCP differisce dalle impostazioni locali generali. I server MCP con ambito locale vengono archiviati in `~/.claude.json` (la tua directory home), mentre le impostazioni locali generali utilizzano `.claude/settings.local.json` (nella directory del progetto). Vedi Impostazioni per i dettagli sui percorsi dei file di impostazioni.

```
## Aggiungi un server con ambito locale (predefinito)
claude mcp add --transport http stripe https://mcp.stripe.com

## Specifica esplicitamente l'ambito locale
claude mcp add --transport http stripe --scope local https://mcp.stripe.com
```

Ambito del progetto

I server con ambito del progetto abilitano la collaborazione del team archiviando le configurazioni in un file `.mcp.json` nella directory radice del tuo progetto. Questo file è progettato per essere archiviato nel controllo della versione, assicurando che tutti i membri del team abbiano accesso agli stessi strumenti e servizi MCP. Quando aggiungi un server con ambito del progetto, Claude Code crea o aggiorna automaticamente questo file con la struttura di configurazione appropriata.

```
## Aggiungi un server con ambito del progetto
claude mcp add --transport http paypal --scope project https://mcp.paypal.com/mcp
```

Il file `.mcp.json` risultante segue un formato standardizzato:

```
{
  "mcpServers": {
    "shared-server": {
      "command": "/path/to/server",
      "args": [],
      "env": {}
    }
  }
}
```

Per motivi di sicurezza, Claude Code richiede l'approvazione prima di utilizzare server con ambito del progetto dai file `.mcp.json`. Se devi ripristinare queste scelte di approvazione, utilizza il comando `claude mcp reset-project-choices`.

Ambito utente

I server con ambito utente vengono archiviati in `~/.claude.json` e forniscono accessibilità tra progetti, rendendoli disponibili in tutti i progetti sulla tua macchina mentre rimangono privati al tuo account utente. Questo ambito funziona bene per server di utilità personali, strumenti di sviluppo o servizi che utilizzi frequentemente in diversi progetti.

```
## Aggiungi un server utente
claude mcp add --transport http hubspot --scope user https://mcp.hubspot.com/
anthropic
```

Scelta dell'ambito giusto

Seleziona il tuo ambito in base a:

- **Ambito locale**: Server personali, configurazioni sperimentali o credenziali sensibili specifici di un progetto
- **Ambito del progetto**: Server condivisi dal team, strumenti specifici del progetto o servizi necessari per la collaborazione
- **Ambito utente**: Utilità personali necessarie in più progetti, strumenti di sviluppo o servizi utilizzati frequentemente

> **Note:**

Dove vengono archiviati i server MCP?

- **Ambito utente e locale**: `~/.claude.json` (nel campo `mcpServers` o nei percorsi del progetto)
- **Ambito del progetto**: `.mcp.json` nella radice del tuo progetto (archiviato nel controllo della versione)
- **Gestito**: `managed-mcp.json` nelle directory di sistema (vedi Configurazione MCP gestita)

Gerarchia e precedenza dell'ambito

Le configurazioni del server MCP seguono una chiara gerarchia di precedenza. Quando server con lo stesso nome esistono in più ambiti, il sistema risolve i conflitti dando priorità ai server con ambito locale per primi, seguiti dai server con ambito del progetto e infine dai server con ambito utente. Questo design assicura che le configurazioni personali possano sovrascrivere quelle condivise quando necessario.

Espansione delle variabili di ambiente in `.mcp.json`

Claude Code supporta l'espansione delle variabili di ambiente nei file `.mcp.json`, consentendo ai team di condividere configurazioni mantenendo flessibilità per i percorsi specifici della macchina e i valori sensibili come le chiavi API.

Sintassi supportata:

- `${VAR}` - Si espande al valore della variabile di ambiente `VAR`
- `${VAR:-default}` - Si espande a `VAR` se impostato, altrimenti utilizza `default`

Posizioni di espansione: Le variabili di ambiente possono essere espanse in:

- `command` - Il percorso dell'eseguibile del server
- `args` - Argomenti della riga di comando
- `env` - Variabili di ambiente passate al server
- `url` - Per i tipi di server HTTP
- `headers` - Per l'autenticazione del server HTTP

Esempio con espansione di variabili:

```
{
  "mcpServers": {
    "api-server": {
      "type": "http",
      "url": "${API_BASE_URL:-https://api.example.com}/mcp",
      "headers": {
        "Authorization": "Bearer ${API_KEY}"
      }
    }
  }
}
```

Se una variabile di ambiente richiesta non è impostata e non ha un valore predefinito, Claude Code non riuscirà ad analizzare la configurazione.

Esempi pratici

{/* ### Example: Automate browser testing with Playwright

```
claude mcp add --transport stdio playwright -- npx -y @playwright/mcp@latest
```

Then write and run browser tests:

```
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
  ``` */}

Esempio: Monitora gli errori con Sentry

```bash  theme={null}
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Autenticati con il tuo account Sentry:

```
/mcp
```

Quindi esegui il debug dei problemi di produzione:

```
Quali sono gli errori più comuni nelle ultime 24 ore?
```

```
Mostrami la stack trace per l'errore ID abc123
```

```
Quale deployment ha introdotto questi nuovi errori?
```

Esempio: Connettiti a GitHub per le revisioni del codice

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

Autenticati se necessario selezionando “Authenticate” per GitHub:

```
/mcp
```

Quindi lavora con GitHub:

```
Rivedi la PR #456 e suggerisci miglioramenti
```

```
Crea un nuovo issue per il bug che abbiamo appena trovato
```

```
Mostrami tutte le PR aperte assegnate a me
```

Esempio: Interroga il tuo database PostgreSQL

```
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \
  --dsn "postgresql://readonly:pass@prod.db.com:5432/analytics"
```

Quindi interroga il tuo database naturalmente:

```
Qual è il nostro ricavo totale questo mese?
```

```
Mostrami lo schema per la tabella orders
```

```
Trova i clienti che non hanno effettuato un acquisto negli ultimi 90 giorni
```

Autenticazione con server MCP remoti

Molti server MCP basati su cloud richiedono l'autenticazione. Claude Code supporta OAuth 2.0 per connessioni sicure.

Step 1: Aggiungi il server che richiede l'autenticazione

Per esempio:

```
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Step 2: Utilizza il comando /mcp all'interno di Claude Code

In Claude Code, utilizza il comando:

```
/mcp
```

Quindi segui i passaggi nel tuo browser per accedere.

> **Tip:**
>
> Suggerimenti:
>
> - I token di autenticazione vengono archiviati in modo sicuro e aggiornati automaticamente
> - Utilizza "Clear authentication" nel menu `/mcp` per revocare l'accesso

- Se il tuo browser non si apre automaticamente, copia l'URL fornito e aprilo manualmente
- Se il reindirizzamento del browser non riesce con un errore di connessione dopo l'autenticazione, incolla l'URL di callback completo dalla barra degli indirizzi del tuo browser nel prompt dell'URL che appare in Claude Code
- L'autenticazione OAuth funziona con i server HTTP

Utilizza una porta di callback OAuth fissa

Alcuni server MCP richiedono un URI di reindirizzamento specifico registrato in anticipo. Per impostazione predefinita, Claude Code sceglie una porta disponibile casuale per il callback OAuth. Utilizza `--callback-port` per fissare la porta in modo che corrisponda a un URI di reindirizzamento pre-registrato della forma `http://localhost:PORT/callback`.

Puoi utilizzare `--callback-port` da solo (con registrazione dinamica del client) o insieme a `--client-id` (con credenziali pre-configurate).

```
## Porta di callback fissa con registrazione dinamica del client
claude mcp add --transport http \
  --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

Utilizza credenziali OAuth pre-configurate

Alcuni server MCP non supportano la configurazione OAuth automatica. Se vedi un errore come "Incompatible auth server: does not support dynamic client registration," il server richiede credenziali pre-configurate. Registra prima un'app OAuth tramite il portale degli sviluppatori del server, quindi fornisci le credenziali quando aggiungi il server.

Step 1: Registra un'app OAuth con il server

Crea un'app tramite il portale degli sviluppatori del server e annota il tuo ID client e il segreto client.

Molti server richiedono anche un URI di reindirizzamento. Se è così, scegli una porta e registra un URI di reindirizzamento nel formato `http://localhost:PORT/callback`. Utilizza quella stessa porta con `--callback-port` nel passaggio successivo.

Step 2: Aggiungi il server con le tue credenziali

Scegli uno dei seguenti metodi. La porta utilizzata per `--callback-port` può essere qualsiasi porta disponibile. Deve solo corrispondere all'URI di reindirizzamento che hai registrato nel passaggio precedente.

claude mcp add

Utilizza `--client-id` per passare l'ID client della tua app. Il flag `--client-secret` richiede il segreto con input mascherato:

```
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

Includi l'oggetto `oauth` nella configurazione JSON e passa `--client-secret` come flag separato:

```
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 (solo porta di callback)

Utilizza `--callback-port` senza un ID client per fissare la porta mentre utilizzi la registrazione dinamica del client:

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"callbackPort":8080}}'
```

CI / env var

Imposta il segreto tramite variabile di ambiente per saltare il prompt interattivo:

```
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: Autenticati in Claude Code

Esegui `/mcp` in Claude Code e segui il flusso di accesso del browser.

Tip:

Suggerimenti:

- Il segreto client viene archiviato in modo sicuro nel tuo portachiavi di sistema (macOS) o in un file di credenziali, non nella tua configurazione
- Se il server utilizza un client OAuth pubblico senza segreto, utilizza solo `--client-id` senza `--client-secret`
- `--callback-port` può essere utilizzato con o senza `--client-id`
- Questi flag si applicano solo ai trasporti HTTP e SSE. Non hanno effetto sui server stdio
- Utilizza `claude mcp get <name>` per verificare che le credenziali OAuth siano configurate per un server

Sovrascrivi la scoperta dei metadati OAuth

Se il tuo server MCP restituisce errori sull'endpoint dei metadati OAuth standard (`/.well-known/oauth-authorization-server`) ma espone un endpoint OIDC funzionante, puoi dire a Claude Code di recuperare i metadati OAuth direttamente da un URL che specifichi, bypassando la catena di scoperta standard.

Imposta `authServerMetadataUrl` nell'oggetto `oauth` della configurazione del tuo server in `.mcp.json` :

```
{
  "mcpServers": {
    "my-server": {
      "type": "http",
      "url": "https://mcp.example.com/mcp",
      "oauth": {
        "authServerMetadataUrl": "https://auth.example.com/.well-known/openid-
configuration"
      }
    }
  }
}
```

L'URL deve utilizzare `https://` . Questa opzione richiede Claude Code v2.1.64 o successiva.

Aggiungi server MCP dalla configurazione JSON

Se hai una configurazione JSON per un server MCP, puoi aggiungerla direttamente:

Step 1: Aggiungi un server MCP da JSON

```
## Sintassi di base
claude mcp add-json <name> '<json>'

## Esempio: Aggiunta di un server HTTP con configurazione JSON
claude mcp add-json weather-api '{"type":"http","url":"https://api.weather.com/
mcp","headers":{"Authorization":"Bearer token"}}'

## Esempio: Aggiunta di un server stdio con configurazione JSON
claude mcp add-json local-weather '{"type":"stdio","command":"/path/to/weather-
cli","args":["--api-key","abc123"],"env":{"CACHE_DIR":"/tmp"}}'

## Esempio: Aggiunta di un server HTTP con credenziali OAuth pre-configurate
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: Verifica che il server sia stato aggiunto

```
claude mcp get weather-api
```

> **Tip:**
>
> Suggerimenti:
>
> - Assicurati che il JSON sia correttamente sfuggito nella tua shell
> - Il JSON deve conformarsi allo schema di configurazione del server MCP
> - Puoi utilizzare `--scope user` per aggiungere il server alla tua configurazione utente invece di quella specifica del progetto

Importa server MCP da Claude Desktop

Se hai già configurato server MCP in Claude Desktop, puoi importarli:

Step 1: Importa server da Claude Desktop

```
## Sintassi di base
claude mcp add-from-claude-desktop
```

Step 2: Seleziona quali server importare

Dopo aver eseguito il comando, vedrai una finestra di dialogo interattiva che ti consente di selezionare quali server desideri importare.

Step 3: Verifica che i server siano stati importati

```
claude mcp list
```

> **Tip:**
>
> Suggerimenti:
>
> - Questa funzionalità funziona solo su macOS e Windows Subsystem for Linux (WSL)
> - Legge il file di configurazione di Claude Desktop dalla sua posizione standard su quelle piattaforme
> - Utilizza il flag `--scope user` per aggiungere server alla tua configurazione utente
> - I server importati avranno gli stessi nomi di Claude Desktop
> - Se server con gli stessi nomi esistono già, riceveranno un suffisso numerico (per esempio, `server_1`)

Utilizza server MCP da Claude.ai

Se hai effettuato l'accesso a Claude Code con un account Claude.ai, i server MCP che hai aggiunto in Claude.ai sono automaticamente disponibili in Claude Code:

Step 1: Configura server MCP in Claude.ai

Aggiungi server su claude.ai/settings/connectors. Nei piani Team ed Enterprise, solo gli amministratori possono aggiungere server.

Step 2: Autentica il server MCP

Completa eventuali passaggi di autenticazione richiesti in Claude.ai.

Step 3: Visualizza e gestisci i server in Claude Code

In Claude Code, utilizza il comando:

```
/mcp
```

I server Claude.ai appaiono nell'elenco con indicatori che mostrano che provengono da Claude.ai.

Per disabilitare i server MCP di claude.ai in Claude Code, imposta la variabile di ambiente `ENABLE_CLAUDEAI_MCP_SERVERS` su `false`:

```
ENABLE_CLAUDEAI_MCP_SERVERS=false claude
```

Utilizza Claude Code come server MCP

Puoi utilizzare Claude Code stesso come server MCP a cui altre applicazioni possono connettersi:

```
## Avvia Claude come server MCP stdio
claude mcp serve
```

Puoi utilizzarlo in Claude Desktop aggiungendo questa configurazione a claude_desktop_config.json:

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

> **Warning:**
>
> **Configurazione del percorso dell'eseguibile**: Il campo `command` deve fare riferimento all'eseguibile di Claude Code. Se il comando `claude` non è nel PATH del tuo sistema, dovrai specificare il percorso completo dell'eseguibile.

Per trovare il percorso completo:

```
which claude
```

Quindi utilizza il percorso completo nella tua configurazione:

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "/full/path/to/claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Senza il percorso dell'eseguibile corretto, incontrerai errori come `spawn claude ENOENT`.

Tip:

Suggerimenti:

- Il server fornisce accesso agli strumenti di Claude come View, Edit, LS, ecc.
- In Claude Desktop, prova a chiedere a Claude di leggere file in una directory, fare modifiche e altro ancora.
- Nota che questo server MCP sta solo esponendo gli strumenti di Claude Code al tuo client MCP, quindi il tuo client è responsabile dell'implementazione della conferma dell'utente per le singole chiamate di strumenti.

Limiti di output MCP e avvisi

Quando gli strumenti MCP producono output di grandi dimensioni, Claude Code aiuta a gestire l'utilizzo dei token per evitare di sovraccaricare il contesto della tua conversazione:

- **Soglia di avviso di output**: Claude Code visualizza un avviso quando l'output di qualsiasi strumento MCP supera 10.000 token
- **Limite configurabile**: Puoi regolare il massimo di token di output MCP consentiti utilizzando la variabile di ambiente `MAX_MCP_OUTPUT_TOKENS`
- **Limite predefinito**: Il massimo predefinito è 25.000 token

Per aumentare il limite per gli strumenti che producono output di grandi dimensioni:

```
## Imposta un limite più alto per gli output degli strumenti MCP
export MAX_MCP_OUTPUT_TOKENS=50000
claude
```

Questo è particolarmente utile quando si lavora con server MCP che:

- Interrogano grandi set di dati o database
- Generano report o documentazione dettagliati
- Elaborano file di log estesi o informazioni di debug

> **Warning:**
>
> Se incontri frequentemente avvisi di output con server MCP specifici, considera di aumentare il limite o configurare il server per impaginare o filtrare le sue risposte.

Utilizza risorse MCP

I server MCP possono esporre risorse che puoi referenziare utilizzando menzioni @, simile a come referenzi i file.

Referenzia risorse MCP

Step 1: Elenca le risorse disponibili

Digita `@` nel tuo prompt per vedere le risorse disponibili da tutti i server MCP connessi. Le risorse appaiono insieme ai file nel menu di completamento automatico.

Step 2: Referenzia una risorsa specifica

Utilizza il formato `@server:protocol://resource/path` per referenziare una risorsa:

```
Puoi analizzare @github:issue://123 e suggerire una correzione?
```

```
Per favore rivedi la documentazione API su @docs:file://api/authentication
```

Step 3: Referenze di risorse multiple

Puoi referenziare più risorse in un singolo prompt:

```
Confronta @postgres:schema://users con @docs:file://database/user-model
```

Tip:

Suggerimenti:

- Le risorse vengono recuperate automaticamente e incluse come allegati quando referenziate
- I percorsi delle risorse sono ricercabili con fuzzy search nel completamento automatico della menzione @
- Claude Code fornisce automaticamente strumenti per elencare e leggere risorse MCP quando i server le supportano
- Le risorse possono contenere qualsiasi tipo di contenuto fornito dal server MCP (testo, JSON, dati strutturati, ecc.)

Scala con MCP Tool Search

Quando hai molti server MCP configurati, le definizioni degli strumenti possono consumare una parte significativa della tua finestra di contesto. MCP Tool Search risolve questo caricando gli strumenti su richiesta invece di precaricarli tutti.

Come funziona

Claude Code abilita automaticamente Tool Search quando le descrizioni dei tuoi strumenti MCP consumerebbero più del 10% della finestra di contesto. Puoi regolare questa soglia o disabilitare completamente la ricerca degli strumenti. Quando attivato:

1. Gli strumenti MCP vengono differiti piuttosto che caricati nel contesto in anticipo
2. Claude utilizza uno strumento di ricerca per scoprire gli strumenti MCP rilevanti quando necessario
3. Solo gli strumenti che Claude effettivamente necessita vengono caricati nel contesto
4. Gli strumenti MCP continuano a funzionare esattamente come prima dal tuo punto di vista

Per gli autori di server MCP

Se stai costruendo un server MCP, il campo delle istruzioni del server diventa più utile con Tool Search abilitato. Le istruzioni del server aiutano Claude a capire quando cercare i tuoi strumenti, simile a come funzionano le skills.

Aggiungi istruzioni del server chiare e descrittive che spieghino:

- Quale categoria di attività gestiscono i tuoi strumenti
- Quando Claude dovrebbe cercare i tuoi strumenti
- Capacità chiave che il tuo server fornisce

Configura la ricerca degli strumenti

La ricerca degli strumenti viene eseguita in modalità auto per impostazione predefinita, il che significa che si attiva solo quando le definizioni dei tuoi strumenti MCP superano la soglia di contesto. Se hai pochi strumenti, vengono caricati normalmente senza ricerca degli strumenti. Questa funzionalità richiede modelli che supportano blocchi `tool_reference`: Sonnet 4 e successivi, oppure Opus 4 e successivi. I modelli Haiku non supportano la ricerca degli strumenti.

Controlla il comportamento della ricerca degli strumenti con la variabile di ambiente `ENABLE_TOOL_SEARCH`:

Valore	Comportamento
`auto`	Si attiva quando gli strumenti MCP superano il 10% del contesto (predefinito)
`auto:<N>`	Si attiva a una soglia personalizzata, dove `<N>` è una percentuale (ad es. `auto:5` per il 5%)
`true`	Sempre abilitato
`false`	Disabilitato, tutti gli strumenti MCP caricati in anticipo

```
## Utilizza una soglia personalizzata del 5%
ENABLE_TOOL_SEARCH=auto:5 claude

## Disabilita completamente la ricerca degli strumenti
ENABLE_TOOL_SEARCH=false claude
```

Oppure imposta il valore nel campo `env` del tuo settings.json.

Puoi anche disabilitare lo strumento MCPSearch specificamente utilizzando l'impostazione `disallowedTools`:

```
{
  "permissions": {
    "deny": ["MCPSearch"]
  }
}
```

Utilizza i prompt MCP come comandi

I server MCP possono esporre prompt che diventano disponibili come comandi in Claude Code.

Esegui i prompt MCP

Step 1: Scopri i prompt disponibili

Digita `/` per vedere tutti i comandi disponibili, inclusi quelli dai server MCP. I prompt MCP appaiono con il formato `/mcp__servername__promptname`.

Step 2: Esegui un prompt senza argomenti

```
/mcp__github__list_prs
```

Step 3: Esegui un prompt con argomenti

Molti prompt accettano argomenti. Passali separati da spazi dopo il comando:

```
/mcp__github__pr_review 456
```

```
/mcp__jira__create_issue "Bug nel flusso di accesso" high
```

Tip:

Suggerimenti:

- I prompt MCP vengono scoperto dinamicamente dai server connessi
- Gli argomenti vengono analizzati in base ai parametri definiti del prompt
- I risultati del prompt vengono iniettati direttamente nella conversazione
- I nomi del server e del prompt vengono normalizzati (gli spazi diventano trattini bassi)

Configurazione MCP gestita

Per le organizzazioni che necessitano di un controllo centralizzato sui server MCP, Claude Code supporta due opzioni di configurazione:

1. **Controllo esclusivo con `managed-mcp.json`**: Distribuisci un set fisso di server MCP che gli utenti non possono modificare o estendere
2. **Controllo basato su policy con allowlist/denylist**: Consenti agli utenti di aggiungere i propri server, ma limita quali sono consentiti

Queste opzioni consentono agli amministratori IT di:

- **Controllare a quali server MCP i dipendenti possono accedere**: Distribuisci un set standardizzato di server MCP approvati in tutta l'organizzazione
- **Prevenire server MCP non autorizzati**: Limita gli utenti dall'aggiungere server MCP non approvati
- **Disabilitare completamente MCP**: Rimuovi completamente la funzionalità MCP se necessario

Opzione 1: Controllo esclusivo con managed-mcp.json

Quando distribuisci un file `managed-mcp.json`, assume il **controllo esclusivo** su tutti i server MCP. Gli utenti non possono aggiungere, modificare o utilizzare alcun server MCP diverso da quelli definiti in questo file. Questo è l'approccio più semplice per le organizzazioni che desiderano un controllo completo.

Gli amministratori di sistema distribuiscono il file di configurazione a una directory a livello di sistema:

- macOS: `/Library/Application Support/ClaudeCode/managed-mcp.json`
- Linux e WSL: `/etc/claude-code/managed-mcp.json`
- Windows: `C:\Program Files\ClaudeCode\managed-mcp.json`

> **Note:**
>
> Questi sono percorsi a livello di sistema (non directory home dell'utente come `~/Library/...`) che richiedono privilegi di amministratore. Sono progettati per essere distribuiti dagli amministratori IT.

Il file `managed-mcp.json` utilizza lo stesso formato di un file `.mcp.json` standard:

```
{
  "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"
      }
    }
  }
}
```

Opzione 2: Controllo basato su policy con allowlist e denylist

Invece di assumere il controllo esclusivo, gli amministratori possono consentire agli utenti di configurare i propri server MCP mentre applicano restrizioni su quali server sono consentiti. Questo approccio utilizza `allowedMcpServers` e `deniedMcpServers` nel file di impostazioni gestite.

> **Note:**
>
> **Scelta tra le opzioni**: Utilizza l'Opzione 1 (`managed-mcp.json`) quando desideri distribuire un set fisso di server senza personalizzazione dell'utente. Utilizza l'Opzione 2 (allowlist/denylist) quando desideri consentire agli utenti di aggiungere i propri server entro i vincoli della policy.

Opzioni di restrizione

Ogni voce nell'allowlist o denylist può limitare i server in tre modi:

1. **Per nome del server** (`serverName`): Corrisponde al nome configurato del server

2. **Per comando** (`serverCommand`): Corrisponde al comando esatto e agli argomenti utilizzati per avviare i server stdio
3. **Per pattern URL** (`serverUrl`): Corrisponde agli URL dei server remoti con supporto per i caratteri jolly

Importante: Ogni voce deve avere esattamente uno tra `serverName` , `serverCommand` o `serverUrl` .

Configurazione di esempio

```
{
  "allowedMcpServers": [
    // Consenti per nome del server
    { "serverName": "github" },
    { "serverName": "sentry" },

    // Consenti per comando esatto (per server stdio)
    { "serverCommand": ["npx", "-y", "@modelcontextprotocol/server-filesystem"] },
    { "serverCommand": ["python", "/usr/local/bin/approved-server.py"] },

    // Consenti per pattern URL (per server remoti)
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ],
  "deniedMcpServers": [
    // Blocca per nome del server
    { "serverName": "dangerous-server" },

    // Blocca per comando esatto (per server stdio)
    { "serverCommand": ["npx", "-y", "unapproved-package"] },

    // Blocca per pattern URL (per server remoti)
    { "serverUrl": "https://*.untrusted.com/*" }
  ]
}
```

Come funzionano le restrizioni basate su comando

Corrispondenza esatta:

- Gli array di comando devono corrispondere **esattamente** - sia il comando che tutti gli argomenti nell'ordine corretto
- Esempio: `["npx", "-y", "server"]` NON corrisponderà a `["npx", "server"]` o `["npx", "-y", "server", "--flag"]`

Comportamento del server stdio:

- Quando l'allowlist contiene **qualsiasi** voce `serverCommand`, i server stdio **devono** corrispondere a uno di quei comandi
- I server stdio non possono passare solo per nome quando sono presenti restrizioni di comando
- Questo assicura che gli amministratori possono applicare quali comandi sono consentiti di eseguire

Comportamento del server non-stdio:

- I server remoti (HTTP, SSE, WebSocket) utilizzano la corrispondenza basata su URL quando esistono voci `serverUrl` nell'allowlist
- Se non esistono voci URL, i server remoti ricadono sulla corrispondenza basata su nome
- Le restrizioni di comando non si applicano ai server remoti

Come funzionano le restrizioni basate su URL

I pattern URL supportano i caratteri jolly utilizzando `*` per corrispondere a qualsiasi sequenza di caratteri. Questo è utile per consentire interi domini o sottodomini.

Esempi di caratteri jolly:

- `https://mcp.company.com/*` - Consenti tutti i percorsi su un dominio specifico
- `https://*.example.com/*` - Consenti qualsiasi sottodominio di example.com
- `http://localhost:*/*` - Consenti qualsiasi porta su localhost

Comportamento del server remoto:

- Quando l'allowlist contiene **qualsiasi** voce `serverUrl`, i server remoti **devono** corrispondere a uno di quei pattern URL
- I server remoti non possono passare solo per nome quando sono presenti restrizioni URL

- Questo assicura che gli amministratori possono applicare quali endpoint remoti sono consentiti

```
{
  "allowedMcpServers": [
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ]
}
```

Risultato:

- Server HTTP su `https://mcp.company.com/api`: ✅ Consentito (corrisponde al pattern URL)
- Server HTTP su `https://api.internal.corp/mcp`: ✅ Consentito (corrisponde al sottodominio jolly)
- Server HTTP su `https://external.com/mcp`: ❌ Bloccato (non corrisponde a nessun pattern URL)
- Server stdio con qualsiasi comando: ❌ Bloccato (nessuna voce di nome o comando per corrispondere)

```
{
  "allowedMcpServers": [
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

Risultato:

- Server stdio con `["npx", "-y", "approved-package"]`: ✅ Consentito (corrisponde al comando)
- Server stdio con `["node", "server.js"]`: ❌ Bloccato (non corrisponde al comando)
- Server HTTP denominato “my-api”: ❌ Bloccato (nessuna voce di nome per corrispondere)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

Risultato:

- Server stdio denominato “local-tool” con `["npx", "-y", "approved-package"]`: ✅ Consentito (corrisponde al comando)
- Server stdio denominato “local-tool” con `["node", "server.js"]`: ❌ Bloccato (le voci di comando esistono ma non corrisponde)
- Server stdio denominato “github” con `["node", "server.js"]`: ❌ Bloccato (i server stdio devono corrispondere ai comandi quando le voci di comando esistono)
- Server HTTP denominato “github”: ✅ Consentito (corrisponde al nome)
- Server HTTP denominato “other-api”: ❌ Bloccato (il nome non corrisponde)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverName": "internal-tool" }
  ]
}
```

Risultato:

- Server stdio denominato “github” con qualsiasi comando: ✅ Consentito (nessuna restrizione di comando)
- Server stdio denominato “internal-tool” con qualsiasi comando: ✅ Consentito (nessuna restrizione di comando)
- Server HTTP denominato “github”: ✅ Consentito (corrisponde al nome)
- Qualsiasi server denominato “other”: ❌ Bloccato (il nome non corrisponde)

Comportamento dell’allowlist (`allowedMcpServers`)

- `undefined` (predefinito): Nessuna restrizione - gli utenti possono configurare qualsiasi server MCP

- Array vuoto `[]`: Blocco completo - gli utenti non possono configurare alcun server MCP
- Elenco di voci: Gli utenti possono configurare solo server che corrispondono per nome, comando o pattern URL

Comportamento della denylist (`deniedMcpServers`)

- `undefined` (predefinito): Nessun server è bloccato
- Array vuoto `[]`: Nessun server è bloccato
- Elenco di voci: I server specificati sono esplicitamente bloccati in tutti gli ambiti

Note importanti

- **L'Opzione 1 e l'Opzione 2 possono essere combinate**: Se `managed-mcp.json` esiste, ha il controllo esclusivo e gli utenti non possono aggiungere server. Gli allowlist/denylist si applicano comunque ai server gestiti stessi.
- **La denylist ha precedenza assoluta**: Se un server corrisponde a una voce della denylist (per nome, comando o URL), sarà bloccato anche se è nell'allowlist
- **Le restrizioni basate su nome, comando e URL funzionano insieme**: un server passa se corrisponde a **una** voce di nome, una voce di comando o un pattern URL (a meno che non sia bloccato dalla denylist)

> **Note:**
>
> **Quando utilizzi `managed-mcp.json`**: Gli utenti non possono aggiungere server MCP tramite `claude mcp add` o file di configurazione. Le impostazioni `allowedMcpServers` e `deniedMcpServers` si applicano comunque per filtrare quali server gestiti vengono effettivamente caricati.

Estendi Claude con skills

> *Crea, gestisci e condividi skills per estendere le capacità di Claude in Claude Code. Include comandi personalizzati e skills raggruppate.*

Le skills estendono ciò che Claude può fare. Crea un file `SKILL.md` con istruzioni, e Claude lo aggiunge al suo toolkit. Claude utilizza le skills quando rilevante, oppure puoi invocare una direttamente con `/skill-name`.

> **Note:**
>
> Per i comandi integrati come `/help` e `/compact`, vedi modalità interattiva.
>
> **I comandi personalizzati sono stati uniti alle skills.** Un file in `.claude/commands/deploy.md` e una skill in `.claude/skills/deploy/SKILL.md` creano entrambi `/deploy` e funzionano allo stesso modo. I tuoi file `.claude/commands/` esistenti continuano a funzionare. Le skills aggiungono funzionalità opzionali: una directory per file di supporto, frontmatter per controllare se tu o Claude le invoca, e la capacità per Claude di caricarle automaticamente quando rilevante.

Le skills di Claude Code seguono lo standard aperto Agent Skills, che funziona su più strumenti AI. Claude Code estende lo standard con funzionalità aggiuntive come controllo dell'invocazione, esecuzione in subagent, e iniezione di contesto dinamico.

Skills raggruppate

Le skills raggruppate vengono fornite con Claude Code e sono disponibili in ogni sessione. A differenza dei comandi integrati, che eseguono logica fissa direttamente, le skills raggruppate sono basate su prompt: danno a Claude un playbook dettagliato e gli permettono di orchestrare il lavoro utilizzando i suoi strumenti. Questo significa che le skills raggruppate possono generare agenti paralleli, leggere file e adattarsi al tuo codebase.

Invochi le skills raggruppate allo stesso modo di qualsiasi altra skill: digita `/` seguito dal nome della skill.

- `/simplify`: esamina i tuoi file modificati di recente per problemi di riutilizzo del codice, qualità ed efficienza, quindi li corregge. Eseguila dopo aver implementato una funzionalità o una correzione di bug per pulire il tuo lavoro. Genera tre agenti di revisione in parallelo (riutilizzo del codice, qualità del codice, efficienza), aggrega i loro

risultati e applica le correzioni. Passa testo opzionale per concentrarti su problemi specifici: `/simplify focus on memory efficiency`.

- `/batch <instruction>`: orchestra modifiche su larga scala in un codebase in parallelo. Fornisci una descrizione della modifica e `/batch` ricerca il codebase, scompone il lavoro in 5-30 unità indipendenti e presenta un piano per la tua approvazione. Una volta approvato, genera un agente di background per unità, ciascuno in un git worktree isolato. Ogni agente implementa la sua unità, esegue i test e apre una pull request. Richiede un repository git. Esempio: `/batch migrate src/ from Solid to React`.
- `/debug [description]`: risolve i problemi della tua sessione Claude Code attuale leggendo il log di debug della sessione. Opzionalmente descrivi il problema per focalizzare l'analisi.
- `/loop [interval] <prompt>`: esegue un prompt ripetutamente a intervalli mentre la sessione rimane aperta. Claude analizza l'intervallo, pianifica un'attività cron ricorrente e conferma la cadenza. Utile per eseguire il polling di una distribuzione, prendersi cura di una PR, o eseguire periodicamente un'altra skill. Esempio: `/loop 5m check if the deploy finished`. Vedi Esegui prompt su una pianificazione.
- `/claude-api`: carica il materiale di riferimento dell'API Claude per il linguaggio del tuo progetto (Python, TypeScript, Java, Go, Ruby, C#, PHP, o cURL) e il riferimento Agent SDK per Python e TypeScript. Copre l'uso degli strumenti, lo streaming, i batch, gli output strutturati e le insidie comuni. Si attiva anche automaticamente quando il tuo codice importa `anthropic`, `@anthropic-ai/sdk`, o `claude_agent_sdk`.

Iniziare

Crea la tua prima skill

Questo esempio crea una skill che insegna a Claude a spiegare il codice usando diagrammi visivi e analogie. Poiché utilizza frontmatter predefinito, Claude può caricarla automaticamente quando chiedi come funziona qualcosa, oppure puoi invocarla direttamente con `/explain-code`.

Step 1: Crea la directory della skill

Crea una directory per la skill nella tua cartella di skills personali. Le skills personali sono disponibili su tutti i tuoi progetti.

```
mkdir -p ~/.claude/skills/explain-code
```

Step 2: Scrivi SKILL.md

Ogni skill ha bisogno di un file `SKILL.md` con due parti: frontmatter YAML (tra i marcatori `---`) che dice a Claude quando usare la skill, e contenuto markdown con istruzioni che Claude segue quando la skill viene invocata. Il campo `name` diventa il `/slash-command`, e la `description` aiuta Claude a decidere quando caricarla automaticamente.

Crea `~/.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: Testa la skill

Puoi testarla in due modi:

Lascia che Claude la invochi automaticamente chiedendo qualcosa che corrisponda alla descrizione:

```
How does this code work?
```

O invocala direttamente con il nome della skill:

```
/explain-code src/auth/login.ts
```

In entrambi i casi, Claude dovrebbe includere un'analogia e un diagramma ASCII nella sua spiegazione.

Dove vivono le skills

Dove archivi una skill determina chi può usarla:

Posizione	Percorso	Si applica a
Enterprise	Vedi impostazioni gestite	Tutti gli utenti della tua organizzazione
Personale	`~/.claude/skills/<skill-name>/SKILL.md`	Tutti i tuoi progetti
Progetto	`.claude/skills/<skill-name>/SKILL.md`	Solo questo progetto
Plugin	`<plugin>/skills/<skill-name>/SKILL.md`	Dove il plugin è abilitato

Quando le skills condividono lo stesso nome tra i livelli, le posizioni con priorità più alta vincono: enterprise > personale > progetto. Le skills dei plugin utilizzano uno spazio dei nomi `plugin-name:skill-name`, quindi non possono entrare in conflitto con altri livelli. Se hai file in `.claude/commands/`, funzionano allo stesso modo, ma se una skill e un comando condividono lo stesso nome, la skill ha la precedenza.

Scoperta automatica da directory annidate

Quando lavori con file in sottodirectory, Claude Code scopre automaticamente le skills da directory `.claude/skills/` annidate. Ad esempio, se stai modificando un file in `packages/frontend/`, Claude Code cerca anche le skills in `packages/frontend/.claude/skills/`. Questo supporta configurazioni monorepo dove i pacchetti hanno le loro skill.

Ogni skill è una directory con `SKILL.md` come punto di ingresso:

```
my-skill/
├── SKILL.md           # Main instructions (required)
├── template.md        # Template for Claude to fill in
├── examples/
│   └── sample.md      # Example output showing expected format
└── scripts/
    └── validate.sh    # Script Claude can execute
```

`SKILL.md` contiene le istruzioni principali ed è obbligatorio. Gli altri file sono opzionali e ti permettono di costruire skills più potenti: template per Claude da compilare, output di esempio che mostrano il formato previsto, script che Claude può eseguire, o documentazione di riferimento dettagliata. Fai riferimento a questi file da `SKILL.md` in modo che Claude sappia cosa contengono e quando caricarli. Vedi Aggiungi file di supporto per più dettagli.

> **Note:**
>
> I file in `.claude/commands/` continuano a funzionare e supportano lo stesso frontmatter. Le skills sono consigliate poiché supportano funzionalità aggiuntive come file di supporto.

Skills da directory aggiuntive

Le skills definite in `.claude/skills/` all'interno di directory aggiunte tramite `--add-dir` vengono caricate automaticamente e rilevate dal rilevamento dei cambiamenti in tempo reale, quindi puoi modificarle durante una sessione senza riavviare.

> **Note:**
>
> I file CLAUDE.md da directory `--add-dir` non vengono caricati per impostazione predefinita. Per caricarli, imposta `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1`. Vedi Carica da directory aggiuntive.

Configura le skills

Le skills vengono configurate tramite frontmatter YAML in cima a `SKILL.md` e il contenuto markdown che segue.

Tipi di contenuto della skill

I file delle skills possono contenere qualsiasi istruzione, ma pensare a come vuoi invocarle aiuta a guidare cosa includere:

Contenuto di riferimento aggiunge conoscenza che Claude applica al tuo lavoro attuale. Convenzioni, pattern, guide di stile, conoscenza del dominio. Questo contenuto viene eseguito inline in modo che Claude possa usarlo insieme al contesto della tua conversazione.

```
---
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
```

Contenuto di attività dà a Claude istruzioni passo-passo per un'azione specifica, come distribuzioni, commit o generazione di codice. Spesso sono azioni che vuoi invocare direttamente con `/skill-name` piuttosto che lasciare che Claude decida quando eseguirle. Aggiungi `disable-model-invocation: true` per impedire a Claude di attivarla automaticamente.

```
---
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
```

Il tuo `SKILL.md` può contenere qualsiasi cosa, ma pensare a come vuoi che la skill venga invocata (da te, da Claude, o entrambi) e dove vuoi che venga eseguita (inline o in un subagent) aiuta a guidare cosa includere. Per skills complesse, puoi anche aggiungere file di supporto per mantenere la skill principale focalizzata.

Riferimento frontmatter

Oltre al contenuto markdown, puoi configurare il comportamento della skill utilizzando campi frontmatter YAML tra i marcatori `---` in cima al tuo file `SKILL.md`:

```
---
name: my-skill
description: What this skill does
disable-model-invocation: true
allowed-tools: Read, Grep
---

Your skill instructions here...
```

Tutti i campi sono opzionali. Solo `description` è consigliato in modo che Claude sappia quando usare la skill.

Campo	Obbligatorio	Descrizione
`name`	No	Nome visualizzato per la skill. Se omesso, utilizza il nome della directory. Solo lettere minuscole, numeri e trattini (max 64 caratteri).
`description`	Consigliato	Cosa fa la skill e quando usarla. Claude usa questo per decidere quando applicare la skill. Se omesso, utilizza il primo paragrafo del contenuto markdown.
`argument-hint`	No	Suggerimento mostrato durante l'autocompletamento per indicare gli argomenti previsti. Esempio: `[issue-number]` o `[filename] [format]`.
`disable-model-invocation`	No	Imposta su `true` per impedire a Claude di caricare automaticamente questa skill. Usa per flussi di lavoro che vuoi attivare manualmente con `/name`. Predefinito: `false`.

Campo	Obbligatorio	Descrizione
`user-invocable`	No	Imposta su `false` per nascondere dal menu `/`. Usa per conoscenze di background che gli utenti non dovrebbero invocare direttamente. Predefinito: `true`.
`allowed-tools`	No	Strumenti che Claude può usare senza chiedere il permesso quando questa skill è attiva.
`model`	No	Modello da usare quando questa skill è attiva.
`context`	No	Imposta su `fork` per eseguire in un contesto subagent biforcato.
`agent`	No	Quale tipo di subagent usare quando `context: fork` è impostato.
`hooks`	No	Hooks scoped a questo ciclo di vita della skill. Vedi Hooks in skills e agents per il formato di configurazione.

Sostituzioni di stringhe disponibili

Le skills supportano la sostituzione di stringhe per valori dinamici nel contenuto della skill:

Variabile	Descrizione
`$ARGUMENTS`	Tutti gli argomenti passati quando si invoca la skill. Se `$ARGUMENTS` non è presente nel contenuto, gli argomenti vengono aggiunti come `ARGUMENTS: <value>`.
`$ARGUMENTS[N]`	Accedi a un argomento specifico per indice a base 0, come `$ARGUMENTS[0]` per il primo argomento.

Variabile	Descrizione
`$N`	Abbreviazione per `$ARGUMENTS[N]`, come `$0` per il primo argomento o `$1` per il secondo.
`${CLAUDE_SESSION_ID}`	L'ID della sessione attuale. Utile per il logging, la creazione di file specifici della sessione, o la correlazione dell'output della skill con le sessioni.
`${CLAUDE_SKILL_DIR}`	La directory contenente il file `SKILL.md` della skill. Per le skills dei plugin, questa è la sottodirectory della skill all'interno del plugin, non la radice del plugin. Usa questo nei comandi di iniezione bash per fare riferimento a script o file raggruppati con la skill, indipendentemente dalla directory di lavoro attuale.

Esempio usando sostituzioni:

```
---
name: session-logger
description: Log activity for this session
---

Log the following to logs/${CLAUDE_SESSION_ID}.log:

$ARGUMENTS
```

Aggiungi file di supporto

Le skills possono includere più file nella loro directory. Questo mantiene `SKILL.md` focalizzato sull'essenziale mentre permette a Claude di accedere a materiale di riferimento dettagliato solo quando necessario. Grandi documenti di riferimento, specifiche API, o raccolte di esempi non hanno bisogno di caricarsi nel contesto ogni volta che la skill viene eseguita.

```
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)
```

Fai riferimento ai file di supporto da `SKILL.md` in modo che Claude sappia cosa contiene ogni file e quando caricarlo:

```
### Additional resources

- For complete API details, see [reference.md](reference.md)
- For usage examples, see [examples.md](examples.md)
```

> **Tip:**
>
> Mantieni `SKILL.md` sotto 500 righe. Sposta il materiale di riferimento dettagliato in file separati.

Controlla chi invoca una skill

Per impostazione predefinita, sia tu che Claude potete invocare qualsiasi skill. Puoi digitare `/skill-name` per invocarla direttamente, e Claude può caricarla automaticamente quando rilevante per la tua conversazione. Due campi frontmatter ti permettono di limitare questo:

- `disable-model-invocation: true`: Solo tu puoi invocare la skill. Usa questo per flussi di lavoro con effetti collaterali o che vuoi controllare il timing, come `/commit`, `/deploy`, o `/send-slack-message`. Non vuoi che Claude decida di distribuire perché il tuo codice sembra pronto.
- `user-invocable: false`: Solo Claude può invocare la skill. Usa questo per conoscenze di background che non sono azionabili come comando. Una skill `legacy-system-context` spiega come funziona un vecchio sistema. Claude dovrebbe saperlo quando rilevante, ma `/legacy-system-context` non è un'azione significativa per gli utenti da intraprendere.

Questo esempio crea una skill di distribuzione che solo tu puoi attivare. Il campo `disable-model-invocation: true` impedisce a Claude di eseguirla automaticamente:

```
---
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
```

Ecco come i due campi influenzano l'invocazione e il caricamento del contesto:

Front-matter	Puoi invocare	Claude può invocare	Quando caricato nel contesto
(predefinito)	Sì	Sì	La descrizione è sempre nel contesto, la skill completa si carica quando invocata
`disable-model-invocation: true`	Sì	No	La descrizione non è nel contesto, la skill completa si carica quando la invochi
`user-invocable: false`	No	Sì	La descrizione è sempre nel contesto, la skill completa si carica quando invocata

> **Note:**
>
> In una sessione regolare, le descrizioni delle skills vengono caricate nel contesto in modo che Claude sappia cosa è disponibile, ma il contenuto completo della skill si carica solo quando invocato. Subagents con skills precaricate funzionano diversamente: il contenuto completo della skill viene iniettato all'avvio.

Limita l'accesso agli strumenti

Usa il campo `allowed-tools` per limitare quali strumenti Claude può usare quando una skill è attiva. Questa skill crea una modalità di sola lettura dove Claude può esplorare i file ma non modificarli:

```
---
name: safe-reader
description: Read files without making changes
allowed-tools: Read, Grep, Glob
---
```

Passa argomenti alle skills

Sia tu che Claude potete passare argomenti quando invocate una skill. Gli argomenti sono disponibili tramite il placeholder `$ARGUMENTS`.

Questa skill corregge un problema GitHub per numero. Il placeholder `$ARGUMENTS` viene sostituito con qualsiasi cosa segua il nome della skill:

```
---
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
```

Quando esegui `/fix-issue 123`, Claude riceve "Fix GitHub issue 123 following our coding standards..."

Se invochi una skill con argomenti ma la skill non include `$ARGUMENTS`, Claude Code aggiunge `ARGUMENTS: <your input>` alla fine del contenuto della skill in modo che Claude veda comunque quello che hai digitato.

Per accedere agli argomenti individuali per posizione, usa `$ARGUMENTS[N]` o il più breve `$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.
```

Eseguendo `/migrate-component SearchBar React Vue` sostituisce `$ARGUMENTS[0]` con `SearchBar`, `$ARGUMENTS[1]` con `React`, e `$ARGUMENTS[2]` con `Vue`. La stessa skill usando la scorciatoia `$N`:

```
---
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.
```

Pattern avanzati

Inietta contesto dinamico

La sintassi `!` command`` esegue comandi shell prima che il contenuto della skill venga inviato a Claude. L'output del comando sostituisce il placeholder, quindi Claude riceve dati effettivi, non il comando stesso.

Questa skill riassume una pull request recuperando dati PR live con GitHub CLI. I comandi `!` gh pr diff`` e altri vengono eseguiti per primi, e il loro output viene inserito nel prompt:

```
---
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...
```

Quando questa skill viene eseguita:

1. Ogni `!` command`` viene eseguito immediatamente (prima che Claude veda qualsiasi cosa)
2. L'output sostituisce il placeholder nel contenuto della skill
3. Claude riceve il prompt completamente renderizzato con dati PR effettivi

Questo è preprocessing, non qualcosa che Claude esegue. Claude vede solo il risultato finale.

> **Tip:**
>
> Per abilitare il pensiero esteso in una skill, includi la parola "ultrathink" da qualche parte nel contenuto della tua skill.

Esegui skills in un subagent

Aggiungi `context: fork` al tuo frontmatter quando vuoi che una skill venga eseguita in isolamento. Il contenuto della skill diventa il prompt che guida il subagent. Non avrà accesso alla tua cronologia di conversazione.

> **Warning:**

> `context: fork` ha senso solo per skills con istruzioni esplicite. Se la tua skill contiene linee guida come "usa queste convenzioni API" senza un'attività, il subagent riceve le linee guida ma nessun prompt azionabile, e ritorna senza output significativo.

Le skills e i subagents funzionano insieme in due direzioni:

Approccio	System prompt	Attività	Carica anche
Skill con `context: fork`	Dal tipo di agent (`Explore` , `Plan` , ecc.)	Contenuto SKILL.md	CLAUDE.md
Subagent con campo `skills`	Corpo markdown del subagent	Messaggio di delega di Claude	Skills precaricate + CLAUDE.md

Con `context: fork` , scrivi l'attività nella tua skill e scegli un tipo di agent per eseguirla. Per l'inverso (definire un subagent personalizzato che usa le skills come materiale di riferimento), vedi Subagents.

Esempio: Skill di ricerca usando agent Explore

Questa skill esegue ricerca in un agent Explore biforcato. Il contenuto della skill diventa l'attività, e l'agent fornisce strumenti di sola lettura ottimizzati per l'esplorazione del codebase:

```
---
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
```

Quando questa skill viene eseguita:

1. Viene creato un nuovo contesto isolato
2. Il subagent riceve il contenuto della skill come suo prompt ("Research $ARGUMENTS thoroughly...")
3. Il campo `agent` determina l'ambiente di esecuzione (modello, strumenti e permessi)
4. I risultati vengono riassunti e restituiti alla tua conversazione principale

Il campo `agent` specifica quale configurazione di subagent usare. Le opzioni includono agent integrati (`Explore` , `Plan` , `general-purpose`) o qualsiasi subagent personalizzato da `.claude/agents/` . Se omesso, usa `general-purpose` .

Limita l'accesso alle skills di Claude

Per impostazione predefinita, Claude può invocare qualsiasi skill che non abbia `disable-model-invocation: true` impostato. Le skills che definiscono `allowed-tools` concedono a Claude l'accesso a quegli strumenti senza approvazione per uso quando la skill è attiva. Le tue impostazioni di permesso governano comunque il comportamento di approvazione di base per tutti gli altri strumenti. I comandi integrati come `/compact` e `/init` non sono disponibili tramite lo strumento Skill.

Tre modi per controllare quali skills Claude può invocare:

Disabilita tutte le skills negando lo strumento Skill in `/permissions` :

```
## Add to deny rules:
Skill
```

Consenti o nega skills specifiche usando regole di permesso:

```
## Allow only specific skills
Skill(commit)
Skill(review-pr *)

## Deny specific skills
Skill(deploy *)
```

Sintassi di permesso: `Skill(name)` per corrispondenza esatta, `Skill(name *)` per corrispondenza di prefisso con qualsiasi argomento.

Nascondi skills individuali aggiungendo `disable-model-invocation: true` al loro frontmatter. Questo rimuove la skill dal contesto di Claude interamente.

> **Note:**
>
> Il campo `user-invocable` controlla solo la visibilità del menu, non l'accesso allo strumento Skill. Usa `disable-model-invocation: true` per bloccare l'invocazione programmatica.

Condividi skills

Le skills possono essere distribuite a diversi ambiti a seconda del tuo pubblico:

- **Skills di progetto**: Esegui il commit di `.claude/skills/` al controllo di versione
- **Plugin**: Crea una directory `skills/` nel tuo plugin
- **Gestito**: Distribuisci a livello di organizzazione tramite impostazioni gestite

Genera output visivo

Le skills possono raggruppare ed eseguire script in qualsiasi linguaggio, dando a Claude capacità oltre ciò che è possibile in un singolo prompt. Un pattern potente è generare output visivo: file HTML interattivi che si aprono nel tuo browser per esplorare dati, eseguire il debug o creare report.

Questo esempio crea un esploratore di codebase: una vista ad albero interattiva dove puoi espandere e comprimere directory, vedere le dimensioni dei file a colpo d'occhio e identificare i tipi di file per colore.

Crea la directory della Skill:

```
mkdir -p ~/.claude/skills/codebase-visualizer/scripts
```

Crea `~/.claude/skills/codebase-visualizer/SKILL.md`. La descrizione dice a Claude quando attivare questa Skill, e le istruzioni dicono a Claude di eseguire lo script raggruppato:

```
---
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
```

Crea `~/.claude/skills/codebase-visualizer/scripts/visualize.py`. Questo script scansiona un albero di directory e genera un file HTML autonomo con:

- Una **barra laterale di riepilogo** che mostra il conteggio dei file, il conteggio delle directory, la dimensione totale e il numero di tipi di file
- Un **grafico a barre** che suddivide il codebase per tipo di file (i primi 8 per dimensione)

- Un **albero comprimibile** dove puoi espandere e comprimere directory, con indicatori di tipo di file codificati per colore

Lo script richiede Python ma utilizza solo librerie integrate, quindi non ci sono pacchetti da installare:

```
#!/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()}')
```

Per testare, apri Claude Code in qualsiasi progetto e chiedi “Visualizza questo codebase.” Claude esegue lo script, genera `codebase-map.html`, e lo apre nel tuo browser.

Questo pattern funziona per qualsiasi output visivo: grafici di dipendenza, report di copertura dei test, documentazione API, o visualizzazioni di schema del database. Lo script raggruppato fa il lavoro pesante mentre Claude gestisce l’orchestrazione.

Risoluzione dei problemi

Skill non si attiva

Se Claude non usa la tua skill quando previsto:

1. Controlla che la descrizione includa parole chiave che gli utenti direbbero naturalmente
2. Verifica che la skill appaia in `What skills are available?`
3. Prova a riformulare la tua richiesta per corrispondere più strettamente alla descrizione
4. Invocala direttamente con `/skill-name` se la skill è invocabile dall'utente

Skill si attiva troppo spesso

Se Claude usa la tua skill quando non vuoi:

1. Rendi la descrizione più specifica
2. Aggiungi `disable-model-invocation: true` se vuoi solo l'invocazione manuale

Claude non vede tutte le mie skills

Le descrizioni delle skills vengono caricate nel contesto in modo che Claude sappia cosa è disponibile. Se hai molte skills, potrebbero superare il budget dei caratteri. Il budget si ridimensiona dinamicamente al 2% della finestra di contesto, con un fallback di 16.000 caratteri. Esegui `/context` per verificare un avviso sulle skills escluse.

Per sovrascrivere il limite, imposta la variabile di ambiente `SLASH_COMMAND_TOOL_CHAR_BUDGET`.

Risorse correlate

- **Subagents**: delega attività ad agenti specializzati
- **Plugins**: pacchetto e distribuisci skills con altre estensioni
- **Hooks**: automatizza flussi di lavoro attorno agli eventi degli strumenti
- **Memory**: gestisci file CLAUDE.md per il contesto persistente
- **Modalità interattiva**: comandi integrati e scorciatoie
- **Permissions**: controlla l'accesso agli strumenti e alle skills

Creare plugin

> *Crea plugin personalizzati per estendere Claude Code con skills, agents, hooks e MCP servers.*

I plugin ti permettono di estendere Claude Code con funzionalità personalizzate che possono essere condivise tra progetti e team. Questa guida copre la creazione dei tuoi plugin con skills, agents, hooks e MCP servers.

Stai cercando di installare plugin esistenti? Vedi Scopri e installa plugin. Per le specifiche tecniche complete, vedi Riferimento plugin.

Quando usare plugin rispetto alla configurazione standalone

Claude Code supporta due modi per aggiungere skills, agents e hooks personalizzati:

Approccio	Nomi skill	Migliore per
Standalone (directory `.claude/`)	`/hello`	Flussi di lavoro personali, personalizzazioni specifiche del progetto, esperimenti rapidi
Plugin (directory con `.claude-plugin/plugin.json`)	`/plugin-name:hello`	Condivisione con i colleghi, distribuzione alla comunità, rilasci versionati, riutilizzabili tra progetti

Usa la configurazione standalone quando:

- Stai personalizzando Claude Code per un singolo progetto
- La configurazione è personale e non ha bisogno di essere condivisa
- Stai sperimentando con skills o hooks prima di pacchettizzarli
- Vuoi nomi skill brevi come `/hello` o `/deploy`

Usa i plugin quando:

- Vuoi condividere funzionalità con il tuo team o comunità
- Hai bisogno degli stessi skills/agents in più progetti

- Vuoi il controllo della versione e aggiornamenti facili per le tue estensioni
- Stai distribuendo tramite un marketplace
- Sei d'accordo con skills con namespace come `/my-plugin:hello` (il namespace previene conflitti tra plugin)

> **Tip:**
>
> Inizia con la configurazione standalone in `.claude/` per un'iterazione rapida, poi converti in un plugin quando sei pronto a condividere.

Quickstart

Questo quickstart ti guida attraverso la creazione di un plugin con uno skill personalizzato. Creerai un manifest (il file di configurazione che definisce il tuo plugin), aggiungerai uno skill e lo testerai localmente usando il flag `--plugin-dir`.

Prerequisiti

- Claude Code installato e autenticato
- Claude Code versione 1.0.33 o successiva (esegui `claude --version` per verificare)

> **Note:**
>
> Se non vedi il comando `/plugin`, aggiorna Claude Code all'ultima versione. Vedi Troubleshooting per le istruzioni di aggiornamento.

Crea il tuo primo plugin

Step 1: Crea la directory del plugin

Ogni plugin vive nella sua directory contenente un manifest e i tuoi skills, agents o hooks. Creane uno ora:

```
mkdir my-first-plugin
```

Step 2: Crea il manifest del plugin

Il file manifest in `.claude-plugin/plugin.json` definisce l'identità del tuo plugin: il suo nome, descrizione e versione. Claude Code usa questi metadati per visualizzare il tuo plugin nel plugin manager.

Crea la directory `.claude-plugin` dentro la cartella del tuo plugin:

```
mkdir my-first-plugin/.claude-plugin
```

Poi crea `my-first-plugin/.claude-plugin/plugin.json` con questo contenuto:

```
{
"name": "my-first-plugin",
"description": "A greeting plugin to learn the basics",
"version": "1.0.0",
"author": {
"name": "Your Name"
}
}
```

Campo	Scopo
`name`	Identificatore univoco e namespace dello skill. Gli skill sono prefissati con questo (ad es., `/my-first-plugin:hello`).
`description`	Mostrato nel plugin manager quando si sfogliano o si installano plugin.
`version`	Traccia i rilasci usando il versionamento semantico.
`author`	Opzionale. Utile per l'attribuzione.

Per campi aggiuntivi come `homepage`, `repository` e `license`, vedi lo schema manifest completo.

Step 3: Aggiungi uno skill

Gli skill vivono nella directory `skills/`. Ogni skill è una cartella contenente un file `SKILL.md`. Il nome della cartella diventa il nome dello skill, prefissato con il namespace del plugin (`hello/` in un plugin denominato `my-first-plugin` crea `/my-first-plugin:hello`).

Crea una directory skill nella cartella del tuo plugin:

```
mkdir -p my-first-plugin/skills/hello
```

Poi crea `my-first-plugin/skills/hello/SKILL.md` con questo contenuto:

```
---
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: Testa il tuo plugin

Esegui Claude Code con il flag `--plugin-dir` per caricare il tuo plugin:

```
claude --plugin-dir ./my-first-plugin
```

Una volta che Claude Code si avvia, prova il tuo nuovo skill:

```
/my-first-plugin:hello
```

Vedrai Claude rispondere con un saluto. Esegui `/help` per vedere il tuo skill elencato sotto il namespace del plugin.

> **Note:**
>
> **Perché il namespace?** Gli skill del plugin hanno sempre il namespace (come `/greet:hello`) per prevenire conflitti quando più plugin hanno skill con lo stesso nome.
>
> Per cambiare il prefisso del namespace, aggiorna il campo `name` in `plugin.json`.

Step 5: Aggiungi argomenti dello skill

Rendi il tuo skill dinamico accettando input dell'utente. Il placeholder `$ARGUMENTS` cattura qualsiasi testo che l'utente fornisce dopo il nome dello skill.

Aggiorna il tuo file `SKILL.md`:

```
---
description: Greet the user with a personalized message
---

## Hello Skill

Greet the user named "$ARGUMENTS" warmly and ask how you can help them today. Make
the greeting personal and encouraging.
```

Esegui `/reload-plugins` per raccogliere i cambiamenti, poi prova lo skill con il tuo nome:

```
/my-first-plugin:hello Alex
```

Claude ti saluterà per nome. Per ulteriori informazioni sul passaggio di argomenti agli skill, vedi Skills.

Hai creato e testato con successo un plugin con questi componenti chiave:

- **Plugin manifest** (`.claude-plugin/plugin.json`): descrive i metadati del tuo plugin
- **Directory skills** (`skills/`): contiene i tuoi skill personalizzati
- **Argomenti dello skill** (`$ARGUMENTS`): cattura l'input dell'utente per il comportamento dinamico

> **Tip:**
>
> Il flag `--plugin-dir` è utile per lo sviluppo e il test. Quando sei pronto a condividere il tuo plugin con altri, vedi Crea e distribuisci un marketplace di plugin.

Panoramica della struttura del plugin

Hai creato un plugin con uno skill, ma i plugin possono includere molto di più: agents personalizzati, hooks, MCP servers e LSP servers.

> **Warning:**
>
> **Errore comune**: Non mettere `commands/` , `agents/` , `skills/` o `hooks/` dentro la directory `.claude-plugin/` . Solo `plugin.json` va dentro `.claude-plugin/` . Tutte le altre directory devono essere al livello radice del plugin.

Directory	Posizione	Scopo
`.claude-plugin/`	Radice del plugin	Contiene il manifest `plugin.json` (opzionale se i componenti usano posizioni predefinite)
`commands/`	Radice del plugin	Skills come file Markdown
`agents/`	Radice del plugin	Definizioni di agent personalizzati
`skills/`	Radice del plugin	Agent Skills con file `SKILL.md`
`hooks/`	Radice del plugin	Gestori di eventi in `hooks.json`
`.mcp.json`	Radice del plugin	Configurazioni del server MCP
`.lsp.json`	Radice del plugin	Configurazioni del server LSP per l'intelligenza del codice
`settings.json`	Radice del plugin	Impostazioni predefinite applicate quando il plugin è abilitato

Note:

Prossimi passi: Pronto ad aggiungere più funzionalità? Vai a Sviluppa plugin più complessi per aggiungere agents, hooks, MCP servers e LSP servers. Per le specifiche tecniche complete di tutti i componenti del plugin, vedi Riferimento plugin.

Sviluppa plugin più complessi

Una volta che hai familiarità con i plugin di base, puoi creare estensioni più sofisticate.

Aggiungi Skills al tuo plugin

I plugin possono includere Agent Skills per estendere le capacità di Claude. Gli skill sono invocati dal modello: Claude li usa automaticamente in base al contesto dell'attività.

Aggiungi una directory `skills/` alla radice del tuo plugin con cartelle Skill contenenti file `SKILL.md`:

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json
└── skills/
    └── code-review/
        └── SKILL.md
```

Ogni `SKILL.md` ha bisogno di frontmatter con campi `name` e `description`, seguiti da istruzioni:

```
---
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
```

Dopo aver installato il plugin, esegui `/reload-plugins` per caricare gli Skills. Per una guida completa sulla creazione di Skill inclusa la divulgazione progressiva e le restrizioni degli strumenti, vedi Agent Skills.

Aggiungi server LSP al tuo plugin

> **Tip:**
>
> Per linguaggi comuni come TypeScript, Python e Rust, installa i plugin LSP pre-costruiti dal marketplace ufficiale. Crea plugin LSP personalizzati solo quando hai bisogno di supporto per linguaggi non ancora coperti.

I plugin LSP (Language Server Protocol) danno a Claude l'intelligenza del codice in tempo reale. Se hai bisogno di supportare un linguaggio che non ha un plugin LSP ufficiale, puoi crearne uno tuo aggiungendo un file `.lsp.json` al tuo plugin:

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

Gli utenti che installano il tuo plugin devono avere il binario del language server installato sulla loro macchina.

Per le opzioni di configurazione LSP complete, vedi LSP servers.

Spedisci impostazioni predefinite con il tuo plugin

I plugin possono includere un file `settings.json` alla radice del plugin per applicare la configurazione predefinita quando il plugin è abilitato. Attualmente, è supportata solo la chiave `agent`.

Impostare `agent` attiva uno dei custom agents del plugin come thread principale, applicando il suo system prompt, le restrizioni degli strumenti e il modello. Questo consente a un plugin di cambiare il comportamento predefinito di Claude Code quando abilitato.

```
{
  "agent": "security-reviewer"
}
```

Questo esempio attiva l'agent `security-reviewer` definito nella directory `agents/` del plugin. Le impostazioni da `settings.json` hanno priorità rispetto alle `settings` dichiarate in `plugin.json`. Le chiavi sconosciute vengono silenziosamente ignorate.

Organizza plugin complessi

Per i plugin con molti componenti, organizza la tua struttura di directory per funzionalità. Per i layout di directory completi e i modelli di organizzazione, vedi Struttura della directory del plugin.

Testa i tuoi plugin localmente

Usa il flag `--plugin-dir` per testare i plugin durante lo sviluppo. Questo carica il tuo plugin direttamente senza richiedere l'installazione.

```
claude --plugin-dir ./my-plugin
```

Man mano che apporti modifiche al tuo plugin, esegui `/reload-plugins` per raccogliere gli aggiornamenti senza riavviare. Le modifiche alla configurazione del server LSP richiedono comunque un riavvio completo. Testa i componenti del tuo plugin:

- Prova i tuoi skill con `/plugin-name:skill-name`
- Verifica che gli agents appaiano in `/agents`
- Verifica che gli hooks funzionino come previsto

> **Tip:**
>
> Puoi caricare più plugin contemporaneamente specificando il flag più volte:
>
> ```
> claude --plugin-dir ./plugin-one --plugin-dir ./plugin-two
> ```

Esegui il debug dei problemi del plugin

Se il tuo plugin non funziona come previsto:

1. **Controlla la struttura**: Assicurati che le tue directory siano alla radice del plugin, non dentro `.claude-plugin/`
2. **Testa i componenti individualmente**: Controlla ogni comando, agent e hook separatamente
3. **Usa strumenti di validazione e debug**: Vedi Strumenti di debug e sviluppo per i comandi CLI e le tecniche di troubleshooting

Condividi i tuoi plugin

Quando il tuo plugin è pronto per essere condiviso:

1. **Aggiungi documentazione**: Includi un `README.md` con istruzioni di installazione e utilizzo
2. **Versiona il tuo plugin**: Usa il versionamento semantico nel tuo `plugin.json`
3. **Crea o usa un marketplace**: Distribuisci tramite marketplace di plugin per l'installazione
4. **Testa con altri**: Fai testare il plugin ai colleghi del team prima di una distribuzione più ampia

Una volta che il tuo plugin è in un marketplace, altri possono installarlo usando le istruzioni in Scopri e installa plugin.

Invia il tuo plugin al marketplace ufficiale

Per inviare un plugin al marketplace ufficiale di Anthropic, usa uno dei moduli di invio in-app:

- **Claude.ai**: claude.ai/settings/plugins/submit
- **Console**: platform.claude.com/plugins/submit

> **Note:**
>
> Per le specifiche tecniche complete, le tecniche di debug e le strategie di distribuzione, vedi Riferimento plugin.

Converti configurazioni esistenti in plugin

Se hai già skill o hooks nella tua directory `.claude/`, puoi convertirli in un plugin per una condivisione e distribuzione più facile.

Passaggi di migrazione

Step 1: Crea la struttura del plugin

Crea una nuova directory plugin:

```
mkdir -p my-plugin/.claude-plugin
```

Crea il file manifest in `my-plugin/.claude-plugin/plugin.json`:

```
{
  "name": "my-plugin",
  "description": "Migrated from standalone configuration",
  "version": "1.0.0"
}
```

Step 2: Copia i tuoi file esistenti

Copia le tue configurazioni esistenti nella directory del plugin:

```
## 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: Migra gli hook

Se hai hook nelle tue impostazioni, crea una directory hooks:

```
mkdir my-plugin/hooks
```

Crea `my-plugin/hooks/hooks.json` con la tua configurazione degli hook. Copia l'oggetto `hooks` dal tuo `.claude/settings.json` o `settings.local.json`, poiché il formato è lo stesso. Il comando riceve l'input dell'hook come JSON su stdin, quindi usa `jq` per estrarre il percorso del file:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [{ "type": "command", "command":
"jq -r '.tool_input.file_path' | xargs npm run lint:fix" }]
      }
    ]
  }
}
```

Step 4: Testa il tuo plugin migrato

Carica il tuo plugin per verificare che tutto funzioni:

```
claude --plugin-dir ./my-plugin
```

Testa ogni componente: esegui i tuoi comandi, verifica che gli agents appaiano in `/agents` e verifica che gli hook si attivino correttamente.

Cosa cambia durante la migrazione

Standalone (`.claude/`)	**Plugin**
Disponibile solo in un progetto	Può essere condiviso tramite marketplace
File in `.claude/commands/`	File in `plugin-name/commands/`
Hook in `settings.json`	Hook in `hooks/hooks.json`
Deve essere copiato manualmente per condividere	Installa con `/plugin install`

> **Note:**
>
> Dopo la migrazione, puoi rimuovere i file originali da `.claude/` per evitare duplicati. La versione del plugin avrà la precedenza quando caricata.

Prossimi passi

Ora che comprendi il sistema di plugin di Claude Code, ecco i percorsi suggeriti per diversi obiettivi:

Per gli utenti di plugin

- Scopri e installa plugin: sfoglia i marketplace e installa i plugin
- Configura marketplace del team: configura plugin a livello di repository per il tuo team

Per gli sviluppatori di plugin

- Crea e distribuisci un marketplace: pacchetto e condividi i tuoi plugin
- Riferimento plugin: specifiche tecniche complete
- Approfondisci componenti specifici del plugin:
  - Skills: dettagli dello sviluppo dello skill
  - Subagents: configurazione e capacità dell'agent
  - Hooks: gestione degli eventi e automazione
  - MCP: integrazione di strumenti esterni

Riferimento dei plugin

> *Riferimento tecnico completo per il sistema di plugin di Claude Code, inclusi schemi, comandi CLI e specifiche dei componenti.*

> **Tip:**
>
> Stai cercando di installare plugin? Vedi Scopri e installa plugin. Per creare plugin, vedi Plugin. Per distribuire plugin, vedi Marketplace dei plugin.

Questo riferimento fornisce specifiche tecniche complete per il sistema di plugin di Claude Code, inclusi schemi dei componenti, comandi CLI e strumenti di sviluppo.

Un **plugin** è una directory autonoma di componenti che estende Claude Code con funzionalità personalizzate. I componenti del plugin includono skills, agents, hooks, MCP servers e LSP servers.

Riferimento dei componenti del plugin

Skills

I plugin aggiungono skills a Claude Code, creando scorciatoie `/name` che tu o Claude potete invocare.

Posizione: Directory `skills/` o `commands/` nella radice del plugin

Formato file: Le skills sono directory con `SKILL.md` ; i comandi sono semplici file markdown

Struttura della skill:

```
skills/
├── pdf-processor/
│   ├── SKILL.md
│   ├── reference.md (opzionale)
│   └── scripts/ (opzionale)
└── code-reviewer/
    └── SKILL.md
```

Comportamento dell'integrazione:

- Le skills e i comandi vengono rilevati automaticamente quando il plugin viene installato
- Claude può invocarli automaticamente in base al contesto dell'attività
- Le skills possono includere file di supporto insieme a SKILL.md

Per i dettagli completi, vedi Skills.

Agents

I plugin possono fornire subagents specializzati per attività specifiche che Claude può invocare automaticamente quando appropriato.

Posizione: Directory `agents/` nella radice del plugin

Formato file: File markdown che descrivono le capacità dell'agent

Struttura dell'agent:

```
---
name: agent-name
description: In cosa si specializza questo agent e quando Claude dovrebbe
invocarlo
---

Prompt di sistema dettagliato per l'agent che descrive il suo ruolo, competenza e
comportamento.
```

Punti di integrazione:

- Gli agents appaiono nell'interfaccia `/agents`
- Claude può invocare gli agents automaticamente in base al contesto dell'attività
- Gli agents possono essere invocati manualmente dagli utenti
- I plugin agents funzionano insieme agli agents Claude integrati

Per i dettagli completi, vedi Subagents.

Hooks

I plugin possono fornire gestori di eventi che rispondono automaticamente agli eventi di Claude Code.

Posizione: `hooks/hooks.json` nella radice del plugin, o inline in plugin.json

Formato: Configurazione JSON con matcher di eventi e azioni

Configurazione dell'hook:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format-code.sh"
          }
        ]
      }
    ]
  }
}
```

Eventi disponibili:

- `PreToolUse` : Prima che Claude usi qualsiasi tool
- `PostToolUse` : Dopo che Claude ha usato con successo qualsiasi tool
- `PostToolUseFailure` : Dopo che l'esecuzione del tool di Claude non riesce
- `PermissionRequest` : Quando viene mostrata una finestra di dialogo di autorizzazione
- `UserPromptSubmit` : Quando l'utente invia un prompt
- `Notification` : Quando Claude Code invia notifiche
- `Stop` : Quando Claude tenta di fermarsi
- `SubagentStart` : Quando un subagent viene avviato
- `SubagentStop` : Quando un subagent tenta di fermarsi
- `SessionStart` : All'inizio delle sessioni
- `SessionEnd` : Alla fine delle sessioni
- `TeammateIdle` : Quando un compagno di squadra di un team di agent sta per andare inattivo
- `TaskCompleted` : Quando un'attività viene contrassegnata come completata
- `PreCompact` : Prima che la cronologia della conversazione venga compattata

Tipi di hook:

- `command` : Esegui comandi shell o script
- `prompt` : Valuta un prompt con un LLM (usa il placeholder `$ARGUMENTS` per il contesto)
- `agent` : Esegui un verificatore agentico con strumenti per attività di verifica complesse

MCP servers

I plugin possono raggruppare server Model Context Protocol (MCP) per connettere Claude Code con strumenti e servizi esterni.

Posizione: `.mcp.json` nella radice del plugin, o inline in plugin.json

Formato: Configurazione standard del server MCP

Configurazione del server MCP:

```
{
  "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}"
    }
  }
}
```

Comportamento dell'integrazione:

- I server MCP del plugin si avviano automaticamente quando il plugin è abilitato
- I server appaiono come strumenti MCP standard nel toolkit di Claude
- Le capacità del server si integrano perfettamente con gli strumenti esistenti di Claude

- I server del plugin possono essere configurati indipendentemente dai server MCP dell'utente

LSP servers

> **Tip:**
>
> Stai cercando di usare plugin LSP? Installali dal marketplace ufficiale: cerca "lsp" nella scheda Discover di `/plugin`. Questa sezione documenta come creare plugin LSP per linguaggi non coperti dal marketplace ufficiale.

I plugin possono fornire server Language Server Protocol (LSP) per dare a Claude intelligenza del codice in tempo reale mentre lavora sul tuo codebase.

L'integrazione LSP fornisce:

- **Diagnostica istantanea**: Claude vede errori e avvisi immediatamente dopo ogni modifica
- **Navigazione del codice**: vai alla definizione, trova riferimenti e informazioni al passaggio del mouse
- **Consapevolezza del linguaggio**: informazioni sul tipo e documentazione per i simboli del codice

Posizione: `.lsp.json` nella radice del plugin, o inline in `plugin.json`

Formato: Configurazione JSON che mappa i nomi dei server di linguaggio alle loro configurazioni

Formato del file `.lsp.json`:

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

Inline in `plugin.json`:

```
{
  "name": "my-plugin",
  "lspServers": {
    "go": {
      "command": "gopls",
      "args": ["serve"],
      "extensionToLanguage": {
        ".go": "go"
      }
    }
  }
}
```

Campi obbligatori:

Campo	Descrizione
`command`	Il binario LSP da eseguire (deve essere in PATH)
`extensionToLanguage`	Mappa le estensioni di file agli identificatori di linguaggio

Campi opzionali:

Campo	Descrizione
`args`	Argomenti della riga di comando per il server LSP
`transport`	Trasporto di comunicazione: `stdio` (predefinito) o `socket`
`env`	Variabili di ambiente da impostare all'avvio del server
`initializationOptions`	Opzioni passate al server durante l'inizializzazione
`settings`	Impostazioni passate tramite `workspace/didChangeConfiguration`
`workspaceFolder`	Percorso della cartella di lavoro per il server
`startupTimeout`	Tempo massimo di attesa per l'avvio del server (millisecondi)
`shutdownTimeout`	Tempo massimo di attesa per l'arresto graduale (millisecondi)
`restartOnCrash`	Se riavviare automaticamente il server in caso di arresto anomalo

Campo	Descrizione
`maxRestarts`	Numero massimo di tentativi di riavvio prima di rinunciare

> **Warning:**
>
> **Devi installare il binario del server di linguaggio separatamente.** I plugin LSP configurano come Claude Code si connette a un server di linguaggio, ma non includono il server stesso. Se vedi `Executable not found in $PATH` nella scheda Errors di `/plugin`, installa il binario richiesto per il tuo linguaggio.

Plugin LSP disponibili:

Plugin	Server di linguaggio	Comando di installazione
`pyright-lsp`	Pyright (Python)	`pip install pyright` o `npm install -g pyright`
`typescript-lsp`	TypeScript Language Server	`npm install -g typescript-language-server typescript`
`rust-lsp`	rust-analyzer	Vedi installazione di rust-analyzer

Installa il server di linguaggio per primo, quindi installa il plugin dal marketplace.

Ambiti di installazione del plugin

Quando installi un plugin, scegli un **ambito** che determina dove il plugin è disponibile e chi altro può usarlo:

Ambito	File di impostazioni	Caso d'uso
`user`	`~/.claude/settings.json`	Plugin personali disponibili in tutti i progetti (predefinito)
`project`	`.claude/settings.json`	Plugin del team condivisi tramite controllo della versione

Ambito	File di impostazioni	Caso d'uso
`local`	`.claude/settings.local.json`	Plugin specifici del progetto, gitignored
`managed`	Impostazioni gestite	Plugin gestiti (sola lettura, solo aggiornamento)

I plugin utilizzano lo stesso sistema di ambito di altre configurazioni di Claude Code. Per le istruzioni di installazione e i flag di ambito, vedi Installa plugin. Per una spiegazione completa degli ambiti, vedi Ambiti di configurazione.

Schema del manifest del plugin

Il file `.claude-plugin/plugin.json` definisce i metadati e la configurazione del tuo plugin. Questa sezione documenta tutti i campi e le opzioni supportati.

Il manifest è opzionale. Se omesso, Claude Code scopre automaticamente i componenti nelle posizioni predefinite e deriva il nome del plugin dal nome della directory. Usa un manifest quando hai bisogno di fornire metadati o percorsi di componenti personalizzati.

Schema completo

```
{
  "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"
}
```

Campi obbligatori

Se includi un manifest, `name` è l'unico campo obbligatorio.

Campo	Tipo	Descrizione	Esempio
`name`	string	Identificatore univoco (kebab-case, senza spazi)	`"deployment-tools"`

Questo nome viene utilizzato per lo spazio dei nomi dei componenti. Ad esempio, nell'interfaccia utente, l'agent `agent-creator` per il plugin con nome `plugin-dev` apparirà come `plugin-dev:agent-creator`.

Campi di metadati

Campo	Tipo	Descrizione	Esempio
`version`	string	Versione semantica. Se impostata anche nella voce del marketplace, `plugin.json` ha priorità. Devi impostarla in un solo posto.	`"2.1.0"`
`description`	string	Breve spiegazione dello scopo del plugin	`"Deployment automation tools"`
`author`	object	Informazioni sull'autore	`{"name": "Dev Team", "email": "dev@company.com"}`
`homepage`	string	URL della documentazione	`"https://docs.example.com"`
`repository`	string	URL del codice sorgente	`"https://github.com/user/plugin"`
`license`	string	Identificatore della licenza	`"MIT"`, `"Apache-2.0"`
`keywords`	array	Tag di scoperta	`["deployment", "ci-cd"]`

Campi del percorso del componente

Campo	Tipo	Descrizione	Esempio
`commands`	string|array	File/directory di comandi aggiuntivi	`"./custom/cmd.md"` o `["./cmd1.md"]`
`agents`	string|array	File di agent aggiuntivi	`"./custom/agents/reviewer.md"`

Campo	Tipo	Descrizione	Esempio
`skills`	string|array	Directory di skills aggiuntive	`"./custom/skills/"`
`hooks`	string|array|object	Percorsi di configurazione degli hook o configurazione inline	`"./my-extra-hooks.json"`
`mcpServers`	string|array|object	Percorsi di configurazione MCP o configurazione inline	`"./my-extra-mcp-config.json"`
`outputStyles`	string|array	File/directory di stili di output aggiuntivi	`"./styles/"`
`lspServers`	string|array|object	Configurazioni Language Server Protocol per l'intelligenza del codice (vai alla definizione, trova riferimenti, ecc.)	`"./.lsp.json"`

Regole del comportamento del percorso

Importante: I percorsi personalizzati integrano le directory predefinite - non le sostituiscono.

- Se `commands/` esiste, viene caricato in aggiunta ai percorsi di comandi personalizzati
- Tutti i percorsi devono essere relativi alla radice del plugin e iniziare con `./`
- I comandi dai percorsi personalizzati utilizzano le stesse regole di denominazione e spazio dei nomi
- È possibile specificare più percorsi come array per flessibilità

Esempi di percorso:

```
{
  "commands": [
    "./specialized/deploy.md",
    "./utilities/batch-process.md"
  ],
  "agents": [
    "./custom-agents/reviewer.md",
    "./custom-agents/tester.md"
  ]
}
```

Variabili di ambiente

`${CLAUDE_PLUGIN_ROOT}` : Contiene il percorso assoluto della directory del tuo plugin. Usalo in hooks, server MCP e script per garantire percorsi corretti indipendentemente dalla posizione di installazione.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/process.sh"
          }
        ]
      }
    ]
  }
}
```

Caching del plugin e risoluzione dei file

I plugin vengono specificati in uno di due modi:

- Tramite `claude --plugin-dir`, per la durata di una sessione.
- Tramite un marketplace, installato per sessioni future.

Per motivi di sicurezza e verifica, Claude Code copia i plugin del *marketplace* nella **cache dei plugin** locale dell'utente (`~/.claude/plugins/cache`) piuttosto che usarli sul posto. Comprendere questo comportamento è importante quando si sviluppano plugin che fanno riferimento a file esterni.

Limitazioni dell'attraversamento dei percorsi

I plugin installati non possono fare riferimento a file al di fuori della loro directory. I percorsi che attraversano al di fuori della radice del plugin (come `../shared-utils`) non funzioneranno dopo l'installazione perché questi file esterni non vengono copiati nella cache.

Lavorare con dipendenze esterne

Se il tuo plugin ha bisogno di accedere a file al di fuori della sua directory, puoi creare link simbolici a file esterni all'interno della directory del tuo plugin. I symlink vengono rispettati durante il processo di copia:

```
## All'interno della directory del tuo plugin
ln -s /path/to/shared-utils ./shared-utils
```

Il contenuto collegato simbolicamente verrà copiato nella cache del plugin. Questo fornisce flessibilità mantenendo i vantaggi di sicurezza del sistema di caching.

Struttura della directory del plugin

Layout standard del plugin

Un plugin completo segue questa struttura:

```
enterprise-plugin/
├── .claude-plugin/           # Directory dei metadati (opzionale)
│   └── plugin.json             # manifest del plugin
├── commands/                 # Posizione predefinita del comando
│   ├── status.md
│   └── logs.md
├── agents/                   # Posizione predefinita dell'agent
│   ├── security-reviewer.md
│   ├── performance-tester.md
│   └── compliance-checker.md
├── skills/                   # Skills dell'agent
│   ├── code-reviewer/
│   │   └── SKILL.md
│   └── pdf-processor/
│       ├── SKILL.md
│       └── scripts/
├── hooks/                    # Configurazioni degli hook
│   ├── hooks.json           # Configurazione principale degli hook
│   └── security-hooks.json  # Hook aggiuntivi
├── settings.json             # Impostazioni predefinite per il plugin
├── .mcp.json                 # Definizioni del server MCP
├── .lsp.json                 # Configurazioni del server LSP
├── scripts/                  # Script degli hook e utilità
│   ├── security-scan.sh
│   ├── format-code.py
│   └── deploy.js
├── LICENSE                   # File di licenza
└── CHANGELOG.md              # Cronologia delle versioni
```

Warning:

La directory `.claude-plugin/` contiene il file `plugin.json`. Tutte le altre directory (commands/, agents/, skills/, hooks/) devono essere nella radice del plugin, non all'interno di `.claude-plugin/`.

Riferimento delle posizioni dei file

Componente	Posizione predefinita	Scopo
Manifest	`.claude-plugin/plugin.json`	Metadati e configurazione del plugin (opzionale)
Comandi	`commands/`	File Markdown della skill (legacy; usa `skills/` per le nuove skills)
Agents	`agents/`	File Markdown del subagent
Skills	`skills/`	Skills con struttura `<name>/SKILL.md`
Hooks	`hooks/hooks.json`	Configurazione degli hook
Server MCP	`.mcp.json`	Definizioni del server MCP
Server LSP	`.lsp.json`	Configurazioni del server di linguaggio
Impostazioni	`settings.json`	Configurazione predefinita applicata quando il plugin è abilitato. Attualmente sono supportate solo le impostazioni `agent`

Riferimento dei comandi CLI

Claude Code fornisce comandi CLI per la gestione non interattiva dei plugin, utile per scripting e automazione.

plugin install

Installa un plugin dai marketplace disponibili.

```
claude plugin install <plugin> [options]
```

Argomenti:

- `<plugin>` : Nome del plugin o `plugin-name@marketplace-name` per un marketplace specifico

Opzioni:

Opzione	Descrizione	Predefinito
`-s, --scope <scope>`	Ambito di installazione: `user` , `project` , o `local`	`user`
`-h, --help`	Visualizza la guida per il comando	

L'ambito determina quale file di impostazioni viene aggiunto al plugin installato. Ad esempio, –scope project scrive in `enabledPlugins` in .claude/settings.json, rendendo il plugin disponibile a tutti coloro che clonano il repository del progetto.

Esempi:

```
## Installa nell'ambito utente (predefinito)
claude plugin install formatter@my-marketplace

## Installa nell'ambito del progetto (condiviso con il team)
claude plugin install formatter@my-marketplace --scope project

## Installa nell'ambito locale (gitignored)
claude plugin install formatter@my-marketplace --scope local
```

plugin uninstall

Rimuovi un plugin installato.

```
claude plugin uninstall <plugin> [options]
```

Argomenti:

- `<plugin>` : Nome del plugin o `plugin-name@marketplace-name`

Opzioni:

Opzione	Descrizione	Predefinito
`-s, --scope <scope>`	Disinstalla dall'ambito: `user`, `project`, o `local`	`user`
`-h, --help`	Visualizza la guida per il comando	

Alias: `remove`, `rm`

plugin enable

Abilita un plugin disabilitato.

```
claude plugin enable <plugin> [options]
```

Argomenti:

- `<plugin>`: Nome del plugin o `plugin-name@marketplace-name`

Opzioni:

Opzione	Descrizione	Predefinito
`-s, --scope <scope>`	Ambito da abilitare: `user`, `project`, o `local`	`user`
`-h, --help`	Visualizza la guida per il comando	

plugin disable

Disabilita un plugin senza disinstallarlo.

```
claude plugin disable <plugin> [options]
```

Argomenti:

- `<plugin>`: Nome del plugin o `plugin-name@marketplace-name`

Opzioni:

Opzione	Descrizione	Predefinito
`-s, --scope <scope>`	Ambito da disabilitare: `user`, `project`, o `local`	`user`

Opzione	Descrizione	Predefinito
`-h, --help`	Visualizza la guida per il comando	

plugin update

Aggiorna un plugin all'ultima versione.

```
claude plugin update <plugin> [options]
```

Argomenti:

- `<plugin>`: Nome del plugin o `plugin-name@marketplace-name`

Opzioni:

Opzione	Descrizione	Predefinito
`-s, --scope <scope>`	Ambito da aggiornare: `user`, `project`, `local`, o `managed`	`user`
`-h, --help`	Visualizza la guida per il comando	

Strumenti di debug e sviluppo

Comandi di debug

Usa `claude --debug` (o `/debug` all'interno del TUI) per vedere i dettagli del caricamento del plugin:

Questo mostra:

- Quali plugin vengono caricati
- Eventuali errori nei manifest del plugin
- Registrazione di comandi, agents e hook
- Inizializzazione del server MCP

Problemi comuni

Problema	Causa	Soluzione
Plugin non caricato	`plugin.json` non valido	Convalida la sintassi JSON con `claude plugin validate` o `/plugin validate`
Comandi non visualizzati	Struttura della directory errata	Assicurati che `commands/` sia alla radice, non in `.claude-plugin/`
Hook non attivati	Script non eseguibile	Esegui `chmod +x script.sh`
Server MCP non riesce	`${CLAUDE_PLUGIN_ROOT}` mancante	Usa la variabile per tutti i percorsi del plugin
Errori di percorso	Percorsi assoluti utilizzati	Tutti i percorsi devono essere relativi e iniziare con `./`
LSP `Executable not found in $PATH`	Server di linguaggio non installato	Installa il binario (ad es., `npm install -g typescript-language-server typescript`)

Messaggi di errore di esempio

Errori di convalida del manifest:

- `Invalid JSON syntax: Unexpected token } in JSON at position 142` : controlla la mancanza di virgole, virgole extra o stringhe non quotate
- `Plugin has an invalid manifest file at .claude-plugin/plugin.json. Validation errors: name: Required` : un campo obbligatorio è mancante
- `Plugin has a corrupt manifest file at .claude-plugin/plugin.json. JSON parse error: ...` : errore di sintassi JSON

Errori di caricamento del plugin:

- `Warning: No commands found in plugin my-plugin custom directory: ./cmds. Expected .md files or SKILL.md in subdirectories.` : il percorso del comando esiste ma non contiene file di comando validi

- `Plugin directory not found at path: ./plugins/my-plugin. Check that the marketplace entry has the correct path.`: il percorso `source` in marketplace.json punta a una directory inesistente
- `Plugin my-plugin has conflicting manifests: both plugin.json and marketplace entry specify components.`: rimuovi le definizioni di componenti duplicate o rimuovi `strict: false` nella voce del marketplace

Risoluzione dei problemi degli hook

Script dell'hook non in esecuzione:

1. Controlla che lo script sia eseguibile: `chmod +x ./scripts/your-script.sh`
2. Verifica la riga shebang: La prima riga dovrebbe essere `#!/bin/bash` o `#!/usr/bin/env bash`
3. Controlla che il percorso usi `${CLAUDE_PLUGIN_ROOT}`: `"command": "${CLAUDE_PLUGIN_ROOT}/scripts/your-script.sh"`
4. Testa lo script manualmente: `./scripts/your-script.sh`

Hook non attivato su eventi previsti:

1. Verifica che il nome dell'evento sia corretto (sensibile alle maiuscole): `PostToolUse`, non `postToolUse`
2. Controlla che il pattern del matcher corrisponda ai tuoi tool: `"matcher": "Write|Edit"` per le operazioni sui file
3. Conferma che il tipo di hook sia valido: `command`, `prompt`, o `agent`

Risoluzione dei problemi del server MCP

Server non si avvia:

1. Controlla che il comando esista e sia eseguibile
2. Verifica che tutti i percorsi utilizzino la variabile `${CLAUDE_PLUGIN_ROOT}`
3. Controlla i log del server MCP: `claude --debug` mostra gli errori di inizializzazione
4. Testa il server manualmente al di fuori di Claude Code

Strumenti del server non visualizzati:

1. Assicurati che il server sia configurato correttamente in `.mcp.json` o `plugin.json`
2. Verifica che il server implementi correttamente il protocollo MCP
3. Controlla i timeout di connessione nell'output di debug

Errori di struttura della directory

Sintomi: Il plugin si carica ma i componenti (comandi, agents, hook) sono mancanti.

Struttura corretta: I componenti devono essere nella radice del plugin, non all'interno di `.claude-plugin/`. Solo `plugin.json` appartiene a `.claude-plugin/`.

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json      ← Solo manifest qui
├── commands/            ← A livello di radice
├── agents/              ← A livello di radice
└── hooks/               ← A livello di radice
```

Se i tuoi componenti sono all'interno di `.claude-plugin/`, spostali nella radice del plugin.

Checklist di debug:

1. Esegui `claude --debug` e cerca i messaggi "loading plugin"
2. Controlla che ogni directory di componenti sia elencata nell'output di debug
3. Verifica che i permessi dei file consentano la lettura dei file del plugin

---

Riferimento di distribuzione e versioning

Gestione della versione

Segui il versionamento semantico per i rilasci del plugin:

```
{
  "name": "my-plugin",
  "version": "2.1.0"
}
```

Formato della versione: `MAJOR.MINOR.PATCH`

- **MAJOR**: Modifiche di rilievo (modifiche API incompatibili)
- **MINOR**: Nuove funzionalità (aggiunte compatibili con le versioni precedenti)
- **PATCH**: Correzioni di bug (correzioni compatibili con le versioni precedenti)

Best practice:

- Inizia con `1.0.0` per il tuo primo rilascio stabile
- Aggiorna la versione in `plugin.json` prima di distribuire le modifiche
- Documenta le modifiche in un file `CHANGELOG.md`
- Usa versioni pre-release come `2.0.0-beta.1` per i test

> **Warning:**
>
> Claude Code utilizza la versione per determinare se aggiornare il tuo plugin. Se modifichi il codice del tuo plugin ma non aumenti la versione in `plugin.json`, gli utenti esistenti del tuo plugin non vedranno le tue modifiche a causa del caching.
>
> Se il tuo plugin si trova all'interno di una directory marketplace, puoi gestire la versione tramite `marketplace.json` e omettere il campo `version` da `plugin.json`.

Vedi anche

- Plugin - Tutorial e utilizzo pratico
- Marketplace dei plugin - Creazione e gestione dei marketplace
- Skills - Dettagli dello sviluppo delle skill
- Subagents - Configurazione e capacità dell'agent
- Hooks - Gestione degli eventi e automazione
- MCP - Integrazione di strumenti esterni
- Impostazioni - Opzioni di configurazione per i plugin

Creare e distribuire un marketplace di plugin

> *Crea e ospita marketplace di plugin per distribuire estensioni Claude Code tra team e comunità.*

Un **plugin marketplace** è un catalogo che ti consente di distribuire plugin ad altri. I marketplace forniscono scoperta centralizzata, tracciamento delle versioni, aggiornamenti automatici e supporto per più tipi di fonte (repository git, percorsi locali e altro). Questa guida ti mostra come creare il tuo marketplace per condividere plugin con il tuo team o comunità.

Stai cercando di installare plugin da un marketplace esistente? Vedi Scopri e installa plugin precostruiti.

Panoramica

La creazione e la distribuzione di un marketplace comporta:

1. **Creazione di plugin**: crea uno o più plugin con comandi, agenti, hooks, MCP servers o LSP servers. Questa guida presuppone che tu abbia già plugin da distribuire; vedi Crea plugin per i dettagli su come crearli.
2. **Creazione di un file marketplace**: definisci un `marketplace.json` che elenca i tuoi plugin e dove trovarli (vedi Crea il file marketplace).
3. **Ospita il marketplace**: esegui il push su GitHub, GitLab o un altro host git (vedi Ospita e distribuisci marketplace).
4. **Condividi con gli utenti**: gli utenti aggiungono il tuo marketplace con `/plugin marketplace add` e installano singoli plugin (vedi Scopri e installa plugin).

Una volta che il tuo marketplace è attivo, puoi aggiornarlo eseguendo il push delle modifiche al tuo repository. Gli utenti aggiornano la loro copia locale con `/plugin marketplace update`.

Procedura dettagliata: creare un marketplace locale

Questo esempio crea un marketplace con un plugin: una skill `/quality-review` per le revisioni del codice. Creerai la struttura delle directory, aggiungerai una skill, creerai il manifest del plugin e il catalogo del marketplace, quindi lo installerai e lo testerai.

Step 1: Crea la struttura delle directory

```
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: Crea la skill

Crea un file `SKILL.md` che definisce cosa fa la skill `/quality-review`.

```
---
description: Rivedi il codice per bug, sicurezza e prestazioni
disable-model-invocation: true
---

Rivedi il codice che ho selezionato o i cambiamenti recenti per:
- Potenziali bug o casi limite
- Problemi di sicurezza
- Problemi di prestazioni
- Miglioramenti di leggibilità

Sii conciso e pratico.
```

Step 3: Crea il manifest del plugin

Crea un file `plugin.json` che descrive il plugin. Il manifest va nella directory `.claude-plugin/`.

```
{
  "name": "quality-review-plugin",
  "description": "Aggiunge una skill /quality-review per revisioni rapide del codice",
  "version": "1.0.0"
}
```

Step 4: Crea il file marketplace

Crea il catalogo marketplace che elenca il tuo plugin.

```
{
  "name": "my-plugins",
  "owner": {
    "name": "Your Name"
  },
  "plugins": [
    {
      "name": "quality-review-plugin",
      "source": "./plugins/quality-review-plugin",
      "description": "Aggiunge una skill /quality-review per revisioni rapide del
codice"
    }
  ]
}
```

Step 5: Aggiungi e installa

Aggiungi il marketplace e installa il plugin.

```
/plugin marketplace add ./my-marketplace
/plugin install quality-review-plugin@my-plugins
```

Step 6: Provalo

Seleziona del codice nel tuo editor ed esegui il tuo nuovo comando.

```
/review
```

Per saperne di più su cosa possono fare i plugin, inclusi hooks, agenti, MCP servers e LSP servers, vedi Plugin.

> **Note:**
>
> **Come vengono installati i plugin**: Quando gli utenti installano un plugin, Claude Code copia la directory del plugin in una posizione cache. Ciò significa che i plugin non possono fare riferimento a file al di fuori della loro directory utilizzando percorsi come `../shared-utils`, perché quei file non verranno copiati.

> Se hai bisogno di condividere file tra plugin, usa symlink (che vengono seguiti durante la copia). Vedi Plugin caching and file resolution per i dettagli.

Crea il file marketplace

Crea `.claude-plugin/marketplace.json` nella radice del tuo repository. Questo file definisce il nome del tuo marketplace, le informazioni del proprietario e un elenco di plugin con le loro fonti.

Ogni voce di plugin ha bisogno almeno di un `name` e di una `source` (da dove recuperarla). Vedi lo schema completo di seguito per tutti i campi disponibili.

```
{
  "name": "company-tools",
  "owner": {
    "name": "DevTools Team",
    "email": "devtools@example.com"
  },
  "plugins": [
    {
      "name": "code-formatter",
      "source": "./plugins/formatter",
      "description": "Formattazione automatica del codice al salvataggio",
      "version": "2.1.0",
      "author": {
        "name": "DevTools Team"
      }
    },
    {
      "name": "deployment-tools",
      "source": {
        "source": "github",
        "repo": "company/deploy-plugin"
      },
      "description": "Strumenti di automazione della distribuzione"
    }
  ]
}
```

Schema del marketplace

Campi obbligatori

Campo	Tipo	Descrizione	Esempio
`name`	string	Identificatore del marketplace (kebab-case, senza spazi). Questo è pubblico: gli utenti lo vedono quando installano plugin (ad esempio, `/plugin install my-tool@your-marketplace`).	`"acme-tools"`
`owner`	object	Informazioni sul manutentore del marketplace (vedi i campi di seguito)	
`plugins`	array	Elenco dei plugin disponibili	Vedi di seguito

Note:

Nomi riservati: I seguenti nomi di marketplace sono riservati per uso ufficiale di Anthropic e non possono essere utilizzati da marketplace di terze parti: `claude-code-marketplace`, `claude-code-plugins`, `claude-plugins-official`, `anthropic-marketplace`, `anthropic-plugins`, `agent-skills`, `life-sciences`. Anche i nomi che impersonano marketplace ufficiali (come `official-claude-plugins` o `anthropic-tools-v2`) sono bloccati.

Campi del proprietario

Campo	Tipo	Obbligatorio	Descrizione
`name`	string	Sì	Nome del manutentore o del team
`email`	string	No	Email di contatto per il manutentore

Metadati opzionali

Campo	Tipo	Descrizione
`metadata.description`	string	Breve descrizione del marketplace
`metadata.version`	string	Versione del marketplace
`metadata.pluginRoot`	string	Directory di base anteposta ai percorsi di fonte del plugin relativo (ad esempio, `"./plugins"` ti consente di scrivere `"source": "formatter"` invece di `"source": "./plugins/formatter"`)

Voci di plugin

Ogni voce di plugin nell'array `plugins` descrive un plugin e dove trovarlo. Puoi includere qualsiasi campo dallo schema del manifest del plugin (come `description` , `version` , `author` , `commands` , `hooks` , ecc.), più questi campi specifici del marketplace: `source` , `category` , `tags` e `strict` .

Campi obbligatori

Campo	Tipo	Descrizione
`name`	string	Identificatore del plugin (kebab-case, senza spazi). Questo è pubblico: gli utenti lo vedono quando installano (ad esempio, `/plugin install my-plugin@marketplace`).
`source`	string|object	Da dove recuperare il plugin (vedi Plugin sources di seguito)

Campi di plugin opzionali

Campi di metadati standard:

Campo	Tipo	Descrizione
`description`	string	Breve descrizione del plugin
`version`	string	Versione del plugin
`author`	object	Informazioni sull'autore del plugin (`name` obbligatorio, `email` opzionale)
`homepage`	string	URL della homepage o della documentazione del plugin
`repository`	string	URL del repository del codice sorgente
`license`	string	Identificatore di licenza SPDX (ad esempio, MIT, Apache-2.0)
`keywords`	array	Tag per la scoperta e la categorizzazione dei plugin
`category`	string	Categoria del plugin per l'organizzazione
`tags`	array	Tag per la ricercabilità
`strict`	boolean	Controlla se `plugin.json` è l'autorità per le definizioni dei componenti (predefinito: true). Vedi Strict mode di seguito.

Campi di configurazione dei componenti:

Campo	Tipo	Descrizione
`commands`	string|array	Percorsi personalizzati ai file o alle directory dei comandi

Campo	Tipo	Descrizione
`agents`	string|array	Percorsi personalizzati ai file degli agenti
`hooks`	string|object	Configurazione degli hook personalizzati o percorso al file degli hook
`mcpServers`	string|object	Configurazioni del server MCP o percorso alla configurazione MCP
`lspServers`	string|object	Configurazioni del server LSP o percorso alla configurazione LSP

Plugin sources

Le plugin sources indicano a Claude Code dove recuperare ogni singolo plugin elencato nel tuo marketplace. Questi sono impostati nel campo `source` di ogni voce di plugin in `marketplace.json`.

Una volta che un plugin viene clonato o copiato nella macchina locale, viene copiato nella cache del plugin locale con versione in `~/.claude/plugins/cache`.

Source	Tipo	Campi	Note
Percorso relativo	`string` (ad es. `"./my-plugin"`)	—	Directory locale all'interno del repository del marketplace. Deve iniziare con `./`
`github`	object	`repo`, `ref?`, `sha?`	
`url`	object	`url` (deve terminare con .git), `ref?`, `sha?`	Fonte URL Git

Source	Tipo	Campi	Note
`git-subdir`	object	`url`, `path`, `ref?`, `sha?`	Sottodirectory all'interno di un repository git. Clona in modo sparso per ridurre al minimo la larghezza di banda per i monorepo
`npm`	object	`package`, `version?`, `registry?`	Installato tramite `npm install`
`pip`	object	`package`, `version?`, `registry?`	Installato tramite pip

Note:

Marketplace sources vs plugin sources: Questi sono concetti diversi che controllano cose diverse.

- **Marketplace source** — dove recuperare il catalogo `marketplace.json` stesso. Impostato quando gli utenti eseguono `/plugin marketplace add` o nelle impostazioni `extraKnownMarketplaces`. Supporta `ref` (branch/tag) ma non `sha`.
- **Plugin source** — dove recuperare un singolo plugin elencato nel marketplace. Impostato nel campo `source` di ogni voce di plugin all'interno di `marketplace.json`. Supporta sia `ref` (branch/tag) che `sha` (commit esatto).

Ad esempio, un marketplace ospitato in `acme-corp/plugin-catalog` (marketplace source) può elencare un plugin recuperato da `acme-corp/code-formatter` (plugin source). La marketplace source e la plugin source puntano a repository diversi e sono fissate indipendentemente.

Percorsi relativi

Per i plugin nello stesso repository:

```
{
  "name": "my-plugin",
  "source": "./plugins/my-plugin"
}
```

> **Note:**
>
> I percorsi relativi funzionano solo quando gli utenti aggiungono il tuo marketplace tramite Git (GitHub, GitLab o URL git). Se gli utenti aggiungono il tuo marketplace tramite un URL diretto al file `marketplace.json`, i percorsi relativi non si risolveranno correttamente. Per la distribuzione basata su URL, usa invece GitHub, npm o fonti URL git. Vedi Troubleshooting per i dettagli.

Repository GitHub

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo"
  }
}
```

Puoi fissare a un branch, tag o commit specifico:

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

Campo	Tipo	Descrizione
`repo`	string	Obbligatorio. Repository GitHub nel formato `owner/repo`
`ref`	string	Opzionale. Branch o tag Git (predefinito al branch predefinito del repository)

Campo	Tipo	Descrizione
`sha`	string	Opzionale. SHA del commit git completo a 40 caratteri per fissare a una versione esatta

Repository Git

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git"
  }
}
```

Puoi fissare a un branch, tag o commit specifico:

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git",
    "ref": "main",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

Campo	Tipo	Descrizione
`url`	string	Obbligatorio. URL completo del repository git (deve terminare con `.git`)
`ref`	string	Opzionale. Branch o tag Git (predefinito al branch predefinito del repository)

Campo	Tipo	Descrizione
`sha`	string	Opzionale. SHA del commit git completo a 40 caratteri per fissare a una versione esatta

Sottodirectory Git

Usa `git-subdir` per puntare a un plugin che si trova all'interno di una sottodirectory di un repository git. Claude Code utilizza un clone parziale e sparso per recuperare solo la sottodirectory, riducendo al minimo la larghezza di banda per i grandi monorepo.

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin"
  }
}
```

Puoi fissare a un branch, tag o commit specifico:

```
{
  "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"
  }
}
```

Il campo `url` accetta anche una scorciatoia GitHub (`owner/repo`) o URL SSH (`git@github.com:owner/repo.git`).

Campo	Tipo	Descrizione
`url`	string	Obbligatorio. URL del repository Git, scorciatoia GitHub `owner/repo` o URL SSH
`path`	string	Obbligatorio. Percorso della sottodirectory all'interno del repo contenente il plugin (ad esempio, `"tools/claude-plugin"`)
`ref`	string	Opzionale. Branch o tag Git (predefinito al branch predefinito del repository)
`sha`	string	Opzionale. SHA del commit git completo a 40 caratteri per fissare a una versione esatta

Pacchetti npm

I plugin distribuiti come pacchetti npm vengono installati utilizzando `npm install`. Questo funziona con qualsiasi pacchetto nel registro npm pubblico o in un registro privato ospitato dal tuo team.

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin"
  }
}
```

Per fissare a una versione specifica, aggiungi il campo `version`:

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "2.1.0"
  }
}
```

Per installare da un registro privato o interno, aggiungi il campo `registry`:

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "^2.0.0",
    "registry": "https://npm.example.com"
  }
}
```

Campo	Tipo	Descrizione
`package`	string	Obbligatorio. Nome del pacchetto o pacchetto con scope (ad esempio, `@org/plugin`)
`version`	string	Opzionale. Versione o intervallo di versione (ad esempio, `2.1.0`, `^2.0.0`, `~1.5.0`)
`registry`	string	Opzionale. URL del registro npm personalizzato. Predefinito al registro npm del sistema (tipicamente npmjs.org)

Voci di plugin avanzate

Questo esempio mostra una voce di plugin che utilizza molti dei campi opzionali, inclusi percorsi personalizzati per comandi, agenti, hooks e server MCP:

```
{
  "name": "enterprise-tools",
  "source": {
    "source": "github",
    "repo": "company/enterprise-plugin"
  },
  "description": "Strumenti di automazione del flusso di lavoro aziendale",
  "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
}
```

Cose importanti da notare:

- `commands` e `agents` : Puoi specificare più directory o singoli file. I percorsi sono relativi alla radice del plugin.
- `${CLAUDE_PLUGIN_ROOT}` : Usa questa variabile nelle configurazioni degli hook e del server MCP per fare riferimento ai file all'interno della directory di installazione del plugin. Questo è necessario perché i plugin vengono copiati in una posizione cache quando installati.
- `strict: false` : Poiché è impostato su false, il plugin non ha bisogno del suo `plugin.json` . La voce del marketplace definisce tutto. Vedi Strict mode di seguito.

Strict mode

Il campo `strict` controlla se `plugin.json` è l'autorità per le definizioni dei componenti (comandi, agenti, hooks, skill, server MCP, stili di output).

Valore	Comportamento
`true` (predefinito)	`plugin.json` è l'autorità. La voce del marketplace può integrarla con componenti aggiuntivi e entrambe le fonti vengono unite.
`false`	La voce del marketplace è la definizione completa. Se il plugin ha anche un `plugin.json` che dichiara componenti, è un conflitto e il plugin non riesce a caricarsi.

Quando usare ogni modalità:

- `strict: true` : il plugin ha il suo `plugin.json` e gestisce i suoi componenti. La voce del marketplace può aggiungere comandi o hook extra in cima. Questo è il predefinito e funziona per la maggior parte dei plugin.
- `strict: false` : l'operatore del marketplace vuole il controllo completo. Il repository del plugin fornisce file grezzi e la voce del marketplace definisce quali di questi file sono esposti come comandi, agenti, hook, ecc. Utile quando il marketplace ristruttura

o cura i componenti di un plugin diversamente da quanto previsto dall'autore del plugin.

Ospita e distribuisci marketplace

Ospita su GitHub (consigliato)

GitHub fornisce il metodo di distribuzione più semplice:

1. **Crea un repository**: Configura un nuovo repository per il tuo marketplace
2. **Aggiungi il file marketplace**: Crea `.claude-plugin/marketplace.json` con le tue definizioni di plugin
3. **Condividi con i team**: Gli utenti aggiungono il tuo marketplace con `/plugin marketplace add owner/repo`

Vantaggi: Controllo della versione integrato, tracciamento dei problemi e funzionalità di collaborazione del team.

Ospita su altri servizi git

Qualsiasi servizio di hosting git funziona, come GitLab, Bitbucket e server self-hosted. Gli utenti aggiungono con l'URL completo del repository:

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

Repository privati

Claude Code supporta l'installazione di plugin da repository privati. Per l'installazione manuale e gli aggiornamenti, Claude Code utilizza i tuoi helper di credenziali git esistenti. Se `git clone` funziona per un repository privato nel tuo terminale, funziona anche in Claude Code. Gli helper di credenziali comuni includono `gh auth login` per GitHub, Keychain di macOS e `git-credential-store`.

Gli aggiornamenti automatici in background vengono eseguiti all'avvio senza helper di credenziali, poiché i prompt interattivi bloccherebbero l'avvio di Claude Code. Per abilitare gli aggiornamenti automatici per i marketplace privati, imposta il token di autenticazione appropriato nel tuo ambiente:

Provider	Variabili di ambiente	Note
GitHub	`GITHUB_TOKEN` o `GH_TOKEN`	Token di accesso personale o token di GitHub App

Provider	Variabili di ambiente	Note
GitLab	`GITLAB_TOKEN` o `GL_TOKEN`	Token di accesso personale o token di progetto
Bitbucket	`BITBUCKET_TOKEN`	Password dell'app o token di accesso al repository

Imposta il token nella configurazione della tua shell (ad esempio, `.bashrc`, `.zshrc`) o passalo quando esegui Claude Code:

```
export GITHUB_TOKEN=ghp_xxxxxxxxxxxxxxxxxxxx
```

> **Note:**
>
> Per gli ambienti CI/CD, configura il token come variabile di ambiente segreta. GitHub Actions fornisce automaticamente `GITHUB_TOKEN` per i repository nella stessa organizzazione.

Testa localmente prima della distribuzione

Testa il tuo marketplace localmente prima di condividerlo:

```
/plugin marketplace add ./my-local-marketplace
/plugin install test-plugin@my-local-marketplace
```

Per l'intera gamma di comandi add (GitHub, URL Git, percorsi locali, URL remoti), vedi Aggiungi marketplace.

Richiedi marketplace per il tuo team

Puoi configurare il tuo repository in modo che i membri del team vengano automaticamente invitati a installare il tuo marketplace quando fidano della cartella del progetto. Aggiungi il tuo marketplace a `.claude/settings.json`:

```
{
  "extraKnownMarketplaces": {
    "company-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

Puoi anche specificare quali plugin devono essere abilitati per impostazione predefinita:

```
{
  "enabledPlugins": {
    "code-formatter@company-tools": true,
    "deployment-tools@company-tools": true
  }
}
```

Per le opzioni di configurazione complete, vedi Plugin settings.

Restrizioni del marketplace gestito

Per le organizzazioni che richiedono un controllo rigoroso sulle fonti dei plugin, gli amministratori possono limitare quali marketplace di plugin gli utenti possono aggiungere utilizzando l'impostazione `strictKnownMarketplaces` nelle impostazioni gestite.

Quando `strictKnownMarketplaces` è configurato nelle impostazioni gestite, il comportamento della restrizione dipende dal valore:

Valore	Comportamento
Non definito (predefinito)	Nessuna restrizione. Gli utenti possono aggiungere qualsiasi marketplace
Array vuoto `[]`	Blocco completo. Gli utenti non possono aggiungere nuovi marketplace

Valore	Comportamento
Elenco di fonti	Gli utenti possono aggiungere solo marketplace che corrispondono esattamente all'elenco di autorizzazione

Configurazioni comuni

Disabilita tutti gli aggiunte di marketplace:

```
{
  "strictKnownMarketplaces": []
}
```

Consenti solo marketplace specifici:

```
{
  "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"
    }
  ]
}
```

Consenti tutti i marketplace da un server git interno utilizzando la corrispondenza del modello regex sull'host:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

Consenti marketplace basati su filesystem da una directory specifica utilizzando la corrispondenza del modello regex sul percorso:

```
{
  "strictKnownMarketplaces": [
    {
      "source": "pathPattern",
      "pathPattern": "^/opt/approved/"
    }
  ]
}
```

Usa `".*"` come `pathPattern` per consentire qualsiasi percorso del filesystem controllando comunque le fonti di rete con `hostPattern`.

Come funzionano le restrizioni

Le restrizioni vengono convalidate all'inizio del processo di installazione del plugin, prima di qualsiasi richiesta di rete o operazione del filesystem. Ciò impedisce i tentativi di accesso non autorizzato al marketplace.

L'elenco di autorizzazione utilizza la corrispondenza esatta per la maggior parte dei tipi di fonte. Affinché un marketplace sia consentito, tutti i campi specificati devono corrispondere esattamente:

- Per le fonti GitHub: `repo` è obbligatorio e `ref` o `path` devono corrispondere anche se specificati nell'elenco di autorizzazione
- Per le fonti URL: l'URL completo deve corrispondere esattamente
- Per le fonti `hostPattern`: l'host del marketplace viene confrontato con il modello regex

- Per le fonti `pathPattern`: il percorso del filesystem del marketplace viene confrontato con il modello regex

Poiché `strictKnownMarketplaces` è impostato nelle impostazioni gestite, le configurazioni individuali degli utenti e dei progetti non possono ignorare queste restrizioni.

Per i dettagli di configurazione completi inclusi tutti i tipi di fonte supportati e il confronto con `extraKnownMarketplaces`, vedi il riferimento strictKnownMarketplaces.

Risoluzione della versione e canali di rilascio

Le versioni dei plugin determinano i percorsi della cache e il rilevamento degli aggiornamenti. Puoi specificare la versione nel manifest del plugin (`plugin.json`) o nella voce del marketplace (`marketplace.json`).

> **Warning:**
>
> Quando possibile, evita di impostare la versione in entrambi i posti. Il manifest del plugin vince sempre silenziosamente, il che può causare l'ignoranza della versione del marketplace. Per i plugin con percorso relativo, imposta la versione nella voce del marketplace. Per tutte le altre fonti di plugin, impostala nel manifest del plugin.

Configura i canali di rilascio

Per supportare i canali di rilascio "stable" e "latest" per i tuoi plugin, puoi configurare due marketplace che puntano a diversi ref o SHA dello stesso repo. Puoi quindi assegnare i due marketplace a diversi gruppi di utenti tramite impostazioni gestite.

> **Warning:**
>
> Il `plugin.json` del plugin deve dichiarare una `version` diversa in ogni ref o commit fissato. Se due ref o commit hanno la stessa versione del manifest, Claude Code li tratta come identici e salta l'aggiornamento.

Esempio

```
{
  "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"
      }
    }
  ]
}
```

Assegna i canali ai gruppi di utenti

Assegna ogni marketplace al gruppo di utenti appropriato tramite impostazioni gestite. Ad esempio, il gruppo stabile riceve:

```
{
  "extraKnownMarketplaces": {
    "stable-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/stable-tools"
      }
    }
  }
}
```

Il gruppo early-access riceve invece `latest-tools`:

```
{
  "extraKnownMarketplaces": {
    "latest-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/latest-tools"
      }
    }
  }
}
```

Validazione e test

Testa il tuo marketplace prima di condividerlo.

Valida la sintassi JSON del tuo marketplace:

```
claude plugin validate .
```

O da Claude Code:

```
/plugin validate .
```

Aggiungi il marketplace per il test:

```
/plugin marketplace add ./path/to/marketplace
```

Installa un plugin di test per verificare che tutto funzioni:

```
/plugin install test-plugin@marketplace-name
```

Per i flussi di lavoro di test completi dei plugin, vedi Testa i tuoi plugin localmente. Per la risoluzione dei problemi tecnici, vedi Plugins reference.

Troubleshooting

Marketplace non carica

Sintomi: Non riesci ad aggiungere il marketplace o a vedere i plugin da esso

Soluzioni:

- Verifica che l'URL del marketplace sia accessibile
- Controlla che `.claude-plugin/marketplace.json` esista nel percorso specificato
- Assicurati che la sintassi JSON sia valida utilizzando `claude plugin validate` o `/plugin validate`
- Per i repository privati, conferma di avere i permessi di accesso

Errori di validazione del marketplace

Esegui `claude plugin validate .` o `/plugin validate .` dalla directory del tuo marketplace per verificare i problemi. Errori comuni:

Errore	Causa	Soluzione
`File not found: .claude-plugin/marketplace.json`	Manifest mancante	Crea `.claude-plugin/marketplace.json` con i campi obbligatori
`Invalid JSON syntax: Unexpected token...`	Errore di sintassi JSON	Controlla le virgole mancanti, le virgole extra o le stringhe non quotate
`Duplicate plugin name "x" found in marketplace`	Due plugin condividono lo stesso nome	Dai a ogni plugin un valore `name` univoco

Errore	Causa	Soluzione
`plugins[0].source: Path traversal not allowed`	Il percorso di fonte contiene `..`	Usa percorsi relativi alla radice del marketplace senza `..`

Avvisi (non bloccanti):

- `Marketplace has no plugins defined` : aggiungi almeno un plugin all'array `plugins`
- `No marketplace description provided` : aggiungi `metadata.description` per aiutare gli utenti a comprendere il tuo marketplace

Errori di installazione del plugin

Sintomi: Il marketplace appare ma l'installazione del plugin non riesce

Soluzioni:

- Verifica che gli URL di fonte del plugin siano accessibili
- Controlla che le directory dei plugin contengano i file richiesti
- Per le fonti GitHub, assicurati che i repository siano pubblici o che tu abbia accesso
- Testa manualmente le fonti dei plugin clonando/scaricando

L'autenticazione del repository privato non riesce

Sintomi: Errori di autenticazione durante l'installazione di plugin da repository privati

Soluzioni:

Per l'installazione manuale e gli aggiornamenti:

- Verifica di essere autenticato con il tuo provider git (ad esempio, esegui `gh auth status` per GitHub)
- Controlla che il tuo helper di credenziali sia configurato correttamente: `git config --global credential.helper`
- Prova a clonare il repository manualmente per verificare che le tue credenziali funzionino

Per gli aggiornamenti automatici in background:

- Imposta il token appropriato nel tuo ambiente: `echo $GITHUB_TOKEN`
- Controlla che il token abbia i permessi richiesti (accesso in lettura al repository)
- Per GitHub, assicurati che il token abbia lo scope `repo` per i repository privati
- Per GitLab, assicurati che il token abbia almeno lo scope `read_repository`

- Verifica che il token non sia scaduto

Le operazioni Git scadono

Sintomi: L'installazione del plugin o gli aggiornamenti del marketplace non riescono con un errore di timeout come "Git clone timed out after 120s" o "Git pull timed out after 120s".

Causa: Claude Code utilizza un timeout di 120 secondi per tutte le operazioni git, inclusa la clonazione dei repository dei plugin e il pull degli aggiornamenti del marketplace. I repository di grandi dimensioni o le connessioni di rete lente possono superare questo limite.

Soluzione: Aumenta il timeout utilizzando la variabile di ambiente `CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS`. Il valore è in millisecondi:

```
export CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS=300000  # 5 minuti
```

I plugin con percorsi relativi non riescono nei marketplace basati su URL

Sintomi: Hai aggiunto un marketplace tramite URL (come `https://example.com/marketplace.json`), ma i plugin con fonti di percorso relativo come `"./plugins/my-plugin"` non riescono a installare con errori "path not found".

Causa: I marketplace basati su URL scaricano solo il file `marketplace.json` stesso. Non scaricano i file dei plugin dal server. I percorsi relativi nella voce del marketplace fanno riferimento a file sul server remoto che non sono stati scaricati.

Soluzioni:

- **Usa fonti esterne**: Cambia le voci dei plugin per usare GitHub, npm o fonti URL git invece di percorsi relativi:

```
{ "name": "my-plugin", "source": { "source": "github", "repo": "owner/repo" } }
```

- **Usa un marketplace basato su Git**: Ospita il tuo marketplace in un repository Git e aggiungilo con l'URL git. I marketplace basati su Git clonano l'intero repository, rendendo i percorsi relativi funzionanti correttamente.

File non trovati dopo l'installazione

Sintomi: Il plugin si installa ma i riferimenti ai file non riescono, specialmente i file al di fuori della directory del plugin

Causa: I plugin vengono copiati in una directory cache piuttosto che utilizzati in-place. I percorsi che fanno riferimento a file al di fuori della directory del plugin (come `../shared-utils`) non funzioneranno perché quei file non vengono copiati.

Soluzioni: Vedi Plugin caching and file resolution per le soluzioni alternative inclusi symlink e ristrutturazione delle directory.

Per ulteriori strumenti di debug e problemi comuni, vedi Debugging and development tools.

Vedi anche

- Scopri e installa plugin precostruiti - Installazione di plugin da marketplace esistenti
- Plugin - Creazione dei tuoi plugin
- Plugins reference - Specifiche tecniche complete e schemi
- Plugin settings - Opzioni di configurazione dei plugin
- strictKnownMarketplaces reference - Restrizioni del marketplace gestito

Scopri e installa plugin precostruiti tramite marketplace

> *Trova e installa plugin dai marketplace per estendere Claude Code con nuovi comandi, agenti e funzionalità.*

I plugin estendono Claude Code con skills, agenti, hooks e MCP servers. I marketplace dei plugin sono cataloghi che vi aiutano a scoprire e installare queste estensioni senza doverle costruire da soli.

Cercate di creare e distribuire il vostro marketplace? Consultate Creare e distribuire un marketplace di plugin.

Come funzionano i marketplace

Un marketplace è un catalogo di plugin che qualcun altro ha creato e condiviso. Utilizzare un marketplace è un processo in due fasi:

Step 1: Aggiungere il marketplace

Questo registra il catalogo con Claude Code in modo che possiate sfogliare ciò che è disponibile. Nessun plugin viene installato ancora.

Step 2: Installare singoli plugin

Sfogliate il catalogo e installate i plugin che desiderate.

Pensatelo come aggiungere un app store: aggiungere lo store vi dà accesso per sfogliare la sua collezione, ma voi scegliete comunque quali app scaricare individualmente.

Marketplace ufficiale Anthropic

Il marketplace ufficiale Anthropic (`claude-plugins-official`) è automaticamente disponibile quando avviate Claude Code. Eseguite `/plugin` e andate alla scheda **Discover** per sfogliare ciò che è disponibile.

Per installare un plugin dal marketplace ufficiale:

```
/plugin install plugin-name@claude-plugins-official
```

> **Note:**
>
> Il marketplace ufficiale è mantenuto da Anthropic. Per inviare un plugin al marketplace ufficiale, utilizzate uno dei moduli di invio in-app:
>
> - **Claude.ai**: claude.ai/settings/plugins/submit
> - **Console**: platform.claude.com/plugins/submit
>
> Per distribuire plugin in modo indipendente, create il vostro marketplace e condividetelo con gli utenti.

Il marketplace ufficiale include diverse categorie di plugin:

Code intelligence

I plugin di code intelligence abilitano lo strumento LSP integrato di Claude Code, dando a Claude la capacità di saltare alle definizioni, trovare riferimenti e vedere errori di tipo immediatamente dopo le modifiche. Questi plugin configurano connessioni Language Server Protocol, la stessa tecnologia che alimenta la code intelligence di VS Code.

Questi plugin richiedono che il binario del language server sia installato sul vostro sistema. Se avete già un language server installato, Claude potrebbe chiedervi di installare il plugin corrispondente quando aprite un progetto.

Linguaggio	Plugin	Binario richiesto
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`

Potete anche creare il vostro plugin LSP per altri linguaggi.

> **Note:**
>
> Se vedete `Executable not found in $PATH` nella scheda `/plugin` Errors dopo aver installato un plugin, installate il binario richiesto dalla tabella sopra.

Cosa Claude guadagna dai plugin di code intelligence

Una volta che un plugin di code intelligence è installato e il suo binario del language server è disponibile, Claude guadagna due capacità:

- **Diagnostica automatica**: dopo ogni modifica di file che Claude fa, il language server analizza i cambiamenti e segnala errori e avvisi automaticamente. Claude vede errori di tipo, import mancanti e problemi di sintassi senza dover eseguire un compilatore o linter. Se Claude introduce un errore, lo nota e corregge il problema nello stesso turno. Questo non richiede alcuna configurazione oltre all'installazione del plugin. Potete vedere la diagnostica inline premendo **Ctrl+O** quando appare l'indicatore "diagnostics found".
- **Navigazione del codice**: Claude può utilizzare il language server per saltare alle definizioni, trovare riferimenti, ottenere informazioni sul tipo al passaggio del mouse, elencare simboli, trovare implementazioni e tracciare gerarchie di chiamate. Queste operazioni danno a Claude una navigazione più precisa rispetto alla ricerca basata su grep, anche se la disponibilità può variare a seconda del linguaggio e dell'ambiente.

Se riscontrate problemi, consultate Risoluzione dei problemi di code intelligence.

Integrazioni esterne

Questi plugin raggruppano MCP servers preconfigurati in modo che possiate connettere Claude a servizi esterni senza configurazione manuale:

- **Controllo del codice sorgente**: `github`, `gitlab`
- **Gestione dei progetti**: `atlassian` (Jira/Confluence), `asana`, `linear`, `notion`
- **Design**: `figma`
- **Infrastruttura**: `vercel`, `firebase`, `supabase`
- **Comunicazione**: `slack`
- **Monitoraggio**: `sentry`

Flussi di lavoro di sviluppo

Plugin che aggiungono comandi e agenti per attività di sviluppo comuni:

- **commit-commands**: Flussi di lavoro di commit Git inclusi commit, push e creazione di PR
- **pr-review-toolkit**: Agenti specializzati per la revisione delle pull request
- **agent-sdk-dev**: Strumenti per la costruzione con Claude Agent SDK
- **plugin-dev**: Toolkit per la creazione dei vostri plugin

Stili di output

Personalizzate come Claude risponde:

- **explanatory-output-style**: Approfondimenti educativi sulle scelte di implementazione
- **learning-output-style**: Modalità di apprendimento interattivo per la costruzione di competenze

Provate: aggiungere il marketplace demo

Anthropic mantiene anche un marketplace di plugin demo (`claude-code-plugins`) con plugin di esempio che mostrano cosa è possibile con il sistema di plugin. A differenza del marketplace ufficiale, dovete aggiungere questo manualmente.

Step 1: Aggiungere il marketplace

Da Claude Code, eseguite il comando `plugin marketplace add` per il marketplace `anthropics/claude-code`:

```
/plugin marketplace add anthropics/claude-code
```

Questo scarica il catalogo del marketplace e rende i suoi plugin disponibili per voi.

Step 2: Sfogliare i plugin disponibili

Eseguite `/plugin` per aprire il gestore dei plugin. Questo apre un'interfaccia a schede con quattro schede che potete scorrere utilizzando **Tab** (o **Shift+Tab** per andare indietro):

- **Discover**: sfogliate i plugin disponibili da tutti i vostri marketplace
- **Installed**: visualizzate e gestite i vostri plugin installati
- **Marketplaces**: aggiungete, rimuovete o aggiornate i vostri marketplace aggiunti
- **Errors**: visualizzate eventuali errori di caricamento dei plugin

Andate alla scheda **Discover** per vedere i plugin dal marketplace che avete appena aggiunto.

Step 3: Installare un plugin

Selezionate un plugin per visualizzare i suoi dettagli, quindi scegliete un ambito di installazione:

- **User scope**: installate per voi stessi in tutti i progetti
- **Project scope**: installate per tutti i collaboratori su questo repository
- **Local scope**: installate per voi stessi solo in questo repository

Ad esempio, selezionate **commit-commands** (un plugin che aggiunge comandi di flusso di lavoro git) e installatelo nel vostro ambito utente.

Potete anche installare direttamente dalla riga di comando:

```
/plugin install commit-commands@anthropics-claude-code
```

Consultate Ambiti di configurazione per saperne di più sugli ambiti.

Step 4: Utilizzare il vostro nuovo plugin

Dopo l'installazione, i comandi del plugin sono immediatamente disponibili. I comandi dei plugin sono nello spazio dei nomi del nome del plugin, quindi **commit-commands** fornisce comandi come `/commit-commands:commit`.

Provate eseguendo:

```
/commit-commands:commit
```

Questo mette in stage le vostre modifiche, genera un messaggio di commit e crea il commit.

Ogni plugin funziona diversamente. Controllate la descrizione del plugin nella scheda **Discover** o la sua homepage per sapere quali comandi e funzionalità fornisce.

Il resto di questa guida copre tutti i modi in cui potete aggiungere marketplace, installare plugin e gestire la vostra configurazione.

Aggiungere marketplace

Utilizzate il comando `/plugin marketplace add` per aggiungere marketplace da diverse fonti.

> **Tip:**
>
> **Scorciatoie**: Potete utilizzare `/plugin market` invece di `/plugin marketplace` e `rm` invece di `remove`.

- **Repository GitHub**: formato `owner/repo` (ad esempio, `anthropics/claude-code`)
- **URL Git**: qualsiasi URL di repository git (GitLab, Bitbucket, self-hosted)
- **Percorsi locali**: directory o percorsi diretti ai file `marketplace.json`
- **URL remoti**: URL diretti ai file `marketplace.json` ospitati

Aggiungere da GitHub

Aggiungete un repository GitHub che contiene un file `.claude-plugin/marketplace.json` utilizzando il formato `owner/repo`—dove `owner` è il nome utente GitHub o l'organizzazione e `repo` è il nome del repository.

Ad esempio, `anthropics/claude-code` si riferisce al repository `claude-code` di proprietà di `anthropics`:

```
/plugin marketplace add anthropics/claude-code
```

Aggiungere da altri host Git

Aggiungete qualsiasi repository git fornendo l'URL completo. Questo funziona con qualsiasi host Git, inclusi GitLab, Bitbucket e server self-hosted:

Utilizzando HTTPS:

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

Utilizzando SSH:

```
/plugin marketplace add git@gitlab.com:company/plugins.git
```

Per aggiungere un branch o tag specifico, aggiungete `#` seguito dal ref:

```
/plugin marketplace add https://gitlab.com/company/plugins.git#v1.0.0
```

Aggiungere da percorsi locali

Aggiungete una directory locale che contiene un file `.claude-plugin/marketplace.json`:

```
/plugin marketplace add ./my-marketplace
```

Potete anche aggiungere un percorso diretto a un file `marketplace.json`:

```
/plugin marketplace add ./path/to/marketplace.json
```

Aggiungere da URL remoti

Aggiungete un file `marketplace.json` remoto tramite URL:

```
/plugin marketplace add https://example.com/marketplace.json
```

> **Note:**
>
> I marketplace basati su URL hanno alcune limitazioni rispetto ai marketplace basati su Git. Se riscontrate errori "path not found" durante l'installazione di plugin, consultate Risoluzione dei problemi.

Installare plugin

Una volta aggiunti i marketplace, potete installare plugin direttamente (installa nell'ambito utente per impostazione predefinita):

```
/plugin install plugin-name@marketplace-name
```

Per scegliere un ambito di installazione diverso, utilizzate l'interfaccia interattiva: eseguite `/plugin`, andate alla scheda **Discover** e premete **Enter** su un plugin. Vedrete opzioni per:

- **User scope** (predefinito): installate per voi stessi in tutti i progetti
- **Project scope**: installate per tutti i collaboratori su questo repository (aggiunge a `.claude/settings.json`)
- **Local scope**: installate per voi stessi solo in questo repository (non condiviso con i collaboratori)

Potete anche vedere plugin con ambito **managed**—questi sono installati dagli amministratori tramite impostazioni gestite e non possono essere modificati.

Eseguite `/plugin` e andate alla scheda **Installed** per vedere i vostri plugin raggruppati per ambito.

> **Warning:**
>
> Assicuratevi di fidarvi di un plugin prima di installarlo. Anthropic non controlla quali MCP server, file o altro software sono inclusi nei plugin e non può verificare che funzionino come previsto. Controllate la homepage di ogni plugin per ulteriori informazioni.

Gestire i plugin installati

Eseguite `/plugin` e andate alla scheda **Installed** per visualizzare, abilitare, disabilitare o disinstallare i vostri plugin. Digitate per filtrare l'elenco per nome o descrizione del plugin.

Potete anche gestire i plugin con comandi diretti.

Disabilitate un plugin senza disinstallarlo:

```
/plugin disable plugin-name@marketplace-name
```

Riabilitate un plugin disabilitato:

```
/plugin enable plugin-name@marketplace-name
```

Rimuovete completamente un plugin:

```
/plugin uninstall plugin-name@marketplace-name
```

L'opzione `--scope` vi consente di indirizzare un ambito specifico con comandi CLI:

```
claude plugin install formatter@your-org --scope project
claude plugin uninstall formatter@your-org --scope project
```

Applicare le modifiche dei plugin senza riavviare

Quando installate, abilitate o disabilitate plugin durante una sessione, alcune modifiche (come nuovi comandi e hook) hanno effetto immediatamente. Altre, inclusi gli aggiornamenti del server LSP, richiedono un riavvio.

Per attivare tutte le modifiche dei plugin in sospeso senza riavviare, eseguite:

```
/reload-plugins
```

Claude Code ricarica tutti i plugin attivi e segnala cosa è stato caricato. Se sono stati aggiunti o aggiornati server LSP, vi farà sapere che richiedono un riavvio per avere effetto.

Gestire i marketplace

Potete gestire i marketplace tramite l'interfaccia interattiva `/plugin` o con comandi CLI.

Utilizzare l'interfaccia interattiva

Eseguite `/plugin` e andate alla scheda **Marketplaces** per:

- Visualizzare tutti i vostri marketplace aggiunti con le loro fonti e stato
- Aggiungere nuovi marketplace
- Aggiornare gli elenchi dei marketplace per recuperare i plugin più recenti
- Rimuovere i marketplace di cui non avete più bisogno

Utilizzare comandi CLI

Potete anche gestire i marketplace con comandi diretti.

Elencate tutti i marketplace configurati:

```
/plugin marketplace list
```

Aggiornate gli elenchi dei plugin da un marketplace:

```
/plugin marketplace update marketplace-name
```

Rimuovete un marketplace:

```
/plugin marketplace remove marketplace-name
```

> **Warning:**
>
> La rimozione di un marketplace disinstallerà tutti i plugin che avete installato da esso.

Configurare gli aggiornamenti automatici

Claude Code può aggiornare automaticamente i marketplace e i loro plugin installati all'avvio. Quando l'aggiornamento automatico è abilitato per un marketplace, Claude Code aggiorna i dati del marketplace e aggiorna i plugin installati alle loro versioni più recenti. Se sono stati aggiornati plugin, vedrete una notifica che vi chiede di eseguire `/reload-plugins`.

Attivate/disattivate l'aggiornamento automatico per singoli marketplace tramite l'interfaccia utente:

1. Eseguite `/plugin` per aprire il gestore dei plugin
2. Selezionate **Marketplaces**
3. Scegliete un marketplace dall'elenco
4. Selezionate **Enable auto-update** o **Disable auto-update**

I marketplace ufficiali Anthropic hanno l'aggiornamento automatico abilitato per impostazione predefinita. I marketplace di terze parti e di sviluppo locale hanno l'aggiornamento automatico disabilitato per impostazione predefinita.

Per disabilitare completamente tutti gli aggiornamenti automatici sia per Claude Code che per tutti i plugin, impostate la variabile di ambiente `DISABLE_AUTOUPDATER`. Consultate Aggiornamenti automatici per i dettagli.

Per mantenere gli aggiornamenti automatici dei plugin abilitati mentre disabilitate gli aggiornamenti di Claude Code, impostate `FORCE_AUTOUPDATE_PLUGINS=true` insieme a `DISABLE_AUTOUPDATER`:

```
export DISABLE_AUTOUPDATER=true
export FORCE_AUTOUPDATE_PLUGINS=true
```

Questo è utile quando volete gestire gli aggiornamenti di Claude Code manualmente ma ricevere comunque aggiornamenti automatici dei plugin.

Configurare i marketplace del team

Gli amministratori del team possono configurare l'installazione automatica del marketplace per i progetti aggiungendo la configurazione del marketplace a `.claude/settings.json`. Quando i membri del team si fidano della cartella del repository, Claude Code li invita a installare questi marketplace e plugin.

Aggiungete `extraKnownMarketplaces` al file `.claude/settings.json` del vostro progetto:

```
{
  "extraKnownMarketplaces": {
    "my-team-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

Per le opzioni di configurazione complete incluse `extraKnownMarketplaces` e `enabledPlugins`, consultate Impostazioni dei plugin.

Sicurezza

I plugin e i marketplace sono componenti altamente affidabili che possono eseguire codice arbitrario sulla vostra macchina con i vostri privilegi utente. Installate solo plugin e aggiungete marketplace da fonti di cui vi fidate. Le organizzazioni possono limitare quali marketplace gli utenti sono autorizzati ad aggiungere utilizzando restrizioni gestite dei marketplace.

Risoluzione dei problemi

Comando /plugin non riconosciuto

Se vedete "unknown command" o il comando `/plugin` non appare:

1. **Controllate la vostra versione**: Eseguite `claude --version`. I plugin richiedono la versione 1.0.33 o successiva.
2. **Aggiornate Claude Code**:
   - **Homebrew**: `brew upgrade claude-code`

   - **npm**: `npm update -g @anthropic-ai/claude-code`
   - **Programma di installazione nativo**: Rieseguite il comando di installazione da Setup

3. **Riavviate Claude Code**: Dopo l'aggiornamento, riavviate il vostro terminale ed eseguite `claude` di nuovo.

Problemi comuni

- **Marketplace non caricato**: Verificate che l'URL sia accessibile e che `.claude-plugin/marketplace.json` esista nel percorso
- **Errori di installazione dei plugin**: Controllate che gli URL di origine dei plugin siano accessibili e che i repository siano pubblici (o che abbiate accesso)
- **File non trovati dopo l'installazione**: I plugin vengono copiati in una cache, quindi i percorsi che fanno riferimento a file al di fuori della directory del plugin non funzioneranno
- **Le skill dei plugin non appaiono**: Cancellate la cache con `rm -rf ~/.claude/plugins/cache`, riavviate Claude Code e reinstallate il plugin.

Per la risoluzione dettagliata dei problemi con soluzioni, consultate Risoluzione dei problemi nella guida del marketplace. Per gli strumenti di debug, consultate Strumenti di debug e sviluppo.

Problemi di code intelligence

- **Language server non avviato**: verificate che il binario sia installato e disponibile nel vostro `$PATH`. Controllate la scheda `/plugin` Errors per i dettagli.
- **Utilizzo elevato della memoria**: i language server come `rust-analyzer` e `pyright` possono consumare memoria significativa su progetti di grandi dimensioni. Se riscontrate problemi di memoria, disabilitate il plugin con `/plugin disable <plugin-name>` e affidatevi invece agli strumenti di ricerca integrati di Claude.
- **Diagnostica falsa positiva nei monorepo**: i language server possono segnalare errori di import non risolti per i pacchetti interni se l'area di lavoro non è configurata correttamente. Questi non influiscono sulla capacità di Claude di modificare il codice.

Passaggi successivi

- **Costruite i vostri plugin**: Consultate Plugin per creare skills, agenti e hook
- **Create un marketplace**: Consultate Creare un marketplace di plugin per distribuire plugin al vostro team o comunità
- **Riferimento tecnico**: Consultate Riferimento dei plugin per le specifiche complete

Part 6: Advanced & Automation

Creare subagent personalizzati

> *Creare e utilizzare subagent AI specializzati in Claude Code per flussi di lavoro specifici per attività e una migliore gestione del contesto.*

I subagent sono assistenti AI specializzati che gestiscono tipi specifici di attività. Ogni subagent viene eseguito nel proprio context window con un prompt di sistema personalizzato, accesso a strumenti specifici e autorizzazioni indipendenti. Quando Claude incontra un'attività che corrisponde alla descrizione di un subagent, la delega a quel subagent, che lavora in modo indipendente e restituisce i risultati.

> **Note:**
>
> Se hai bisogno di più agenti che lavorano in parallelo e comunicano tra loro, consulta invece agent teams. I subagent lavorano all'interno di una singola sessione; i team di agenti coordinano tra sessioni separate.

I subagent ti aiutano a:

- **Preservare il contesto** mantenendo l'esplorazione e l'implementazione fuori dalla tua conversazione principale
- **Applicare vincoli** limitando quali strumenti un subagent può utilizzare
- **Riutilizzare configurazioni** tra progetti con subagent a livello utente
- **Specializzare il comportamento** con prompt di sistema focalizzati per domini specifici
- **Controllare i costi** instradando le attività a modelli più veloci e economici come Haiku

Claude utilizza la descrizione di ogni subagent per decidere quando delegare le attività. Quando crei un subagent, scrivi una descrizione chiara in modo che Claude sappia quando utilizzarlo.

Claude Code include diversi subagent integrati come **Explore**, **Plan** e **general-purpose**. Puoi anche creare subagent personalizzati per gestire attività specifiche. Questa pagina copre i subagent integrati, come creare i tuoi, opzioni di configurazione complete, pattern per lavorare con i subagent e subagent di esempio.

Subagent integrati

Claude Code include subagent integrati che Claude utilizza automaticamente quando appropriato. Ognuno eredita le autorizzazioni della conversazione principale con restrizioni aggiuntive sugli strumenti.

Explore

Un agente veloce e di sola lettura ottimizzato per la ricerca e l'analisi delle basi di codice.

- **Model**: Haiku (veloce, bassa latenza)
- **Tools**: Strumenti di sola lettura (accesso negato agli strumenti Write e Edit)
- **Purpose**: Scoperta di file, ricerca di codice, esplorazione della base di codice

Claude delega a Explore quando ha bisogno di cercare o comprendere una base di codice senza apportare modifiche. Questo mantiene i risultati dell'esplorazione fuori dal contesto della tua conversazione principale.

Quando invoca Explore, Claude specifica un livello di accuratezza: **quick** per ricerche mirate, **medium** per esplorazione equilibrata, o **very thorough** per analisi completa.

Plan

Un agente di ricerca utilizzato durante la plan mode per raccogliere contesto prima di presentare un piano.

- **Model**: Eredita dalla conversazione principale
- **Tools**: Strumenti di sola lettura (accesso negato agli strumenti Write e Edit)
- **Purpose**: Ricerca della base di codice per la pianificazione

Quando sei in plan mode e Claude ha bisogno di comprendere la tua base di codice, delega la ricerca al subagent Plan. Questo previene l'annidamento infinito (i subagent non possono generare altri subagent) mentre raccoglie comunque il contesto necessario.

General-purpose

Un agente capace per attività complesse e multi-step che richiedono sia esplorazione che azione.

- **Model**: Eredita dalla conversazione principale

- **Tools**: Tutti gli strumenti
- **Purpose**: Ricerca complessa, operazioni multi-step, modifiche al codice

Claude delega a general-purpose quando l'attività richiede sia esplorazione che modifica, ragionamento complesso per interpretare i risultati, o più step dipendenti.

Other

Claude Code include agenti helper aggiuntivi per attività specifiche. Questi vengono in genere invocati automaticamente, quindi non hai bisogno di utilizzarli direttamente.

Agent	Model	Quando Claude lo utilizza
Bash	Eredita	Esecuzione di comandi terminali in un contesto separato
statusline-setup	Sonnet	Quando esegui `/statusline` per configurare la tua status line
Claude Code Guide	Haiku	Quando fai domande sulle funzionalità di Claude Code

Oltre a questi subagent integrati, puoi creare i tuoi con prompt personalizzati, restrizioni sugli strumenti, modalità di autorizzazione, hooks e skills. Le sezioni seguenti mostrano come iniziare e personalizzare i subagent.

Quickstart: crea il tuo primo subagent

I subagent sono definiti in file Markdown con frontmatter YAML. Puoi crearli manualmente o utilizzare il comando `/agents`.

Questa procedura ti guida attraverso la creazione di un subagent a livello utente con il comando `/agent`. Il subagent esamina il codice e suggerisce miglioramenti per la base di codice.

Step 1: Apri l'interfaccia dei subagent

In Claude Code, esegui:

```
/agents
```

Step 2: Crea un nuovo agente a livello utente

Seleziona **Create new agent**, quindi scegli **User-level**. Questo salva il subagent in `~/.claude/agents/` in modo che sia disponibile in tutti i tuoi progetti.

Step 3: Genera con Claude

Seleziona **Generate with Claude**. Quando richiesto, descrivi il 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 genera il prompt di sistema e la configurazione. Premi `e` per aprirlo nel tuo editor se desideri personalizzarlo.

Step 4: Seleziona gli strumenti

Per un revisore di sola lettura, deseleziona tutto tranne **Read-only tools**. Se mantieni tutti gli strumenti selezionati, il subagent eredita tutti gli strumenti disponibili per la conversazione principale.

Step 5: Seleziona il modello

Scegli quale modello utilizza il subagent. Per questo agente di esempio, seleziona **Sonnet**, che bilancia capacità e velocità per analizzare i pattern del codice.

Step 6: Scegli un colore

Scegli un colore di sfondo per il subagent. Questo ti aiuta a identificare quale subagent è in esecuzione nell'interfaccia utente.

Step 7: Salva e provalo

Salva il subagent. È disponibile immediatamente (non è necessario riavviare). Provalo:

```
Use the code-improver agent to suggest improvements in this project
```

Claude delega al tuo nuovo subagent, che scansiona la base di codice e restituisce suggerimenti di miglioramento.

Ora hai un subagent che puoi utilizzare in qualsiasi progetto sulla tua macchina per analizzare le basi di codice e suggerire miglioramenti.

Puoi anche creare subagent manualmente come file Markdown, definirli tramite flag CLI, o distribuirli tramite plugin. Le sezioni seguenti coprono tutte le opzioni di configurazione.

Configura i subagent

Usa il comando /agents

Il comando `/agents` fornisce un'interfaccia interattiva per gestire i subagent. Esegui `/agents` per:

- Visualizzare tutti i subagent disponibili (integrati, utente, progetto e plugin)
- Creare nuovi subagent con configurazione guidata o generazione Claude
- Modificare la configurazione del subagent esistente e l'accesso agli strumenti
- Eliminare subagent personalizzati
- Vedere quali subagent sono attivi quando esistono duplicati

Questo è il modo consigliato per creare e gestire i subagent. Per la creazione manuale o l'automazione, puoi anche aggiungere file subagent direttamente.

Per elencare tutti i subagent configurati dalla riga di comando senza avviare una sessione interattiva, esegui `claude agents`. Questo mostra gli agenti raggruppati per fonte e indica quali vengono sovrascritti da definizioni di priorità più alta.

Scegli l'ambito del subagent

I subagent sono file Markdown con frontmatter YAML. Archiviali in posizioni diverse a seconda dell'ambito. Quando più subagent condividono lo stesso nome, la posizione con priorità più alta vince.

Location	Scope	Priority	Come creare
Flag CLI `--agents`	Sessione corrente	1 (massima)	Passa JSON quando avvii Claude Code
`.claude/agents/`	Progetto corrente	2	Interattivo o manuale
`~/.claude/agents/`	Tutti i tuoi progetti	3	Interattivo o manuale
Directory `agents/` del plugin	Dove il plugin è abilitato	4 (minima)	Installato con plugins

I **subagent di progetto** (`.claude/agents/`) sono ideali per subagent specifici di una base di codice. Archiviali nel controllo della versione in modo che il tuo team possa utilizzarli e migliorarli in modo collaborativo.

I **subagent utente** (`~/.claude/agents/`) sono subagent personali disponibili in tutti i tuoi progetti.

I **subagent definiti da CLI** vengono passati come JSON quando avvii Claude Code. Esistono solo per quella sessione e non vengono salvati su disco, rendendoli utili per test rapidi o script di automazione:

```
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"
  }
}'
```

Il flag `--agents` accetta JSON con gli stessi campi frontmatter dei subagent basati su file: `description` , `prompt` , `tools` , `disallowedTools` , `model` , `permissionMode` , `mcpServers` , `hooks` , `maxTurns` , `skills` e `memory` . Usa `prompt` per il prompt di sistema, equivalente al corpo markdown nei subagent basati su file. Consulta il riferimento CLI per il formato JSON completo.

I **subagent plugin** provengono da plugins che hai installato. Appaiono in `/agents` insieme ai tuoi subagent personalizzati. Consulta il riferimento dei componenti plugin per i dettagli sulla creazione di subagent plugin.

Scrivi file subagent

I file subagent utilizzano frontmatter YAML per la configurazione, seguito dal prompt di sistema in Markdown:

> **Note:**
>
> I subagent vengono caricati all'avvio della sessione. Se crei un subagent aggiungendo manualmente un file, riavvia la tua sessione o usa `/agents` per caricarlo immediatamente.

```
---
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.
```

Il frontmatter definisce i metadati e la configurazione del subagent. Il corpo diventa il prompt di sistema che guida il comportamento del subagent. I subagent ricevono solo questo prompt di sistema (più dettagli di base sull'ambiente come la directory di lavoro), non il prompt di sistema completo di Claude Code.

Campi frontmatter supportati

I seguenti campi possono essere utilizzati nel frontmatter YAML. Solo `name` e `description` sono obbligatori.

Field	Required	Description
`name`	Yes	Identificatore univoco utilizzando lettere minuscole e trattini
`description`	Yes	Quando Claude dovrebbe delegare a questo subagent
`tools`	No	Tools che il subagent può utilizzare. Eredita tutti gli strumenti se omesso
`disallowedTools`	No	Strumenti da negare, rimossi dall'elenco ereditato o specificato
`model`	No	Model da utilizzare: `sonnet`, `opus`, `haiku`, o `inherit`. Predefinito: `inherit`

Field	Required	Description
`permissionMode`	No	Permission mode: `default`, `acceptEdits`, `dontAsk`, `bypassPermissions`, o `plan`
`maxTurns`	No	Numero massimo di turni agentici prima che il subagent si fermi
`skills`	No	Skills da caricare nel contesto del subagent all'avvio. Il contenuto completo della skill viene iniettato, non solo reso disponibile per l'invocazione. I subagent non ereditano skills dalla conversazione principale
`mcpServers`	No	MCP servers disponibili per questo subagent. Ogni voce è un nome di server che fa riferimento a un server già configurato (ad es. `"slack"`) o una definizione inline con il nome del server come chiave e una configurazione MCP server completa come valore
`hooks`	No	Lifecycle hooks scoped a questo subagent
`memory`	No	Persistent memory scope: `user`, `project`, o `local`. Abilita l'apprendimento tra sessioni
`background`	No	Imposta su `true` per eseguire sempre questo subagent come background task. Predefinito: `false`

Field	Required	Description
`isolation`	No	Imposta su `worktree` per eseguire il subagent in un git worktree temporaneo, dandogli una copia isolata del repository. Il worktree viene automaticamente pulito se il subagent non apporta modifiche

Scegli un modello

Il campo `model` controlla quale AI model utilizza il subagent:

- **Model alias**: Usa uno degli alias disponibili: `sonnet`, `opus`, o `haiku`
- **inherit**: Usa lo stesso modello della conversazione principale
- **Omitted**: Se non specificato, predefinito a `inherit` (usa lo stesso modello della conversazione principale)

Controlla le capacità del subagent

Puoi controllare cosa possono fare i subagent attraverso l'accesso agli strumenti, le modalità di autorizzazione e le regole condizionali.

Strumenti disponibili

I subagent possono utilizzare qualsiasi strumento interno di Claude Code. Per impostazione predefinita, i subagent ereditano tutti gli strumenti dalla conversazione principale, inclusi gli strumenti MCP.

Per limitare gli strumenti, usa il campo `tools` (allowlist) o il campo `disallowedTools` (denylist):

```
---
name: safe-researcher
description: Research agent with restricted capabilities
tools: Read, Grep, Glob, Bash
disallowedTools: Write, Edit
---
```

Limita quali subagent possono essere generati

Quando un agente viene eseguito come thread principale con `claude --agent`, può generare subagent utilizzando lo strumento Agent. Per limitare quali tipi di subagent può generare, usa la sintassi `Agent(agent_type)` nel campo `tools`.

> **Note:**
>
> Nella versione 2.1.63, lo strumento Task è stato rinominato in Agent. I riferimenti `Task(...)` esistenti nelle impostazioni e nelle definizioni degli agenti continuano a funzionare come alias.

```
---
name: coordinator
description: Coordinates work across specialized agents
tools: Agent(worker, researcher), Read, Bash
---
```

Questo è un allowlist: solo i subagent `worker` e `researcher` possono essere generati. Se l'agente tenta di generare qualsiasi altro tipo, la richiesta fallisce e l'agente vede solo i tipi consentiti nel suo prompt. Per bloccare agenti specifici mentre consenti tutti gli altri, usa `permissions.deny` invece.

Per consentire la generazione di qualsiasi subagent senza restrizioni, usa `Agent` senza parentesi:

```
tools: Agent, Read, Bash
```

Se `Agent` viene omesso completamente dall'elenco `tools`, l'agente non può generare alcun subagent. Questa restrizione si applica solo agli agenti eseguiti come thread principale con `claude --agent`. I subagent non possono generare altri subagent, quindi `Agent(agent_type)` non ha effetto nelle definizioni dei subagent.

Modalità di autorizzazione

Il campo `permissionMode` controlla come il subagent gestisce i prompt di autorizzazione. I subagent ereditano il contesto di autorizzazione dalla conversazione principale ma possono sovrascrivere la modalità.

Mode	Behavior
`default`	Controllo di autorizzazione standard con prompt
`acceptEdits`	Auto-accetta modifiche ai file
`dontAsk`	Auto-nega prompt di autorizzazione (gli strumenti esplicitamente consentiti continuano a funzionare)
`bypassPermissions`	Salta tutti i controlli di autorizzazione
`plan`	Plan mode (esplorazione di sola lettura)

> **Warning:**
>
> Usa `bypassPermissions` con cautela. Salta tutti i controlli di autorizzazione, consentendo al subagent di eseguire qualsiasi operazione senza approvazione.

Se il principale utilizza `bypassPermissions`, questo ha la precedenza e non può essere sovrascritto.

Precarica skills nei subagent

Usa il campo `skills` per iniettare il contenuto della skill nel contesto di un subagent all'avvio. Questo dà al subagent conoscenza del dominio senza richiedere di scoprire e caricare skills durante l'esecuzione.

```
---
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.
```

Il contenuto completo di ogni skill viene iniettato nel contesto del subagent, non solo reso disponibile per l'invocazione. I subagent non ereditano skills dalla conversazione principale; devi elencarle esplicitamente.

> **Note:**
>
> Questo è l'inverso di eseguire una skill in un subagent. Con `skills` in un subagent, il subagent controlla il prompt di sistema e carica il contenuto della skill. Con `context: fork` in una skill, il contenuto della skill viene iniettato nell'agente che specifichi. Entrambi utilizzano lo stesso sistema sottostante.

Abilita memoria persistente

Il campo `memory` dà al subagent una directory persistente che sopravvive tra le conversazioni. Il subagent utilizza questa directory per accumulare conoscenza nel tempo, come pattern della base di codice, intuizioni di debug e decisioni architettoniche.

```
---
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.
```

Scegli un ambito in base a quanto ampiamente la memoria dovrebbe applicarsi:

Scope	Location	Use when
`user`	`~/.claude/agent-memory/<name-of-agent>/`	il subagent dovrebbe ricordare gli insegnamenti tra tutti i progetti
`project`	`.claude/agent-memory/<name-of-agent>/`	la conoscenza del subagent è specifica del progetto e condivisibile tramite controllo della versione
`local`	`.claude/agent-memory-local/<name-of-agent>/`	la conoscenza del subagent è specifica del progetto ma non dovrebbe essere archiviata nel controllo della versione

Quando la memoria è abilitata:

- Il prompt di sistema del subagent include istruzioni per leggere e scrivere nella directory di memoria.
- Il prompt di sistema del subagent include anche le prime 200 righe di `MEMORY.md` nella directory di memoria, con istruzioni per curare `MEMORY.md` se supera 200 righe.
- Gli strumenti Read, Write e Edit vengono automaticamente abilitati in modo che il subagent possa gestire i suoi file di memoria.

Suggerimenti per la memoria persistente

- `user` è l'ambito predefinito consigliato. Usa `project` o `local` quando la conoscenza del subagent è rilevante solo per una base di codice specifica.
- Chiedi al subagent di consultare la sua memoria prima di iniziare il lavoro: "Review this PR, and check your memory for patterns you've seen before."
- Chiedi al subagent di aggiornare la sua memoria dopo aver completato un'attività: "Now that you're done, save what you learned to your memory." Nel tempo, questo costruisce una base di conoscenza che rende il subagent più efficace.
- Includi istruzioni di memoria direttamente nel file markdown del subagent in modo che mantenga proattivamente la sua stessa base di conoscenza:

```
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.
```

Regole condizionali con hooks

Per un controllo più dinamico sull'utilizzo degli strumenti, usa gli hook `PreToolUse` per convalidare le operazioni prima che vengono eseguite. Questo è utile quando hai bisogno di consentire alcune operazioni di uno strumento mentre ne blocchi altre.

Questo esempio crea un subagent che consente solo query di database di sola lettura. L'hook `PreToolUse` esegue lo script specificato in `command` prima di ogni comando Bash:

```
---
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 passa l'input dell'hook come JSON tramite stdin ai comandi dell'hook. Lo script di convalida legge questo JSON, estrae il comando Bash e esce con codice 2 per bloccare le operazioni di scrittura:

```
#!/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
```

Consulta Hook input per lo schema di input completo e exit codes per come i codici di uscita influenzano il comportamento.

Disabilita subagent specifici

Puoi impedire a Claude di utilizzare subagent specifici aggiungendoli all'array `deny` nelle tue impostazioni. Usa il formato `Agent(subagent-name)` dove `subagent-name` corrisponde al campo name del subagent.

```
{
  "permissions": {
    "deny": ["Agent(Explore)", "Agent(my-custom-agent)"]
  }
}
```

Questo funziona sia per i subagent integrati che personalizzati. Puoi anche usare il flag CLI `--disallowedTools`:

```
claude --disallowedTools "Agent(Explore)"
```

Consulta la documentazione Permissions per ulteriori dettagli sulle regole di autorizzazione.

Definisci hook per i subagent

I subagent possono definire hooks che vengono eseguiti durante il ciclo di vita del subagent. Ci sono due modi per configurare gli hook:

1. **Nel frontmatter del subagent**: Definisci hook che vengono eseguiti solo mentre quel subagent è attivo
2. **In `settings.json`**: Definisci hook che vengono eseguiti nella sessione principale quando i subagent iniziano o si fermano

Hook nel frontmatter del subagent

Definisci gli hook direttamente nel file markdown del subagent. Questi hook vengono eseguiti solo mentre quel subagent specifico è attivo e vengono puliti quando finisce.

Tutti gli hook events sono supportati. Gli eventi più comuni per i subagent sono:

Event	Matcher input	When it fires
`PreToolUse`	Tool name	Prima che il subagent utilizzi uno strumento

Event	Matcher input	When it fires
`PostToolUse`	Tool name	Dopo che il subagent ha utilizzato uno strumento
`Stop`	(none)	Quando il subagent finisce (convertito in `SubagentStop` al runtime)

Questo esempio convalida i comandi Bash con l'hook `PreToolUse` ed esegue un linter dopo le modifiche ai file con `PostToolUse`:

```
---
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"
---
```

Gli hook `Stop` nel frontmatter vengono automaticamente convertiti in eventi `SubagentStop`.

Hook a livello di progetto per gli eventi dei subagent

Configura gli hook in `settings.json` che rispondono agli eventi del ciclo di vita dei subagent nella sessione principale.

Event	Matcher input	When it fires
`SubagentStart`	Agent type name	Quando un subagent inizia l'esecuzione

Event	Matcher input	When it fires
`SubagentStop`	Agent type name	Quando un subagent si completa

Entrambi gli eventi supportano matcher per indirizzare tipi di agenti specifici per nome. Questo esempio esegue uno script di configurazione solo quando il subagent `db-agent` inizia e uno script di pulizia quando qualsiasi subagent si ferma:

```
{
  "hooks": {
    "SubagentStart": [
      {
        "matcher": "db-agent",
        "hooks": [
          { "type": "command", "command": "./scripts/setup-db-connection.sh" }
        ]
      }
    ],
    "SubagentStop": [
      {
        "hooks": [
          { "type": "command", "command": "./scripts/cleanup-db-connection.sh" }
        ]
      }
    ]
  }
}
```

Consulta Hooks per il formato di configurazione dell'hook completo.

Lavora con i subagent

Comprendi la delegazione automatica

Claude delega automaticamente le attività in base alla descrizione dell'attività nella tua richiesta, al campo `description` nelle configurazioni dei subagent e al contesto attuale. Per incoraggiare la delegazione proattiva, includi frasi come "use proactively" nel campo description del tuo subagent.

Puoi anche richiedere un subagent specifico esplicitamente:

```
Use the test-runner subagent to fix failing tests
Have the code-reviewer subagent look at my recent changes
```

Esegui i subagent in primo piano o in background

I subagent possono essere eseguiti in primo piano (bloccante) o in background (concorrente):

- I **subagent in primo piano** bloccano la conversazione principale fino al completamento. I prompt di autorizzazione e le domande di chiarimento (come `AskUserQuestion`) vengono passati a te.
- I **subagent in background** vengono eseguiti contemporaneamente mentre continui a lavorare. Prima di avviare, Claude Code richiede le autorizzazioni degli strumenti di cui il subagent avrà bisogno, assicurando che abbia le approvazioni necessarie in anticipo. Una volta in esecuzione, il subagent eredita queste autorizzazioni e auto-nega qualsiasi cosa non pre-approvata. Se un subagent in background ha bisogno di fare domande di chiarimento, quella chiamata dello strumento fallisce ma il subagent continua.

Se un subagent in background fallisce a causa di autorizzazioni mancanti, puoi riprendere in primo piano per riprovare con prompt interattivi.

Claude decide se eseguire i subagent in primo piano o in background in base all'attività. Puoi anche:

- Chiedere a Claude di "run this in the background"
- Premere **Ctrl+B** per mettere in background un'attività in esecuzione

Per disabilitare tutta la funzionalità di background task, imposta la variabile di ambiente `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` su `1`. Consulta Environment variables.

Pattern comuni

Isola operazioni ad alto volume

Uno degli usi più efficaci per i subagent è isolare le operazioni che producono grandi quantità di output. L'esecuzione di test, il recupero della documentazione o l'elaborazione di file di log possono consumare un contesto significativo. Delegando questi a un subagent, l'output dettagliato rimane nel contesto del subagent mentre solo il riassunto rilevante ritorna alla tua conversazione principale.

```
Use a subagent to run the test suite and report only the failing tests with their error messages
```

Esegui ricerca parallela

Per indagini indipendenti, genera più subagent per lavorare simultaneamente:

```
Research the authentication, database, and API modules in parallel using separate subagents
```

Ogni subagent esplora la sua area in modo indipendente, quindi Claude sintetizza i risultati. Questo funziona meglio quando i percorsi di ricerca non dipendono l'uno dall'altro.

> **Warning:**
>
> Quando i subagent si completano, i loro risultati ritornano alla tua conversazione principale. L'esecuzione di molti subagent che ognuno restituisce risultati dettagliati può consumare un contesto significativo.

Per attività che necessitano di parallelismo sostenuto o superano la tua finestra di contesto, agent teams danno a ogni worker il suo contesto indipendente.

Concatena i subagent

Per flussi di lavoro multi-step, chiedi a Claude di utilizzare i subagent in sequenza. Ogni subagent completa la sua attività e restituisce i risultati a Claude, che poi passa il contesto rilevante al subagent successivo.

```
Use the code-reviewer subagent to find performance issues, then use the optimizer subagent to fix them
```

Scegli tra subagent e conversazione principale

Usa la **conversazione principale** quando:

- L'attività ha bisogno di frequenti scambi o raffinamento iterativo
- Più fasi condividono un contesto significativo (pianificazione → implementazione → test)
- Stai facendo un cambio rapido e mirato

- La latenza è importante. I subagent iniziano da zero e potrebbero aver bisogno di tempo per raccogliere il contesto

Usa i **subagent** quando:

- L'attività produce output dettagliato di cui non hai bisogno nel tuo contesto principale
- Vuoi applicare restrizioni specifiche sugli strumenti o autorizzazioni
- Il lavoro è autonomo e può restituire un riassunto

Considera invece Skills quando vuoi prompt o flussi di lavoro riutilizzabili che vengono eseguiti nel contesto della conversazione principale piuttosto che nel contesto isolato del subagent.

Per una domanda rapida su qualcosa già nella tua conversazione, usa `/btw` invece di un subagent. Vede il tuo contesto completo ma non ha accesso agli strumenti e la risposta viene scartata piuttosto che aggiunta alla cronologia.

> **Note:**
>
> I subagent non possono generare altri subagent. Se il tuo flusso di lavoro richiede delegazione annidata, usa Skills o concatena i subagent dalla conversazione principale.

Gestisci il contesto del subagent

Riprendi i subagent

Ogni invocazione di subagent crea una nuova istanza con contesto fresco. Per continuare il lavoro di un subagent esistente invece di ricominciare da capo, chiedi a Claude di riprendere.

I subagent ripresi mantengono la loro cronologia di conversazione completa, incluse tutte le chiamate di strumenti precedenti, i risultati e il ragionamento. Il subagent riprende esattamente da dove si era fermato piuttosto che ricominciare da capo.

Quando un subagent si completa, Claude riceve il suo ID agente. Per riprendere un subagent, chiedi a Claude di continuare il lavoro precedente:

```
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]
```

Puoi anche chiedere a Claude l'ID agente se vuoi fare riferimento ad esso esplicitamente, o trovare gli ID nei file di trascrizione in `~/.claude/projects/{project}/{sessionId}/subagents/`. Ogni trascrizione è archiviata come `agent-{agentId}.jsonl`.

Le trascrizioni dei subagent persistono indipendentemente dalla conversazione principale:

- **Compattazione della conversazione principale**: Quando la conversazione principale si compatta, le trascrizioni dei subagent non vengono influenzate. Sono archiviate in file separati.
- **Persistenza della sessione**: Le trascrizioni dei subagent persistono all'interno della loro sessione. Puoi riprendere un subagent dopo aver riavviato Claude Code riprendendo la stessa sessione.
- **Pulizia automatica**: Le trascrizioni vengono pulite in base all'impostazione `cleanupPeriodDays` (predefinito: 30 giorni).

Auto-compattazione

I subagent supportano la compattazione automatica utilizzando la stessa logica della conversazione principale. Per impostazione predefinita, la compattazione automatica si attiva a circa il 95% della capacità. Per attivare la compattazione prima, imposta `CLAUDE_AUTOCOMPACT_PCT_OVERRIDE` su una percentuale inferiore (ad esempio, `50`). Consulta environment variables per i dettagli.

Gli eventi di compattazione vengono registrati nei file di trascrizione dei subagent:

```
{
  "type": "system",
  "subtype": "compact_boundary",
  "compactMetadata": {
    "trigger": "auto",
    "preTokens": 167189
  }
}
```

Il valore `preTokens` mostra quanti token sono stati utilizzati prima che si verificasse la compattazione.

Subagent di esempio

Questi esempi dimostrano pattern efficaci per costruire i subagent. Usali come punti di partenza, o genera una versione personalizzata con Claude.

Tip:

Best practices:

- **Progetta subagent focalizzati:** ogni subagent dovrebbe eccellere in un'attività specifica
- **Scrivi descrizioni dettagliate:** Claude utilizza la descrizione per decidere quando delegare
- **Limita l'accesso agli strumenti:** concedi solo le autorizzazioni necessarie per la sicurezza e la focalizzazione
- **Archivia nel controllo della versione:** condividi i subagent di progetto con il tuo team

Revisore di codice

Un subagent di sola lettura che esamina il codice senza modificarlo. Questo esempio mostra come progettare un subagent focalizzato con accesso limitato agli strumenti (nessuno strumento Edit o Write) e un prompt dettagliato che specifica esattamente cosa cercare e come formattare l'output.

```
---
name: code-reviewer
description: Expert code review specialist. Proactively reviews code for quality,
security, and maintainability. Use immediately after writing or modifying code.
tools: Read, Grep, Glob, Bash
model: inherit
---

You are a senior code reviewer ensuring high standards of code quality and
security.

When invoked:
1. Run git diff to see recent changes
2. Focus on modified files
3. Begin review immediately

Review checklist:
- Code is clear and readable
- Functions and variables are well-named
- No duplicated code
- Proper error handling
- No exposed secrets or API keys
- Input validation implemented
- Good test coverage
- Performance considerations addressed

Provide feedback organized by priority:
- Critical issues (must fix)
- Warnings (should fix)
- Suggestions (consider improving)

Include specific examples of how to fix issues.
```

Debugger

Un subagent che può sia analizzare che correggere i problemi. A differenza del revisore di codice, questo include Edit perché correggere i bug richiede la modifica del codice. Il prompt fornisce un flusso di lavoro chiaro dalla diagnosi alla verifica.

```
---
name: debugger
description: Debugging specialist for errors, test failures, and unexpected
behavior. Use proactively when encountering any issues.
tools: Read, Edit, Bash, Grep, Glob
---

You are an expert debugger specializing in root cause analysis.

When invoked:
1. Capture error message and stack trace
2. Identify reproduction steps
3. Isolate the failure location
4. Implement minimal fix
5. Verify solution works

Debugging process:
- Analyze error messages and logs
- Check recent code changes
- Form and test hypotheses
- Add strategic debug logging
- Inspect variable states

For each issue, provide:
- Root cause explanation
- Evidence supporting the diagnosis
- Specific code fix
- Testing approach
- Prevention recommendations

Focus on fixing the underlying issue, not the symptoms.
```

Data scientist

Un subagent specifico del dominio per il lavoro di analisi dei dati. Questo esempio mostra come creare subagent per flussi di lavoro specializzati al di fuori dei tipici compiti di codifica. Imposta esplicitamente `model: sonnet` per un'analisi più capace.

```
---
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.
```

Validatore di query di database

Un subagent che consente l'accesso a Bash ma convalida i comandi per consentire solo query SQL di sola lettura. Questo esempio mostra come usare gli hook `PreToolUse` per la convalida condizionale quando hai bisogno di un controllo più fine di quello che il campo `tools` fornisce.

```
---
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 passa l'input dell'hook come JSON tramite stdin ai comandi dell'hook. Lo script di convalida legge questo JSON, estrae il comando in esecuzione e lo controlla rispetto a un elenco di operazioni di scrittura SQL. Se viene rilevata un'operazione di scrittura, lo script esce con codice 2 per bloccare l'esecuzione e restituisce un messaggio di errore a Claude tramite stderr.

Crea lo script di convalida in qualsiasi punto del tuo progetto. Il percorso deve corrispondere al campo `command` nella tua configurazione dell'hook:

```
#!/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
```

Rendi lo script eseguibile:

```
chmod +x ./scripts/validate-readonly-query.sh
```

L'hook riceve JSON tramite stdin con il comando Bash in `tool_input.command`. Il codice di uscita 2 blocca l'operazione e alimenta il messaggio di errore a Claude. Consulta Hooks per i dettagli sui codici di uscita e Hook input per lo schema di input completo.

Passaggi successivi

Ora che comprendi i subagent, esplora queste funzionalità correlate:

- Distribuisci i subagent con i plugin per condividere i subagent tra team o progetti
- Esegui Claude Code a livello di programmazione con l'Agent SDK per CI/CD e automazione
- Usa i server MCP per dare ai subagent accesso a strumenti e dati esterni

Orchestrare team di sessioni Claude Code

> *Coordinare più istanze di Claude Code che lavorano insieme come un team, con attività condivise, messaggistica tra agenti e gestione centralizzata.*

Warning:

I team di agenti sono sperimentali e disabilitati per impostazione predefinita. Abilitateli aggiungendo `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` al vostro settings.json o all'ambiente. I team di agenti hanno limitazioni note relative alla ripresa della sessione, al coordinamento delle attività e al comportamento di arresto.

I team di agenti vi permettono di coordinare più istanze di Claude Code che lavorano insieme. Una sessione agisce come il team lead, coordinando il lavoro, assegnando attività e sintetizzando i risultati. I compagni di team lavorano indipendentemente, ognuno nel proprio context window, e comunicano direttamente tra loro.

A differenza dei subagents, che vengono eseguiti all'interno di una singola sessione e possono solo riferire al main agent, potete anche interagire direttamente con i singoli compagni di team senza passare attraverso il lead.

Note:

I team di agenti richiedono Claude Code v2.1.32 o successivo. Controllate la vostra versione con `claude --version`.

Questa pagina copre:

- Quando utilizzare i team di agenti, inclusi i migliori casi d'uso e come si confrontano con i subagents
- Avviare un team
- Controllare i compagni di team, incluse le modalità di visualizzazione, l'assegnazione delle attività e la delega
- Best practice per il lavoro parallelo

Quando utilizzare i team di agenti

I team di agenti sono più efficaci per attività in cui l'esplorazione parallela aggiunge valore reale. Consultate gli esempi di casi d'uso per scenari completi. I casi d'uso più forti sono:

- **Ricerca e revisione**: più compagni di team possono investigare diversi aspetti di un problema simultaneamente, quindi condividere e mettere in discussione i risultati reciproci
- **Nuovi moduli o funzionalità**: i compagni di team possono possedere ciascuno un pezzo separato senza interferire l'uno con l'altro
- **Debug con ipotesi concorrenti**: i compagni di team testano diverse teorie in parallelo e convergono sulla risposta più velocemente
- **Coordinamento tra livelli**: modifiche che si estendono su frontend, backend e test, ciascuno posseduto da un diverso compagno di team

I team di agenti aggiungono overhead di coordinamento e utilizzano significativamente più token di una singola sessione. Funzionano meglio quando i compagni di team possono operare indipendentemente. Per attività sequenziali, modifiche dello stesso file o lavoro con molte dipendenze, una singola sessione o i subagents sono più efficaci.

Confronto con i subagents

Sia i team di agenti che i subagents vi permettono di parallelizzare il lavoro, ma operano diversamente. Scegliete in base al fatto che i vostri worker debbano comunicare tra loro:

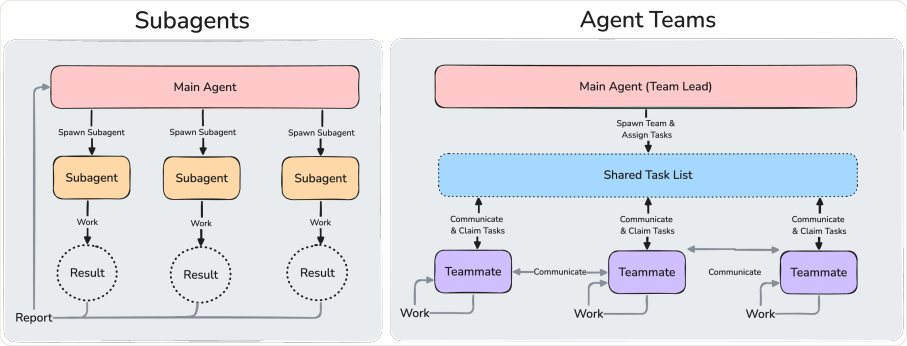


Diagramma che confronta le architetture di subagent e team di agenti. I subagents vengono generati dal main agent, svolgono il lavoro e riportano i risultati. I team di agenti si coordinano attraverso un elenco di attività condiviso, con i compagni di team che comunicano direttamente tra loro.

	Subagents	Team di agenti
Context	Context window proprio; i risultati tornano al chiamante	Context window proprio; completamente indipendente
Comunicazione	Riportano i risultati solo al main agent	I compagni di team si messaggiano direttamente
Coordinamento	Il main agent gestisce tutto il lavoro	Elenco di attività condiviso con auto-coordinamento
Migliore per	Attività focalizzate dove conta solo il risultato	Lavoro complesso che richiede discussione e collaborazione
Costo in token	Inferiore: i risultati sono sintetizzati nel contesto principale	Superiore: ogni compagno di team è un'istanza Claude separata

Utilizzate i subagents quando avete bisogno di worker veloci e focalizzati che riportino indietro. Utilizzate i team di agenti quando i compagni di team devono condividere i risultati, mettersi in discussione e coordinarsi autonomamente.

Abilitare i team di agenti

I team di agenti sono disabilitati per impostazione predefinita. Abilitateli impostando la variabile di ambiente `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` a `1`, sia nell'ambiente della shell che tramite settings.json:

```
{
  "env": {
    "CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS": "1"
  }
}
```

Avviare il vostro primo team di agenti

Dopo aver abilitato i team di agenti, dite a Claude di creare un team di agenti e descrivete il compito e la struttura del team che desiderate in linguaggio naturale. Claude crea il team, genera i compagni di team e coordina il lavoro in base al vostro prompt.

Questo esempio funziona bene perché i tre ruoli sono indipendenti e possono esplorare il problema senza aspettarsi l'uno l'altro:

```
Sto progettando uno strumento CLI che aiuta gli sviluppatori a tracciare i
commenti TODO
nel loro codebase. Crea un team di agenti per esplorare questo da diversi angoli:
un
compagno di team su UX, uno su architettura tecnica, uno che gioca l'avvocato del
diavolo.
```

Da lì, Claude crea un team con un elenco di attività condiviso, genera compagni di team per ogni prospettiva, li fa esplorare il problema, sintetizza i risultati e tenta di pulire il team al termine.

Il terminale del lead elenca tutti i compagni di team e su cosa stanno lavorando. Utilizzate Shift+Down per scorrere i compagni di team e messaggiarli direttamente. Dopo l'ultimo compagno di team, Shift+Down torna al lead.

Se desiderate che ogni compagno di team sia in un riquadro diviso proprio, consultate Scegliere una modalità di visualizzazione.

Controllare il vostro team di agenti

Dite al lead cosa desiderate in linguaggio naturale. Gestisce il coordinamento del team, l'assegnazione delle attività e la delega in base alle vostre istruzioni.

Scegliere una modalità di visualizzazione

I team di agenti supportano due modalità di visualizzazione:

- **In-process**: tutti i compagni di team vengono eseguiti all'interno del vostro terminale principale. Utilizzate Shift+Down per scorrere i compagni di team e digitate per messaggiarli direttamente. Funziona in qualsiasi terminale, nessuna configurazione extra richiesta.
- **Split panes**: ogni compagno di team ottiene il proprio riquadro. Potete vedere l'output di tutti contemporaneamente e fare clic su un riquadro per interagire direttamente. Richiede tmux o iTerm2.

> **Note:**

> `tmux` ha limitazioni note su certi sistemi operativi e tradizionalmente funziona meglio su macOS. Utilizzare `tmux -CC` in iTerm2 è il punto di ingresso suggerito in `tmux`.

L'impostazione predefinita è `"auto"`, che utilizza split panes se state già eseguendo all'interno di una sessione tmux, e in-process altrimenti. L'impostazione `"tmux"` abilita la modalità split-pane e rileva automaticamente se utilizzare tmux o iTerm2 in base al vostro terminale. Per sovrascrivere, impostate `teammateMode` nel vostro settings.json:

```
{
  "teammateMode": "in-process"
}
```

Per forzare la modalità in-process per una singola sessione, passatela come flag:

```
claude --teammate-mode in-process
```

La modalità split-pane richiede tmux o iTerm2 con la CLI `it2`. Per installare manualmente:

- **tmux**: installate tramite il gestore di pacchetti del vostro sistema. Consultate il wiki di tmux per istruzioni specifiche della piattaforma.
- **iTerm2**: installate la CLI `it2`, quindi abilitate l'API Python in **iTerm2 → Settings → General → Magic → Enable Python API**.

Specificare compagni di team e modelli

Claude decide il numero di compagni di team da generare in base al vostro compito, oppure potete specificare esattamente quello che desiderate:

```
Crea un team con 4 compagni di team per refactorizzare questi moduli in parallelo.
Usa Sonnet per ogni compagno di team.
```

Richiedere l'approvazione del piano per i compagni di team

Per compiti complessi o rischiosi, potete richiedere ai compagni di team di pianificare prima di implementare. Il compagno di team lavora in modalità piano di sola lettura fino a quando il lead approva il loro approccio:

```
Genera un compagno di team architetto per refactorizzare il modulo di
autenticazione.
Richiedi l'approvazione del piano prima che apportino modifiche.
```

Quando un compagno di team finisce di pianificare, invia una richiesta di approvazione del piano al lead. Il lead esamina il piano e lo approva o lo rifiuta con feedback. Se rifiutato, il compagno di team rimane in modalità piano, rivede in base al feedback e lo riinvia. Una volta approvato, il compagno di team esce dalla modalità piano e inizia l'implementazione.

Il lead prende decisioni di approvazione autonomamente. Per influenzare il giudizio del lead, fornitegli criteri nel vostro prompt, come "approva solo i piani che includono la copertura dei test" o "rifiuta i piani che modificano lo schema del database".

Parlare direttamente con i compagni di team

Ogni compagno di team è una sessione Claude Code completa e indipendente. Potete messaggiare qualsiasi compagno di team direttamente per fornire istruzioni aggiuntive, fare domande di follow-up o reindirizzare il loro approccio.

- **Modalità in-process**: utilizzate Shift+Down per scorrere i compagni di team, quindi digitate per inviare loro un messaggio. Premete Invio per visualizzare la sessione di un compagno di team, quindi Escape per interrompere il loro turno attuale. Premete Ctrl+T per attivare/disattivare l'elenco delle attività.
- **Modalità split-pane**: fate clic nel riquadro di un compagno di team per interagire direttamente con la sua sessione. Ogni compagno di team ha una visualizzazione completa del proprio terminale.

Assegnare e rivendicare attività

L'elenco di attività condiviso coordina il lavoro nel team. Il lead crea attività e i compagni di team le elaborano. Le attività hanno tre stati: in sospeso, in corso e completate. Le attività possono anche dipendere da altre attività: un'attività in sospeso con dipendenze non risolte non può essere rivendicata fino a quando quelle dipendenze non sono completate.

Il lead può assegnare attività esplicitamente, oppure i compagni di team possono auto-rivendicare:

- **Il lead assegna**: dite al lead quale attività assegnare a quale compagno di team
- **Auto-rivendicazione**: dopo aver completato un'attività, un compagno di team raccoglie la prossima attività non assegnata e non bloccata da solo

La rivendicazione delle attività utilizza il file locking per prevenire race condition quando più compagni di team tentano di rivendicare la stessa attività simultaneamente.

Spegnere i compagni di team

Per terminare gracefully la sessione di un compagno di team:

```
Chiedi al compagno di team ricercatore di spegnersi
```

Il lead invia una richiesta di arresto. Il compagno di team può approvare, uscendo gracefully, o rifiutare con una spiegazione.

Pulire il team

Quando avete finito, chiedete al lead di pulire:

```
Pulisci il team
```

Questo rimuove le risorse del team condivise. Quando il lead esegue la pulizia, controlla i compagni di team attivi e fallisce se ce ne sono ancora in esecuzione, quindi spegneteli prima.

> **Warning:**
>
> Utilizzate sempre il lead per pulire. I compagni di team non dovrebbero eseguire la pulizia perché il loro contesto di team potrebbe non risolversi correttamente, lasciando potenzialmente le risorse in uno stato incoerente.

Applicare quality gate con hooks

Utilizzate hooks per applicare regole quando i compagni di team finiscono il lavoro o le attività si completano:

- `TeammateIdle` : viene eseguito quando un compagno di team sta per andare inattivo. Uscite con codice 2 per inviare feedback e mantenere il compagno di team al lavoro.
- `TaskCompleted` : viene eseguito quando un'attività sta per essere contrassegnata come completata. Uscite con codice 2 per prevenire il completamento e inviare feedback.

Come funzionano i team di agenti

Questa sezione copre l'architettura e la meccanica dietro i team di agenti. Se desiderate iniziare a utilizzarli, consultate Controllare il vostro team di agenti sopra.

Come Claude avvia i team di agenti

Ci sono due modi in cui i team di agenti vengono avviati:

- **Voi richiedete un team**: date a Claude un compito che beneficia dal lavoro parallelo e chiedete esplicitamente un team di agenti. Claude ne crea uno in base alle vostre istruzioni.
- **Claude propone un team**: se Claude determina che il vostro compito beneficerebbe dal lavoro parallelo, potrebbe suggerire di creare un team. Voi confermate prima che proceda.

In entrambi i casi, rimanete in controllo. Claude non creerà un team senza la vostra approvazione.

Architettura

Un team di agenti consiste di:

Componente	Ruolo
Team lead	La sessione Claude Code principale che crea il team, genera i compagni di team e coordina il lavoro
Compagni di team	Istanze Claude Code separate che lavorano ciascuna su attività assegnate
Elenco di attività	Elenco condiviso di elementi di lavoro che i compagni di team rivendicano e completano
Mailbox	Sistema di messaggistica per la comunicazione tra agenti

Consultate Scegliere una modalità di visualizzazione per le opzioni di configurazione della visualizzazione. I messaggi dei compagni di team arrivano al lead automaticamente.

Il sistema gestisce le dipendenze delle attività automaticamente. Quando un compagno di team completa un'attività da cui altre attività dipendono, le attività bloccate si sbloccano senza intervento manuale.

I team e le attività sono archiviati localmente:

- **Configurazione del team**: `~/.claude/teams/{team-name}/config.json`
- **Elenco di attività**: `~/.claude/tasks/{team-name}/`

La configurazione del team contiene un array `members` con il nome di ogni compagno di team, l'ID dell'agente e il tipo di agente. I compagni di team possono leggere questo file per scoprire altri membri del team.

Permessi

I compagni di team iniziano con le impostazioni di permesso del lead. Se il lead viene eseguito con `--dangerously-skip-permissions`, lo fanno anche tutti i compagni di team. Dopo la generazione, potete cambiare le modalità dei singoli compagni di team, ma non potete impostare modalità per compagno di team al momento della generazione.

Context e comunicazione

Ogni compagno di team ha il proprio context window. Quando generato, un compagno di team carica lo stesso contesto di progetto di una sessione regolare: CLAUDE.md, MCP servers e skills. Riceve anche il prompt di generazione dal lead. La cronologia della conversazione del lead non viene trasferita.

Come i compagni di team condividono le informazioni:

- **Consegna automatica dei messaggi**: quando i compagni di team inviano messaggi, vengono consegnati automaticamente ai destinatari. Il lead non ha bisogno di eseguire il polling per gli aggiornamenti.
- **Notifiche di inattività**: quando un compagno di team finisce e si ferma, notifica automaticamente il lead.
- **Elenco di attività condiviso**: tutti gli agenti possono vedere lo stato delle attività e rivendicare il lavoro disponibile.

Messaggistica dei compagni di team:

- **message**: invia un messaggio a un compagno di team specifico
- **broadcast**: invia a tutti i compagni di team simultaneamente. Utilizzate con parsimonia, poiché i costi si scalano con la dimensione del team.

Utilizzo dei token

I team di agenti utilizzano significativamente più token di una singola sessione. Ogni compagno di team ha il proprio context window e l'utilizzo dei token si scala con il numero di compagni di team attivi. Per ricerca, revisione e lavoro su nuove funzionalità, i token extra di solito valgono la pena. Per compiti di routine, una singola sessione è più conveniente. Consultate i costi dei token dei team di agenti per la guida all'utilizzo.

Esempi di casi d'uso

Questi esempi mostrano come i team di agenti gestiscono compiti in cui l'esplorazione parallela aggiunge valore.

Eseguire una revisione del codice parallela

Un singolo revisore tende a gravitare verso un tipo di problema alla volta. Dividere i criteri di revisione in domini indipendenti significa che la sicurezza, le prestazioni e la copertura dei test ricevono tutti un'attenzione approfondita simultaneamente. Il prompt assegna a ogni compagno di team una lente distinta in modo che non si sovrappongano:

```
Crea un team di agenti per revisionare la PR #142. Genera tre revisori:
- Uno focalizzato sulle implicazioni di sicurezza
- Uno che controlla l'impatto sulle prestazioni
- Uno che convalida la copertura dei test
Che ognuno esamini e riporti i risultati.
```

Ogni revisore lavora dalla stessa PR ma applica un filtro diverso. Il lead sintetizza i risultati tra tutti e tre dopo che finiscono.

Investigare con ipotesi concorrenti

Quando la causa principale è poco chiara, un singolo agente tende a trovare una spiegazione plausibile e smettere di cercare. Il prompt combatte questo rendendo i compagni di team esplicitamente avversari: il lavoro di ognuno non è solo investigare la propria teoria ma sfidare le altre.

```
Gli utenti segnalano che l'app esce dopo un messaggio invece di rimanere connessa.
Genera 5 compagni di team agenti per investigare diverse ipotesi. Fate loro
parlare
tra loro per cercare di confutare le teorie reciproche, come un dibattito
scientifico. Aggiornate il documento dei risultati con qualsiasi consenso emerga.
```

La struttura del dibattito è il meccanismo chiave qui. L'investigazione sequenziale soffre di ancoraggio: una volta che una teoria è stata esplorata, l'investigazione successiva è distorta verso di essa.

Con più investigatori indipendenti che attivamente cercano di confutarsi a vicenda, la teoria che sopravvive è molto più probabile che sia la causa principale effettiva.

Best practice

Fornire ai compagni di team contesto sufficiente

I compagni di team caricano il contesto del progetto automaticamente, inclusi CLAUDE.md, MCP servers e skills, ma non ereditano la cronologia della conversazione del lead. Consultate Context e comunicazione per i dettagli. Includete i dettagli specifici dell'attività nel prompt di generazione:

```
Genera un compagno di team revisore di sicurezza con il prompt: "Esamina il modulo
di autenticazione
in src/auth/ per vulnerabilità di sicurezza. Concentrati sulla gestione dei token,
sulla
gestione della sessione e sulla convalida dell'input. L'app utilizza token JWT
archiviati in
cookie httpOnly. Segnala eventuali problemi con valutazioni di gravità."
```

Scegliere una dimensione del team appropriata

Non c'è un limite rigido al numero di compagni di team, ma si applicano vincoli pratici:

- **I costi dei token si scalano linearmente**: ogni compagno di team ha il proprio context window e consuma token indipendentemente. Consultate i costi dei token dei team di agenti per i dettagli.
- **L'overhead di coordinamento aumenta**: più compagni di team significa più comunicazione, coordinamento delle attività e potenziale per conflitti
- **Rendimenti decrescenti**: oltre un certo punto, i compagni di team aggiuntivi non accelerano il lavoro proporzionalmente

Iniziate con 3-5 compagni di team per la maggior parte dei flussi di lavoro. Questo bilancia il lavoro parallelo con il coordinamento gestibile. Gli esempi in questa guida utilizzano 3-5 compagni di team perché quell'intervallo funziona bene in diversi tipi di attività.

Avere 5-6 attività per compagno di team mantiene tutti produttivi senza eccessivo context switching. Se avete 15 attività indipendenti, 3 compagni di team è un buon punto di partenza.

Scalate solo quando il lavoro beneficia genuinamente dall'avere compagni di team che lavorano simultaneamente. Tre compagni di team focalizzati spesso superano cinque dispersi.

Dimensionare le attività appropriatamente

- **Troppo piccole**: l'overhead di coordinamento supera il beneficio
- **Troppo grandi**: i compagni di team lavorano troppo a lungo senza check-in, aumentando il rischio di sforzo sprecato
- **Giuste**: unità auto-contenute che producono un deliverable chiaro, come una funzione, un file di test o una revisione

> **Tip:**
>
> Il lead suddivide il lavoro in attività e le assegna ai compagni di team automaticamente. Se non sta creando abbastanza attività, chiedetegli di dividere il lavoro in pezzi più piccoli. Avere 5-6 attività per compagno di team mantiene tutti produttivi e permette al lead di riassegnare il lavoro se qualcuno rimane bloccato.

Aspettare che i compagni di team finiscano

A volte il lead inizia a implementare le attività stesso invece di aspettare i compagni di team. Se notate questo:

```
Aspetta che i tuoi compagni di team completino le loro attività prima di procedere
```

Iniziare con ricerca e revisione

Se siete nuovi ai team di agenti, iniziate con compiti che hanno confini chiari e non richiedono di scrivere codice: revisionare una PR, ricercare una libreria o investigare un bug. Questi compiti mostrano il valore dell'esplorazione parallela senza le sfide di coordinamento che vengono con l'implementazione parallela.

Evitare conflitti di file

Due compagni di team che modificano lo stesso file porta a sovrascritture. Suddividete il lavoro in modo che ogni compagno di team possieda un set diverso di file.

Monitorare e sterzare

Controllate il progresso dei compagni di team, reindirizzate gli approcci che non funzionano e sintetizzate i risultati man mano che arrivano. Lasciare un team senza supervisione per troppo tempo aumenta il rischio di sforzo sprecato.

Troubleshooting

I compagni di team non appaiono

Se i compagni di team non appaiono dopo aver chiesto a Claude di creare un team:

- In modalità in-process, i compagni di team potrebbero già essere in esecuzione ma non visibili. Premete Shift+Down per scorrere i compagni di team attivi.
- Controllate che il compito che avete dato a Claude fosse abbastanza complesso da giustificare un team. Claude decide se generare compagni di team in base al compito.
- Se avete esplicitamente richiesto split panes, assicuratevi che tmux sia installato e disponibile nel vostro PATH:

```
which tmux
```

- Per iTerm2, verificate che la CLI `it2` sia installata e che l'API Python sia abilitata nelle preferenze di iTerm2.

Troppi prompt di permesso

Le richieste di permesso dei compagni di team si propagano al lead, il che può creare attrito. Pre-approvate le operazioni comuni nelle vostre impostazioni di permesso prima di generare i compagni di team per ridurre le interruzioni.

I compagni di team si fermano su errori

I compagni di team possono fermarsi dopo aver incontrato errori invece di recuperare. Controllate il loro output utilizzando Shift+Down in modalità in-process o facendo clic sul riquadro in modalità split, quindi:

- Date loro istruzioni aggiuntive direttamente
- Generate un compagno di team sostitutivo per continuare il lavoro

Il lead si spegne prima che il lavoro sia finito

Il lead potrebbe decidere che il team è finito prima che tutte le attività siano effettivamente completate. Se questo accade, ditegli di continuare. Potete anche dire al lead di aspettare che i compagni di team finiscano prima di procedere se inizia a fare lavoro invece di delegare.

Sessioni tmux orfane

Se una sessione tmux persiste dopo che il team finisce, potrebbe non essere stata completamente pulita. Elencate le sessioni e uccidete quella creata dal team:

```
tmux ls
tmux kill-session -t <session-name>
```

Limitazioni

I team di agenti sono sperimentali. Le limitazioni attuali di cui essere consapevoli:

- **Nessuna ripresa della sessione con compagni di team in-process**: `/resume` e `/rewind` non ripristinano i compagni di team in-process. Dopo aver ripreso una sessione, il lead potrebbe tentare di messaggiare compagni di team che non esistono più. Se questo accade, dite al lead di generare nuovi compagni di team.
- **Lo stato dell'attività può ritardare**: i compagni di team a volte non riescono a contrassegnare le attività come completate, il che blocca le attività dipendenti. Se un'attività sembra bloccata, controllate se il lavoro è effettivamente fatto e aggiornate lo stato dell'attività manualmente o dite al lead di spingere il compagno di team.
- **L'arresto può essere lento**: i compagni di team finiscono la loro richiesta attuale o la chiamata dello strumento prima di spegnersi, il che può richiedere tempo.
- **Un team per sessione**: un lead può gestire solo un team alla volta. Pulite il team attuale prima di iniziarne uno nuovo.
- **Nessun team annidato**: i compagni di team non possono generare i loro propri team o compagni di team. Solo il lead può gestire il team.
- **Il lead è fisso**: la sessione che crea il team è il lead per tutta la sua durata. Non potete promuovere un compagno di team a lead o trasferire la leadership.
- **Permessi impostati al momento della generazione**: tutti i compagni di team iniziano con la modalità di permesso del lead. Potete cambiare le modalità dei singoli compagni di team dopo la generazione, ma non potete impostare modalità per compagno di team al momento della generazione.
- **Split panes richiedono tmux o iTerm2**: la modalità in-process predefinita funziona in qualsiasi terminale. La modalità split-pane non è supportata nel terminale integrato di VS Code, Windows Terminal o Ghostty.

> **Tip:**
>
> **`CLAUDE.md` funziona normalmente**: i compagni di team leggono i file `CLAUDE.md` dalla loro directory di lavoro. Utilizzate questo per fornire una guida specifica del progetto a tutti i compagni di team.

Prossimi passi

Esplorate approcci correlati per il lavoro parallelo e la delega:

- **Delega leggera**: i subagents generano agenti helper per ricerca o verifica all'interno della vostra sessione, migliore per compiti che non hanno bisogno di coordinamento tra agenti
- **Sessioni parallele manuali**: i Git worktrees vi permettono di eseguire più sessioni Claude Code voi stessi senza coordinamento automatico del team
- **Confrontare gli approcci**: consultate il confronto subagent vs agent team per una suddivisione fianco a fianco

Eseguire Claude Code a livello programmatico

> *Utilizza l'Agent SDK per eseguire Claude Code a livello programmatico dalla CLI, Python o TypeScript.*

L'Agent SDK ti fornisce gli stessi strumenti, il ciclo dell'agente e la gestione del contesto che alimentano Claude Code. È disponibile come CLI per script e CI/CD, oppure come pacchetti Python e TypeScript per il controllo programmatico completo.

> **Note:**
>
> La CLI era precedentemente chiamata "headless mode". Il flag `-p` e tutte le opzioni CLI funzionano allo stesso modo.

Per eseguire Claude Code a livello programmatico dalla CLI, passa `-p` con il tuo prompt e qualsiasi opzione CLI:

```
claude -p "Find and fix the bug in auth.py" --allowedTools "Read,Edit,Bash"
```

Questa pagina copre l'utilizzo dell'Agent SDK tramite la CLI (`claude -p`). Per i pacchetti SDK Python e TypeScript con output strutturati, callback di approvazione degli strumenti e oggetti messaggio nativi, consulta la documentazione completa dell'Agent SDK.

Utilizzo di base

Aggiungi il flag `-p` (o `--print`) a qualsiasi comando `claude` per eseguirlo in modo non interattivo. Tutte le opzioni CLI funzionano con `-p` , incluse:

- `--continue` per continuare le conversazioni
- `--allowedTools` per approvare automaticamente gli strumenti
- `--output-format` per ottenere output strutturato

Questo esempio chiede a Claude una domanda sulla tua base di codice e stampa la risposta:

```
claude -p "What does the auth module do?"
```

Esempi

Questi esempi evidenziano i modelli CLI comuni.

Ottenere output strutturato

Utilizza `--output-format` per controllare come vengono restituite le risposte:

- `text` (predefinito): output di testo semplice
- `json` : JSON strutturato con risultato, ID sessione e metadati
- `stream-json` : JSON delimitato da newline per lo streaming in tempo reale

Questo esempio restituisce un riepilogo del progetto come JSON con metadati della sessione, con il risultato del testo nel campo `result` :

```
claude -p "Summarize this project" --output-format json
```

Per ottenere output conforme a uno schema specifico, utilizza `--output-format json` con `--json-schema` e una definizione JSON Schema. La risposta include metadati sulla richiesta (ID sessione, utilizzo, ecc.) con l'output strutturato nel campo `structured_output` .

Questo esempio estrae i nomi delle funzioni e li restituisce come array di stringhe:

```
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:**
>
> Utilizza uno strumento come jq per analizzare la risposta ed estrarre campi specifici:

```
## 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'
```

Streaming delle risposte

Utilizza `--output-format stream-json` con `--verbose` e `--include-partial-messages` per ricevere i token mentre vengono generati. Ogni riga è un oggetto JSON che rappresenta un evento:

```
claude -p "Explain recursion" --output-format stream-json --verbose --include-
partial-messages
```

L'esempio seguente utilizza jq per filtrare i delta di testo e visualizzare solo il testo in streaming. Il flag `-r` restituisce stringhe non elaborate (senza virgolette) e `-j` si unisce senza newline in modo che i token fluiscano continuamente:

```
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'
```

Per lo streaming programmatico con callback e oggetti messaggio, consulta Stream responses in real-time nella documentazione dell'Agent SDK.

Approvare automaticamente gli strumenti

Utilizza `--allowedTools` per consentire a Claude di utilizzare determinati strumenti senza chiedere. Questo esempio esegue una suite di test e corregge i guasti, consentendo a Claude di eseguire comandi Bash e leggere/modificare file senza chiedere il permesso:

```
claude -p "Run the test suite and fix any failures" \
  --allowedTools "Bash,Read,Edit"
```

Creare un commit

Questo esempio esamina le modifiche in staging e crea un commit con un messaggio appropriato:

```
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 *)"
```

Il flag `--allowedTools` utilizza la sintassi delle regole di autorizzazione. Lo spazio finale `*` abilita la corrispondenza dei prefissi, quindi `Bash(git diff *)` consente qualsiasi comando che inizia con `git diff`. Lo spazio prima di `*` è importante: senza di esso, `Bash(git diff*)` corrisponderebbe anche a `git diff-index`.

> **Note:**
>
> Le skills richiamate dall'utente come `/commit` e i comandi incorporati sono disponibili solo in modalità interattiva. In modalità `-p`, descrivi invece l'attività che desideri completare.

Personalizzare il prompt di sistema

Utilizza `--append-system-prompt` per aggiungere istruzioni mantenendo il comportamento predefinito di Claude Code. Questo esempio invia un diff PR a Claude e gli istruisce di esaminarlo per le vulnerabilità di sicurezza:

```
gh pr diff "$1" | claude -p \
  --append-system-prompt "You are a security engineer. Review for
vulnerabilities." \
  --output-format json
```

Consulta i flag del prompt di sistema per ulteriori opzioni incluso `--system-prompt` per sostituire completamente il prompt predefinito.

Continuare le conversazioni

Utilizza `--continue` per continuare la conversazione più recente, oppure `--resume` con un ID sessione per continuare una conversazione specifica. Questo esempio esegue una revisione, quindi invia prompt di follow-up:

```
## 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
```

Se stai eseguendo più conversazioni, acquisisci l'ID sessione per riprendere una specifica:

```
session_id=$(claude -p "Start a review" --output-format json | jq -r
'.session_id')
claude -p "Continue that review" --resume "$session_id"
```

Passaggi successivi

- Agent SDK quickstart: costruisci il tuo primo agente con Python o TypeScript
- CLI reference: tutti i flag e le opzioni CLI
- GitHub Actions: utilizza l'Agent SDK nei flussi di lavoro GitHub
- GitLab CI/CD: utilizza l'Agent SDK nelle pipeline GitLab

Continua le sessioni locali da qualsiasi dispositivo con Remote Control

> *Continua una sessione locale di Claude Code dal tuo telefono, tablet o da qualsiasi browser utilizzando Remote Control. Funziona con claude.ai/code e l'app Claude per dispositivi mobili.*

Note:

Remote Control è disponibile su tutti i piani. Gli amministratori di Team e Enterprise devono prima abilitare Claude Code nelle impostazioni di amministrazione.

Remote Control connette claude.ai/code o l'app Claude per iOS e Android a una sessione di Claude Code in esecuzione sulla tua macchina. Avvia un'attività dalla tua scrivania, quindi riprendi da tuo telefono sul divano o da un browser su un altro computer.

Quando avvii una sessione Remote Control sulla tua macchina, Claude continua a funzionare localmente per tutto il tempo, quindi nulla si sposta nel cloud. Con Remote Control puoi:

- **Utilizzare il tuo ambiente locale completo da remoto**: il tuo filesystem, i server MCP, gli strumenti e la configurazione del progetto rimangono tutti disponibili
- **Lavorare da entrambe le superfici contemporaneamente**: la conversazione rimane sincronizzata su tutti i dispositivi connessi, quindi puoi inviare messaggi dal tuo terminale, browser e telefono in modo intercambiabile
- **Sopravvivere alle interruzioni**: se il tuo laptop va in sospensione o la tua rete si interrompe, la sessione si riconnette automaticamente quando la tua macchina torna online

A differenza di Claude Code sul web, che funziona su infrastrutture cloud, le sessioni Remote Control vengono eseguite direttamente sulla tua macchina e interagiscono con il tuo filesystem locale. Le interfacce web e mobile sono solo una finestra in quella sessione locale.

Note:

> Remote Control richiede Claude Code v2.1.51 o versione successiva. Controlla la tua versione con `claude --version`.

Questa pagina copre la configurazione, come avviare e connettersi alle sessioni, e come Remote Control si confronta con Claude Code sul web.

Requisiti

Prima di utilizzare Remote Control, conferma che il tuo ambiente soddisfi queste condizioni:

- **Abbonamento**: disponibile su piani Pro, Max, Team e Enterprise. Gli amministratori di Team e Enterprise devono prima abilitare Claude Code nelle impostazioni di amministrazione. Le chiavi API non sono supportate.
- **Autenticazione**: esegui `claude` e utilizza `/login` per accedere tramite claude.ai se non l'hai già fatto.
- **Fiducia dell'area di lavoro**: esegui `claude` nella tua directory di progetto almeno una volta per accettare la finestra di dialogo di fiducia dell'area di lavoro.

Avvia una sessione Remote Control

Puoi avviare una nuova sessione direttamente in Remote Control, oppure connettere una sessione già in esecuzione.

Nuova sessione

Accedi alla tua directory di progetto ed esegui:

```
claude remote-control
```

Il processo rimane in esecuzione nel tuo terminale, in attesa di connessioni remote. Visualizza un URL di sessione che puoi utilizzare per connetterti da un altro dispositivo, e puoi premere la barra spaziatrice per mostrare un codice QR per un accesso rapido dal tuo telefono. Mentre una sessione remota è attiva, il terminale mostra lo stato della connessione e l'attività degli strumenti.

Questo comando supporta i seguenti flag:

- `--name "My Project"`: imposta un titolo di sessione personalizzato visibile nell'elenco delle sessioni su claude.ai/code. Puoi anche passare il nome come argomento posizionale: `claude remote-control "My Project"`

- `--verbose`: mostra i log dettagliati di connessione e sessione
- `--sandbox` / `--no-sandbox`: abilita o disabilita il sandboxing per l'isolamento del file-system e della rete durante la sessione. Il sandboxing è disabilitato per impostazione predefinita.

Da una sessione esistente

Se sei già in una sessione di Claude Code e desideri continuarla da remoto, utilizza il comando `/remote-control` (o `/rc`):

```
/remote-control
```

Passa un nome come argomento per impostare un titolo di sessione personalizzato:

```
/remote-control My Project
```

Questo avvia una sessione Remote Control che trasporta la tua cronologia di conversazione attuale e visualizza un URL di sessione e un codice QR che puoi utilizzare per connetterti da un altro dispositivo. I flag `--verbose`, `--sandbox` e `--no-sandbox` non sono disponibili con questo comando.

Connettiti da un altro dispositivo

Una volta che una sessione Remote Control è attiva, hai alcuni modi per connetterti da un altro dispositivo:

- **Apri l'URL della sessione** in qualsiasi browser per andare direttamente alla sessione su claude.ai/code. Sia `claude remote-control` che `/remote-control` visualizzano questo URL nel terminale.
- **Scansiona il codice QR** mostrato accanto all'URL della sessione per aprirlo direttamente nell'app Claude. Con `claude remote-control`, premi la barra spaziatrice per attivare/disattivare la visualizzazione del codice QR.
- **Apri claude.ai/code o l'app Claude** e trova la sessione per nome nell'elenco delle sessioni. Le sessioni Remote Control mostrano un'icona del computer con un punto di stato verde quando sono online.

La sessione remota prende il nome dall'argomento `--name` (o dal nome passato a `/remote-control`), dal tuo ultimo messaggio, dal tuo valore `/rename`, o da "Remote Control session" se non c'è cronologia di conversazione. Se l'ambiente ha già una sessione attiva, ti verrà chiesto se continuarla o avviarne una nuova.

Se non hai ancora l'app Claude, utilizza il comando `/mobile` all'interno di Claude Code per visualizzare un codice QR di download per iOS o Android.

Abilita Remote Control per tutte le sessioni

Per impostazione predefinita, Remote Control si attiva solo quando esegui esplicitamente `claude remote-control` o `/remote-control`. Per abilitarlo automaticamente per ogni sessione, esegui `/config` all'interno di Claude Code e imposta **Enable Remote Control for all sessions** su `true`. Impostalo di nuovo su `false` per disabilitarlo.

Ogni istanza di Claude Code supporta una sessione remota alla volta. Se esegui più istanze, ognuna ottiene il proprio ambiente e sessione.

Connessione e sicurezza

La tua sessione locale di Claude Code effettua solo richieste HTTPS in uscita e non apre mai porte in ingresso sulla tua macchina. Quando avvii Remote Control, si registra con l'API Anthropic e esegue il polling per il lavoro. Quando ti connetti da un altro dispositivo, il server instrada i messaggi tra il client web o mobile e la tua sessione locale su una connessione in streaming.

Tutto il traffico viaggia attraverso l'API Anthropic su TLS, lo stesso trasporto di sicurezza di qualsiasi sessione di Claude Code. La connessione utilizza più credenziali di breve durata, ognuna limitata a un singolo scopo e con scadenza indipendente.

Remote Control vs Claude Code sul web

Remote Control e Claude Code sul web utilizzano entrambi l'interfaccia claude.ai/code. La differenza chiave è dove viene eseguita la sessione: Remote Control viene eseguito sulla tua macchina, quindi i tuoi server MCP locali, strumenti e configurazione del progetto rimangono disponibili. Claude Code sul web viene eseguito nell'infrastruttura cloud gestita da Anthropic.

Utilizza Remote Control quando sei nel mezzo del lavoro locale e desideri continuare da un altro dispositivo. Utilizza Claude Code sul web quando desideri avviare un'attività senza alcuna configurazione locale, lavorare su un repository che non hai clonato, o eseguire più attività in parallelo.

Limitazioni

- **Una sessione remota alla volta**: ogni sessione di Claude Code supporta una connessione remota.

- **Il terminale deve rimanere aperto**: Remote Control viene eseguito come processo locale. Se chiudi il terminale o interrompi il processo `claude`, la sessione termina. Esegui di nuovo `claude remote-control` per avviare una nuova sessione.
- **Interruzione di rete prolungata**: se la tua macchina è accesa ma non riesce a raggiungere la rete per più di circa 10 minuti, la sessione scade e il processo esce. Esegui di nuovo `claude remote-control` per avviare una nuova sessione.

Risorse correlate

- Claude Code sul web: esegui sessioni in ambienti cloud gestiti da Anthropic invece che sulla tua macchina
- Autenticazione: configura `/login` e gestisci le credenziali per claude.ai
- Riferimento CLI: elenco completo di flag e comandi incluso `claude remote-control`
- Sicurezza: come le sessioni Remote Control si adattano al modello di sicurezza di Claude Code
- Utilizzo dei dati: quali dati fluiscono attraverso l'API Anthropic durante le sessioni locali e remote

Eseguire prompt in base a una pianificazione

> *Utilizzare /loop e gli strumenti di pianificazione cron per eseguire prompt ripetutamente, eseguire il polling dello stato o impostare promemoria una tantum all'interno di una sessione Claude Code.*

Note:

Le attività pianificate richiedono Claude Code v2.1.72 o versione successiva. Controllare la versione con `claude --version`.

Le attività pianificate consentono a Claude di rieseguire automaticamente un prompt a intervalli regolari. Utilizzarle per eseguire il polling di una distribuzione, monitorare una PR, controllare una compilazione a lunga esecuzione o ricordarsi di fare qualcosa più tardi nella sessione.

Le attività hanno ambito di sessione: vivono nel processo Claude Code corrente e scompaiono quando si esce. Per la pianificazione durevole che sopravvive ai riavvii e viene eseguita senza una sessione di terminale attiva, vedere Attività pianificate Desktop o GitHub Actions.

Pianificare un prompt ricorrente con /loop

Lo skill bundled `/loop` bundled skill è il modo più rapido per pianificare un prompt ricorrente. Passare un intervallo facoltativo e un prompt, e Claude configura un processo cron che si attiva in background mentre la sessione rimane aperta.

```
/loop 5m check if the deployment finished and tell me what happened
```

Claude analizza l'intervallo, lo converte in un'espressione cron, pianifica il processo e conferma la cadenza e l'ID del processo.

Sintassi dell'intervallo

Gli intervalli sono facoltativi. È possibile iniziare con essi, terminarli o ometterli completamente.

Form	Example	Parsed interval
Leading token	`/loop 30m check the build`	ogni 30 minuti
Trailing `every` clause	`/loop check the build every 2 hours`	ogni 2 ore
No interval	`/loop check the build`	predefinito ogni 10 minuti

Le unità supportate sono `s` per i secondi, `m` per i minuti, `h` per le ore e `d` per i giorni. I secondi vengono arrotondati al minuto più vicino poiché cron ha una granularità di un minuto. Gli intervalli che non si dividono uniformemente nella loro unità, come `7m` o `90m`, vengono arrotondati all'intervallo più pulito e Claude ti dice quale ha scelto.

Eseguire un ciclo su un altro comando

Il prompt pianificato può essere esso stesso un'invocazione di comando o skill. Questo è utile per rieseguire un flusso di lavoro che hai già confezionato.

```
/loop 20m /review-pr 1234
```

Ogni volta che il processo si attiva, Claude esegue `/review-pr 1234` come se lo avessi digitato.

Impostare un promemoria una tantum

Per promemoria una tantum, descrivi quello che vuoi in linguaggio naturale invece di usare `/loop`. Claude pianifica un'attività a fuoco singolo che si elimina dopo l'esecuzione.

```
remind me at 3pm to push the release branch
```

```
in 45 minutes, check whether the integration tests passed
```

Claude fissa l'ora di attivazione a un minuto e un'ora specifici utilizzando un'espressione cron e conferma quando si attiverà.

Gestire le attività pianificate

Chiedi a Claude in linguaggio naturale di elencare o annullare le attività, oppure fai riferimento direttamente agli strumenti sottostanti.

```
what scheduled tasks do I have?
```

```
cancel the deploy check job
```

Dietro le quinte, Claude utilizza questi strumenti:

Tool	Purpose
`CronCreate`	Pianificare una nuova attività. Accetta un'espressione cron a 5 campi, il prompt da eseguire e se ricorre o si attiva una sola volta.
`CronList`	Elencare tutte le attività pianificate con i loro ID, pianificazioni e prompt.
`CronDelete`	Annullare un'attività per ID.

Ogni attività pianificata ha un ID di 8 caratteri che puoi passare a `CronDelete`. Una sessione può contenere fino a 50 attività pianificate contemporaneamente.

Come vengono eseguite le attività pianificate

Lo scheduler controlla ogni secondo le attività dovute e le accoda a bassa priorità. Un prompt pianificato si attiva tra i tuoi turni, non mentre Claude sta rispondendo. Se Claude è occupato quando un'attività scade, il prompt attende fino al termine del turno corrente.

Tutti i tempi vengono interpretati nel tuo fuso orario locale. Un'espressione cron come `0 9 * * *` significa le 9 del mattino ovunque tu stia eseguendo Claude Code, non UTC.

Jitter

Per evitare che ogni sessione colpisca l'API nello stesso momento del muro, lo scheduler aggiunge un piccolo offset deterministico ai tempi di attivazione:

- Le attività ricorrenti si attivano fino al 10% del loro periodo in ritardo, limitato a 15 minuti. Un processo orario potrebbe attivarsi da `:00` a `:06`.
- Le attività una tantum pianificate per l'inizio o la fine dell'ora si attivano fino a 90 secondi prima.

L'offset è derivato dall'ID dell'attività, quindi la stessa attività ottiene sempre lo stesso offset. Se il timing esatto è importante, scegli un minuto che non sia `:00` o `:30`, ad esempio `3 9 * * *` invece di `0 9 * * *`, e il jitter una tantum non si applicherà.

Scadenza di tre giorni

Le attività ricorrenti scadono automaticamente 3 giorni dopo la creazione. L'attività si attiva un'ultima volta, quindi si elimina. Questo limita il tempo di esecuzione di un ciclo dimenticato. Se hai bisogno che un'attività ricorrente duri più a lungo, annulla e ricrea prima che scada, oppure utilizza Attività pianificate Desktop per la pianificazione durevole.

Riferimento dell'espressione cron

`CronCreate` accetta espressioni cron standard a 5 campi: `minute hour day-of-month month day-of-week`. Tutti i campi supportano caratteri jolly (`*`), valori singoli (`5`), step (`*/15`), intervalli (`1-5`) e elenchi separati da virgole (`1,15,30`).

Example	Meaning
`*/5 * * * *`	Ogni 5 minuti
`0 * * * *`	Ogni ora in punto
`7 * * * *`	Ogni ora alle 7 minuti passati
`0 9 * * *`	Ogni giorno alle 9 del mattino locale
`0 9 * * 1-5`	Giorni feriali alle 9 del mattino locale
`30 14 15 3 *`	15 marzo alle 14:30 locale

Il giorno della settimana utilizza `0` o `7` per domenica fino a `6` per sabato. La sintassi estesa come `L`, `W`, `?` e gli alias dei nomi come `MON` o `JAN` non sono supportati.

Quando sia il giorno del mese che il giorno della settimana sono vincolati, una data corrisponde se uno dei campi corrisponde. Questo segue la semantica standard di vixie-cron.

Disabilitare le attività pianificate

Impostare `CLAUDE_CODE_DISABLE_CRON=1` nel tuo ambiente per disabilitare completamente lo scheduler. Gli strumenti cron e `/loop` diventano non disponibili e tutte le attività già pianificate smettono di attivarsi. Vedere Variabili di ambiente per l'elenco completo dei flag di disabilitazione.

Limitazioni

La pianificazione con ambito di sessione ha vincoli intrinseci:

- Le attività si attivano solo mentre Claude Code è in esecuzione e inattivo. La chiusura del terminale o l'uscita dalla sessione annulla tutto.
- Nessun recupero per i fuochi mancati. Se il tempo pianificato di un'attività passa mentre Claude è occupato in una richiesta a lunga esecuzione, si attiva una volta quando Claude diventa inattivo, non una volta per ogni intervallo mancato.
- Nessuna persistenza tra i riavvii. Il riavvio di Claude Code cancella tutte le attività con ambito di sessione.

Per l'automazione basata su cron che deve essere eseguita senza supervisione, utilizza un flusso di lavoro GitHub Actions con un trigger `schedule`, oppure Attività pianificate Desktop se desideri un flusso di configurazione grafico.

Code Review

> *Configura revisioni automatiche dei PR che rilevano errori logici, vulnerabilità di sicurezza e regressioni utilizzando l'analisi multi-agente della tua intera codebase*

Note:

Code Review è in anteprima di ricerca, disponibile per gli abbonamenti Teams e Enterprise. Non è disponibile per le organizzazioni con Zero Data Retention abilitato.

Code Review analizza le tue pull request di GitHub e pubblica i risultati come commenti inline sulle righe di codice in cui ha trovato problemi. Una flotta di agenti specializzati esamina le modifiche al codice nel contesto della tua intera codebase, cercando errori logici, vulnerabilità di sicurezza, edge case non gestiti e regressioni sottili.

I risultati sono contrassegnati per gravità e non approvano o bloccano il tuo PR, quindi i flussi di lavoro di revisione esistenti rimangono intatti. Puoi regolare cosa Claude segnala aggiungendo un file `CLAUDE.md` o `REVIEW.md` al tuo repository.

Per eseguire Claude nella tua infrastruttura CI invece di questo servizio gestito, vedi GitHub Actions o GitLab CI/CD.

Questa pagina copre:

- Come funzionano le revisioni
- Setup
- Personalizzazione delle revisioni con `CLAUDE.md` e `REVIEW.md`
- Prezzi

Come funzionano le revisioni

Una volta che un amministratore abilita Code Review per la tua organizzazione, le revisioni vengono eseguite automaticamente quando una pull request si apre o si aggiorna. Più agenti analizzano il diff e il codice circostante in parallelo sull'infrastruttura Anthropic. Ogni agente cerca una classe diversa di problema, quindi un passaggio di verifica controlla i candidati rispetto al comportamento effettivo del codice per filtrare i falsi positivi. I

risultati vengono deduplicati, classificati per gravità e pubblicati come commenti inline sulle righe specifiche in cui sono stati trovati i problemi. Se non vengono trovati problemi, Claude pubblica un breve commento di conferma sul PR.

Le revisioni si scalano in costo con la dimensione e la complessità del PR, completandosi in media in 20 minuti. Gli amministratori possono monitorare l'attività di revisione e la spesa tramite il dashboard di analisi.

Livelli di gravità

Ogni risultato è contrassegnato con un livello di gravità:

Marcatore	Gravità	Significato
🔴	Normale	Un bug che dovrebbe essere corretto prima del merge
🟡	Nit	Un problema minore, vale la pena correggerlo ma non bloccante
🟣	Pre-esistente	Un bug che esiste nella codebase ma non è stato introdotto da questo PR

I risultati includono una sezione di ragionamento esteso comprimibile che puoi espandere per capire perché Claude ha segnalato il problema e come ha verificato il problema.

Cosa controlla Code Review

Per impostazione predefinita, Code Review si concentra sulla correttezza: bug che interromperebbero la produzione, non preferenze di formattazione o copertura di test mancante. Puoi espandere cosa controlla aggiungendo file di guida al tuo repository.

Configura Code Review

Un amministratore abilita Code Review una volta per l'organizzazione e seleziona quali repository includere.

Step 1: Apri le impostazioni di amministrazione di Claude Code

Vai a claude.ai/admin-settings/claude-code e trova la sezione Code Review. Hai bisogno dell'accesso amministratore alla tua organizzazione Claude e del permesso di installare GitHub Apps nella tua organizzazione GitHub.

Step 2: Avvia il setup

Fai clic su **Setup**. Questo avvia il flusso di installazione dell'app GitHub.

Step 3: Installa l'app GitHub di Claude

Segui i prompt per installare l'app GitHub di Claude nella tua organizzazione GitHub. L'app richiede questi permessi del repository:

- **Contents**: lettura e scrittura
- **Issues**: lettura e scrittura
- **Pull requests**: lettura e scrittura

Code Review utilizza l'accesso in lettura ai contenuti e l'accesso in scrittura alle pull request. L'insieme di permessi più ampio supporta anche GitHub Actions se lo abiliti in seguito.

Step 4: Seleziona i repository

Scegli quali repository abilitare per Code Review. Se non vedi un repository, assicurati di aver dato all'app GitHub di Claude l'accesso durante l'installazione. Puoi aggiungere altri repository in seguito.

Step 5: Imposta i trigger di revisione per repository

Dopo il completamento del setup, la sezione Code Review mostra i tuoi repository in una tabella. Per ogni repository, usa il dropdown per scegliere quando vengono eseguite le revisioni:

- **After PR creation only**: la revisione viene eseguita una volta quando un PR viene aperto o contrassegnato come pronto per la revisione
- **After every push to PR branch**: la revisione viene eseguita ad ogni push, rilevando nuovi problemi mentre il PR evolve e risolvendo automaticamente i thread quando correggi i problemi segnalati

La revisione ad ogni push esegue più revisioni e costa di più. Inizia con la creazione del PR e passa a on-push per i repository in cui desideri una copertura continua e la pulizia automatica dei thread.

La tabella dei repository mostra anche il costo medio per revisione per ogni repository in base all'attività recente. Usa il menu delle azioni della riga per attivare o disattivare Code Review per repository, o per rimuovere completamente un repository.

Per verificare il setup, apri un PR di test. Un check run denominato **Claude Code Review** appare entro pochi minuti. Se non appare, conferma che il repository è elencato nelle tue impostazioni di amministrazione e che l'app GitHub di Claude ha accesso ad esso.

Personalizza le revisioni

Code Review legge due file dal tuo repository per guidare cosa segnala. Entrambi sono additivi rispetto ai controlli di correttezza predefiniti:

- `CLAUDE.md` : istruzioni di progetto condivise che Claude Code utilizza per tutti i compiti, non solo le revisioni. Usalo quando la guida si applica anche alle sessioni interattive di Claude Code.
- `REVIEW.md` : guida solo per la revisione, letta esclusivamente durante le revisioni del codice. Usalo per le regole che riguardano strettamente cosa segnalare o saltare durante la revisione e che ingombrerebbero il tuo `CLAUDE.md` generale.

CLAUDE.md

Code Review legge i file `CLAUDE.md` del tuo repository e tratta le violazioni appena introdotte come risultati a livello di nit. Questo funziona bidirezionalmente: se il tuo PR modifica il codice in modo da rendere obsoleta un'affermazione `CLAUDE.md`, Claude segnala che i documenti devono essere aggiornati.

Claude legge i file `CLAUDE.md` a ogni livello della tua gerarchia di directory, quindi le regole nel `CLAUDE.md` di una sottodirectory si applicano solo ai file sotto quel percorso. Vedi la documentazione della memoria per ulteriori informazioni su come funziona `CLAUDE.md`.

Per la guida specifica della revisione che non desideri applicare alle sessioni generali di Claude Code, usa `REVIEW.md` invece.

REVIEW.md

Aggiungi un file `REVIEW.md` alla radice del tuo repository per le regole specifiche della revisione. Usalo per codificare:

- Linee guida di stile dell'azienda o del team: "preferisci i ritorni anticipati ai condizionali annidati"
- Convenzioni specifiche del linguaggio o del framework non coperte dai linter
- Cose che Claude dovrebbe sempre segnalare: "qualsiasi nuova rotta API deve avere un test di integrazione"
- Cose che Claude dovrebbe saltare: "non commentare la formattazione nel codice generato sotto `/gen/`"

Esempio di `REVIEW.md`:

```
## Linee guida per la revisione del codice

### Controlla sempre
- I nuovi endpoint API hanno test di integrazione corrispondenti
- Le migrazioni del database sono retrocompatibili
- I messaggi di errore non perdono dettagli interni agli utenti

### Stile
- Preferisci le istruzioni `match` ai controlli `isinstance` concatenati
- Usa la registrazione strutturata, non l'interpolazione di stringhe f nelle
chiamate di log

### Salta
- File generati sotto `src/gen/`
- Modifiche solo di formattazione nei file `*.lock`
```

Claude scopre automaticamente `REVIEW.md` alla radice del repository. Nessuna configurazione necessaria.

Visualizza l'utilizzo

Vai a claude.ai/analytics/code-review per vedere l'attività di Code Review in tutta la tua organizzazione. Il dashboard mostra:

Sezione	Cosa mostra
PRs reviewed	Conteggio giornaliero delle pull request revisionate nell'intervallo di tempo selezionato
Cost weekly	Spesa settimanale su Code Review
Feedback	Conteggio dei commenti di revisione che sono stati risolti automaticamente perché uno sviluppatore ha affrontato il problema
Repository breakdown	Conteggi per repository dei PR revisionati e dei commenti risolti

La tabella dei repository nelle impostazioni di amministrazione mostra anche il costo medio per revisione per ogni repository.

Prezzi

Code Review viene fatturato in base all'utilizzo dei token. Le revisioni costano in media $15-25, scalando con la dimensione del PR, la complessità della codebase e quanti problemi richiedono verifica. L'utilizzo di Code Review viene fatturato separatamente tramite extra usage e non conta rispetto all'utilizzo incluso nel tuo piano.

Il trigger di revisione che scegli influisce sul costo totale:

- **After PR creation only**: viene eseguito una volta per PR
- **After every push**: viene eseguito ad ogni commit, moltiplicando il costo per il numero di push

I costi appaiono sulla tua fattura Anthropic indipendentemente dal fatto che la tua organizzazione utilizzi AWS Bedrock o Google Vertex AI per altre funzionalità di Claude Code. Per impostare un limite di spesa mensile per Code Review, vai a claude.ai/admin-settings/usage e configura il limite per il servizio Claude Code Review.

Monitora la spesa tramite il grafico dei costi settimanali in analytics o la colonna del costo medio per repository nelle impostazioni di amministrazione.

Risorse correlate

Code Review è progettato per funzionare insieme al resto di Claude Code. Se desideri eseguire revisioni localmente prima di aprire un PR, hai bisogno di una configurazione self-hosted, o desideri approfondire come `CLAUDE.md` modella il comportamento di Claude in tutti gli strumenti, queste pagine sono buoni prossimi passi:

- Plugins: sfoglia il marketplace dei plugin, incluso un plugin `code-review` per eseguire revisioni on-demand localmente prima di eseguire il push
- GitHub Actions: esegui Claude nei tuoi flussi di lavoro GitHub Actions per l'automazione personalizzata oltre la revisione del codice
- GitLab CI/CD: integrazione Claude self-hosted per le pipeline GitLab
- Memory: come funzionano i file `CLAUDE.md` in Claude Code
- Analytics: traccia l'utilizzo di Claude Code oltre la revisione del codice

Part 7: CI/CD & Integrations

Claude Code GitHub Actions

> *Scopri come integrare Claude Code nel tuo flusso di lavoro di sviluppo con Claude Code GitHub Actions*

Claude Code GitHub Actions porta l'automazione basata su AI al tuo flusso di lavoro GitHub. Con una semplice menzione `@claude` in qualsiasi PR o issue, Claude può analizzare il tuo codice, creare pull request, implementare funzionalità e correggere bug - il tutto seguendo gli standard del tuo progetto. Per le revisioni automatiche pubblicate su ogni PR senza un trigger, vedi GitHub Code Review.

Note:

Claude Code GitHub Actions è costruito sulla base dell'Claude Agent SDK, che consente l'integrazione programmatica di Claude Code nelle tue applicazioni. Puoi utilizzare l'SDK per costruire flussi di lavoro di automazione personalizzati oltre GitHub Actions.

Info:

Claude Opus 4.6 è ora disponibile. Claude Code GitHub Actions utilizza per impostazione predefinita Sonnet. Per utilizzare Opus 4.6, configura il parametro model per utilizzare `claude-opus-4-6`.

Perché utilizzare Claude Code GitHub Actions?

- **Creazione istantanea di PR**: Descrivi ciò di cui hai bisogno e Claude crea una PR completa con tutti i cambiamenti necessari
- **Implementazione automatica del codice**: Trasforma gli issue in codice funzionante con un singolo comando
- **Segue i tuoi standard**: Claude rispetta le tue linee guida `CLAUDE.md` e i pattern di codice esistenti
- **Configurazione semplice**: Inizia in pochi minuti con il nostro installer e la chiave API
- **Sicuro per impostazione predefinita**: Il tuo codice rimane sui runner di Github

Cosa può fare Claude?

Claude Code fornisce un potente GitHub Action che trasforma il modo in cui lavori con il codice:

Claude Code Action

Questo GitHub Action ti consente di eseguire Claude Code all'interno dei tuoi flussi di lavoro GitHub Actions. Puoi utilizzarlo per costruire qualsiasi flusso di lavoro personalizzato sulla base di Claude Code.

Visualizza repository →

Setup

Configurazione rapida

Il modo più semplice per configurare questa action è attraverso Claude Code nel terminale. Basta aprire claude ed eseguire `/install-github-app`.

Questo comando ti guiderà attraverso la configurazione dell'app GitHub e dei secret richiesti.

> **Note:**
>
> - Devi essere un amministratore del repository per installare l'app GitHub e aggiungere secret
> - L'app GitHub richiederà autorizzazioni di lettura e scrittura per Contents, Issues e Pull requests
> - Questo metodo di avvio rapido è disponibile solo per gli utenti diretti dell'API Claude. Se stai utilizzando AWS Bedrock o Google Vertex AI, consulta la sezione Utilizzo con AWS Bedrock e Google Vertex AI.

Configurazione manuale

Se il comando `/install-github-app` non riesce o preferisci una configurazione manuale, segui queste istruzioni di configurazione manuale:

1. **Installa l'app GitHub di Claude** nel tuo repository: https://github.com/apps/claude

   L'app GitHub di Claude richiede le seguenti autorizzazioni del repository:

   - **Contents**: Lettura e scrittura (per modificare i file del repository)
   - **Issues**: Lettura e scrittura (per rispondere agli issue)

- **Pull requests**: Lettura e scrittura (per creare PR e spingere i cambiamenti)

Per ulteriori dettagli sulla sicurezza e le autorizzazioni, vedi la documentazione sulla sicurezza.

2. **Aggiungi ANTHROPIC_API_KEY** ai tuoi secret del repository (Scopri come utilizzare i secret in GitHub Actions)
3. **Copia il file del flusso di lavoro** da examples/claude.yml nella cartella `.github/workflows/` del tuo repository

> **Tip:**
>
> Dopo aver completato la configurazione rapida o manuale, testa l'action taggando `@claude` in un commento di issue o PR.

Aggiornamento dalla versione Beta

> **Warning:**
>
> Claude Code GitHub Actions v1.0 introduce breaking changes che richiedono l'aggiornamento dei tuoi file di flusso di lavoro per eseguire l'upgrade a v1.0 dalla versione beta.

Se stai attualmente utilizzando la versione beta di Claude Code GitHub Actions, ti consigliamo di aggiornare i tuoi flussi di lavoro per utilizzare la versione GA. La nuova versione semplifica la configurazione aggiungendo potenti nuove funzionalità come il rilevamento automatico della modalità.

Cambiamenti essenziali

Tutti gli utenti beta devono apportare questi cambiamenti ai loro file di flusso di lavoro per eseguire l'upgrade:

1. **Aggiorna la versione dell'action**: Cambia `@beta` a `@v1`
2. **Rimuovi la configurazione della modalità**: Elimina `mode: "tag"` o `mode: "agent"` (ora rilevata automaticamente)
3. **Aggiorna gli input del prompt**: Sostituisci `direct_prompt` con `prompt`
4. **Sposta le opzioni CLI**: Converti `max_turns`, `model`, `custom_instructions`, ecc. in `claude_args`

Breaking Changes Reference

Old Beta Input	New v1.0 Input
`mode`	*(Removed - auto-detected)*
`direct_prompt`	`prompt`
`override_prompt`	`prompt` with GitHub variables
`custom_instructions`	`claude_args: --append-system-prompt`
`max_turns`	`claude_args: --max-turns`
`model`	`claude_args: --model`
`allowed_tools`	`claude_args: --allowedTools`
`disallowed_tools`	`claude_args: --disallowedTools`
`claude_env`	`settings` JSON format

Esempio Prima e Dopo

Versione beta:

```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"
```

Versione GA (v1.0):

```
- uses: anthropics/claude-code-action@v1
  with:
    prompt: "Review this PR for security issues"
    anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
    claude_args: |
      --append-system-prompt "Follow our coding standards"
      --max-turns 10
      --model claude-sonnet-4-6
```

> **Tip:**
>
> L'action ora rileva automaticamente se eseguire in modalità interattiva (risponde alle menzioni `@claude`) o in modalità automazione (viene eseguita immediatamente con un prompt) in base alla tua configurazione.

Esempi di casi d'uso

Claude Code GitHub Actions può aiutarti con una varietà di attività. La directory degli esempi contiene flussi di lavoro pronti all'uso per diversi scenari.

Flusso di lavoro di base

```
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
```

Utilizzo di 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"
```

Automazione personalizzata con prompt

```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"
```

Casi d’uso comuni

Nei commenti di issue o PR:

```
@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 analizzerà automaticamente il contesto e risponderà in modo appropriato.

Best practices

Configurazione CLAUDE.md

Crea un file `CLAUDE.md` nella radice del tuo repository per definire le linee guida dello stile di codice, i criteri di revisione, le regole specifiche del progetto e i pattern preferiti. Questo file guida la comprensione di Claude degli standard del tuo progetto.

Considerazioni sulla sicurezza

> **Warning:**
>
> Non eseguire mai il commit delle chiavi API direttamente nel tuo repository.

Per una guida completa sulla sicurezza inclusa autorizzazioni, autenticazione e best practices, vedi la documentazione sulla sicurezza di Claude Code Action.

Utilizza sempre GitHub Secrets per le chiavi API:

- Aggiungi la tua chiave API come secret del repository denominato `ANTHROPIC_API_KEY`
- Fai riferimento ad essa nei flussi di lavoro: `anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}`
- Limita le autorizzazioni dell'action solo a ciò che è necessario
- Rivedi i suggerimenti di Claude prima di eseguire il merge

Utilizza sempre GitHub Secrets (ad esempio, `${{ secrets.ANTHROPIC_API_KEY }}`) piuttosto che hardcodare le chiavi API direttamente nei tuoi file di flusso di lavoro.

Ottimizzazione delle prestazioni

Utilizza i template di issue per fornire contesto, mantieni il tuo `CLAUDE.md` conciso e focalizzato, e configura timeout appropriati per i tuoi flussi di lavoro.

Costi CI

Quando utilizzi Claude Code GitHub Actions, tieni presente i costi associati:

Costi di GitHub Actions:

- Claude Code viene eseguito su runner ospitati da GitHub, che consumano i tuoi minuti di GitHub Actions
- Vedi la documentazione sulla fatturazione di GitHub per i dettagli sui prezzi e i limiti di minuti

Costi API:

- Ogni interazione di Claude consuma token API in base alla lunghezza dei prompt e delle risposte
- L'utilizzo dei token varia in base alla complessità dell'attività e alla dimensione della codebase
- Vedi la pagina dei prezzi di Claude per i tassi di token attuali

Suggerimenti per l'ottimizzazione dei costi:

- Utilizza comandi specifici `@claude` per ridurre le chiamate API non necessarie
- Configura `--max-turns` appropriato in `claude_args` per prevenire iterazioni eccessive
- Imposta timeout a livello di flusso di lavoro per evitare job fuori controllo
- Considera l'utilizzo dei controlli di concorrenza di GitHub per limitare le esecuzioni parallele

Esempi di configurazione

Claude Code Action v1 semplifica la configurazione con parametri unificati:

```
- 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
```

Caratteristiche principali:

- **Interfaccia prompt unificata** - Utilizza `prompt` per tutte le istruzioni
- **Skills** - Richiama skills installate direttamente dal prompt
- **Passthrough CLI** - Qualsiasi argomento CLI di Claude Code tramite `claude_args`
- **Trigger flessibili** - Funziona con qualsiasi evento GitHub

Visita la directory degli esempi per i file di flusso di lavoro completi.

> **Tip:**
>
> Quando rispondi ai commenti di issue o PR, Claude risponde automaticamente alle menzioni @claude. Per altri eventi, utilizza il parametro `prompt` per fornire istruzioni.

Utilizzo con AWS Bedrock e Google Vertex AI

Per ambienti aziendali, puoi utilizzare Claude Code GitHub Actions con la tua infrastruttura cloud. Questo approccio ti dà il controllo sulla residenza dei dati e sulla fatturazione mantenendo la stessa funzionalità.

Prerequisiti

Prima di configurare Claude Code GitHub Actions con i provider cloud, hai bisogno di:

Per Google Cloud Vertex AI:

1. Un progetto Google Cloud con Vertex AI abilitato
2. Workload Identity Federation configurato per GitHub Actions
3. Un account di servizio con le autorizzazioni richieste
4. Un'app GitHub (consigliato) o utilizza il GITHUB_TOKEN predefinito

Per AWS Bedrock:

1. Un account AWS con Amazon Bedrock abilitato
2. GitHub OIDC Identity Provider configurato in AWS
3. Un ruolo IAM con autorizzazioni Bedrock
4. Un'app GitHub (consigliato) o utilizza il GITHUB_TOKEN predefinito

Step 1: Crea un'app GitHub personalizzata (Consigliato per provider di terze parti)

Per il miglior controllo e sicurezza quando utilizzi provider di terze parti come Vertex AI o Bedrock, ti consigliamo di creare la tua app GitHub:

1. Vai a https://github.com/settings/apps/new
2. Compila le informazioni di base:

- **GitHub App name**: Scegli un nome univoco (ad es. "YourOrg Claude Assistant")

- **Homepage URL**: Il sito web della tua organizzazione o l'URL del repository

1. Configura le impostazioni dell'app:

- **Webhooks**: Deseleziona "Active" (non necessario per questa integrazione)

1. Imposta le autorizzazioni richieste:

- **Repository permissions**:
- Contents: Read & Write
- Issues: Read & Write
- Pull requests: Read & Write

1. Fai clic su "Create GitHub App"
2. Dopo la creazione, fai clic su "Generate a private key" e salva il file `.pem` scaricato
3. Annota il tuo App ID dalla pagina delle impostazioni dell'app
4. Installa l'app nel tuo repository:

- Dalla pagina delle impostazioni della tua app, fai clic su "Install App" nella barra laterale sinistra
- Seleziona il tuo account o organizzazione
- Scegli "Only select repositories" e seleziona il repository specifico
- Fai clic su "Install"

1. Aggiungi la chiave privata come secret al tuo repository:

- Vai a Settings → Secrets and variables → Actions del tuo repository
- Crea un nuovo secret denominato `APP_PRIVATE_KEY` con il contenuto del file `.pem`

1. Aggiungi l'App ID come secret:

- Crea un nuovo secret denominato `APP_ID` con l'ID della tua app GitHub

> **Note:**
>
> Questa app verrà utilizzata con l'action actions/create-github-app-token per generare token di autenticazione nei tuoi flussi di lavoro.

Alternativa per Claude API o se non vuoi configurare la tua app Github: Utilizza l'app ufficiale di Anthropic:

1. Installa da: https://github.com/apps/claude
2. Nessuna configurazione aggiuntiva necessaria per l'autenticazione

Step 2: Configura l'autenticazione del provider cloud

Scegli il tuo provider cloud e configura l'autenticazione sicura:

Configura AWS per consentire a GitHub Actions di autenticarsi in modo sicuro senza archiviare le credenziali.

> ***Security Note**: Utilizza configurazioni specifiche del repository e concedi solo le autorizzazioni minime richieste.*

Required Setup:

1. **Enable Amazon Bedrock**:

- Richiedi l'accesso ai modelli Claude in Amazon Bedrock
- Per i modelli tra regioni, richiedi l'accesso in tutte le regioni richieste

1. **Set up GitHub OIDC Identity Provider**:

- Provider URL: `https://token.actions.githubusercontent.com`
- Audience: `sts.amazonaws.com`

1. **Create IAM Role for GitHub Actions**:

- Trusted entity type: Web identity
- Identity provider: `token.actions.githubusercontent.com`
- Permissions: `AmazonBedrockFullAccess` policy
- Configure trust policy for your specific repository

Required Values:

Dopo la configurazione, avrai bisogno di:

- **AWS_ROLE_TO_ASSUME**: L'ARN del ruolo IAM che hai creato

> **Tip:**
>
> OIDC è più sicuro rispetto all'utilizzo di chiavi di accesso AWS statiche perché le credenziali sono temporanee e ruotate automaticamente.

Vedi la documentazione AWS per le istruzioni dettagliate sulla configurazione di OIDC.

Configura Google Cloud per consentire a GitHub Actions di autenticarsi in modo sicuro senza archiviare le credenziali.

> ***Security Note**: Utilizza configurazioni specifiche del repository e concedi solo le autorizzazioni minime richieste.*

Required Setup:

1. **Enable APIs** nel tuo progetto Google Cloud:

- IAM Credentials API
- Security Token Service (STS) API
- Vertex AI API

1. **Create Workload Identity Federation resources**:

- Crea un Workload Identity Pool
- Aggiungi un provider OIDC GitHub con:
- Issuer: `https://token.actions.githubusercontent.com`
- Attribute mappings for repository and owner
- **Security recommendation**: Utilizza condizioni di attributo specifiche del repository

1. **Create a Service Account**:

- Concedi solo il ruolo `Vertex AI User`
- **Security recommendation**: Crea un account di servizio dedicato per repository

1. **Configure IAM bindings**:

- Consenti al Workload Identity Pool di rappresentare l'account di servizio
- **Security recommendation**: Utilizza set di principali specifici del repository

Required Values:

Dopo la configurazione, avrai bisogno di:

- **GCP_WORKLOAD_IDENTITY_PROVIDER**: Il nome completo della risorsa provider
- **GCP_SERVICE_ACCOUNT**: L'indirizzo email dell'account di servizio

> **Tip:**
>
> Workload Identity Federation elimina la necessità di chiavi di account di servizio scaricabili, migliorando la sicurezza.

Per le istruzioni di configurazione dettagliate, consulta la documentazione di Google Cloud Workload Identity Federation.

Step 3: Aggiungi Secret Richiesti

Aggiungi i seguenti secret al tuo repository (Settings → Secrets and variables → Actions):

Per Claude API (Direct):

1. **Per l'autenticazione API**:

- `ANTHROPIC_API_KEY` : La tua chiave API Claude da console.anthropic.com

1. **Per GitHub App (se utilizzi la tua app)**:

- `APP_ID` : L'ID della tua app GitHub
- `APP_PRIVATE_KEY` : Il contenuto della chiave privata (.pem)

Per Google Cloud Vertex AI

1. **Per l'autenticazione GCP**:

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

1. **Per GitHub App (se utilizzi la tua app)**:

- `APP_ID` : L'ID della tua app GitHub
- `APP_PRIVATE_KEY` : Il contenuto della chiave privata (.pem)

Per AWS Bedrock

1. **Per l'autenticazione AWS**:

- `AWS_ROLE_TO_ASSUME`

1. **Per GitHub App (se utilizzi la tua app)**:

- `APP_ID` : L'ID della tua app GitHub
- `APP_PRIVATE_KEY` : Il contenuto della chiave privata (.pem)

Step 4: Crea file di flusso di lavoro

Crea file di flusso di lavoro GitHub Actions che si integrano con il tuo provider cloud. Gli esempi seguenti mostrano configurazioni complete sia per AWS Bedrock che per Google Vertex AI:

Prerequisites:

- AWS Bedrock access enabled with Claude model permissions
- GitHub configured as an OIDC identity provider in AWS
- IAM role with Bedrock permissions that trusts GitHub Actions

Required GitHub secrets:

Secret Name	Description
`AWS_ROLE_TO_ASSUME`	ARN of the IAM role for Bedrock access
`APP_ID`	Your GitHub App ID (from app settings)
`APP_PRIVATE_KEY`	The private key you generated for your GitHub App

```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:**
>
> Il formato dell'ID del modello per Bedrock include un prefisso di regione (ad esempio, `us.anthropic.claude-sonnet-4-6`).

Prerequisites:

- Vertex AI API enabled in your GCP project
- Workload Identity Federation configured for GitHub
- Service account with Vertex AI permissions

Required GitHub secrets:

Secret Name	Description
`GCP_WORKLOAD_IDENTITY_PROVIDER`	Workload identity provider resource name
`GCP_SERVICE_ACCOUNT`	Service account email with Vertex AI access
`APP_ID`	Your GitHub App ID (from app settings)
`APP_PRIVATE_KEY`	The private key you generated for your GitHub App

```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:**
>
> L'ID del progetto viene recuperato automaticamente dal passaggio di autenticazione di Google Cloud, quindi non è necessario codificarlo.

Troubleshooting

Claude non risponde ai comandi @claude

Verifica che l'app GitHub sia installata correttamente, controlla che i flussi di lavoro siano abilitati, assicurati che la chiave API sia impostata nei secret del repository e conferma che il commento contenga `@claude` (non `/claude`).

CI non in esecuzione sui commit di Claude

Assicurati di utilizzare l'app GitHub o l'app personalizzata (non l'utente Actions), controlla che i trigger del flusso di lavoro includano gli eventi necessari e verifica che le autorizzazioni dell'app includano i trigger CI.

Errori di autenticazione

Conferma che la chiave API sia valida e abbia autorizzazioni sufficienti. Per Bedrock/Vertex, controlla la configurazione delle credenziali e assicurati che i secret siano denominati correttamente nei flussi di lavoro.

Configurazione avanzata

Parametri dell'action

Claude Code Action v1 utilizza una configurazione semplificata:

Parameter	Description	Required
`prompt`	Istruzioni per Claude (testo semplice o un nome di skill)	No*
`claude_args`	Argomenti CLI passati a Claude Code	No
`anthropic_api_key`	Chiave API Claude	Yes**
`github_token`	Token GitHub per l'accesso API	No
`trigger_phrase`	Frase trigger personalizzata (predefinito: "@claude")	No
`use_bedrock`	Utilizza AWS Bedrock invece dell'API Claude	No
`use_vertex`	Utilizza Google Vertex AI invece dell'API Claude	No

*Prompt è opzionale - quando omesso per i commenti di issue/PR, Claude risponde alla frase trigger
**Richiesto per l'API Claude diretta, non per Bedrock/Vertex

Passa argomenti CLI

Il parametro `claude_args` accetta qualsiasi argomento CLI di Claude Code:

```
claude_args: "--max-turns 5 --model claude-sonnet-4-6 --mcp-config /path/to/
config.json"
```

Argomenti comuni:

- `--max-turns` : Numero massimo di turni di conversazione (predefinito: 10)
- `--model` : Modello da utilizzare (ad es. `claude-sonnet-4-6`)
- `--mcp-config` : Percorso della configurazione MCP
- `--allowed-tools` : Elenco separato da virgole degli strumenti consentiti
- `--debug` : Abilita l'output di debug

Metodi di integrazione alternativi

Mentre il comando `/install-github-app` è l'approccio consigliato, puoi anche:

- **Custom GitHub App**: Per le organizzazioni che necessitano di nomi utente personalizzati o flussi di autenticazione personalizzati. Crea la tua app GitHub con le autorizzazioni richieste (contents, issues, pull requests) e utilizza l'action actions/create-github-app-token per generare token nei tuoi flussi di lavoro.
- **Manual GitHub Actions**: Configurazione diretta del flusso di lavoro per la massima flessibilità
- **MCP Configuration**: Caricamento dinamico dei server Model Context Protocol

Vedi la documentazione di Claude Code Action per guide dettagliate su autenticazione, sicurezza e configurazione avanzata.

Personalizzazione del comportamento di Claude

Puoi configurare il comportamento di Claude in due modi:

1. **CLAUDE.md**: Definisci gli standard di codifica, i criteri di revisione e le regole specifiche del progetto in un file `CLAUDE.md` nella radice del tuo repository. Claude seguirà queste linee guida quando crea PR e risponde alle richieste. Consulta la nostra documentazione Memory per ulteriori dettagli.
2. **Custom prompts**: Utilizza il parametro `prompt` nel file del flusso di lavoro per fornire istruzioni specifiche del flusso di lavoro. Questo ti consente di personalizzare il comportamento di Claude per diversi flussi di lavoro o attività.

Claude seguirà queste linee guida quando crea PR e risponde alle richieste.

Claude Code GitLab CI/CD

Scopri come integrare Claude Code nel tuo flusso di lavoro di sviluppo con GitLab CI/CD

Info:

Claude Code per GitLab CI/CD è attualmente in beta. Le funzionalità e la funzionalità possono evolversi mentre perfezzioniamo l'esperienza.

Questa integrazione è mantenuta da GitLab. Per il supporto, consultare il seguente problema GitLab.

Note:

Questa integrazione è costruita sulla base di Claude Code CLI e Agent SDK, consentendo l'uso programmatico di Claude nei vostri lavori CI/CD e flussi di lavoro di automazione personalizzati.

Perché utilizzare Claude Code con GitLab?

- **Creazione istantanea di MR**: Descrivete ciò di cui avete bisogno e Claude propone un MR completo con modifiche e spiegazione
- **Implementazione automatizzata**: Trasformate i problemi in codice funzionante con un singolo comando o menzione
- **Consapevole del progetto**: Claude segue le vostre linee guida `CLAUDE.md` e i modelli di codice esistenti
- **Configurazione semplice**: Aggiungete un lavoro a `.gitlab-ci.yml` e una variabile CI/CD mascherata
- **Pronto per l'azienda**: Scegliete Claude API, AWS Bedrock o Google Vertex AI per soddisfare le esigenze di residenza dei dati e approvvigionamento
- **Sicuro per impostazione predefinita**: Viene eseguito nei vostri runner GitLab con la vostra protezione dei rami e approvazioni

Come funziona

Claude Code utilizza GitLab CI/CD per eseguire attività di intelligenza artificiale in lavori isolati e eseguire il commit dei risultati tramite MR:

1. **Orchestrazione basata su eventi**: GitLab ascolta i trigger scelti (ad esempio, un commento che menziona `@claude` in un problema, MR o thread di revisione). Il lavoro raccoglie il contesto dal thread e dal repository, costruisce prompt da tale input ed esegue Claude Code.
2. **Astrazione del provider**: Utilizzate il provider che si adatta al vostro ambiente:
   - Claude API (SaaS)
   - AWS Bedrock (accesso basato su IAM, opzioni multi-regione)
   - Google Vertex AI (nativo GCP, Workload Identity Federation)
3. **Esecuzione in sandbox**: Ogni interazione viene eseguita in un contenitore con regole rigorose di rete e filesystem. Claude Code applica autorizzazioni con ambito workspace per limitare le scritture. Ogni modifica passa attraverso un MR in modo che i revisori vedano il diff e le approvazioni si applichino ancora.

Scegliete endpoint regionali per ridurre la latenza e soddisfare i requisiti di sovranità dei dati mentre utilizzate gli accordi cloud esistenti.

Cosa può fare Claude?

Claude Code abilita potenti flussi di lavoro CI/CD che trasformano il modo in cui lavorate con il codice:

- Creare e aggiornare MR da descrizioni di problemi o commenti
- Analizzare regressioni di prestazioni e proporre ottimizzazioni
- Implementare funzionalità direttamente in un ramo, quindi aprire un MR
- Correggere bug e regressioni identificati da test o commenti
- Rispondere ai commenti di follow-up per iterare sulle modifiche richieste

Configurazione

Configurazione rapida

Il modo più veloce per iniziare è aggiungere un lavoro minimo al vostro `.gitlab-ci.yml` e impostare la vostra chiave API come variabile mascherata.

1. **Aggiungere una variabile CI/CD mascherata**
   - Andate a **Impostazioni → CI/CD → Variabili**

- Aggiungete `ANTHROPIC_API_KEY` (mascherata, protetta secondo necessità)

2. **Aggiungere un lavoro Claude a `.gitlab-ci.yml`**

```
stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  # Regolate le regole per adattarsi a come desiderate attivare il lavoro:
  # - esecuzioni manuali
  # - eventi di merge request
  # - trigger web/API quando un commento contiene '@claude'
  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:
    # Facoltativo: avviare un server GitLab MCP se la vostra configurazione lo
fornisce
    - /bin/gitlab-mcp-server || true
    # Utilizzate le variabili AI_FLOW_* quando richiamate tramite trigger web/API
con payload di contesto
    - 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
```

Dopo aver aggiunto il lavoro e la vostra variabile `ANTHROPIC_API_KEY`, testate eseguendo il lavoro manualmente da **CI/CD → Pipeline**, oppure attivate da un MR per consentire a Claude di proporre aggiornamenti in un ramo e aprire un MR se necessario.

> **Note:**
>
> Per eseguire su AWS Bedrock o Google Vertex AI invece dell'API Claude, consultate la sezione Utilizzo con AWS Bedrock e Google Vertex AI di seguito per la configurazione dell'autenticazione e dell'ambiente.

Configurazione manuale (consigliata per la produzione)

Se preferite una configurazione più controllata o avete bisogno di provider aziendali:

1. **Configurare l'accesso al provider**:
   - **Claude API**: Creare e archiviare `ANTHROPIC_API_KEY` come variabile CI/CD mascherata
   - **AWS Bedrock**: **Configurare GitLab → AWS OIDC** e creare un ruolo IAM per Bedrock
   - **Google Vertex AI**: **Configurare Workload Identity Federation per GitLab → GCP**
2. **Aggiungere credenziali di progetto per le operazioni dell'API GitLab**:
   - Utilizzate `CI_JOB_TOKEN` per impostazione predefinita, oppure create un Project Access Token con ambito `api`
   - Archiviate come `GITLAB_ACCESS_TOKEN` (mascherato) se utilizzate un PAT
3. **Aggiungere il lavoro Claude a `.gitlab-ci.yml`** (vedere gli esempi di seguito)
4. **(Facoltativo) Abilitare trigger basati su menzioni**:
   - Aggiungere un webhook di progetto per "Commenti (note)" al vostro listener di eventi (se ne utilizzate uno)
   - Fare in modo che il listener chiami l'API di attivazione della pipeline con variabili come `AI_FLOW_INPUT` e `AI_FLOW_CONTEXT` quando un commento contiene `@claude`

Esempi di casi d'uso

Trasformare i problemi in MR

In un commento di problema:

```
@claude implement this feature based on the issue description
```

Claude analizza il problema e la base di codice, scrive le modifiche in un ramo e apre un MR per la revisione.

Ottenere aiuto nell'implementazione

In una discussione MR:

```
@claude suggest a concrete approach to cache the results of this API call
```

Claude propone modifiche, aggiunge codice con caching appropriato e aggiorna il MR.

Correggere i bug rapidamente

In un commento di problema o MR:

```
@claude fix the TypeError in the user dashboard component
```

Claude individua il bug, implementa una correzione e aggiorna il ramo o apre un nuovo MR.

Utilizzo con AWS Bedrock e Google Vertex AI

Per ambienti aziendali, potete eseguire Claude Code interamente sulla vostra infrastruttura cloud con la stessa esperienza per gli sviluppatori.

AWS Bedrock

Prerequisiti

Prima di configurare Claude Code con AWS Bedrock, avete bisogno di:

1. Un account AWS con accesso ad Amazon Bedrock ai modelli Claude desiderati
2. GitLab configurato come provider di identità OIDC in AWS IAM
3. Un ruolo IAM con autorizzazioni Bedrock e una politica di trust limitata al vostro progetto/rami GitLab
4. Variabili CI/CD GitLab per l'assunzione del ruolo:

- `AWS_ROLE_TO_ASSUME` (ARN del ruolo)
- `AWS_REGION` (regione Bedrock)

Istruzioni di configurazione

Configurate AWS per consentire ai lavori CI di GitLab di assumere un ruolo IAM tramite OIDC (nessuna chiave statica).

Configurazione richiesta:

1. Abilitare Amazon Bedrock e richiedere l'accesso ai vostri modelli Claude target
2. Creare un provider OIDC IAM per GitLab se non già presente
3. Creare un ruolo IAM attendibile dal provider OIDC di GitLab, limitato al vostro progetto e rami protetti
4. Allegare autorizzazioni con privilegi minimi per le API di invocazione Bedrock

Valori richiesti da archiviare nelle variabili CI/CD:

- `AWS_ROLE_TO_ASSUME`
- `AWS_REGION`

Aggiungete le variabili in Impostazioni → CI/CD → Variabili:

```
## Per AWS Bedrock:
- AWS_ROLE_TO_ASSUME
- AWS_REGION
```

Utilizzate l'esempio di lavoro AWS Bedrock sopra per scambiare il token di lavoro GitLab con credenziali AWS temporanee in fase di esecuzione.

Google Vertex AI

Prerequisiti

Prima di configurare Claude Code con Google Vertex AI, avete bisogno di:

1. Un progetto Google Cloud con:
   - API Vertex AI abilitata
   - Workload Identity Federation configurata per attendere OIDC di GitLab
1. Un account di servizio dedicato con solo i ruoli Vertex AI richiesti
2. Variabili CI/CD GitLab per WIF:
   - `GCP_WORKLOAD_IDENTITY_PROVIDER` (nome completo della risorsa)
   - `GCP_SERVICE_ACCOUNT` (email dell'account di servizio)

Istruzioni di configurazione

Configurate Google Cloud per consentire ai lavori CI di GitLab di rappresentare un account di servizio tramite Workload Identity Federation.

Configurazione richiesta:

1. Abilitare API IAM Credentials, STS API e Vertex AI API
2. Creare un Workload Identity Pool e un provider per OIDC di GitLab
3. Creare un account di servizio dedicato con ruoli Vertex AI
4. Concedere al principale WIF l'autorizzazione per rappresentare l'account di servizio

Valori richiesti da archiviare nelle variabili CI/CD:

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

Aggiungete le variabili in Impostazioni → CI/CD → Variabili:

```
## Per Google Vertex AI:
- GCP_WORKLOAD_IDENTITY_PROVIDER
- GCP_SERVICE_ACCOUNT
- CLOUD_ML_REGION (ad esempio, us-east5)
```

Utilizzate l'esempio di lavoro Google Vertex AI sopra per autenticarvi senza archiviare le chiavi.

Esempi di configurazione

Di seguito sono riportati frammenti pronti all'uso che potete adattare alla vostra pipeline.

.gitlab-ci.yml di base (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 utilizzerà ANTHROPIC_API_KEY dalle variabili CI/CD
```

Esempio di lavoro AWS Bedrock (OIDC)

Prerequisiti:

- Amazon Bedrock abilitato con accesso ai vostri modelli Claude scelti
- OIDC di GitLab configurato in AWS con un ruolo che attendibile al vostro progetto e rami GitLab
- Ruolo IAM con autorizzazioni Bedrock (privilegi minimi consigliati)

Variabili CI/CD richieste:

- `AWS_ROLE_TO_ASSUME` : ARN del ruolo IAM per l'accesso a Bedrock
- `AWS_REGION` : Regione Bedrock (ad esempio, `us-west-2`)

```
claude-bedrock:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apk add --no-cache bash curl jq git python3 py3-pip
    - pip install --no-cache-dir awscli
    - curl -fsSL https://claude.ai/install.sh | bash
    # Scambiare il token OIDC di GitLab con credenziali AWS
    - export AWS_WEB_IDENTITY_TOKEN_FILE="${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:

> Gli ID modello per Bedrock includono prefissi specifici della regione (ad esempio, `us.anthropic.claude-sonnet-4-6`). Passate il modello desiderato tramite la configurazione del lavoro o il prompt se il vostro flusso di lavoro lo supporta.

Esempio di lavoro Google Vertex AI (Workload Identity Federation)

Prerequisiti:

- API Vertex AI abilitata nel vostro progetto GCP
- Workload Identity Federation configurata per attendere OIDC di GitLab
- Un account di servizio con autorizzazioni Vertex AI

Variabili CI/CD richieste:

- `GCP_WORKLOAD_IDENTITY_PROVIDER`: Nome completo della risorsa del provider
- `GCP_SERVICE_ACCOUNT`: Email dell'account di servizio
- `CLOUD_ML_REGION`: Regione Vertex (ad esempio, `us-east5`)

```
claude-vertex:
  stage: ai
  image: gcr.io/google.com/cloudsdktool/google-cloud-cli:slim
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apt-get update && apt-get install -y git && apt-get clean
    - curl -fsSL https://claude.ai/install.sh | bash
    # Autenticarsi a Google Cloud tramite WIF (nessuna chiave scaricata)
    - >
      gcloud auth login --cred-file=<(cat <<EOF
      {
        "type": "external_account",
        "audience": "${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:**
>
> Con Workload Identity Federation, non è necessario archiviare le chiavi dell'account di servizio. Utilizzate condizioni di trust specifiche del repository e account di servizio con privilegi minimi.

Best practice

Configurazione CLAUDE.md

Create un file `CLAUDE.md` nella radice del repository per definire standard di codifica, criteri di revisione e regole specifiche del progetto. Claude legge questo file durante le esecuzioni e segue le vostre convenzioni quando propone modifiche.

Considerazioni sulla sicurezza

Non eseguite mai il commit di chiavi API o credenziali cloud nel vostro repository. Utilizzate sempre le variabili CI/CD di GitLab:

- Aggiungete `ANTHROPIC_API_KEY` come variabile mascherata (e proteggetela se necessario)
- Utilizzate OIDC specifico del provider dove possibile (nessuna chiave di lunga durata)
- Limitate le autorizzazioni dei lavori e l'uscita di rete
- Revisionate i MR di Claude come qualsiasi altro contributore

Ottimizzazione delle prestazioni

- Mantenete `CLAUDE.md` focalizzato e conciso
- Fornite descrizioni chiare di problemi/MR per ridurre le iterazioni
- Configurate timeout di lavoro ragionevoli per evitare esecuzioni incontrollate
- Memorizzate nella cache npm e installazioni di pacchetti nei runner dove possibile

Costi CI

Quando utilizzate Claude Code con GitLab CI/CD, siate consapevoli dei costi associati:

- **Tempo del runner GitLab**:
  - Claude viene eseguito sui vostri runner GitLab e consuma minuti di calcolo
  - Consultate la fatturazione del runner del vostro piano GitLab per i dettagli
- **Costi API**:
  - Ogni interazione Claude consuma token in base alla dimensione del prompt e della risposta

    - L'utilizzo dei token varia in base alla complessità dell'attività e alla dimensione della base di codice
    - Consultate Prezzi Anthropic per i dettagli

- **Suggerimenti per l'ottimizzazione dei costi**:
    - Utilizzate comandi `@claude` specifici per ridurre i turni non necessari
    - Impostate valori `max_turns` e timeout di lavoro appropriati
    - Limitate la concorrenza per controllare le esecuzioni parallele

Sicurezza e governance

- Ogni lavoro viene eseguito in un contenitore isolato con accesso di rete limitato
- Le modifiche di Claude passano attraverso MR in modo che i revisori vedano ogni diff
- Le regole di protezione dei rami e approvazione si applicano al codice generato da IA
- Claude Code utilizza autorizzazioni con ambito workspace per limitare le scritture
- I costi rimangono sotto il vostro controllo perché portate le vostre credenziali del provider

Risoluzione dei problemi

Claude non risponde ai comandi @claude

- Verificate che la vostra pipeline sia attivata (manualmente, evento MR o tramite listener di note/webhook)
- Assicuratevi che le variabili CI/CD (`ANTHROPIC_API_KEY` o impostazioni del provider cloud) siano presenti e non mascherate
- Controllate che il commento contenga `@claude` (non `/claude`) e che il vostro trigger di menzione sia configurato

Il lavoro non può scrivere commenti o aprire MR

- Assicuratevi che `CI_JOB_TOKEN` abbia autorizzazioni sufficienti per il progetto, oppure utilizzate un Project Access Token con ambito `api`
- Controllate che lo strumento `mcp__gitlab` sia abilitato in `--allowedTools`
- Confermate che il lavoro viene eseguito nel contesto del MR o abbia contesto sufficiente tramite variabili `AI_FLOW_*`

Errori di autenticazione

- **Per Claude API**: Confermate che `ANTHROPIC_API_KEY` sia valida e non scaduta

- **Per Bedrock/Vertex**: Verificate la configurazione OIDC/WIF, l'impersonificazione del ruolo e i nomi segreti; confermate la disponibilità della regione e del modello

Configurazione avanzata

Parametri e variabili comuni

Claude Code supporta questi input comunemente utilizzati:

- `prompt` / `prompt_file`: Fornite istruzioni inline (`-p`) o tramite un file
- `max_turns`: Limitate il numero di iterazioni avanti e indietro
- `timeout_minutes`: Limitate il tempo di esecuzione totale
- `ANTHROPIC_API_KEY`: Richiesto per l'API Claude (non utilizzato per Bedrock/Vertex)
- Ambiente specifico del provider: `AWS_REGION`, variabili di progetto/regione per Vertex

> **Note:**
>
> I flag e i parametri esatti possono variare in base alla versione di `@anthropic-ai/claude-code`. Eseguite `claude --help` nel vostro lavoro per vedere le opzioni supportate.

Personalizzazione del comportamento di Claude

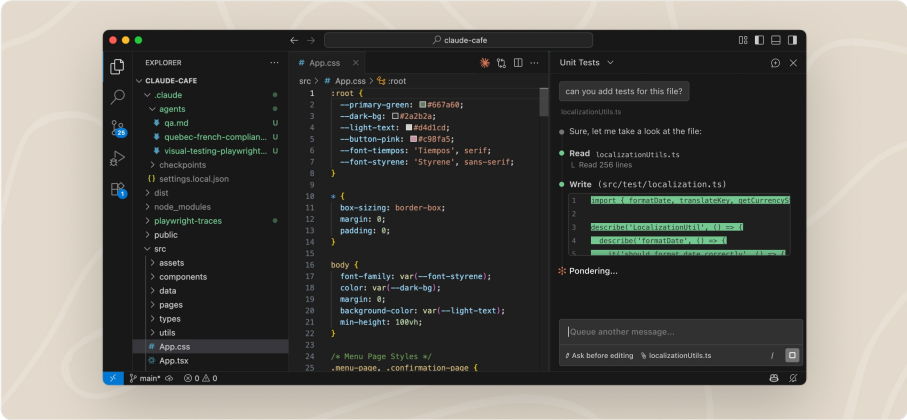
Potete guidare Claude in due modi principali:

1. **CLAUDE.md**: Definite standard di codifica, requisiti di sicurezza e convenzioni di progetto. Claude legge questo durante le esecuzioni e segue le vostre regole.
2. **Prompt personalizzati**: Passate istruzioni specifiche dell'attività tramite `prompt` / `prompt_file` nel lavoro. Utilizzate prompt diversi per lavori diversi (ad esempio, revisione, implementazione, refactoring).

Part 8: IDE & Platform Integration

Usa Claude Code in VS Code

> *Installa e configura l'estensione Claude Code per VS Code. Ottieni assistenza di codifica con IA con diff inline, @-mention, revisione del piano e scorciatoie da tastiera.*



Editor VS Code con il pannello dell'estensione Claude Code aperto sul lato destro, che mostra una conversazione con Claude

L'estensione VS Code fornisce un'interfaccia grafica nativa per Claude Code, integrata direttamente nel tuo IDE. Questo è il modo consigliato per utilizzare Claude Code in VS Code.

Con l'estensione, puoi rivedere e modificare i piani di Claude prima di accettarli, accettare automaticamente le modifiche mentre vengono apportate, @-mention file con intervalli di righe specifici dalla tua selezione, accedere alla cronologia delle conversazioni e aprire più conversazioni in schede o finestre separate.

Prerequisiti

Prima di installare, assicurati di avere:

- VS Code 1.98.0 o superiore

- Un account Anthropic (accederai quando aprirai l'estensione per la prima volta). Se stai utilizzando un provider di terze parti come Amazon Bedrock o Google Vertex AI, consulta invece Usa provider di terze parti.

> **Tip:**
>
> L'estensione include la CLI (interfaccia della riga di comando), a cui puoi accedere dal terminale integrato di VS Code per funzioni avanzate. Consulta Estensione VS Code vs. Claude Code CLI per i dettagli.

Installa l'estensione

Fai clic sul link per il tuo IDE per installare direttamente:

- Installa per VS Code
- Installa per Cursor

Oppure in VS Code, premi `Cmd+Shift+X` (Mac) o `Ctrl+Shift+X` (Windows/Linux) per aprire la visualizzazione Estensioni, cerca "Claude Code" e fai clic su **Installa**.

> **Note:**
>
> Se l'estensione non appare dopo l'installazione, riavvia VS Code o esegui "Developer: Reload Window" dalla Tavolozza dei comandi.

Inizia

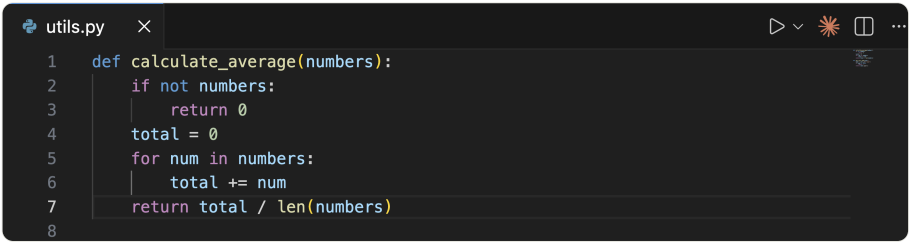
Una volta installata, puoi iniziare a utilizzare Claude Code tramite l'interfaccia VS Code:

Step 1: Apri il pannello Claude Code

In tutto VS Code, l'icona Spark indica Claude Code:

Icona Spark

Il modo più veloce per aprire Claude è fare clic sull'icona Spark nella **Barra degli strumenti dell'editor** (angolo in alto a destra dell'editor). L'icona appare solo quando hai un file aperto.



Editor VS Code che mostra l'icona Spark nella Barra degli strumenti dell'editor

Altri modi per aprire Claude Code:

- **Barra attività**: fai clic sull'icona Spark nella barra laterale sinistra per aprire l'elenco delle sessioni. Fai clic su qualsiasi sessione per aprirla come scheda dell'editor completa, o avvia una nuova. Questa icona è sempre visibile nella Barra attività.
- **Tavolozza dei comandi**: `Cmd+Shift+P` (Mac) o `Ctrl+Shift+P` (Windows/Linux), digita "Claude Code" e seleziona un'opzione come "Open in New Tab"
- **Barra di stato**: fai clic su **Claude Code** nell'angolo in basso a destra della finestra. Funziona anche quando nessun file è aperto.

Quando apri il pannello per la prima volta, appare una checklist **Learn Claude Code**. Completa ogni elemento facendo clic su **Show me**, oppure chiudila con la X. Per riaprirla in seguito, deseleziona **Hide Onboarding** nelle impostazioni di VS Code in Estensioni → Claude Code.

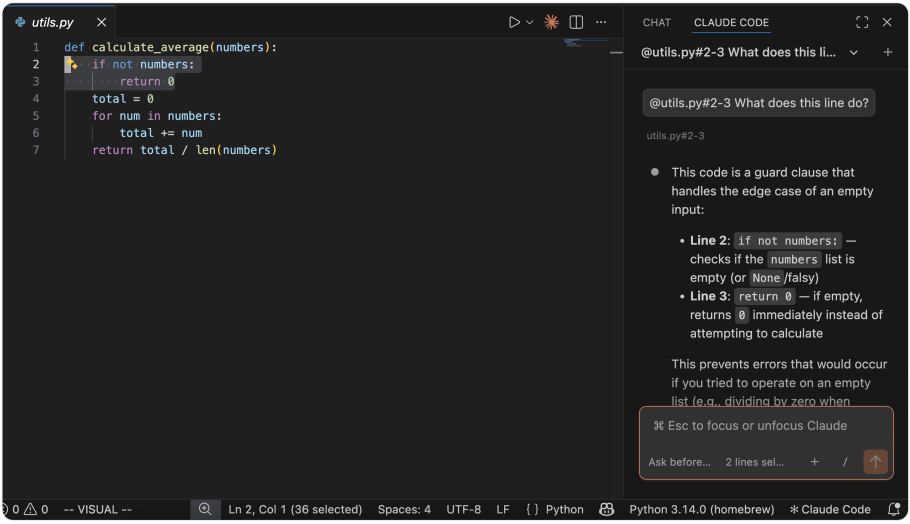
Puoi trascinare il pannello Claude per riposizionarlo ovunque in VS Code. Consulta Personalizza il tuo flusso di lavoro per i dettagli.

Step 2: Invia un prompt

Chiedi a Claude di aiutarti con il tuo codice o i tuoi file, che si tratti di spiegare come funziona qualcosa, eseguire il debug di un problema o apportare modifiche.

> **Tip:**
>
> Claude vede automaticamente il testo selezionato. Premi `Option+K` (Mac) / `Alt+K` (Windows/Linux) per inserire anche un riferimento @-mention (come `@file.ts#5-10`) nel tuo prompt.

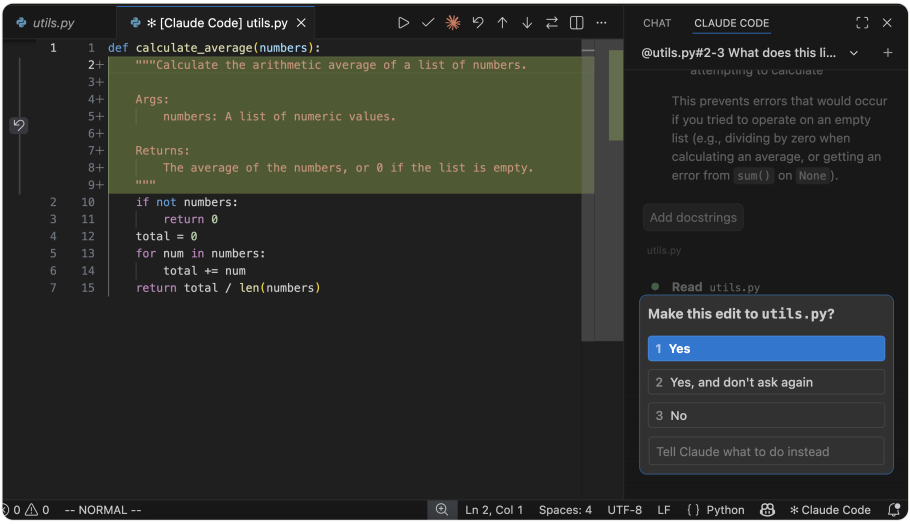
Ecco un esempio di domanda su una riga particolare in un file:



Editor VS Code con le righe 2-3 selezionate in un file Python e il pannello Claude Code che mostra una domanda su quelle righe con un riferimento @-mention

Step 3: Rivedi le modifiche

Quando Claude vuole modificare un file, mostra un confronto affiancato dell'originale e delle modifiche proposte, quindi chiede il permesso. Puoi accettare, rifiutare o dire a Claude cosa fare invece.



VS Code che mostra un diff delle modifiche proposte da Claude con un prompt di permesso che chiede se apportare la modifica

Per altre idee su cosa puoi fare con Claude Code, consulta Flussi di lavoro comuni.

> **Tip:**
>
> Esegui “Claude Code: Open Walkthrough” dalla Tavolozza dei comandi per una visita guidata delle nozioni di base.

Usa la casella dei prompt

La casella dei prompt supporta diverse funzioni:

- **Modalità di permesso**: fai clic sull’indicatore di modalità in fondo alla casella dei prompt per cambiare modalità. In modalità normale, Claude chiede il permesso prima di ogni azione. In Plan Mode, Claude descrive cosa farà e attende l’approvazione prima di apportare modifiche. VS Code apre automaticamente il piano come documento markdown completo dove puoi aggiungere commenti inline per fornire feedback prima che Claude inizi. In modalità auto-accept, Claude apporta modifiche senza chiedere. Imposta il valore predefinito nelle impostazioni di VS Code in `claudeCode.initialPermissionMode`.
- **Menu dei comandi**: fai clic su `/` o digita `/` per aprire il menu dei comandi. Le opzioni includono l’allegato di file, il cambio di modelli, l’attivazione del pensiero esteso e la visualizzazione dell’utilizzo del piano (`/usage`). La sezione Personalizza fornisce accesso ai server MCP, hooks, memoria, autorizzazioni e plugin. Gli elementi con un’icona del terminale si aprono nel terminale integrato.
- **Indicatore di contesto**: la casella dei prompt mostra quanto della finestra di contesto di Claude stai utilizzando. Claude si compatta automaticamente quando necessario, oppure puoi eseguire `/compact` manualmente.
- **Pensiero esteso**: consente a Claude di dedicare più tempo al ragionamento su problemi complessi. Attivalo tramite il menu dei comandi (`/`). Consulta Pensiero esteso per i dettagli.
- **Input multi-riga**: premi `Shift+Enter` per aggiungere una nuova riga senza inviare. Funziona anche nell’input di testo libero “Altro” dei dialoghi delle domande.

Riferisci file e cartelle

Usa @-mention per dare a Claude il contesto su file o cartelle specifiche. Quando digiti `@` seguito da un nome di file o cartella, Claude legge quel contenuto e può rispondere a domande su di esso o apportare modifiche. Claude Code supporta la corrispondenza fuzzy, quindi puoi digitare nomi parziali per trovare quello che ti serve:

```
> Explain the logic in @auth (fuzzy matches auth.js, AuthService.ts, etc.)
> What's in @src/components/ (include a trailing slash for folders)
```

Per i PDF di grandi dimensioni, puoi chiedere a Claude di leggere pagine specifiche invece dell'intero file: una singola pagina, un intervallo come pagine 1-10, o un intervallo aperto come pagina 3 in poi.

Quando selezioni il testo nell'editor, Claude può vedere il tuo codice evidenziato automaticamente. Il piè di pagina della casella dei prompt mostra quante righe sono selezionate. Premi `Option+K` (Mac) / `Alt+K` (Windows/Linux) per inserire un @-mention con il percorso del file e i numeri di riga (ad es. `@app.ts#5-10`). Fai clic sull'indicatore di selezione per attivare/disattivare se Claude può vedere il tuo testo evidenziato - l'icona occhio-barra significa che la selezione è nascosta a Claude.

Puoi anche tenere premuto `Shift` mentre trascini i file nella casella dei prompt per aggiungerli come allegati. Fai clic sulla X su qualsiasi allegato per rimuoverlo dal contesto.

Riprendi conversazioni passate

Fai clic sul menu a discesa in cima al pannello Claude Code per accedere alla cronologia delle conversazioni. Puoi cercare per parola chiave o sfogliare per tempo (Oggi, Ieri, Ultimi 7 giorni, ecc.). Fai clic su qualsiasi conversazione per riprenderla con la cronologia completa dei messaggi. Passa il mouse su una sessione per rivelare le azioni di ridenominazione e rimozione: rinomina per darle un titolo descrittivo, o rimuovi per eliminarla dall'elenco. Per ulteriori informazioni sulla ripresa delle sessioni, consulta Flussi di lavoro comuni.

Riprendi sessioni remote da Claude.ai

Se utilizzi Claude Code sul web, puoi riprendere quelle sessioni remote direttamente in VS Code. Ciò richiede l'accesso con **Claude.ai Subscription**, non Anthropic Console.

Step 1: Apri Conversazioni passate

Fai clic sul menu a discesa **Past Conversations** in cima al pannello Claude Code.

Step 2: Seleziona la scheda Remote

La finestra di dialogo mostra due schede: Local e Remote. Fai clic su **Remote** per vedere le sessioni da claude.ai.

Step 3: Seleziona una sessione da riprendere

Sfoglia o cerca le tue sessioni remote. Fai clic su qualsiasi sessione per scaricarla e continuare la conversazione localmente.

> **Note:**
>
> Solo le sessioni web avviate con un repository GitHub appaiono nella scheda Remote. La ripresa carica la cronologia della conversazione localmente; le modifiche non vengono sincronizzate di nuovo su claude.ai.

Personalizza il tuo flusso di lavoro

Una volta che sei operativo, puoi riposizionare il pannello Claude, eseguire più sessioni o passare alla modalità terminale.

Scegli dove vive Claude

Puoi trascinare il pannello Claude per riposizionarlo ovunque in VS Code. Afferra la scheda o la barra del titolo del pannello e trascinalo a:

- **Barra laterale secondaria**: il lato destro della finestra. Mantiene Claude visibile mentre codifichi.
- **Barra laterale primaria**: la barra laterale sinistra con icone per Explorer, Search, ecc.
- **Area dell'editor**: apre Claude come scheda insieme ai tuoi file. Utile per attività secondarie.

> **Tip:**
>
> Usa la barra laterale per la tua sessione Claude principale e apri schede aggiuntive per attività secondarie. Claude ricorda la tua posizione preferita. L'icona dell'elenco delle sessioni della Barra attività è separata dal pannello Claude: l'elenco delle sessioni è sempre visibile nella Barra attività, mentre l'icona del pannello Claude appare lì solo quando il pannello è ancorato alla barra laterale sinistra.

Esegui più conversazioni

Usa **Open in New Tab** o **Open in New Window** dalla Tavolozza dei comandi per avviare conversazioni aggiuntive. Ogni conversazione mantiene la propria cronologia e contesto, permettendoti di lavorare su diversi compiti in parallelo.

Quando utilizzi le schede, un piccolo punto colorato sull'icona spark indica lo stato: blu significa che una richiesta di permesso è in sospeso, arancione significa che Claude ha finito mentre la scheda era nascosta.

Passa alla modalità terminale

Per impostazione predefinita, l'estensione apre un pannello di chat grafico. Se preferisci l'interfaccia in stile CLI, apri l'impostazione Use Terminal e seleziona la casella.

Puoi anche aprire le impostazioni di VS Code (`Cmd+,` su Mac o `Ctrl+,` su Windows/Linux), vai a Estensioni → Claude Code e seleziona **Use Terminal**.

Gestisci i plugin

L'estensione VS Code include un'interfaccia grafica per installare e gestire i plugin. Digita `/plugins` nella casella dei prompt per aprire l'interfaccia **Manage plugins**.

Installa i plugin

La finestra di dialogo del plugin mostra due schede: **Plugins** e **Marketplaces**.

Nella scheda Plugins:

- I **plugin installati** appaiono in cima con interruttori per abilitarli o disabilitarli
- I **plugin disponibili** dai tuoi marketplace configurati appaiono sotto
- Cerca per filtrare i plugin per nome o descrizione
- Fai clic su **Install** su qualsiasi plugin disponibile

Quando installi un plugin, scegli l'ambito di installazione:

- **Install for you**: disponibile in tutti i tuoi progetti (ambito utente)
- **Install for this project**: condiviso con i collaboratori del progetto (ambito progetto)
- **Install locally**: solo per te, solo in questo repository (ambito locale)

Gestisci i marketplace

Passa alla scheda **Marketplaces** per aggiungere o rimuovere fonti di plugin:

- Inserisci un repository GitHub, URL o percorso locale per aggiungere un nuovo marketplace

- Fai clic sull'icona di aggiornamento per aggiornare l'elenco dei plugin di un marketplace
- Fai clic sull'icona del cestino per rimuovere un marketplace

Dopo aver apportato modifiche, un banner ti chiede di riavviare Claude Code per applicare gli aggiornamenti.

> **Note:**
>
> La gestione dei plugin in VS Code utilizza gli stessi comandi CLI sotto il cofano. I plugin e i marketplace che configuri nell'estensione sono disponibili anche nella CLI e viceversa.

Per ulteriori informazioni sul sistema dei plugin, consulta Plugin e Plugin marketplaces.

Automatizza le attività del browser con Chrome

Connetti Claude al tuo browser Chrome per testare app web, eseguire il debug con i log della console e automatizzare i flussi di lavoro del browser senza lasciare VS Code. Ciò richiede l'estensione Claude in Chrome versione 1.0.36 o superiore.

Digita `@browser` nella casella dei prompt seguito da quello che vuoi che Claude faccia:

```
@browser go to localhost:3000 and check the console for errors
```

Puoi anche aprire il menu degli allegati per selezionare strumenti specifici del browser come aprire una nuova scheda o leggere il contenuto della pagina.

Claude apre nuove schede per le attività del browser e condivide lo stato di accesso del tuo browser, quindi può accedere a qualsiasi sito a cui sei già connesso.

Per le istruzioni di configurazione, l'elenco completo delle funzionalità e la risoluzione dei problemi, consulta Usa Claude Code con Chrome.

Comandi e scorciatoie da tastiera di VS Code

Apri la Tavolozza dei comandi (`Cmd+Shift+P` su Mac o `Ctrl+Shift+P` su Windows/Linux) e digita "Claude Code" per vedere tutti i comandi VS Code disponibili per l'estensione Claude Code.

Alcuni scorciatoie dipendono da quale pannello è “focalizzato” (riceve input da tastiera). Quando il tuo cursore è in un file di codice, l’editor è focalizzato. Quando il tuo cursore è nella casella dei prompt di Claude, Claude è focalizzato. Usa `Cmd+Esc` / `Ctrl+Esc` per alternare tra loro.

> **Note:**
>
> Questi sono comandi VS Code per controllare l’estensione. Non tutti i comandi Claude Code incorporati sono disponibili nell’estensione. Consulta Estensione VS Code vs. Claude Code CLI per i dettagli.

Comando	Scorciatoia	Descrizione
Focus Input	`Cmd+Esc` (Mac) / `Ctrl+Esc` (Windows/Linux)	Alterna lo stato attivo tra editor e Claude
Open in Side Bar	-	Apri Claude nella barra laterale sinistra
Open in Terminal	-	Apri Claude in modalità terminale
Open in New Tab	`Cmd+Shift+Esc` (Mac) / `Ctrl+Shift+Esc` (Windows/Linux)	Apri una nuova conversazione come scheda dell’editor
Open in New Window	-	Apri una nuova conversazione in una finestra separata
New Conversation	`Cmd+N` (Mac) / `Ctrl+N` (Windows/Linux)	Avvia una nuova conversazione (richiede che Claude sia focalizzato)
Insert @-Mention Reference	`Option+K` (Mac) / `Alt+K` (Windows/Linux)	Inserisci un riferimento al file corrente e alla selezione (richiede che l’editor sia focalizzato)
Show Logs	-	Visualizza i log di debug dell’estensione
Logout	-	Esci dal tuo account Anthropic

Configura le impostazioni

L'estensione ha due tipi di impostazioni:

- **Impostazioni dell'estensione** in VS Code: controllano il comportamento dell'estensione all'interno di VS Code. Apri con `Cmd+,` (Mac) o `Ctrl+,` (Windows/Linux), quindi vai a Estensioni → Claude Code. Puoi anche digitare `/` e selezionare **General Config** per aprire le impostazioni.
- **Impostazioni Claude Code** in `~/.claude/settings.json` : condivise tra l'estensione e la CLI. Usa per comandi consentiti, variabili di ambiente, hooks e server MCP. Consulta Impostazioni per i dettagli.

> **Tip:**
>
> Aggiungi `"$schema": "https://json.schemastore.org/claude-code-settings.json"` al tuo `settings.json` per ottenere l'autocompletamento e la convalida inline per tutte le impostazioni disponibili direttamente in VS Code.

Impostazioni dell'estensione

Impostazione	Predefinito	Descrizione
`selectedModel`	`default`	Modello per le nuove conversazioni. Cambia per sessione con `/model` .
`useTerminal`	`false`	Avvia Claude in modalità terminale invece di pannello grafico
`initialPermissionMode`	`default`	Controlla i prompt di approvazione: `default` (chiedi ogni volta), `plan` , `acceptEdits` , o `bypassPermissions`
`preferredLocation`	`panel`	Dove Claude si apre: `sidebar` (destra) o `panel` (nuova scheda)
`autosave`	`true`	Salva automaticamente i file prima che Claude li legga o scriva

Impostazione	Predefinito	Descrizione
`useCtrlEnterToSend`	`false`	Usa Ctrl/Cmd+Enter invece di Enter per inviare i prompt
`enableNewConversationShortcut`	`true`	Abilita Cmd/Ctrl+N per avviare una nuova conversazione
`hideOnboarding`	`false`	Nascondi la checklist di onboarding (icona del berretto di laurea)
`respectGitIgnore`	`true`	Escludi i modelli .gitignore dalle ricerche di file
`environmentVariables`	`[]`	Imposta le variabili di ambiente per il processo Claude. Usa invece le impostazioni Claude Code per la configurazione condivisa.
`disableLoginPrompt`	`false`	Salta i prompt di autenticazione (per configurazioni di provider di terze parti)
`allowDangerouslySkipPermissions`	`false`	Ignora tutti i prompt di permesso. **Usa con estrema cautela.**
`claudeProcessWrapper`	-	Percorso eseguibile utilizzato per avviare il processo Claude

Estensione VS Code vs. Claude Code CLI

Claude Code è disponibile sia come estensione VS Code (pannello grafico) che come CLI (interfaccia della riga di comando nel terminale). Alcune funzioni sono disponibili solo nella CLI. Se hai bisogno di una funzione solo CLI, esegui `claude` nel terminale integrato di VS Code.

Funzione	CLI	Estensione VS Code
Comandi e skills	Tutti	Sottoinsieme (digita `/` per vedere quelli disponibili)
Configurazione del server MCP	Sì	Parziale (aggiungi server tramite CLI; gestisci i server esistenti con `/mcp` nel pannello di chat)
Checkpoints	Sì	Sì
Scorciatoia bash `!`	Sì	No
Completamento scheda	Sì	No

Riavvolgi con i checkpoint

L'estensione VS Code supporta i checkpoint, che tracciano le modifiche ai file di Claude e ti permettono di riavvolgere a uno stato precedente. Passa il mouse su qualsiasi messaggio per rivelare il pulsante di riavvolgimento, quindi scegli tra tre opzioni:

- **Fork conversation from here**: avvia un nuovo ramo di conversazione da questo messaggio mantenendo intatte tutte le modifiche al codice
- **Rewind code to here**: ripristina le modifiche ai file a questo punto nella conversazione mantenendo la cronologia completa della conversazione
- **Fork conversation and rewind code**: avvia un nuovo ramo di conversazione e ripristina le modifiche ai file a questo punto

Per i dettagli completi su come funzionano i checkpoint e le loro limitazioni, consulta Checkpointing.

Esegui CLI in VS Code

Per utilizzare la CLI mentre rimani in VS Code, apri il terminale integrato (`Ctrl+`` ` su Windows/Linux o `Cmd+`` ` su Mac) ed esegui `claude` . La CLI si integra automaticamente con il tuo IDE per funzioni come la visualizzazione dei diff e la condivisione dei diagnostici.

Se utilizzi un terminale esterno, esegui `/ide` all'interno di Claude Code per connetterlo a VS Code.

Passa tra l'estensione e la CLI

L'estensione e la CLI condividono la stessa cronologia delle conversazioni. Per continuare una conversazione dell'estensione nella CLI, esegui `claude --resume` nel terminale. Questo apre un selettore interattivo dove puoi cercare e selezionare la tua conversazione.

Includi l'output del terminale nei prompt

Fai riferimento all'output del terminale nei tuoi prompt usando `@terminal:name` dove `name` è il titolo del terminale. Questo consente a Claude di vedere l'output dei comandi, i messaggi di errore o i log senza copia-incolla.

Monitora i processi in background

Quando Claude esegue comandi di lunga durata, l'estensione mostra l'avanzamento nella barra di stato. Tuttavia, la visibilità per le attività in background è limitata rispetto alla CLI. Per una migliore visibilità, chiedi a Claude di emettere il comando in modo da poterlo eseguire nel terminale integrato di VS Code.

Connettiti a strumenti esterni con MCP

I server MCP (Model Context Protocol) danno a Claude accesso a strumenti esterni, database e API.

Per aggiungere un server MCP, apri il terminale integrato (`Ctrl+`` ` o `Cmd+`` `) ed esegui:

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

Una volta configurato, chiedi a Claude di utilizzare gli strumenti (ad es. "Review PR #456").

Per gestire i server MCP senza lasciare VS Code, digita `/mcp` nel pannello di chat. La finestra di dialogo di gestione MCP ti consente di abilitare o disabilitare i server, riconnetterti a un server e gestire l'autenticazione OAuth. Consulta la documentazione MCP per i server disponibili.

Lavora con git

Claude Code si integra con git per aiutare con i flussi di lavoro di controllo della versione direttamente in VS Code. Chiedi a Claude di eseguire il commit delle modifiche, creare pull request o lavorare tra i rami.

Crea commit e pull request

Claude può mettere in stage le modifiche, scrivere messaggi di commit e creare pull request in base al tuo lavoro:

```
> commit my changes with a descriptive message
> create a pr for this feature
> summarize the changes I've made to the auth module
```

Quando crei pull request, Claude genera descrizioni in base alle modifiche effettive del codice e può aggiungere contesto su test o decisioni di implementazione.

Usa git worktrees per attività parallele

Usa il flag `--worktree` (`-w`) per avviare Claude in un worktree isolato con i suoi file e ramo:

```
claude --worktree feature-auth
```

Ogni worktree mantiene uno stato di file indipendente mentre condivide la cronologia git. Ciò impedisce alle istanze di Claude di interferire l'una con l'altra quando lavorano su compiti diversi. Per ulteriori dettagli, consulta Esegui sessioni parallele di Claude Code con Git worktrees.

Usa provider di terze parti

Per impostazione predefinita, Claude Code si connette direttamente all'API di Anthropic. Se la tua organizzazione utilizza Amazon Bedrock, Google Vertex AI o Microsoft Foundry per accedere a Claude, configura l'estensione per utilizzare il tuo provider:

Step 1: Disabilita il prompt di accesso

Apri l'impostazione Disable Login Prompt e seleziona la casella.

Puoi anche aprire le impostazioni di VS Code (`Cmd+,` su Mac o `Ctrl+,` su Windows/Linux), cercare "Claude Code login" e selezionare **Disable Login Prompt**.

Step 2: Configura il tuo provider

Segui la guida di configurazione per il tuo provider:

- Claude Code su Amazon Bedrock
- Claude Code su Google Vertex AI

- Claude Code su Microsoft Foundry

Queste guide coprono la configurazione del tuo provider in `~/.claude/settings.json`, che garantisce che le tue impostazioni siano condivise tra l'estensione VS Code e la CLI.

Sicurezza e privacy

Il tuo codice rimane privato. Claude Code elabora il tuo codice per fornire assistenza ma non lo utilizza per addestrare i modelli. Per i dettagli sulla gestione dei dati e su come rinunciare alla registrazione, consulta Dati e privacy.

Con le autorizzazioni di modifica automatica abilitate, Claude Code può modificare i file di configurazione di VS Code (come `settings.json` o `tasks.json`) che VS Code potrebbe eseguire automaticamente. Per ridurre il rischio quando si lavora con codice non attendibile:

- Abilita la Modalità limitata di VS Code per gli spazi di lavoro non attendibili
- Usa la modalità di approvazione manuale invece di auto-accept per le modifiche
- Rivedi attentamente le modifiche prima di accettarle

Risolvi i problemi comuni

L'estensione non si installa

- Assicurati di avere una versione compatibile di VS Code (1.98.0 o successiva)
- Verifica che VS Code abbia il permesso di installare estensioni
- Prova a installare direttamente dal VS Code Marketplace

L'icona Spark non è visibile

L'icona Spark appare nella **Barra degli strumenti dell'editor** (in alto a destra dell'editor) quando hai un file aperto. Se non la vedi:

1. **Apri un file**: L'icona richiede che un file sia aperto. Avere solo una cartella aperta non è sufficiente.
2. **Controlla la versione di VS Code**: Richiede 1.98.0 o superiore (Aiuto → Informazioni)
3. **Riavvia VS Code**: Esegui "Developer: Reload Window" dalla Tavolozza dei comandi
4. **Disabilita le estensioni in conflitto**: Disabilita temporaneamente altre estensioni AI (Cline, Continue, ecc.)
5. **Controlla l'attendibilità dell'area di lavoro**: L'estensione non funziona in Modalità limitata

In alternativa, fai clic su “ Claude Code” nella **Barra di stato** (angolo in basso a destra). Funziona anche senza un file aperto. Puoi anche usare la **Tavolozza dei comandi** (`Cmd+Shift+P` / `Ctrl+Shift+P`) e digitare “Claude Code”.

Claude Code non risponde mai

Se Claude Code non risponde ai tuoi prompt:

1. **Controlla la tua connessione Internet**: Assicurati di avere una connessione Internet stabile
2. **Avvia una nuova conversazione**: Prova ad avviare una conversazione nuova per vedere se il problema persiste
3. **Prova la CLI**: Esegui `claude` dal terminale per vedere se ottieni messaggi di errore più dettagliati

Se i problemi persistono, apri un problema su GitHub con i dettagli dell’errore.

Disinstalla l’estensione

Per disinstallare l’estensione Claude Code:

1. Apri la visualizzazione Estensioni (`Cmd+Shift+X` su Mac o `Ctrl+Shift+X` su Windows/Linux)
2. Cerca “Claude Code”
3. Fai clic su **Disinstalla**

Per rimuovere anche i dati dell’estensione e ripristinare tutte le impostazioni:

```
rm -rf ~/.vscode/globalStorage/anthropic.claude-code
```

Per ulteriore aiuto, consulta la guida alla risoluzione dei problemi.

Passaggi successivi

Ora che hai Claude Code configurato in VS Code:

- Esplora i flussi di lavoro comuni per ottenere il massimo da Claude Code
- Configura i server MCP per estendere le capacità di Claude con strumenti esterni. Aggiungi i server usando la CLI, quindi gestiscili con `/mcp` nel pannello di chat.
- Configura le impostazioni di Claude Code per personalizzare i comandi consentiti, gli hooks e altro. Queste impostazioni sono condivise tra l’estensione e la CLI.

JetBrains IDEs

> *Usa Claude Code con JetBrains IDEs inclusi IntelliJ, PyCharm, WebStorm e altri*

Claude Code si integra con JetBrains IDEs attraverso un plugin dedicato, fornendo funzionalità come la visualizzazione interattiva dei diff, la condivisione del contesto della selezione e altro ancora.

IDE supportati

Il plugin Claude Code funziona con la maggior parte dei JetBrains IDEs, inclusi:

- IntelliJ IDEA
- PyCharm
- Android Studio
- WebStorm
- PhpStorm
- GoLand

Funzionalità

- **Avvio rapido**: Usa `Cmd+Esc` (Mac) o `Ctrl+Esc` (Windows/Linux) per aprire Claude Code direttamente dal tuo editor, oppure fai clic sul pulsante Claude Code nell'interfaccia utente
- **Visualizzazione dei diff**: Le modifiche al codice possono essere visualizzate direttamente nel visualizzatore diff dell'IDE invece del terminale
- **Contesto della selezione**: La selezione/scheda corrente nell'IDE viene automaticamente condivisa con Claude Code
- **Scorciatoie di riferimento file**: Usa `Cmd+Option+K` (Mac) o `Alt+Ctrl+K` (Linux/Windows) per inserire riferimenti ai file (ad esempio, @File#L1-99)
- **Condivisione diagnostica**: Gli errori diagnostici (lint, sintassi, ecc.) dall'IDE vengono automaticamente condivisi con Claude mentre lavori

Installazione

Installazione da Marketplace

Trova e installa il plugin Claude Code dal marketplace di JetBrains e riavvia il tuo IDE.

Se non hai ancora installato Claude Code, consulta la nostra guida di avvio rapido per le istruzioni di installazione.

> **Note:**
>
> Dopo aver installato il plugin, potrebbe essere necessario riavviare completamente il tuo IDE affinché abbia effetto.

Utilizzo

Dal tuo IDE

Esegui `claude` dal terminale integrato del tuo IDE e tutte le funzionalità di integrazione saranno attive.

Da terminali esterni

Usa il comando `/ide` in qualsiasi terminale esterno per connettere Claude Code al tuo JetBrains IDE e attivare tutte le funzionalità:

```
claude
```

```
/ide
```

Se desideri che Claude abbia accesso agli stessi file del tuo IDE, avvia Claude Code dalla stessa directory della radice del progetto del tuo IDE.

Configurazione

Impostazioni di Claude Code

Configura l'integrazione dell'IDE attraverso le impostazioni di Claude Code:

1. Esegui `claude`
2. Inserisci il comando `/config`
3. Imposta lo strumento diff su `auto` per il rilevamento automatico dell'IDE

Impostazioni del plugin

Configura il plugin Claude Code andando a **Impostazioni → Strumenti → Claude Code [Beta]**:

Impostazioni generali

- **Comando Claude**: Specifica un comando personalizzato per eseguire Claude (ad esempio, `claude`, `/usr/local/bin/claude`, o `npx @anthropic/claude`)
- **Sopprimere la notifica per il comando Claude non trovato**: Salta le notifiche relative al mancato reperimento del comando Claude
- **Abilita l'uso di Option+Invio per prompt multi-riga** (solo macOS): Quando abilitato, Option+Invio inserisce nuove righe nei prompt di Claude Code. Disabilita se riscontri problemi con il tasto Option catturato inaspettatamente (richiede il riavvio del terminale)
- **Abilita aggiornamenti automatici**: Controlla automaticamente e installa gli aggiornamenti del plugin (applicati al riavvio)

> **Tip:**
>
> Per gli utenti WSL: Imposta `wsl -d Ubuntu -- bash -lic "claude"` come comando Claude (sostituisci `Ubuntu` con il nome della tua distribuzione WSL)

Configurazione del tasto ESC

Se il tasto ESC non interrompe le operazioni di Claude Code nei terminali JetBrains:

1. Vai a **Impostazioni → Strumenti → Terminale**
2. Oppure:
   - Deseleziona "Sposta il focus sull'editor con Escape", oppure
   - Fai clic su "Configura scorciatoie da tastiera del terminale" e elimina la scorciatoia "Sposta il focus sull'editor"
3. Applica le modifiche

Questo consente al tasto ESC di interrompere correttamente le operazioni di Claude Code.

Configurazioni speciali

Sviluppo remoto

> **Warning:**

> Quando usi JetBrains Remote Development, devi installare il plugin nell'host remoto tramite **Impostazioni → Plugin (Host)**.

Il plugin deve essere installato sull'host remoto, non sulla tua macchina client locale.

Configurazione WSL

> **Warning:**
>
> Gli utenti WSL potrebbero aver bisogno di configurazione aggiuntiva affinché il rilevamento dell'IDE funzioni correttamente. Consulta la nostra guida alla risoluzione dei problemi WSL per istruzioni di configurazione dettagliate.

La configurazione WSL potrebbe richiedere:

- Configurazione corretta del terminale
- Regolazioni della modalità di rete
- Aggiornamenti delle impostazioni del firewall

Risoluzione dei problemi

Plugin non funzionante

- Assicurati di eseguire Claude Code dalla directory radice del progetto
- Verifica che il plugin JetBrains sia abilitato nelle impostazioni dell'IDE
- Riavvia completamente l'IDE (potrebbe essere necessario farlo più volte)
- Per Remote Development, assicurati che il plugin sia installato nell'host remoto

IDE non rilevato

- Verifica che il plugin sia installato e abilitato
- Riavvia completamente l'IDE
- Verifica che tu stia eseguendo Claude Code dal terminale integrato
- Per gli utenti WSL, consulta la guida alla risoluzione dei problemi WSL

Comando non trovato

Se facendo clic sull'icona Claude viene visualizzato "comando non trovato":

1. Verifica che Claude Code sia installato: `npm list -g @anthropic-ai/claude-code`
2. Configura il percorso del comando Claude nelle impostazioni del plugin

3. Per gli utenti WSL, usa il formato del comando WSL menzionato nella sezione di configurazione

Considerazioni sulla sicurezza

Quando Claude Code viene eseguito in un JetBrains IDE con le autorizzazioni di modifica automatica abilitate, potrebbe essere in grado di modificare i file di configurazione dell'IDE che possono essere eseguiti automaticamente dal tuo IDE. Questo potrebbe aumentare il rischio di eseguire Claude Code in modalità di modifica automatica e consentire di aggirare i prompt di autorizzazione di Claude Code per l'esecuzione bash.

Quando si esegue in JetBrains IDEs, considera:

- Utilizzare la modalità di approvazione manuale per le modifiche
- Prestare particolare attenzione per assicurarsi che Claude sia utilizzato solo con prompt affidabili
- Essere consapevole di quali file Claude Code ha accesso per modificare

Per ulteriore aiuto, consulta la nostra guida alla risoluzione dei problemi.

Usa Claude Code Desktop

> *Sfrutta al massimo Claude Code Desktop: sessioni parallele con isolamento Git, revisione visiva dei diff, anteprime delle app, monitoraggio dei PR, modalità di autorizzazione, connettori e configurazione aziendale.*

La scheda Code all'interno dell'app Claude Desktop ti consente di utilizzare Claude Code attraverso un'interfaccia grafica invece del terminale.

Desktop aggiunge queste funzionalità all'esperienza standard di Claude Code:

- Revisione visiva dei diff con commenti inline
- Anteprima live dell'app con server di sviluppo
- Monitoraggio dei PR su GitHub con correzione automatica e merge automatico
- Sessioni parallele con isolamento automatico dei worktree Git
- Attività pianificate che eseguono Claude secondo una pianificazione ricorrente
- Connettori per GitHub, Slack, Linear e altri
- Ambienti locali, SSH e cloud

> **Tip:**
>
> Nuovo a Desktop? Inizia con Guida introduttiva per installare l'app e fare il tuo primo edit.

Questa pagina copre lavorare con il codice, gestire le sessioni, estendere Claude Code, attività pianificate e configurazione. Include anche un confronto CLI e risoluzione dei problemi.

Avvia una sessione

Prima di inviare il tuo primo messaggio, configura quattro cose nell'area del prompt:

- **Ambiente**: scegli dove Claude viene eseguito. Seleziona **Local** per la tua macchina, **Remote** per sessioni cloud ospitate da Anthropic, o una **connessione SSH** per una macchina remota che gestisci. Vedi configurazione dell'ambiente.
- **Cartella del progetto**: seleziona la cartella o il repository su cui Claude lavora. Per le sessioni remote, puoi aggiungere più repository.

- **Modello**: scegli un modello dal menu a discesa accanto al pulsante di invio. Il modello viene bloccato una volta avviata la sessione.
- **Modalità di autorizzazione**: scegli quanta autonomia ha Claude dal selettore di modalità. Puoi cambiare questo durante la sessione.

Digita il tuo compito e premi **Invio** per iniziare. Ogni sessione traccia il suo proprio contesto e le modifiche in modo indipendente.

Lavora con il codice

Dai a Claude il contesto giusto, controlla quanto fa da solo e rivedi cosa ha cambiato.

Usa la casella del prompt

Digita quello che vuoi che Claude faccia e premi **Invio** per inviare. Claude legge i file del tuo progetto, apporta modifiche ed esegue comandi in base alla tua modalità di autorizzazione. Puoi interrompere Claude in qualsiasi momento: fai clic sul pulsante di arresto o digita la tua correzione e premi **Invio**. Claude smette di fare quello che sta facendo e si adatta in base al tuo input.

Il pulsante + accanto alla casella del prompt ti dà accesso agli allegati di file, skills, connettori e plugin.

Aggiungi file e contesto ai prompt

La casella del prompt supporta due modi per portare contesto esterno:

- **@mention file**: digita `@` seguito da un nome di file per aggiungere un file al contesto della conversazione. Claude può quindi leggere e fare riferimento a quel file.
- **Allega file**: allega immagini, PDF e altri file al tuo prompt usando il pulsante di allegato, o trascina e rilascia i file direttamente nel prompt. Questo è utile per condividere screenshot di bug, mockup di design o documenti di riferimento.

Scegli una modalità di autorizzazione

Le modalità di autorizzazione controllano quanta autonomia ha Claude durante una sessione: se chiede prima di modificare file, eseguire comandi o entrambi. Puoi cambiare modalità in qualsiasi momento usando il selettore di modalità accanto al pulsante di invio. Inizia con Chiedi autorizzazioni per vedere esattamente cosa fa Claude, quindi passa a Accetta automaticamente gli edit o Plan mode man mano che acquisisci familiarità.

Modalità	Chiave di impostazione	Comportamento
Chiedi autorizzazioni	`default`	Claude chiede prima di modificare file o eseguire comandi. Vedi un diff e puoi accettare o rifiutare ogni modifica. Consigliato per i nuovi utenti.
Accetta automaticamente gli edit	`acceptEdits`	Claude accetta automaticamente gli edit dei file ma chiede comunque prima di eseguire comandi di terminale. Usa questo quando ti fidi dei cambiamenti ai file e vuoi un'iterazione più veloce.
Plan mode	`plan`	Claude analizza il tuo codice e crea un piano senza modificare file o eseguire comandi. Buono per compiti complessi dove vuoi rivedere l'approccio prima.
Bypass permissions	`bypassPermissions`	Claude viene eseguito senza alcun prompt di autorizzazione, equivalente a `--dangerously-skip-permissions` nella CLI. Abilita in Impostazioni → Claude Code sotto "Allow bypass permissions mode". Usa solo in container sandbox o VM. Gli amministratori aziendali possono disabilitare questa opzione.

La modalità di autorizzazione `dontAsk` è disponibile solo nella CLI.

> **Tip:**

> Inizia compiti complessi in Plan mode in modo che Claude mappi un approccio prima di apportare modifiche. Una volta approvato il piano, passa a Accetta automaticamente gli edit o Chiedi autorizzazioni per eseguirlo. Vedi esplora prima, poi pianifica, poi codifica per ulteriori informazioni su questo flusso di lavoro.

Le sessioni remote supportano Accetta automaticamente gli edit e Plan mode. Chiedi autorizzazioni non è disponibile perché le sessioni remote accettano automaticamente gli edit dei file per impostazione predefinita, e Bypass permissions non è disponibile perché l'ambiente remoto è già sandbox.

Gli amministratori aziendali possono limitare quali modalità di autorizzazione sono disponibili. Vedi configurazione aziendale per i dettagli.

Anteprima della tua app

Claude può avviare un server di sviluppo e aprire un browser incorporato per verificare le sue modifiche. Questo funziona per app web frontend così come per server backend: Claude può testare endpoint API, visualizzare log del server e iterare su problemi che trova. Nella maggior parte dei casi, Claude avvia il server automaticamente dopo aver modificato i file del progetto. Puoi anche chiedere a Claude di visualizzare un'anteprima in qualsiasi momento. Per impostazione predefinita, Claude verifica automaticamente le modifiche dopo ogni edit.

Dal pannello di anteprima, puoi:

- Interagire con la tua app in esecuzione direttamente nel browser incorporato
- Guardare Claude verificare automaticamente le sue stesse modifiche: scatta screenshot, ispeziona il DOM, fa clic su elementi, compila moduli e corregge i problemi che trova
- Avviare o arrestare server dal menu a discesa **Preview** nella barra degli strumenti della sessione
- Persistere cookie e archiviazione locale tra i riavvii del server selezionando **Persist sessions** nel menu a discesa, in modo da non dover effettuare di nuovo l'accesso durante lo sviluppo
- Modificare la configurazione del server o arrestare tutti i server contemporaneamente

Claude crea la configurazione iniziale del server in base al tuo progetto. Se la tua app utilizza un comando dev personalizzato, modifica `.claude/launch.json` per adattarlo alla tua configurazione. Vedi Configura server di anteprima per il riferimento completo.

Per cancellare i dati della sessione salvati, attiva/disattiva **Persist preview sessions** in Impostazioni → Claude Code. Per disabilitare completamente l'anteprima, attiva/disattiva **Preview** in Impostazioni → Claude Code.

Rivedi le modifiche con la visualizzazione diff

Dopo che Claude apporta modifiche al tuo codice, la visualizzazione diff ti consente di rivedere le modifiche file per file prima di creare una pull request.

Quando Claude modifica i file, appare un indicatore di statistiche diff che mostra il numero di righe aggiunte e rimosse, come `+12 -1`. Fai clic su questo indicatore per aprire il visualizzatore diff, che visualizza un elenco di file a sinistra e le modifiche per ogni file a destra.

Per commentare righe specifiche, fai clic su qualsiasi riga nel diff per aprire una casella di commento. Digita il tuo feedback e premi **Invio** per aggiungere il commento. Dopo aver aggiunto commenti a più righe, invia tutti i commenti contemporaneamente:

- **macOS**: premi **Cmd+Invio**
- **Windows**: premi **Ctrl+Invio**

Claude legge i tuoi commenti e apporta le modifiche richieste, che appaiono come un nuovo diff che puoi rivedere.

Rivedi il tuo codice

Nella visualizzazione diff, fai clic su **Review code** nella barra degli strumenti in alto a destra per chiedere a Claude di valutare le modifiche prima di eseguire il commit. Claude esamina i diff attuali e lascia commenti direttamente nella visualizzazione diff. Puoi rispondere a qualsiasi commento o chiedere a Claude di rivedere.

La revisione si concentra su problemi ad alto segnale: errori di compilazione, errori logici definitivi, vulnerabilità di sicurezza e bug ovvi. Non contrassegna stile, formattazione, problemi preesistenti o qualsiasi cosa che un linter catturebbe.

Monitora lo stato della pull request

Dopo aver aperto una pull request, una barra di stato CI appare nella sessione. Claude Code utilizza GitHub CLI per eseguire il polling dei risultati dei controlli e far emergere i guasti.

- **Auto-fix**: quando abilitato, Claude tenta automaticamente di correggere i controlli CI falliti leggendo l'output del guasto e iterando.

- **Auto-merge**: quando abilitato, Claude unisce il PR una volta che tutti i controlli passano. Il metodo di merge è squash. Auto-merge deve essere abilitato nelle impostazioni del tuo repository GitHub affinché questo funzioni.

Usa gli interruttori **Auto-fix** e **Auto-merge** nella barra di stato CI per abilitare una delle due opzioni. Claude Code invia anche una notifica desktop quando CI termina.

> **Note:**
>
> Il monitoraggio dei PR richiede che GitHub CLI (`gh`) sia installato e autenticato sulla tua macchina. Se `gh` non è installato, Desktop ti chiede di installarlo la prima volta che tenti di creare un PR.

Gestisci le sessioni

Ogni sessione è una conversazione indipendente con il suo proprio contesto e modifiche. Puoi eseguire più sessioni in parallelo o inviare il lavoro al cloud.

Lavora in parallelo con le sessioni

Fai clic su **+ New session** nella barra laterale per lavorare su più compiti in parallelo. Per i repository Git, ogni sessione ottiene la sua copia isolata del tuo progetto usando Git worktrees, in modo che le modifiche in una sessione non influiscano su altre sessioni fino a quando non le esegui il commit.

I worktree sono archiviati in `<project-root>/.claude/worktrees/` per impostazione predefinita. Puoi cambiare questo in una directory personalizzata in Impostazioni → Claude Code sotto “Worktree location”. Puoi anche impostare un prefisso di ramo che viene anteposto a ogni nome di ramo worktree, il che è utile per mantenere organizzati i rami creati da Claude. Per rimuovere un worktree quando hai finito, passa il mouse sulla sessione nella barra laterale e fai clic sull’icona di archivio.

> **Note:**
>
> L’isolamento della sessione richiede Git. La maggior parte dei Mac include Git per impostazione predefinita. Esegui `git --version` in Terminal per verificare. Su Windows, Git è richiesto affinché la scheda Code funzioni: scarica Git per Windows, installalo e riavvia l’app. Se riscontri errori Git, prova una sessione Cowork per aiutare a risolvere i problemi della tua configurazione.

Usa l'icona del filtro in cima alla barra laterale per filtrare le sessioni per stato (Active, Archived) e ambiente (Local, Cloud). Per rinominare una sessione o controllare l'utilizzo del contesto, fai clic sul titolo della sessione nella barra degli strumenti in cima alla sessione attiva. Quando il contesto si riempie, Claude riassume automaticamente la conversazione e continua a lavorare. Puoi anche digitare `/compact` per attivare la compressione prima e liberare spazio di contesto. Vedi la finestra di contesto per i dettagli su come funziona la compressione.

Esegui attività a lunga esecuzione in remoto

Per grandi refactor, suite di test, migrazioni o altre attività a lunga esecuzione, seleziona **Remote** invece di **Local** quando avvii una sessione. Le sessioni remote vengono eseguite sull'infrastruttura cloud di Anthropic e continuano anche se chiudi l'app o spegni il computer. Torna indietro in qualsiasi momento per vedere i progressi o indirizzare Claude in una direzione diversa. Puoi anche monitorare le sessioni remote da claude.ai/code o dall'app Claude iOS.

Le sessioni remote supportano anche più repository. Dopo aver selezionato un ambiente cloud, fai clic sul pulsante + accanto alla pillola del repo per aggiungere repository aggiuntivi alla sessione. Ogni repo ottiene il suo selettore di ramo. Questo è utile per compiti che si estendono su più codebase, come l'aggiornamento di una libreria condivisa e dei suoi consumatori.

Vedi Claude Code sul web per ulteriori informazioni su come funzionano le sessioni remote.

Continua su un'altra superficie

Il menu **Continue in**, accessibile dall'icona VS Code in basso a destra della barra degli strumenti della sessione, ti consente di spostare la tua sessione su un'altra superficie:

- **Claude Code sul web**: invia la tua sessione locale per continuare l'esecuzione in remoto. Desktop esegue il push del tuo ramo, genera un riepilogo della conversazione e crea una nuova sessione remota con il contesto completo. Puoi quindi scegliere di archiviare la sessione locale o mantenerla. Questo richiede un albero di lavoro pulito e non è disponibile per le sessioni SSH.
- **Il tuo IDE**: apre il tuo progetto in un IDE supportato nella directory di lavoro corrente.

Estendi Claude Code

Connetti servizi esterni, aggiungi flussi di lavoro riutilizzabili, personalizza il comportamento di Claude e configura i server di anteprima.

Connetti strumenti esterni

Per le sessioni locali e SSH, fai clic sul pulsante + accanto alla casella del prompt e seleziona **Connectors** per aggiungere integrazioni come Google Calendar, Slack, GitHub, Linear, Notion e altri. Puoi aggiungere connettori prima o durante una sessione. I connettori non sono disponibili per le sessioni remote.

Per gestire o disconnettere i connettori, vai a Impostazioni → Connectors nell'app desktop, o seleziona **Manage connectors** dal menu Connectors nella casella del prompt.

Una volta connesso, Claude può leggere il tuo calendario, inviare messaggi, creare problemi e interagire direttamente con i tuoi strumenti. Puoi chiedere a Claude quali connettori sono configurati nella tua sessione.

I connettori sono MCP servers con un flusso di configurazione grafico. Usali per l'integrazione rapida con i servizi supportati. Per le integrazioni non elencate in Connectors, aggiungi MCP servers manualmente tramite file di impostazioni. Puoi anche creare connettori personalizzati.

Usa skills

Skills estendono quello che Claude può fare. Claude le carica automaticamente quando rilevante, o puoi invocarne una direttamente: digita `/` nella casella del prompt o fai clic sul pulsante + e seleziona **Slash commands** per sfogliare cosa è disponibile. Questo include comandi incorporati, le tue skill personalizzate, skill del progetto dalla tua codebase e skill da qualsiasi plugin installato. Selezionane una e appare evidenziata nel campo di input. Digita il tuo compito dopo di essa e invia come al solito.

Installa plugin

Plugins sono pacchetti riutilizzabili che aggiungono skills, agents, hooks, MCP servers e configurazioni LSP a Claude Code. Puoi installare plugin dall'app desktop senza usare il terminale.

Per le sessioni locali e SSH, fai clic sul pulsante + accanto alla casella del prompt e seleziona **Plugins** per vedere i tuoi plugin installati e i loro comandi. Per aggiungere un plugin, seleziona **Add plugin** dal sottomenu per aprire il browser dei plugin, che mostra i plugin disponibili dai tuoi marketplace configurati incluso il marketplace ufficiale di Anthropic. Seleziona **Manage plugins** per abilitare, disabilitare o disinstallare plugin.

I plugin possono essere limitati al tuo account utente, a un progetto specifico o solo locali. I plugin non sono disponibili per le sessioni remote. Per il riferimento completo dei plugin inclusa la creazione dei tuoi plugin, vedi plugin.

Configura server di anteprima

Claude rileva automaticamente la tua configurazione del server di sviluppo e archivia la configurazione in `.claude/launch.json` alla radice della cartella che hai selezionato quando hai avviato la sessione. Preview utilizza questa cartella come directory di lavoro, quindi se hai selezionato una cartella padre, le sottocartelle con i loro stessi server di sviluppo non verranno rilevate automaticamente. Per lavorare con il server di una sotto-cartella, avvia una sessione in quella cartella direttamente o aggiungi una configurazione manualmente.

Per personalizzare come il tuo server si avvia, ad esempio per usare `yarn dev` invece di `npm run dev` o per cambiare la porta, modifica il file manualmente o fai clic su **Edit configuration** nel menu a discesa Preview per aprirlo nel tuo editor di codice. Il file supporta JSON con commenti.

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000
    }
  ]
}
```

Puoi definire più configurazioni per eseguire server diversi dallo stesso progetto, come un frontend e un'API. Vedi gli esempi di seguito.

Auto-verify changes

Quando `autoVerify` è abilitato, Claude verifica automaticamente le modifiche al codice dopo aver modificato i file. Scatta screenshot, controlla gli errori e conferma che le modifiche funzionano prima di completare la sua risposta.

Auto-verify è abilitato per impostazione predefinita. Disabilitalo per progetto aggiungendo `"autoVerify": false` a `.claude/launch.json`, o attiva/disattivalo dal menu a discesa **Preview**.

```
{
  "version": "0.0.1",
  "autoVerify": false,
  "configurations": [...]
}
```

Quando disabilitato, gli strumenti di anteprima sono ancora disponibili e puoi chiedere a Claude di verificare in qualsiasi momento. Auto-verify lo rende automatico dopo ogni edit.

Configuration fields

Ogni voce nell'array `configurations` accetta i seguenti campi:

Campo	Tipo	Descrizione
`name`	string	Un identificatore univoco per questo server
`runtimeExecutable`	string	Il comando da eseguire, come `npm`, `yarn` o `node`
`runtimeArgs`	string[]	Argomenti passati a `runtimeExecutable`, come `["run", "dev"]`
`port`	number	La porta su cui il tuo server ascolta. Predefinito a 3000
`cwd`	string	Directory di lavoro relativa alla radice del tuo progetto. Predefinito alla radice del progetto. Usa `${workspaceFolder}` per fare riferimento alla radice del progetto esplicitamente

Campo	Tipo	Descrizione
`env`	object	Variabili di ambiente aggiuntive come coppie chiave-valore, come `{ "NODE_ENV": "development" }`. Non mettere segreti qui poiché questo file viene eseguito il commit nel tuo repo. I segreti impostati nel tuo profilo shell vengono ereditati automaticamente.
`autoPort`	boolean	Come gestire i conflitti di porta. Vedi di seguito
`program`	string	Uno script da eseguire con `node`. Vedi quando usare `program` vs `runtimeExecutable`
`args`	string[]	Argomenti passati a `program`. Usato solo quando `program` è impostato

When to use `program` vs `runtimeExecutable`

Usa `runtimeExecutable` con `runtimeArgs` per avviare un server di sviluppo tramite un gestore di pacchetti. Ad esempio, `"runtimeExecutable": "npm"` con `"runtimeArgs": ["run", "dev"]` esegue `npm run dev`.

Usa `program` quando hai uno script autonomo che vuoi eseguire con `node` direttamente. Ad esempio, `"program": "server.js"` esegue `node server.js`. Passa flag aggiuntivi con `args`.

Port conflicts

Il campo `autoPort` controlla cosa succede quando la tua porta preferita è già in uso:

- `true`: Claude trova e utilizza una porta libera automaticamente. Adatto per la maggior parte dei server di sviluppo.
- `false`: Claude fallisce con un errore. Usa questo quando il tuo server deve usare una porta specifica, come per i callback OAuth o gli allowlist CORS.

- **Non impostato (predefinito)**: Claude chiede se il server ha bisogno di quella porta esatta, quindi salva la tua risposta.

Quando Claude sceglie una porta diversa, passa la porta assegnata al tuo server tramite la variabile di ambiente `PORT`.

Examples

Queste configurazioni mostrano configurazioni comuni per diversi tipi di progetto:

Next.js

Questa configurazione esegue un'app Next.js usando Yarn sulla porta 3000:

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "web",
      "runtimeExecutable": "yarn",
      "runtimeArgs": ["dev"],
      "port": 3000
    }
  ]
}
```

Multiple servers

Per un monorepo con un frontend e un server API, definisci più configurazioni. Il frontend usa `autoPort: true` in modo che scelga una porta libera se 3000 è occupata, mentre il server API richiede la porta 8080 esattamente:

```
{
  "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

Per eseguire uno script Node.js direttamente invece di usare un comando del gestore di pacchetti, usa il campo `program`:

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "server",
      "program": "server.js",
      "args": ["--verbose"],
      "port": 4000
    }
  ]
}
```

Pianifica attività ricorrenti

Le attività pianificate avviano una nuova sessione locale automaticamente a un'ora e una frequenza che scegli. Usale per lavori ricorrenti come revisioni di codice giornaliere, controlli di aggiornamento delle dipendenze o briefing mattutini che traggono dal tuo calendario e dalla tua inbox.

Le attività vengono eseguite sulla tua macchina, quindi l'app desktop deve essere aperta e il tuo computer sveglio affinché si attivino. Vedi Come vengono eseguite le attività pianificate per i dettagli sui run mancati e il comportamento di recupero.

> **Note:**
>
> Per impostazione predefinita, le attività pianificate vengono eseguite contro qualsiasi stato la tua directory di lavoro sia, incluse le modifiche non eseguite il commit. Abilita l'interruttore worktree nell'input del prompt per dare a ogni run il suo worktree Git isolato, nello stesso modo in cui funzionano le sessioni parallele.

Per creare un'attività pianificata, fai clic su **Schedule** nella barra laterale, quindi su **+ New task**. Configura questi campi:

Campo	Descrizione
Name	Identificatore per l'attività. Convertito in minuscolo kebab-case e usato come nome della cartella su disco. Deve essere univoco tra le tue attività.

Campo	Descrizione
Description	Breve riepilogo mostrato nell'elenco delle attività.
Prompt	Le istruzioni inviate a Claude quando l'attività viene eseguita. Scrivi questo nello stesso modo in cui scriveresti qualsiasi messaggio nella casella del prompt. L'input del prompt include anche controlli per modello, modalità di autorizzazione, cartella di lavoro e worktree.
Frequency	Con quale frequenza l'attività viene eseguita. Vedi opzioni di frequenza di seguito.

Puoi anche creare un'attività descrivendo quello che vuoi in qualsiasi sessione. Ad esempio, "configura una revisione di codice giornaliera che viene eseguita ogni mattina alle 9am."

Frequency options

- **Manual**: nessuna pianificazione, viene eseguita solo quando fai clic su **Run now**. Utile per salvare un prompt che attivi su richiesta
- **Hourly**: viene eseguita ogni ora. Ogni attività ottiene un offset fisso di fino a 10 minuti dall'inizio dell'ora per scaglionare il traffico API
- **Daily**: mostra un selettore di ora, predefinito alle 9:00 AM ora locale
- **Weekdays**: uguale a Daily ma salta sabato e domenica
- **Weekly**: mostra un selettore di ora e un selettore di giorno

Per intervalli che il selettore non offre (ogni 15 minuti, primo di ogni mese, ecc.), chiedi a Claude in qualsiasi sessione Desktop di impostare la pianificazione. Usa il linguaggio naturale; ad esempio, "pianifica un'attività per eseguire tutti i test ogni 6 ore."

How scheduled tasks run

Le attività pianificate vengono eseguite localmente sulla tua macchina. Desktop controlla la pianificazione ogni minuto mentre l'app è aperta e avvia una sessione nuova quando un'attività è dovuta, indipendentemente da qualsiasi sessione manuale che hai aperta. Ogni attività ottiene un ritardo fisso di fino a 10 minuti dopo l'ora pianificata per scaglionare il traffico API. Il ritardo è deterministico: la stessa attività inizia sempre allo stesso offset.

Quando un'attività si attiva, ricevi una notifica desktop e una nuova sessione appare sotto una sezione **Scheduled** nella barra laterale. Aprila per vedere cosa ha fatto Claude, rivedi le modifiche o rispondi ai prompt di autorizzazione. La sessione funziona come qualsiasi altra: Claude può modificare file, eseguire comandi, creare commit e aprire pull request.

Le attività vengono eseguite solo mentre l'app desktop è in esecuzione e il tuo computer è sveglio. Se il tuo computer dorme durante un'ora pianificata, il run viene saltato. Per prevenire il sonno inattivo, abilita **Keep computer awake** in Impostazioni sotto **Desktop app → General**. Chiudere il coperchio del laptop lo mette comunque a dormire.

Missed runs

Quando l'app si avvia o il tuo computer si sveglia, Desktop controlla se ogni attività ha perso run negli ultimi sette giorni. Se lo ha fatto, Desktop avvia esattamente un run di recupero per l'ora più recentemente mancata e scarta qualsiasi cosa più vecchia. Un'attività giornaliera che ha perso sei giorni viene eseguita una volta al risveglio. Desktop mostra una notifica quando inizia un run di recupero.

Tieni questo in mente quando scrivi i prompt. Un'attività pianificata per le 9am potrebbe essere eseguita alle 23:00 se il tuo computer è stato addormentato tutto il giorno. Se il timing è importante, aggiungi guardrail al prompt stesso, ad esempio: "Rivedi solo i commit di oggi. Se è dopo le 17:00, salta la revisione e pubblica solo un riepilogo di quello che è stato perso."

Permissions for scheduled tasks

Ogni attività ha la sua propria modalità di autorizzazione, che imposti quando crei o modifichi l'attività. Le regole di autorizzazione da `~/.claude/settings.json` si applicano anche alle sessioni di attività pianificate. Se un'attività viene eseguita in modalità Ask e ha bisogno di eseguire uno strumento per il quale non ha autorizzazione, il run si blocca fino a quando non lo approvi. La sessione rimane aperta nella barra laterale in modo che tu possa rispondere in seguito.

Per evitare blocchi, fai clic su **Run now** dopo aver creato un'attività, guarda i prompt di autorizzazione e seleziona "always allow" per ognuno. I run futuri di quell'attività approvano automaticamente gli stessi strumenti senza chiedere. Puoi rivedere e revocare queste approvazioni dalla pagina dei dettagli dell'attività.

Manage scheduled tasks

Fai clic su un'attività nell'elenco **Schedule** per aprire la sua pagina di dettagli. Da qui puoi:

- **Run now**: avvia l'attività immediatamente senza aspettare l'ora pianificata successiva

- **Toggle repeats**: metti in pausa o riprendi i run pianificati senza eliminare l'attività
- **Edit**: cambia il prompt, la frequenza, la cartella o altre impostazioni
- **Review history**: vedi ogni run passato, inclusi quelli che sono stati saltati perché il tuo computer era addormentato
- **Review allowed permissions**: vedi e revoca le approvazioni degli strumenti salvate per questa attività dal pannello **Always allowed**
- **Delete**: rimuovi l'attività e archivia tutte le sessioni che ha creato

Puoi anche gestire le attività chiedendo a Claude in qualsiasi sessione Desktop. Ad esempio, "metti in pausa la mia attività dependency-audit", "elimina l'attività standup-prep" o "mostrami le mie attività pianificate."

Per modificare il prompt di un'attività su disco, apri `~/.claude/scheduled-tasks/<task-name>/SKILL.md` (o sotto `CLAUDE_CONFIG_DIR` se impostato). Il file utilizza frontmatter YAML per `name` e `description`, con il prompt come corpo. Le modifiche hanno effetto al prossimo run. Pianificazione, cartella, modello e stato abilitato non sono in questo file: cambiali tramite il modulo Edit o chiedi a Claude.

Configurazione dell'ambiente

L'ambiente che scegli quando avvii una sessione determina dove Claude viene eseguito e come ti connetti:

- **Local**: viene eseguito sulla tua macchina con accesso diretto ai tuoi file
- **Remote**: viene eseguito sull'infrastruttura cloud di Anthropic. Le sessioni continuano anche se chiudi l'app.
- **SSH**: viene eseguito su una macchina remota a cui ti connetti tramite SSH, come i tuoi stessi server, VM cloud o dev container

Local sessions

Le sessioni locali ereditano variabili di ambiente dalla tua shell. Se hai bisogno di variabili aggiuntive, impostale nel tuo profilo shell, come `~/.zshrc` o `~/.bashrc`, e riavvia l'app desktop. Vedi variabili di ambiente per l'elenco completo delle variabili supportate.

Extended thinking è abilitato per impostazione predefinita, il che migliora le prestazioni su compiti di ragionamento complesso ma utilizza token aggiuntivi. Per disabilitare completamente il thinking, imposta `MAX_THINKING_TOKENS=0` nel tuo profilo shell. Su Opus, `MAX_THINKING_TOKENS` viene ignorato tranne per `0` perché il ragionamento adattivo controlla la profondità del thinking.

Remote sessions

Le sessioni remote continuano in background anche se chiudi l'app. L'utilizzo conta verso i tuoi limiti del piano di abbonamento senza costi di calcolo separati.

Puoi creare ambienti cloud personalizzati con diversi livelli di accesso alla rete e variabili di ambiente. Seleziona il menu a discesa dell'ambiente quando avvii una sessione remota e scegli **Add environment**. Vedi ambienti cloud per i dettagli sulla configurazione dell'accesso alla rete e delle variabili di ambiente.

SSH sessions

Le sessioni SSH ti consentono di eseguire Claude Code su una macchina remota mentre usi l'app desktop come tua interfaccia. Questo è utile per lavorare con codebase che vivono su VM cloud, dev container o server con hardware o dipendenze specifiche.

Per aggiungere una connessione SSH, fai clic sul menu a discesa dell'ambiente prima di avviare una sessione e seleziona **+ Add SSH connection**. La finestra di dialogo chiede:

- **Name**: un'etichetta amichevole per questa connessione
- **SSH Host**: `user@hostname` o un host definito in `~/.ssh/config`
- **SSH Port**: predefinito a 22 se lasciato vuoto, o utilizza la porta dal tuo SSH config
- **Identity File**: percorso della tua chiave privata, come `~/.ssh/id_rsa`. Lascia vuoto per usare la chiave predefinita o il tuo SSH config.

Una volta aggiunta, la connessione appare nel menu a discesa dell'ambiente. Selezionala per avviare una sessione su quella macchina. Claude viene eseguito sulla macchina remota con accesso ai suoi file e strumenti.

Claude Code deve essere installato sulla macchina remota. Una volta connesso, le sessioni SSH supportano modalità di autorizzazione, connettori, plugin e MCP servers.

Configurazione aziendale

Le organizzazioni su piani Teams o Enterprise possono gestire il comportamento dell'app desktop attraverso i controlli della console di amministrazione, i file di impostazioni gestiti e le politiche di gestione dei dispositivi.

Admin console controls

Queste impostazioni sono configurate tramite la console delle impostazioni di amministrazione:

- **Abilita o disabilita la scheda Code**: controlla se gli utenti della tua organizzazione possono accedere a Claude Code nell'app desktop

- **Disabilita modalità Bypass permissions**: impedisci agli utenti della tua organizzazione di abilitare la modalità bypass permissions
- **Disabilita Claude Code sul web**: abilita o disabilita le sessioni remote per la tua organizzazione

Managed settings

Le impostazioni gestite sovrascrivono le impostazioni del progetto e dell'utente e si applicano quando Desktop genera sessioni CLI. Puoi impostare queste chiavi nel file impostazioni gestite della tua organizzazione o inviarle in remoto tramite la console di amministrazione.

Chiave	Descrizione
`disableBypassPermissionsMode`	imposta su `"disable"` per impedire agli utenti di abilitare la modalità bypass permissions. Vedi impostazioni gestite.

Per l'elenco completo delle impostazioni solo gestite incluse `allowManagedPermissionRulesOnly` e `allowManagedHooksOnly`, vedi impostazioni solo gestite.

Le impostazioni gestite remote caricate tramite la console di amministrazione attualmente si applicano solo alle sessioni CLI e IDE. Per le restrizioni specifiche di Desktop, usa i controlli della console di amministrazione sopra.

Device management policies

I team IT possono gestire l'app desktop tramite MDM su macOS o group policy su Windows. Le politiche disponibili includono l'abilitazione o la disabilitazione della funzione Claude Code, il controllo degli aggiornamenti automatici e l'impostazione di un URL di distribuzione personalizzato.

- **macOS**: configura tramite il dominio di preferenza `com.anthropic.Claude` usando strumenti come Jamf o Kandji
- **Windows**: configura tramite il registro a `SOFTWARE\Policies\Claude`

Authentication and SSO

Le organizzazioni aziendali possono richiedere SSO per tutti gli utenti. Vedi autenticazione per i dettagli a livello di piano e Configurazione di SSO per la configurazione SAML e OIDC.

Data handling

Claude Code elabora il tuo codice localmente nelle sessioni locali o sull'infrastruttura cloud di Anthropic nelle sessioni remote. Le conversazioni e il contesto del codice vengono inviati all'API di Anthropic per l'elaborazione. Vedi gestione dei dati per i dettagli sulla conservazione dei dati, la privacy e la conformità.

Deployment

Desktop può essere distribuito tramite strumenti di distribuzione aziendale:

- **macOS**: distribuisci tramite MDM come Jamf o Kandji usando il programma di installazione `.dmg`
- **Windows**: distribuisci tramite pacchetto MSIX o programma di installazione `.exe`. Vedi Distribuisci Claude Desktop per Windows per le opzioni di distribuzione aziendale inclusa l'installazione silenziosa

Per la configurazione di rete come impostazioni proxy, allowlist del firewall e gateway LLM, vedi configurazione di rete.

Per il riferimento completo della configurazione aziendale, vedi la guida alla configurazione aziendale.

Vieni dalla CLI?

Se usi già la CLI di Claude Code, Desktop esegue lo stesso motore sottostante con un'interfaccia grafica. Puoi eseguire entrambi contemporaneamente sulla stessa macchina, anche sullo stesso progetto. Ognuno mantiene una storia di sessione separata, ma condividono la configurazione e la memoria del progetto tramite file CLAUDE.md.

Per spostare una sessione CLI in Desktop, esegui `/desktop` nel terminale. Claude salva la tua sessione e l'apre nell'app desktop, quindi esce dalla CLI. Questo comando è disponibile solo su macOS e Windows.

> **Tip:**
>
> Quando usare Desktop vs CLI: usa Desktop quando vuoi revisione visiva dei diff, allegati di file o gestione della sessione in una barra laterale. Usa la CLI quando hai bisogno di scripting, automazione, provider di terze parti o preferisci un flusso di lavoro da terminale.

CLI flag equivalents

Questa tabella mostra l'equivalente dell'app desktop per i flag CLI comuni. I flag non elencati non hanno equivalente desktop perché sono progettati per scripting o automazione.

CLI	Equivalente desktop
`--model sonnet`	menu a discesa del modello accanto al pulsante di invio, prima di avviare una sessione
`--resume`, `--continue`	fai clic su una sessione nella barra laterale
`--permission-mode`	selettore di modalità accanto al pulsante di invio
`--dangerously-skip-permissions`	Modalità Bypass permissions. Abilita in Impostazioni → Claude Code → "Allow bypass permissions mode". Gli amministratori aziendali possono disabilitare questa impostazione.
`--add-dir`	aggiungi più repo con il pulsante + nelle sessioni remote
`--allowedTools`, `--disallowedTools`	non disponibile in Desktop
`--verbose`	non disponibile. Controlla i log di sistema: Console.app su macOS, Event Viewer → Windows Logs → Application su Windows
`--print`, `--output-format`	non disponibile. Desktop è solo interattivo.
`ANTHROPIC_MODEL` env var	menu a discesa del modello accanto al pulsante di invio
`MAX_THINKING_TOKENS` env var	imposta nel profilo shell; si applica alle sessioni locali. Vedi configurazione dell'ambiente.

Shared configuration

Desktop e CLI leggono gli stessi file di configurazione, quindi la tua configurazione viene trasferita:

- I file **CLAUDE.md** nel tuo progetto vengono utilizzati da entrambi
- I **MCP servers** configurati in `~/.claude.json` o `.mcp.json` funzionano in entrambi
- **Hooks** e **skills** definiti nelle impostazioni si applicano a entrambi

- **Impostazioni** in `~/.claude.json` e `~/.claude/settings.json` sono condivise. Le regole di autorizzazione, gli strumenti consentiti e altre impostazioni in `settings.json` si applicano alle sessioni Desktop.
- **Modelli**: Sonnet, Opus e Haiku sono disponibili in entrambi. In Desktop, seleziona il modello dal menu a discesa accanto al pulsante di invio prima di avviare una sessione. Non puoi cambiare il modello durante una sessione attiva.

> **Note:**
>
> **MCP servers: app desktop chat vs Claude Code**: i MCP servers configurati per l'app desktop chat Claude in `claude_desktop_config.json` sono separati da Claude Code e non appariranno nella scheda Code. Per usare MCP servers in Claude Code, configurali in `~/.claude.json` o nel file `.mcp.json` del tuo progetto. Vedi configurazione MCP per i dettagli.

Feature comparison

Questa tabella confronta le capacità principali tra CLI e Desktop. Per un elenco completo dei flag CLI, vedi il riferimento CLI.

Funzionalità	CLI	Desktop
Modalità di autorizzazione	tutte le modalità inclusa `dontAsk`	Chiedi autorizzazioni, Accetta automaticamente gli edit, Plan mode e Bypass permissions tramite Impostazioni
`--dangerously-skip-permissions`	Flag CLI	Modalità Bypass permissions. Abilita in Impostazioni → Claude Code → "Allow bypass permissions mode"
Provider di terze parti	Bedrock, Vertex, Foundry	non disponibile. Desktop si connette direttamente all'API di Anthropic.
MCP servers	configura nei file di impostazioni	UI Connectors per sessioni locali e SSH, o file di impostazioni
Plugins	comando `/plugin`	UI gestore plugin

Funzionalità	CLI	Desktop
@mention file	basato su testo	con autocomplete
Allegati di file	non disponibile	immagini, PDF
Isolamento della sessione	flag `--worktree`	worktree automatici
Sessioni multiple	terminali separati	schede della barra laterale
Attività ricorrenti	cron job, pipeline CI	attività pianificate
Scripting e automazione	`--print`, Agent SDK	non disponibile

What's not available in Desktop

Le seguenti funzionalità sono disponibili solo nella CLI o nell'estensione VS Code:

- **Provider di terze parti**: Desktop si connette direttamente all'API di Anthropic. Usa la CLI con Bedrock, Vertex o Foundry invece.
- **Linux**: l'app desktop è disponibile solo su macOS e Windows.
- **Suggerimenti di codice inline**: Desktop non fornisce suggerimenti in stile auto-complete. Funziona tramite prompt conversazionali e modifiche di codice esplicite.
- **Team di agent**: l'orchestrazione multi-agent è disponibile tramite la CLI e Agent SDK, non in Desktop.

Risoluzione dei problemi

Controlla la tua versione

Per vedere quale versione dell'app desktop stai eseguendo:

- **macOS**: fai clic su **Claude** nella barra dei menu, quindi su **About Claude**
- **Windows**: fai clic su **Help**, quindi su **About**

Fai clic sul numero di versione per copiarlo negli appunti.

Errori 403 o di autenticazione nella scheda Code

Se vedi `Error 403: Forbidden` o altri errori di autenticazione quando usi la scheda Code:

1. Esci e accedi di nuovo dal menu dell'app. Questo è il fix più comune.
2. Verifica di avere un abbonamento a pagamento attivo: Pro, Max, Teams o Enterprise.
3. Se la CLI funziona ma Desktop no, esci completamente dall'app desktop, non solo chiudere la finestra, quindi riapri e accedi di nuovo.

4. Controlla la tua connessione Internet e le impostazioni del proxy.

Schermata vuota o bloccata al lancio

Se l'app si apre ma mostra una schermata vuota o non reattiva:

1. Riavvia l'app.
2. Controlla gli aggiornamenti in sospeso. L'app si aggiorna automaticamente al lancio.
3. Su Windows, controlla Event Viewer per i log di crash sotto **Windows Logs → Application**.

"Failed to load session"

Se vedi `Failed to load session`, la cartella selezionata potrebbe non esistere più, un repository Git potrebbe richiedere Git LFS che non è installato, o i permessi dei file potrebbero impedire l'accesso. Prova a selezionare una cartella diversa o riavvia l'app.

Session not finding installed tools

Se Claude non riesce a trovare strumenti come `npm`, `node` o altri comandi CLI, verifica che gli strumenti funzionino nel tuo terminale regolare, controlla che il tuo profilo shell configuri correttamente PATH e riavvia l'app desktop per ricaricare le variabili di ambiente.

Git and Git LFS errors

Su Windows, Git è richiesto affinché la scheda Code avvii sessioni locali. Se vedi "Git is required," installa Git per Windows e riavvia l'app.

Se vedi "Git LFS is required by this repository but is not installed," installa Git LFS da git-lfs.com, esegui `git lfs install` e riavvia l'app.

MCP servers not working on Windows

Se gli interruttori MCP server non rispondono o i server non riescono a connettersi su Windows, controlla che il server sia configurato correttamente nelle tue impostazioni, riavvia l'app, verifica che il processo del server sia in esecuzione in Task Manager e rivedi i log del server per gli errori di connessione.

App won't quit

- **macOS**: premi Cmd+Q. Se l'app non risponde, usa Force Quit con Cmd+Option+Esc, seleziona Claude e fai clic su Force Quit.
- **Windows**: usa Task Manager con Ctrl+Shift+Esc per terminare il processo Claude.

Windows-specific issues

- **PATH not updated after install**: apri una nuova finestra di terminale. Gli aggiornamenti di PATH si applicano solo alle nuove sessioni di terminale.
- **Concurrent installation error**: se vedi un errore su un'altra installazione in corso ma non ce n'è una, prova a eseguire il programma di installazione come Amministratore.
- **ARM64**: i dispositivi Windows ARM64 sono completamente supportati.

Cowork tab unavailable on Intel Macs

La scheda Cowork richiede Apple Silicon (M1 o successivo) su macOS. Su Windows, Cowork è disponibile su tutto l'hardware supportato. Le schede Chat e Code funzionano normalmente su Mac Intel.

"Branch doesn't exist yet" when opening in CLI

Le sessioni remote possono creare rami che non esistono sulla tua macchina locale. Fai clic sul nome del ramo nella barra degli strumenti della sessione per copiarlo, quindi recuperalo localmente:

```
git fetch origin <branch-name>
git checkout <branch-name>
```

Still stuck?

- Cerca o segnala un bug su GitHub Issues
- Visita il centro di supporto Claude

Quando segnali un bug, includi la versione dell'app desktop, il tuo sistema operativo, il messaggio di errore esatto e i log pertinenti. Su macOS, controlla Console.app. Su Windows, controlla Event Viewer → Windows Logs → Application.

Usa Claude Code con Chrome (beta)

> *Connetti Claude Code al tuo browser Chrome per testare app web, eseguire il debug con i log della console, automatizzare la compilazione di moduli ed estrarre dati dalle pagine web.*

Claude Code si integra con l'estensione Claude in Chrome per darti capacità di automazione del browser dalla CLI o dall'estensione VS Code. Costruisci il tuo codice, quindi testa ed esegui il debug nel browser senza cambiare contesto.

Claude apre nuove schede per le attività del browser e condivide lo stato di accesso del tuo browser, quindi può accedere a qualsiasi sito in cui sei già connesso. Le azioni del browser vengono eseguite in una finestra Chrome visibile in tempo reale. Quando Claude incontra una pagina di accesso o un CAPTCHA, si ferma e ti chiede di gestirlo manualmente.

> **Note:**
>
> L'integrazione con Chrome è in beta e attualmente funziona solo con Google Chrome. Non è ancora supportata su Brave, Arc o altri browser basati su Chromium. Anche WSL (Windows Subsystem for Linux) non è supportato.

Capacità

Con Chrome connesso, puoi concatenare azioni del browser con attività di codifica in un singolo flusso di lavoro:

- **Debug in tempo reale**: leggi gli errori della console e lo stato del DOM direttamente, quindi correggi il codice che li ha causati
- **Verifica del design**: costruisci un'interfaccia utente da un mock di Figma, quindi aprila nel browser per verificare che corrisponda
- **Test di app web**: testa la convalida dei moduli, verifica la presenza di regressioni visive o verifica i flussi utente
- **App web autenticate**: interagisci con Google Docs, Gmail, Notion o qualsiasi app in cui sei connesso senza connettori API
- **Estrazione di dati**: estrai informazioni strutturate dalle pagine web e salvale localmente

- **Automazione delle attività**: automatizza le attività ripetitive del browser come l'immissione di dati, la compilazione di moduli o i flussi di lavoro multi-sito
- **Registrazione della sessione**: registra le interazioni del browser come GIF per documentare o condividere ciò che è accaduto

Prerequisiti

Prima di utilizzare Claude Code con Chrome, hai bisogno di:

- Browser Google Chrome
- Estensione Claude in Chrome versione 1.0.36 o superiore
- Claude Code versione 2.0.73 o superiore
- Un piano Anthropic diretto (Pro, Max, Team o Enterprise)

> **Note:**
>
> L'integrazione con Chrome non è disponibile tramite provider di terze parti come Amazon Bedrock, Google Cloud Vertex AI o Microsoft Foundry. Se accedi a Claude esclusivamente tramite un provider di terze parti, hai bisogno di un account claude.ai separato per utilizzare questa funzione.

Inizia nella CLI

Step 1: Avvia Claude Code con Chrome

Avvia Claude Code con il flag `--chrome`:

```
claude --chrome
```

Puoi anche abilitare Chrome da una sessione esistente eseguendo `/chrome`.

Step 2: Chiedi a Claude di usare il browser

Questo esempio naviga verso una pagina, interagisce con essa e segnala ciò che trova, tutto dal tuo terminale o editor:

```
Go to code.claude.com/docs, click on the search box,
type "hooks", and tell me what results appear
```

Esegui `/chrome` in qualsiasi momento per verificare lo stato della connessione, gestire le autorizzazioni o riconnettere l'estensione.

Per VS Code, vedi automazione del browser in VS Code.

Abilita Chrome per impostazione predefinita

Per evitare di passare `--chrome` ogni sessione, esegui `/chrome` e seleziona "Enabled by default".

Nell'estensione VS Code, Chrome è disponibile ogni volta che l'estensione Chrome è installata. Non è necessario alcun flag aggiuntivo.

> **Note:**
>
> L'abilitazione di Chrome per impostazione predefinita nella CLI aumenta l'utilizzo del contesto poiché gli strumenti del browser vengono sempre caricati. Se noti un aumento del consumo di contesto, disabilita questa impostazione e utilizza `--chrome` solo quando necessario.

Gestisci le autorizzazioni del sito

Le autorizzazioni a livello di sito vengono ereditate dall'estensione Chrome. Gestisci le autorizzazioni nelle impostazioni dell'estensione Chrome per controllare quali siti Claude può navigare, fare clic e digitare.

Flussi di lavoro di esempio

Questi esempi mostrano i modi comuni per combinare azioni del browser con attività di codifica. Esegui `/mcp` e seleziona `claude-in-chrome` per vedere l'elenco completo degli strumenti del browser disponibili.

Testa un'applicazione web locale

Quando sviluppi un'app web, chiedi a Claude di verificare che le tue modifiche funzionino correttamente:

```
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 naviga verso il tuo server locale, interagisce con il modulo e segnala ciò che osserva.

Debug con i log della console

Claude può leggere l'output della console per aiutare a diagnosticare i problemi. Dì a Claude quali modelli cercare piuttosto che chiedere tutto l'output della console, poiché i log possono essere dettagliati:

```
Open the dashboard page and check the console for any errors when
the page loads.
```

Claude legge i messaggi della console e può filtrare per modelli specifici o tipi di errore.

Automatizza la compilazione dei moduli

Velocizza le attività ripetitive di immissione dati:

```
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 legge il tuo file locale, naviga nell'interfaccia web e immette i dati per ogni record.

Bozza di contenuto in Google Docs

Usa Claude per scrivere direttamente nei tuoi documenti senza configurazione API:

```
Draft a project update based on the recent commits and add it to my
Google Doc at docs.google.com/document/d/abc123
```

Claude apre il documento, fa clic nell'editor e digita il contenuto. Questo funziona con qualsiasi app web in cui sei connesso: Gmail, Notion, Sheets e altro.

Estrai dati dalle pagine web

Estrai informazioni strutturate dai siti web:

```
Go to the product listings page and extract the name, price, and
availability for each item. Save the results as a CSV file.
```

Claude naviga verso la pagina, legge il contenuto e compila i dati in un formato strutturato.

Esegui flussi di lavoro multi-sito

Coordina le attività su più siti web:

```
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 lavora su più schede per raccogliere informazioni e completare il flusso di lavoro.

Registra una GIF demo

Crea registrazioni condivisibili delle interazioni del browser:

```
Record a GIF showing how to complete the checkout flow, from adding
an item to the cart through to the confirmation page.
```

Claude registra la sequenza di interazione e la salva come file GIF.

Troubleshooting

Estensione non rilevata

Se Claude Code mostra “Chrome extension not detected”:

1. Verifica che l’estensione Chrome sia installata e abilitata in `chrome://extensions`
2. Verifica che Claude Code sia aggiornato eseguendo `claude --version`
3. Verifica che Chrome sia in esecuzione
4. Esegui `/chrome` e seleziona “Reconnect extension” per ristabilire la connessione
5. Se il problema persiste, riavvia sia Claude Code che Chrome

La prima volta che abiliti l’integrazione con Chrome, Claude Code installa un file di configurazione dell’host di messaggistica nativa. Chrome legge questo file all’avvio, quindi se l’estensione non viene rilevata al primo tentativo, riavvia Chrome per raccogliere la nuova configurazione.

Se la connessione continua a non funzionare, verifica che il file di configurazione dell’host esista in:

- **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**: controlla `HKCU\Software\Google\Chrome\NativeMessagingHosts\` nel Registro di Windows

Browser non risponde

Se i comandi del browser di Claude smettono di funzionare:

1. Verifica se una finestra di dialogo modale (avviso, conferma, prompt) sta bloccando la pagina. Le finestre di dialogo JavaScript bloccano gli eventi del browser e impediscono a Claude di ricevere comandi. Chiudi manualmente la finestra di dialogo, quindi dì a Claude di continuare.
2. Chiedi a Claude di creare una nuova scheda e riprovare
3. Riavvia l'estensione Chrome disabilitandola e riabilitandola in `chrome://extensions`

La connessione si interrompe durante le sessioni lunghe

Il service worker dell'estensione Chrome può diventare inattivo durante le sessioni estese, il che interrompe la connessione. Se gli strumenti del browser smettono di funzionare dopo un periodo di inattività, esegui `/chrome` e seleziona "Reconnect extension".

Problemi specifici di Windows

Su Windows, potresti riscontrare:

- **Conflitti di named pipe (EADDRINUSE)**: se un altro processo sta utilizzando la stessa named pipe, riavvia Claude Code. Chiudi tutte le altre sessioni di Claude Code che potrebbero utilizzare Chrome.
- **Errori dell'host di messaggistica nativa**: se l'host di messaggistica nativa si arresta in modo anomalo all'avvio, prova a reinstallare Claude Code per rigenerare la configurazione dell'host.

Messaggi di errore comuni

Questi sono gli errori più frequentemente riscontrati e come risolverli:

Errore	Causa	Soluzione
"Browser extension is not connected"	L'host di messaggistica nativa non può raggiungere l'estensione	Riavvia Chrome e Claude Code, quindi esegui `/chrome` per riconnetterti

Errore	Causa	Soluzione
“Extension not detected”	L’estensione Chrome non è installata o è disabilitata	Installa o abilita l’estensione in `chrome://extensions`
“No tab available”	Claude ha tentato di agire prima che una scheda fosse pronta	Chiedi a Claude di creare una nuova scheda e riprovare
“Receiving end does not exist”	Il service worker dell’estensione è diventato inattivo	Esegui `/chrome` e seleziona “Reconnect extension”

Vedi anche

- Usa Claude Code in VS Code: automazione del browser nell’estensione VS Code
- Riferimento CLI: flag della riga di comando incluso `--chrome`
- Flussi di lavoro comuni: altri modi per utilizzare Claude Code
- Dati e privacy: come Claude Code gestisce i tuoi dati
- Introduzione a Claude in Chrome: documentazione completa per l’estensione Chrome, incluse scorciatoie, pianificazione e autorizzazioni

Claude Code in Slack

> *Delega i compiti di codifica direttamente dal tuo workspace Slack*

Claude Code in Slack porta la potenza di Claude Code direttamente nel tuo workspace Slack. Quando menzioni `@Claude` con un compito di codifica, Claude rileva automaticamente l'intento e crea una sessione Claude Code sul web, permettendoti di delegare il lavoro di sviluppo senza lasciare le conversazioni del tuo team.

Questa integrazione è costruita sull'app Claude for Slack esistente ma aggiunge un instradamento intelligente a Claude Code sul web per le richieste relative alla codifica.

Casi d'uso

- **Investigazione e correzione di bug**: Chiedi a Claude di investigare e correggere i bug non appena vengono segnalati nei canali Slack.
- **Revisioni del codice rapide e modifiche**: Fai in modo che Claude implementi piccole funzionalità o effettui il refactoring del codice in base al feedback del team.
- **Debug collaborativo**: Quando le discussioni del team forniscono contesto cruciale (ad esempio, riproduzioni di errori o segnalazioni di utenti), Claude può utilizzare queste informazioni per informare il suo approccio al debug.
- **Esecuzione di attività parallele**: Avvia compiti di codifica in Slack mentre continui altri lavori, ricevendo notifiche al completamento.

Prerequisiti

Prima di utilizzare Claude Code in Slack, assicurati di avere quanto segue:

Requisito	Dettagli
Piano Claude	Pro, Max, Team o Enterprise con accesso a Claude Code (posti premium)
Claude Code sul web	L'accesso a Claude Code sul web deve essere abilitato
Account GitHub	Connesso a Claude Code sul web con almeno un repository autenticato

Requisito	Dettagli
Autenticazione Slack	Il tuo account Slack collegato al tuo account Claude tramite l'app Claude

Configurazione di Claude Code in Slack

Step 1: Installa l'app Claude in Slack

Un amministratore del workspace deve installare l'app Claude dal Slack App Marketplace. Visita il Slack App Marketplace e fai clic su "Add to Slack" per iniziare il processo di installazione.

Step 2: Connetti il tuo account Claude

Dopo l'installazione dell'app, autentica il tuo account Claude individuale:

1. Apri l'app Claude in Slack facendo clic su "Claude" nella tua sezione App
2. Naviga alla scheda App Home
3. Fai clic su "Connect" per collegare il tuo account Slack al tuo account Claude
4. Completa il flusso di autenticazione nel tuo browser

Step 3: Configura Claude Code sul web

Assicurati che Claude Code sul web sia configurato correttamente:

- Visita claude.ai/code e accedi con lo stesso account che hai connesso a Slack
- Connetti il tuo account GitHub se non è già connesso
- Autentica almeno un repository con cui desideri che Claude lavori

Step 4: Scegli la tua modalità di instradamento

Dopo aver connesso i tuoi account, configura come Claude gestisce i tuoi messaggi in Slack. Naviga alla App Home di Claude in Slack per trovare l'impostazione **Routing Mode**.

Modalità	Comportamento
Code only	Claude instrada tutte le @mention a sessioni Claude Code. Ideale per i team che utilizzano Claude in Slack esclusivamente per compiti di sviluppo.

Modalità	Comportamento
Code + Chat	Claude analizza ogni messaggio e instrada intelligentemente tra Claude Code (per compiti di codifica) e Claude Chat (per scrittura, analisi e domande generali). Ideale per i team che desiderano un unico punto di ingresso @Claude per tutti i tipi di lavoro.

> **Note:**
>
> In modalità Code + Chat, se Claude instrada un messaggio a Chat ma desideravi una sessione di codifica, puoi fare clic su "Retry as Code" per creare una sessione Claude Code. Allo stesso modo, se viene instradato a Code ma desideravi una sessione Chat, puoi scegliere quell'opzione in quel thread.

Come funziona

Rilevamento automatico

Quando menzioni @Claude in un canale o thread Slack, Claude analizza automaticamente il tuo messaggio per determinare se si tratta di un compito di codifica. Se Claude rileva l'intento di codifica, instradarà la tua richiesta a Claude Code sul web invece di rispondere come un assistente chat regolare.

Puoi anche dire esplicitamente a Claude di gestire una richiesta come un compito di codifica, anche se non lo rileva automaticamente.

> **Note:**
>
> Claude Code in Slack funziona solo nei canali (pubblici o privati). Non funziona nei messaggi diretti (DM).

Raccolta del contesto

Da thread: Quando @menzioni Claude in un thread, raccoglie il contesto da tutti i messaggi in quel thread per comprendere la conversazione completa.

Da canali: Quando menzionato direttamente in un canale, Claude guarda i messaggi recenti del canale per il contesto rilevante.

Questo contesto aiuta Claude a comprendere il problema, selezionare il repository appropriato e informare il suo approccio al compito.

> **Warning:**
>
> Quando @Claude viene invocato in Slack, a Claude viene dato accesso al contesto della conversazione per comprendere meglio la tua richiesta. Claude può seguire le indicazioni da altri messaggi nel contesto, quindi gli utenti dovrebbero assicurarsi di utilizzare Claude solo in conversazioni Slack affidabili.

Flusso della sessione

1. **Avvio**: Menzioni @Claude con una richiesta di codifica
2. **Rilevamento**: Claude analizza il tuo messaggio e rileva l'intento di codifica
3. **Creazione della sessione**: Una nuova sessione Claude Code viene creata su claude.ai/code
4. **Aggiornamenti di avanzamento**: Claude pubblica aggiornamenti di stato nel tuo thread Slack mentre il lavoro progredisce
5. **Completamento**: Al termine, Claude ti @menziona con un riepilogo e pulsanti di azione
6. **Revisione**: Fai clic su "View Session" per vedere la trascrizione completa, o "Create PR" per aprire una pull request

Elementi dell'interfaccia utente

App Home

La scheda App Home mostra lo stato della tua connessione e ti consente di connettere o disconnettere il tuo account Claude da Slack.

Azioni sui messaggi

- **View Session**: Apre la sessione Claude Code completa nel tuo browser dove puoi vedere tutto il lavoro eseguito, continuare la sessione o fare richieste aggiuntive.
- **Create PR**: Crea una pull request direttamente dalle modifiche della sessione.
- **Retry as Code**: Se Claude inizialmente risponde come assistente chat ma desideravi una sessione di codifica, fai clic su questo pulsante per riprovare la richiesta come un compito Claude Code.
- **Change Repo**: Ti consente di selezionare un repository diverso se Claude ha scelto in modo errato.

Selezione del repository

Claude seleziona automaticamente un repository in base al contesto della tua conversazione Slack. Se più repository potrebbero applicarsi, Claude potrebbe visualizzare un menu a discesa che ti consente di scegliere quello corretto.

Accesso e autorizzazioni

Accesso a livello di utente

Tipo di accesso	Requisito
Sessioni Claude Code	Ogni utente esegue sessioni con il proprio account Claude
Utilizzo e limiti di velocità	Le sessioni contano rispetto ai limiti del piano del singolo utente
Accesso al repository	Gli utenti possono accedere solo ai repository che hanno personalmente connesso
Cronologia sessioni	Le sessioni appaiono nella tua cronologia Claude Code su claude.ai/code

Autorizzazioni amministratore del workspace

Gli amministratori del workspace Slack controllano se l'app Claude può essere installata nel workspace. I singoli utenti si autenticano quindi con i propri account Claude per utilizzare l'integrazione.

Cosa è accessibile dove

In Slack: Vedrai aggiornamenti di stato, riepiloghi di completamento e pulsanti di azione. La trascrizione completa è preservata e sempre accessibile.

Sul web: La sessione Claude Code completa con la cronologia della conversazione completa, tutte le modifiche al codice, operazioni su file e la possibilità di continuare la sessione o creare pull request.

Best practice

Scrivere richieste efficaci

- **Sii specifico**: Includi nomi di file, nomi di funzioni o messaggi di errore quando rilevante.

- **Fornisci contesto**: Menziona il repository o il progetto se non è chiaro dalla conversazione.
- **Definisci il successo**: Spiega come dovrebbe apparire "fatto"—Claude dovrebbe scrivere test? Aggiornare la documentazione? Creare una PR?
- **Usa thread**: Rispondi nei thread quando discuti di bug o funzionalità in modo che Claude possa raccogliere il contesto completo.

Quando utilizzare Slack rispetto al web

Usa Slack quando: Il contesto esiste già in una discussione Slack, desideri avviare un compito in modo asincrono, o stai collaborando con compagni di team che hanno bisogno di visibilità.

Usa il web direttamente quando: Hai bisogno di caricare file, desideri un'interazione in tempo reale durante lo sviluppo, o stai lavorando su compiti più lunghi e complessi.

Risoluzione dei problemi

Le sessioni non si avviano

1. Verifica che il tuo account Claude sia connesso nella App Home di Claude
2. Controlla di avere l'accesso a Claude Code sul web abilitato
3. Assicurati di avere almeno un repository GitHub connesso a Claude Code

Repository non visualizzato

1. Connetti il repository in Claude Code sul web su claude.ai/code
2. Verifica le tue autorizzazioni GitHub per quel repository
3. Prova a disconnettere e riconnettere il tuo account GitHub

Repository errato selezionato

1. Fai clic sul pulsante "Change Repo" per selezionare un repository diverso
2. Includi il nome del repository nella tua richiesta per una selezione più accurata

Errori di autenticazione

1. Disconnetti e riconnetti il tuo account Claude nella App Home
2. Assicurati di essere connesso all'account Claude corretto nel tuo browser
3. Controlla che il tuo piano Claude includa l'accesso a Claude Code

Scadenza della sessione

1. Le sessioni rimangono accessibili nella tua cronologia Claude Code sul web

2. Puoi continuare o fare riferimento a sessioni passate da claude.ai/code

Limitazioni attuali

- **Solo GitHub**: Attualmente supporta repository su GitHub.
- **Una PR alla volta**: Ogni sessione può creare una pull request.
- **Si applicano i limiti di velocità**: Le sessioni utilizzano i limiti di velocità del tuo piano Claude individuale.
- **Accesso web richiesto**: Gli utenti devono avere accesso a Claude Code sul web; coloro che non lo hanno riceveranno solo risposte di chat Claude standard.

Risorse correlate

- Claude Code sul web Scopri di più su Claude Code sul web
- Claude for Slack Documentazione generale di Claude for Slack
- Slack App Marketplace Installa l'app Claude dal Marketplace di Slack
- Claude Help Center Ottieni supporto aggiuntivo

Claude Code sul web

> *Esegui attività Claude Code in modo asincrono su infrastruttura cloud sicura*

Note:

Claude Code sul web è attualmente in anteprima di ricerca.

Cos'è Claude Code sul web?

Claude Code sul web consente agli sviluppatori di avviare Claude Code dall'app Claude. Questo è perfetto per:

- **Rispondere a domande**: Poni domande sull'architettura del codice e su come vengono implementate le funzionalità
- **Correzioni di bug e attività di routine**: Attività ben definite che non richiedono una guida frequente
- **Lavoro parallelo**: Affronta più correzioni di bug in parallelo
- **Repository non sulla tua macchina locale**: Lavora su codice che non hai controllato localmente
- **Modifiche backend**: Dove Claude Code può scrivere test e poi scrivere codice per superare quei test

Claude Code è disponibile anche nell'app Claude per iOS e Android per avviare attività in movimento e monitorare il lavoro in corso.

Puoi avviare nuove attività sul web dal tuo terminale con `--remote`, oppure teletrasportare sessioni web nel tuo terminale per continuare localmente. Per utilizzare l'interfaccia web mentre esegui Claude Code sulla tua macchina invece dell'infrastruttura cloud, vedi Remote Control.

Chi può utilizzare Claude Code sul web?

Claude Code sul web è disponibile in anteprima di ricerca per:

- **Utenti Pro**
- **Utenti Max**
- **Utenti Team**

- **Utenti Enterprise** con posti premium o posti Chat + Claude Code

Iniziare

1. Visita claude.ai/code
2. Connetti il tuo account GitHub
3. Installa l'app Claude GitHub nei tuoi repository
4. Seleziona il tuo ambiente predefinito
5. Invia la tua attività di codifica
6. Rivedi le modifiche nella vista diff, itera con commenti, quindi crea una pull request

Come funziona

Quando avvii un'attività su Claude Code sul web:

1. **Clonazione del repository**: Il tuo repository viene clonato su una macchina virtuale gestita da Anthropic
2. **Configurazione dell'ambiente**: Claude prepara un ambiente cloud sicuro con il tuo codice, quindi esegue il tuo script di configurazione se configurato
3. **Configurazione della rete**: L'accesso a Internet viene configurato in base alle tue impostazioni
4. **Esecuzione dell'attività**: Claude analizza il codice, apporta modifiche, esegue test e verifica il suo lavoro
5. **Completamento**: Ricevi una notifica al termine e puoi creare una PR con le modifiche
6. **Risultati**: Le modifiche vengono inviate a un ramo, pronto per la creazione della pull request

Rivedi le modifiche con la vista diff

La vista diff ti consente di vedere esattamente cosa ha cambiato Claude prima di creare una pull request. Invece di fare clic su "Crea PR" per rivedere le modifiche in GitHub, visualizza il diff direttamente nell'app e itera con Claude fino a quando le modifiche non sono pronte.

Quando Claude apporta modifiche ai file, appare un indicatore di statistiche diff che mostra il numero di righe aggiunte e rimosse (ad esempio, `+12 -1`). Seleziona questo indicatore per aprire il visualizzatore diff, che visualizza un elenco di file a sinistra e le modifiche per ogni file a destra.

Dalla vista diff, puoi:

- Rivedere le modifiche file per file
- Commentare modifiche specifiche per richiedere modifiche
- Continuare a iterare con Claude in base a quello che vedi

Questo ti consente di perfezionare le modifiche attraverso più round di feedback senza creare PR bozza o passare a GitHub.

Spostamento di attività tra web e terminale

Puoi avviare nuove attività sul web dal tuo terminale, oppure estrarre sessioni web nel tuo terminale per continuare localmente. Le sessioni web persistono anche se chiudi il tuo laptop e puoi monitorarle da qualsiasi luogo, inclusa l'app mobile Claude.

> **Note:**
>
> L'handoff della sessione è unidirezionale: puoi estrarre sessioni web nel tuo terminale, ma non puoi inviare una sessione terminale esistente al web. Il flag `--remote` crea una *nuova* sessione web per il tuo repository attuale.

Dal terminale al web

Avvia una sessione web dalla riga di comando con il flag `--remote`:

```
claude --remote "Fix the authentication bug in src/auth/login.ts"
```

Questo crea una nuova sessione web su claude.ai. L'attività viene eseguita nel cloud mentre continui a lavorare localmente. Usa `/tasks` per controllare l'avanzamento, oppure apri la sessione su claude.ai o l'app mobile Claude per interagire direttamente. Da lì puoi guidare Claude, fornire feedback o rispondere a domande proprio come in qualsiasi altra conversazione.

Suggerimenti per attività remote

Pianifica localmente, esegui da remoto: Per attività complesse, avvia Claude in plan mode per collaborare sull'approccio, quindi invia il lavoro al web:

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

In plan mode, Claude può solo leggere file ed esplorare la base di codice. Una volta soddisfatto del piano, avvia una sessione remota per l'esecuzione autonoma:

```
claude --remote "Execute the migration plan in docs/migration-plan.md"
```

Questo modello ti dà il controllo sulla strategia mentre consente a Claude di eseguire autonomamente nel cloud.

Esegui attività in parallelo: Ogni comando `--remote` crea la sua propria sessione web che viene eseguita indipendentemente. Puoi avviare più attività e verranno tutte eseguite simultaneamente in sessioni separate:

```
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"
```

Monitora tutte le sessioni con `/tasks`. Quando una sessione si completa, puoi creare una PR dall'interfaccia web o teletrasportare la sessione nel tuo terminale per continuare a lavorare.

Dal web al terminale

Ci sono diversi modi per estrarre una sessione web nel tuo terminale:

- **Usando `/teleport`**: Da Claude Code, esegui `/teleport` (o `/tp`) per vedere un selettore interattivo delle tue sessioni web. Se hai modifiche non sottoposte a commit, ti verrà chiesto di archiviarle prima.
- **Usando `--teleport`**: Dalla riga di comando, esegui `claude --teleport` per un selettore di sessione interattivo, oppure `claude --teleport <session-id>` per riprendere una sessione specifica direttamente.
- **Da `/tasks`**: Esegui `/tasks` per vedere le tue sessioni in background, quindi premi `t` per teletrasportarti in una
- **Dall'interfaccia web**: Fai clic su "Apri in CLI" per copiare un comando che puoi incollare nel tuo terminale

Quando teletrasporti una sessione, Claude verifica che sei nel repository corretto, recupera e controlla il ramo dalla sessione remota e carica la cronologia completa della conversazione nel tuo terminale.

Requisiti per il teletrasporto

Il teletrasporto verifica questi requisiti prima di riprendere una sessione. Se un requisito non è soddisfatto, vedrai un errore o ti verrà chiesto di risolvere il problema.

Requisito	Dettagli
Stato git pulito	La tua directory di lavoro non deve avere modifiche non sottoposte a commit. Il teletrasporto ti chiede di archiviare le modifiche se necessario.
Repository corretto	Devi eseguire `--teleport` da un checkout dello stesso repository, non da un fork.
Ramo disponibile	Il ramo dalla sessione web deve essere stato inviato al remoto. Il teletrasporto lo recupera e lo controlla automaticamente.
Stesso account	Devi essere autenticato allo stesso account Claude.ai utilizzato nella sessione web.

Condivisione di sessioni

Per condividere una sessione, attiva/disattiva la sua visibilità in base ai tipi di account di seguito. Dopo di che, condividi il link della sessione così com'è. I destinatari che aprono la tua sessione condivisa vedranno lo stato più recente della sessione al caricamento, ma la pagina del destinatario non si aggiornerà in tempo reale.

Condivisione da un account Enterprise o Teams

Per gli account Enterprise e Teams, le due opzioni di visibilità sono **Private** e **Team**. La visibilità Team rende la sessione visibile agli altri membri della tua organizzazione Claude.ai. La verifica dell'accesso al repository è abilitata per impostazione predefinita, in base all'account GitHub connesso all'account del destinatario. Il nome visualizzato del tuo account è visibile a tutti i destinatari con accesso. Le sessioni Claude in Slack vengono condivise automaticamente con visibilità Team.

Condivisione da un account Max o Pro

Per gli account Max e Pro, le due opzioni di visibilità sono **Private** e **Public**. La visibilità Public rende la sessione visibile a qualsiasi utente connesso a claude.ai.

Controlla la tua sessione per contenuti sensibili prima di condividere. Le sessioni possono contenere codice e credenziali da repository GitHub privati. La verifica dell'accesso al repository non è abilitata per impostazione predefinita.

Abilita la verifica dell'accesso al repository e/o nascondi il tuo nome dalle tue sessioni condivise andando su Impostazioni > Claude Code > Impostazioni di condivisione.

Gestione delle sessioni

Archiviazione di sessioni

Puoi archiviare sessioni per mantenere organizzato il tuo elenco di sessioni. Le sessioni archiviate sono nascoste dall'elenco di sessioni predefinito ma possono essere visualizzate filtrando le sessioni archiviate.

Per archiviare una sessione, passa il mouse sulla sessione nella barra laterale e fai clic sull'icona di archiviazione.

Eliminazione di sessioni

L'eliminazione di una sessione rimuove permanentemente la sessione e i suoi dati. Questa azione non può essere annullata. Puoi eliminare una sessione in due modi:

- **Dalla barra laterale**: Filtra le sessioni archiviate, quindi passa il mouse sulla sessione che desideri eliminare e fai clic sull'icona di eliminazione
- **Dal menu della sessione**: Apri una sessione, fai clic sul menu a discesa accanto al titolo della sessione e seleziona **Elimina**

Ti verrà chiesto di confermare prima che una sessione venga eliminata.

Ambiente cloud

Immagine predefinita

Costruiamo e manteniamo un'immagine universale con toolchain comuni e ecosistemi di linguaggio preinstallati. Questa immagine include:

- Linguaggi di programmazione e runtime popolari
- Strumenti di compilazione comuni e gestori di pacchetti
- Framework di test e linter

Verifica degli strumenti disponibili

Per vedere cosa è preinstallato nel tuo ambiente, chiedi a Claude Code di eseguire:

```
check-tools
```

Questo comando visualizza:

- Linguaggi di programmazione e loro versioni
- Gestori di pacchetti disponibili
- Strumenti di sviluppo installati

Configurazioni specifiche del linguaggio

L'immagine universale include ambienti preconfigurati per:

- **Python**: Python 3.x con pip, poetry e librerie scientifiche comuni
- **Node.js**: Ultime versioni LTS con npm, yarn, pnpm e bun
- **Ruby**: Versioni 3.1.6, 3.2.6, 3.3.6 (predefinita: 3.3.6) con gem, bundler e rbenv per la gestione delle versioni
- **PHP**: Versione 8.4.14
- **Java**: OpenJDK con Maven e Gradle
- **Go**: Ultima versione stabile con supporto dei moduli
- **Rust**: Toolchain Rust con cargo
- **C++**: Compilatori GCC e Clang

Database

L'immagine universale include i seguenti database:

- **PostgreSQL**: Versione 16
- **Redis**: Versione 7.0

Configurazione dell'ambiente

Quando avvii una sessione in Claude Code sul web, ecco cosa accade dietro le quinte:

1. **Preparazione dell'ambiente**: Cloniamo il tuo repository ed eseguiamo qualsiasi script di configurazione configurato. Il repository verrà clonato con il ramo predefinito nel tuo repository GitHub. Se desideri controllare un ramo specifico, puoi specificarlo nel prompt.
2. **Configurazione della rete**: Configuriamo l'accesso a Internet per l'agente. L'accesso a Internet è limitato per impostazione predefinita, ma puoi configurare l'ambiente per non avere accesso a Internet o accesso completo a Internet in base alle tue esigenze.

3. **Esecuzione di Claude Code**: Claude Code viene eseguito per completare la tua attività, scrivendo codice, eseguendo test e verificando il suo lavoro. Puoi guidare e dirigere Claude durante la sessione tramite l'interfaccia web. Claude rispetta il contesto che hai definito nel tuo `CLAUDE.md`.
4. **Risultato**: Quando Claude completa il suo lavoro, invierà il ramo al remoto. Potrai creare una PR per il ramo.

> **Note:**
>
> Claude opera interamente attraverso il terminale e gli strumenti CLI disponibili nell'ambiente. Utilizza gli strumenti preinstallati nell'immagine universale e qualsiasi strumento aggiuntivo che installi tramite hooks o gestione delle dipendenze.

Per aggiungere un nuovo ambiente: Seleziona l'ambiente attuale per aprire il selettore di ambiente, quindi seleziona "Aggiungi ambiente". Questo aprirà una finestra di dialogo in cui puoi specificare il nome dell'ambiente, il livello di accesso alla rete, le variabili di ambiente e uno script di configurazione.

Per aggiornare un ambiente esistente: Seleziona l'ambiente attuale, a destra del nome dell'ambiente, e seleziona il pulsante delle impostazioni. Questo aprirà una finestra di dialogo in cui puoi aggiornare il nome dell'ambiente, l'accesso alla rete, le variabili di ambiente e lo script di configurazione.

Per selezionare il tuo ambiente predefinito dal terminale: Se hai più ambienti configurati, esegui `/remote-env` per scegliere quale utilizzare quando avvii sessioni web dal tuo terminale con `--remote`. Con un singolo ambiente, questo comando mostra la tua configurazione attuale.

> **Note:**
>
> Le variabili di ambiente devono essere specificate come coppie chiave-valore, in formato `.env`. Ad esempio:
>
> ```
> API_KEY=your_api_key
> DEBUG=true
> ```

Script di configurazione

Uno script di configurazione è uno script Bash che viene eseguito quando inizia una nuova sessione cloud, prima che Claude Code si avvii. Utilizza gli script di configurazione per installare dipendenze, configurare strumenti o preparare qualsiasi cosa di cui l'ambiente cloud ha bisogno che non sia nell'immagine predefinita.

Gli script vengono eseguiti come root su Ubuntu 24.04, quindi `apt install` e la maggior parte dei gestori di pacchetti di linguaggio funzionano.

> **Tip:**
>
> Per verificare cosa è già installato prima di aggiungerlo al tuo script, chiedi a Claude di eseguire `check-tools` in una sessione cloud.

Per aggiungere uno script di configurazione, apri la finestra di dialogo delle impostazioni dell'ambiente e inserisci il tuo script nel campo **Setup script**.

Questo esempio installa la CLI `gh`, che non è nell'immagine predefinita:

```
#!/bin/bash
apt update && apt install -y gh
```

Gli script di configurazione vengono eseguiti solo quando si crea una nuova sessione. Vengono saltati quando si riprende una sessione esistente.

Se lo script esce con un codice diverso da zero, la sessione non si avvia. Aggiungi `|| true` ai comandi non critici per evitare di bloccare la sessione su un'installazione instabile.

> **Note:**
>
> Gli script di configurazione che installano pacchetti hanno bisogno di accesso alla rete per raggiungere i registri. L'accesso alla rete predefinito consente connessioni a registri di pacchetti comuni inclusi npm, PyPI, RubyGems e crates.io. Gli script non riusciranno a installare pacchetti se il tuo ambiente ha l'accesso alla rete disabilitato.

Script di configurazione vs. hook SessionStart

Utilizza uno script di configurazione per installare cose di cui il cloud ha bisogno ma il tuo laptop ha già, come un runtime di linguaggio o uno strumento CLI. Utilizza un hook SessionStart per la configurazione del progetto che dovrebbe essere eseguita ovunque, cloud e locale, come `npm install`.

Entrambi vengono eseguiti all'inizio di una sessione, ma appartengono a posti diversi:

	Script di configurazione	Hook SessionStart
Allegato a	L'ambiente cloud	Il tuo repository
Configurato in	Interfaccia utente dell'ambiente cloud	`.claude/settings.json` nel tuo repository
Viene eseguito	Prima che Claude Code si avvii, solo su nuove sessioni	Dopo che Claude Code si avvia, su ogni sessione incluse quelle riprese
Ambito	Solo ambienti cloud	Sia locale che cloud

Gli hook SessionStart possono anche essere definiti nel tuo `~/.claude/settings.json` a livello di utente localmente, ma le impostazioni a livello di utente non vengono trasferite alle sessioni cloud. Nel cloud, vengono eseguiti solo gli hook sottoposti a commit nel repository.

Gestione delle dipendenze

Le immagini di ambiente personalizzate e gli snapshot non sono ancora supportati. Utilizza script di configurazione per installare pacchetti quando inizia una sessione, oppure hook SessionStart per l'installazione di dipendenze che dovrebbe essere eseguita anche negli ambienti locali. Gli hook SessionStart hanno limitazioni note.

Per configurare l'installazione automatica delle dipendenze con uno script di configurazione, apri le impostazioni dell'ambiente e aggiungi uno script:

```
#!/bin/bash
npm install
pip install -r requirements.txt
```

In alternativa, puoi utilizzare gli hook SessionStart nel file `.claude/settings.json` del tuo repository per l'installazione di dipendenze che dovrebbe essere eseguita anche negli ambienti locali:

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/scripts/install_pkgs.sh"
          }
        ]
      }
    ]
  }
}
```

Crea lo script corrispondente in `scripts/install_pkgs.sh`:

```
#!/bin/bash

## Only run in remote environments
if [ "$CLAUDE_CODE_REMOTE" != "true" ]; then
  exit 0
fi

npm install
pip install -r requirements.txt
exit 0
```

Rendilo eseguibile: `chmod +x scripts/install_pkgs.sh`

Persistenza delle variabili di ambiente

Gli hook SessionStart possono persistere le variabili di ambiente per i comandi Bash successivi scrivendo nel file specificato nella variabile di ambiente `CLAUDE_ENV_FILE`. Per i dettagli, vedi Hook SessionStart nel riferimento degli hook.

Limitazioni della gestione delle dipendenze

- **Gli hook si attivano per tutte le sessioni**: Gli hook SessionStart vengono eseguiti sia negli ambienti locali che remoti. Non esiste una configurazione di hook per limitare un hook solo alle sessioni remote. Per saltare l'esecuzione locale, controlla la variabile di ambiente `CLAUDE_CODE_REMOTE` nel tuo script come mostrato sopra.
- **Richiede accesso alla rete**: I comandi di installazione hanno bisogno di accesso alla rete per raggiungere i registri di pacchetti. Se il tuo ambiente è configurato con accesso "No internet", questi hook falliranno. Utilizza accesso alla rete "Limited" (predefinito) o "Full". L'elenco di consentiti predefinito include registri comuni come npm, PyPI, RubyGems e crates.io.
- **Compatibilità proxy**: Tutto il traffico in uscita negli ambienti remoti passa attraverso un proxy di sicurezza. Alcuni gestori di pacchetti non funzionano correttamente con questo proxy. Bun è un esempio noto.
- **Viene eseguito ad ogni avvio della sessione**: Gli hook vengono eseguiti ogni volta che una sessione si avvia o riprende, aggiungendo latenza di avvio. Mantieni gli script di installazione veloci controllando se le dipendenze sono già presenti prima di reinstallarle.

Accesso alla rete e sicurezza

Politica di rete

Proxy GitHub

Per motivi di sicurezza, tutte le operazioni GitHub passano attraverso un servizio proxy dedicato che gestisce in modo trasparente tutte le interazioni git. All'interno della sandbox, il client git si autentica utilizzando una credenziale con ambito personalizzato. Questo proxy:

- Gestisce l'autenticazione GitHub in modo sicuro - il client git utilizza una credenziale con ambito all'interno della sandbox, che il proxy verifica e traduce nel tuo token di autenticazione GitHub effettivo
- Limita le operazioni git push al ramo di lavoro attuale per motivi di sicurezza
- Abilita il clonaggio, il recupero e le operazioni PR senza interruzioni mantenendo i confini di sicurezza

Proxy di sicurezza

Gli ambienti vengono eseguiti dietro un proxy di rete HTTP/HTTPS per motivi di sicurezza e prevenzione degli abusi. Tutto il traffico Internet in uscita passa attraverso questo proxy, che fornisce:

- Protezione contro richieste dannose
- Limitazione della velocità e prevenzione degli abusi
- Filtro dei contenuti per una sicurezza migliorata

Livelli di accesso

Per impostazione predefinita, l'accesso alla rete è limitato ai domini consentiti.

Puoi configurare l'accesso alla rete personalizzato, inclusa la disabilitazione dell'accesso alla rete.

Domini consentiti predefiniti

Quando si utilizza l'accesso alla rete "Limited", i seguenti domini sono consentiti per impostazione predefinita:

Servizi Anthropic

- api.anthropic.com
- statsig.anthropic.com
- platform.claude.com
- code.claude.com
- claude.ai

Controllo versione

- 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

Registri di container

- 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

Piattaforme cloud

- 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

Gestori di pacchetti - JavaScript/Node

- registry.npmjs.org
- www.npmjs.com
- www.npmjs.org
- npmjs.com
- npmjs.org
- yarnpkg.com
- registry.yarnpkg.com

Gestori di pacchetti - Python

- pypi.org
- www.pypi.org
- files.pythonhosted.org
- pythonhosted.org
- test.pypi.org
- pypi.python.org

- pypa.io
- www.pypa.io

Gestori di pacchetti - 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

Gestori di pacchetti - Rust

- crates.io
- www.crates.io
- index.crates.io
- static.crates.io
- rustup.rs
- static.rust-lang.org
- www.rust-lang.org

Gestori di pacchetti - Go

- proxy.golang.org
- sum.golang.org
- index.golang.org
- golang.org
- www.golang.org
- goproxy.io
- pkg.go.dev

Gestori di pacchetti - 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

Gestori di pacchetti - Altri linguaggi

- 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

Distribuzioni Linux

- archive.ubuntu.com
- security.ubuntu.com
- ubuntu.com
- www.ubuntu.com
- *.ubuntu.com
- ppa.launchpad.net
- launchpad.net
- www.launchpad.net

Strumenti di sviluppo e piattaforme

- 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

Servizi cloud e monitoraggio

- statsig.com
- www.statsig.com
- api.statsig.com
- sentry.io
- *.sentry.io
- http-intake.logs.datadoghq.com
- *.datadoghq.com
- *.datadoghq.eu

Distribuzione di contenuti e mirror

- sourceforge.net
- *.sourceforge.net
- packagecloud.io
- *.packagecloud.io

Schema e configurazione

- json-schema.org
- www.json-schema.org
- json.schemastore.org
- www.schemastore.org

Model Context Protocol

- *.modelcontextprotocol.io

> **Note:**
>
> I domini contrassegnati con `*` indicano la corrispondenza dei sottodomini con caratteri jolly. Ad esempio, `*.gcr.io` consente l'accesso a qualsiasi sottodominio di `gcr.io`.

Migliori pratiche di sicurezza per l'accesso alla rete personalizzato

1. **Principio del minimo privilegio**: Abilita solo l'accesso alla rete minimo richiesto
2. **Audit regolari**: Rivedi periodicamente i domini consentiti
3. **Usa HTTPS**: Preferisci sempre gli endpoint HTTPS rispetto a HTTP

Sicurezza e isolamento

Claude Code sul web fornisce forti garanzie di sicurezza:

- **Macchine virtuali isolate**: Ogni sessione viene eseguita in una VM isolata gestita da Anthropic
- **Controlli di accesso alla rete**: L'accesso alla rete è limitato per impostazione predefinita e può essere disabilitato

> **Note:**
>
> Quando viene eseguito con l'accesso alla rete disabilitato, Claude Code è autorizzato a comunicare con l'API Anthropic che potrebbe comunque consentire ai dati di uscire dalla VM Claude Code isolata.

- **Protezione delle credenziali**: Le credenziali sensibili (come le credenziali git o le chiavi di firma) non sono mai all'interno della sandbox con Claude Code. L'autenticazione viene gestita tramite un proxy sicuro utilizzando credenziali con ambito
- **Analisi sicura**: Il codice viene analizzato e modificato all'interno di VM isolate prima di creare PR

Prezzi e limiti di velocità

Claude Code sul web condivide i limiti di velocità con tutti gli altri utilizzi di Claude e Claude Code all'interno del tuo account. L'esecuzione di più attività in parallelo consumerà più limiti di velocità proporzionalmente.

Limitazioni

- **Autenticazione del repository**: Puoi spostare sessioni da web a locale solo quando sei autenticato allo stesso account
- **Restrizioni della piattaforma**: Claude Code sul web funziona solo con codice ospitato in GitHub. I repository non GitHub come GitLab non possono essere utilizzati con sessioni cloud

Migliori pratiche

1. **Automatizza la configurazione dell'ambiente**: Utilizza script di configurazione per installare dipendenze e configurare strumenti prima che Claude Code si avvii. Per scenari più avanzati, configura hook SessionStart.
2. **Documenta i requisiti**: Specifica chiaramente le dipendenze e i comandi nel tuo file `CLAUDE.md`. Se hai un file `AGENTS.md`, puoi fornirlo nel tuo `CLAUDE.md` usando `@AGENTS.md` per mantenere un'unica fonte di verità.

Risorse correlate

- Configurazione degli hook
- Riferimento delle impostazioni
- Sicurezza
- Utilizzo dei dati

Part 9: Security & Privacy

Sicurezza

> *Scopri le misure di sicurezza di Claude Code e le migliori pratiche per un utilizzo sicuro.*

Come affrontiamo la sicurezza

Fondamento della sicurezza

La sicurezza del vostro codice è fondamentale. Claude Code è costruito con la sicurezza al centro, sviluppato secondo il programma di sicurezza completo di Anthropic. Scopri di più e accedi alle risorse (rapporto SOC 2 Type 2, certificato ISO 27001, ecc.) presso Anthropic Trust Center.

Architettura basata su permessi

Claude Code utilizza permessi di sola lettura rigorosi per impostazione predefinita. Quando sono necessarie azioni aggiuntive (modifica di file, esecuzione di test, esecuzione di comandi), Claude Code richiede un'autorizzazione esplicita. Gli utenti controllano se approvare le azioni una sola volta o consentirle automaticamente.

Abbiamo progettato Claude Code per essere trasparente e sicuro. Ad esempio, richiediamo l'approvazione per i comandi bash prima di eseguirli, dandovi il controllo diretto. Questo approccio consente agli utenti e alle organizzazioni di configurare i permessi direttamente.

Per la configurazione dettagliata dei permessi, vedere Permissions.

Protezioni integrate

Per mitigare i rischi nei sistemi agentici:

- **Strumento bash in sandbox**: Sandbox comandi bash con isolamento del filesystem e della rete, riducendo i prompt di permesso mantenendo la sicurezza. Abilita con `/sandbox` per definire i confini dove Claude Code può lavorare autonomamente
- **Restrizione dell'accesso in scrittura**: Claude Code può scrivere solo nella cartella in cui è stato avviato e nelle sue sottocartelle, non può modificare file nelle directory padre senza autorizzazione esplicita. Mentre Claude Code può leggere file al di fuori della directory di lavoro (utile per accedere alle librerie di sistema e alle dipendenze),

le operazioni di scrittura sono strettamente limitate all'ambito del progetto, creando un chiaro confine di sicurezza

- **Mitigazione dell'affaticamento da prompt**: Supporto per l'allowlisting di comandi sicuri utilizzati frequentemente per utente, per codebase o per organizzazione
- **Modalità Accept Edits**: Accetta in batch più modifiche mantenendo i prompt di permesso per i comandi con effetti collaterali

Responsabilità dell'utente

Claude Code ha solo i permessi che gli concedete. Siete responsabili della revisione del codice e dei comandi proposti per la sicurezza prima dell'approvazione.

Proteggiti dall'iniezione di prompt

L'iniezione di prompt è una tecnica in cui un attaccante tenta di ignorare o manipolare le istruzioni di un assistente AI inserendo testo dannoso. Claude Code include diversi meccanismi di protezione contro questi attacchi:

Protezioni fondamentali

- **Sistema di permessi**: Le operazioni sensibili richiedono un'approvazione esplicita
- **Analisi consapevole del contesto**: Rileva istruzioni potenzialmente dannose analizzando la richiesta completa
- **Sanitizzazione dell'input**: Previene l'iniezione di comandi elaborando gli input dell'utente
- **Blocklist di comandi**: Blocca i comandi rischiosi che recuperano contenuti arbitrari dal web come `curl` e `wget` per impostazione predefinita. Quando esplicitamente consentiti, siate consapevoli delle limitazioni del modello di permesso

Misure di protezione della privacy

Abbiamo implementato diversi meccanismi di protezione per proteggere i vostri dati, tra cui:

- Periodi di conservazione limitati per le informazioni sensibili (consultare il Privacy Center per ulteriori informazioni)
- Accesso limitato ai dati della sessione utente
- Controllo dell'utente sulle preferenze di addestramento dei dati. Gli utenti consumer possono modificare le loro impostazioni di privacy in qualsiasi momento.

Per i dettagli completi, consultare i nostri Termini di servizio commerciali (per utenti Team, Enterprise e API) o Termini consumer (per utenti Free, Pro e Max) e Informativa sulla privacy.

Misure di protezione aggiuntive

- **Approvazione della richiesta di rete**: Gli strumenti che effettuano richieste di rete richiedono l'approvazione dell'utente per impostazione predefinita
- **Finestre di contesto isolate**: Web fetch utilizza una finestra di contesto separata per evitare di iniettare prompt potenzialmente dannosi
- **Verifica della fiducia**: Le prime esecuzioni di codebase e i nuovi server MCP richiedono la verifica della fiducia
  - Nota: La verifica della fiducia è disabilitata quando si esegue in modo non interattivo con il flag `-p`
- **Rilevamento dell'iniezione di comandi**: I comandi bash sospetti richiedono l'approvazione manuale anche se precedentemente allowlisted
- **Corrispondenza fail-closed**: I comandi non corrispondenti richiedono per impostazione predefinita l'approvazione manuale
- **Descrizioni in linguaggio naturale**: I comandi bash complessi includono spiegazioni per la comprensione dell'utente
- **Archiviazione sicura delle credenziali**: Le chiavi API e i token sono crittografati. Vedere Credential Management

> **Warning:**
>
> **Rischio di sicurezza WebDAV su Windows**: Quando si esegue Claude Code su Windows, consigliamo di non abilitare WebDAV o di non consentire a Claude Code di accedere a percorsi come `\\*` che potrebbero contenere sottodirectory WebDAV. WebDAV è stato deprecato da Microsoft a causa di rischi di sicurezza. L'abilitazione di WebDAV potrebbe consentire a Claude Code di attivare richieste di rete a host remoti, aggirando il sistema di permessi.

Migliori pratiche per lavorare con contenuti non attendibili:

1. Rivedere i comandi suggeriti prima dell'approvazione
2. Evitare di inviare contenuti non attendibili direttamente a Claude tramite pipe
3. Verificare le modifiche proposte ai file critici
4. Utilizzare macchine virtuali (VM) per eseguire script e effettuare chiamate di strumenti, soprattutto quando si interagisce con servizi web esterni
5. Segnalare comportamenti sospetti con `/bug`

> **Warning:**
>
> Sebbene queste protezioni riducano significativamente il rischio, nessun sistema è completamente immune da tutti gli attacchi. Mantenete sempre buone pratiche di sicurezza quando lavorate con qualsiasi strumento AI.

Sicurezza MCP

Claude Code consente agli utenti di configurare server Model Context Protocol (MCP). L'elenco dei server MCP consentiti è configurato nel vostro codice sorgente, come parte delle impostazioni di Claude Code che gli ingegneri controllano nel controllo del codice sorgente.

Incoraggiamo sia la scrittura dei vostri server MCP che l'utilizzo di server MCP da provider di cui vi fidate. Siete in grado di configurare i permessi di Claude Code per i server MCP. Anthropic non gestisce né controlla alcun server MCP.

Sicurezza dell'IDE

Vedere VS Code security and privacy per ulteriori informazioni sull'esecuzione di Claude Code in un IDE.

Sicurezza dell'esecuzione nel cloud

Quando si utilizza Claude Code sul web, sono in vigore controlli di sicurezza aggiuntivi:

- **Macchine virtuali isolate**: Ogni sessione cloud viene eseguita in una VM isolata gestita da Anthropic
- **Controlli di accesso alla rete**: L'accesso alla rete è limitato per impostazione predefinita e può essere configurato per essere disabilitato o consentire solo domini specifici
- **Protezione delle credenziali**: L'autenticazione viene gestita tramite un proxy sicuro che utilizza una credenziale con ambito all'interno della sandbox, che viene quindi tradotta nel vostro token di autenticazione GitHub effettivo
- **Restrizioni di ramo**: Le operazioni di push Git sono limitate al ramo di lavoro corrente
- **Registrazione di audit**: Tutte le operazioni negli ambienti cloud vengono registrate per scopi di conformità e audit
- **Pulizia automatica**: Gli ambienti cloud vengono terminati automaticamente al completamento della sessione

Per ulteriori dettagli sull'esecuzione nel cloud, vedere Claude Code sul web.

Le sessioni di Remote Control funzionano diversamente: l'interfaccia web si connette a un processo Claude Code in esecuzione sulla vostra macchina locale. Tutta l'esecuzione del codice e l'accesso ai file rimangono locali, e gli stessi dati che fluiscono durante qualsiasi sessione locale di Claude Code viaggiano attraverso l'API Anthropic su TLS. Non sono coinvolte VM cloud o sandbox. La connessione utilizza più credenziali di breve durata e con ambito ristretto, ciascuna limitata a uno scopo specifico e con scadenza indipendente, per limitare il raggio di esplosione di qualsiasi singola credenziale compromessa.

Migliori pratiche di sicurezza

Lavorare con codice sensibile

- Rivedere tutte le modifiche suggerite prima dell'approvazione
- Utilizzare impostazioni di permesso specifiche del progetto per repository sensibili
- Considerare l'utilizzo di devcontainers per un isolamento aggiuntivo
- Controllare regolarmente le impostazioni di permesso con `/permissions`

Sicurezza del team

- Utilizzare managed settings per applicare gli standard organizzativi
- Condividere le configurazioni di permesso approvate tramite il controllo del codice sorgente
- Formare i membri del team sulle migliori pratiche di sicurezza
- Monitorare l'utilizzo di Claude Code tramite metriche OpenTelemetry
- Controllare o bloccare le modifiche alle impostazioni durante le sessioni con `ConfigChange` hooks

Segnalazione di problemi di sicurezza

Se scoprite una vulnerabilità di sicurezza in Claude Code:

1. Non divulgatela pubblicamente
2. Segnalatela tramite il nostro programma HackerOne
3. Includete i passaggi di riproduzione dettagliati
4. Concedete il tempo necessario per affrontare il problema prima della divulgazione pubblica

Risorse correlate

- Sandboxing - Isolamento del filesystem e della rete per i comandi bash

- Permissions - Configurare i permessi e i controlli di accesso
- Monitoring usage - Tracciare e controllare l'attività di Claude Code
- Development containers - Ambienti sicuri e isolati
- Anthropic Trust Center - Certificazioni di sicurezza e conformità

Sandboxing

> *Scopri come lo strumento bash in sandbox di Claude Code fornisce isolamento del filesystem e della rete per un'esecuzione dell'agente più sicura e autonoma.*

Panoramica

Claude Code include il sandboxing nativo per fornire un ambiente più sicuro per l'esecuzione dell'agente, riducendo la necessità di prompt di autorizzazione costanti. Invece di chiedere autorizzazione per ogni comando bash, il sandboxing crea confini definiti in anticipo dove Claude Code può lavorare più liberamente con rischio ridotto.

Lo strumento bash in sandbox utilizza primitive a livello del sistema operativo per applicare sia l'isolamento del filesystem che della rete.

Perché il sandboxing è importante

La sicurezza tradizionale basata su autorizzazioni richiede l'approvazione costante dell'utente per i comandi bash. Sebbene questo fornisca controllo, può portare a:

- **Affaticamento da approvazione**: Fare clic ripetutamente su "approva" può causare agli utenti di prestare meno attenzione a ciò che stanno approvando
- **Produttività ridotta**: Le interruzioni costanti rallentano i flussi di lavoro di sviluppo
- **Autonomia limitata**: Claude Code non può lavorare in modo efficiente quando è in attesa di approvazioni

Il sandboxing affronta queste sfide:

1. **Definendo confini chiari**: Specifica esattamente quali directory e host di rete Claude Code può accedere
2. **Riducendo i prompt di autorizzazione**: I comandi sicuri all'interno della sandbox non richiedono approvazione
3. **Mantenendo la sicurezza**: I tentativi di accedere a risorse al di fuori della sandbox attivano notifiche immediate
4. **Abilitando l'autonomia**: Claude Code può funzionare più indipendentemente entro limiti definiti

> **Warning:**

> Il sandboxing efficace richiede **sia** l'isolamento del filesystem che della rete. Senza isolamento della rete, un agente compromesso potrebbe esfiltare file sensibili come chiavi SSH. Senza isolamento del filesystem, un agente compromesso potrebbe backdoor le risorse di sistema per ottenere accesso alla rete. Quando si configura il sandboxing è importante assicurarsi che le impostazioni configurate non creino bypass in questi sistemi.

Come funziona

Isolamento del filesystem

Lo strumento bash in sandbox limita l'accesso al file system a directory specifiche:

- **Comportamento di scrittura predefinito**: Accesso in lettura e scrittura alla directory di lavoro corrente e alle sue sottodirectory
- **Comportamento di lettura predefinito**: Accesso in lettura all'intero computer, ad eccezione di determinate directory negate
- **Accesso bloccato**: Non è possibile modificare file al di fuori della directory di lavoro corrente senza autorizzazione esplicita
- **Configurabile**: Definisci percorsi consentiti e negati personalizzati tramite le impostazioni

È possibile concedere l'accesso in scrittura a percorsi aggiuntivi utilizzando `sandbox.filesystem.allowWrite` nelle impostazioni. Queste restrizioni sono applicate a livello del sistema operativo (Seatbelt su macOS, bubblewrap su Linux), quindi si applicano a tutti i comandi dei sottoprocessi, inclusi strumenti come `kubectl`, `terraform` e `npm`, non solo agli strumenti di file di Claude.

Isolamento della rete

L'accesso alla rete è controllato tramite un server proxy in esecuzione al di fuori della sandbox:

- **Restrizioni di dominio**: Solo i domini approvati possono essere accessibili
- **Conferma dell'utente**: Le nuove richieste di dominio attivano prompt di autorizzazione (a meno che `allowManagedDomainsOnly` non sia abilitato, che blocca automaticamente i domini non consentiti)
- **Supporto proxy personalizzato**: Gli utenti avanzati possono implementare regole personalizzate sul traffico in uscita
- **Copertura completa**: Le restrizioni si applicano a tutti gli script, programmi e sottoprocessi generati dai comandi

Applicazione a livello del sistema operativo

Lo strumento bash in sandbox sfrutta le primitive di sicurezza del sistema operativo:

- **macOS**: Utilizza Seatbelt per l'applicazione della sandbox
- **Linux**: Utilizza bubblewrap per l'isolamento
- **WSL2**: Utilizza bubblewrap, come Linux

WSL1 non è supportato perché bubblewrap richiede funzionalità del kernel disponibili solo in WSL2.

Queste restrizioni a livello del sistema operativo assicurano che tutti i processi figlio generati dai comandi di Claude Code ereditino gli stessi confini di sicurezza.

Iniziare

Prerequisiti

Su **macOS**, il sandboxing funziona subito utilizzando il framework Seatbelt integrato.

Su **Linux e WSL2**, installa prima i pacchetti richiesti:

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

Abilita il sandboxing

È possibile abilitare il sandboxing eseguendo il comando `/sandbox`:

```
/sandbox
```

Questo apre un menu in cui è possibile scegliere tra le modalità sandbox. Se le dipendenze richieste sono mancanti (come `bubblewrap` o `socat` su Linux), il menu visualizza le istruzioni di installazione per la piattaforma.

Modalità sandbox

Claude Code offre due modalità sandbox:

Modalità auto-allow: I comandi Bash tenteranno di eseguire all'interno della sandbox e sono automaticamente consentiti senza richiedere autorizzazione. I comandi che non possono essere sandboxati (come quelli che necessitano di accesso alla rete a host non consentiti) ricadono nel flusso di autorizzazione regolare. Le regole di chiesta/negazione esplicite che hai configurato sono sempre rispettate.

Modalità autorizzazioni regolari: Tutti i comandi bash passano attraverso il flusso di autorizzazione standard, anche quando sandboxati. Questo fornisce più controllo ma richiede più approvazioni.

In entrambe le modalità, la sandbox applica le stesse restrizioni di filesystem e rete. La differenza è solo se i comandi sandboxati sono auto-approvati o richiedono autorizzazione esplicita.

> **Info:**
>
> La modalità auto-allow funziona indipendentemente dall'impostazione della modalità di autorizzazione. Anche se non sei in modalità "accetta modifiche", i comandi bash sandboxati verranno eseguiti automaticamente quando auto-allow è abilitato. Ciò significa che i comandi bash che modificano file entro i confini della sandbox verranno eseguiti senza richiedere, anche quando gli strumenti di modifica dei file normalmente richiederebbero approvazione.

Configura il sandboxing

Personalizza il comportamento della sandbox tramite il file `settings.json`. Vedi Settings per il riferimento di configurazione completo.

Concessione dell'accesso in scrittura dei sottoprocessi a percorsi specifici

Per impostazione predefinita, i comandi sandboxati possono scrivere solo nella directory di lavoro corrente. Se i comandi dei sottoprocessi come `kubectl`, `terraform` o `npm` devono scrivere al di fuori della directory del progetto, utilizza `sandbox.filesystem.allowWrite` per concedere l'accesso a percorsi specifici:

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "allowWrite": ["~/.kube", "//tmp/build"]
    }
  }
}
```

Questi percorsi sono applicati a livello del sistema operativo, quindi tutti i comandi in esecuzione all'interno della sandbox, inclusi i loro processi figlio, li rispettano. Questo è l'approccio consigliato quando uno strumento ha bisogno di accesso in scrittura a una posizione specifica, piuttosto che escludere completamente lo strumento dalla sandbox con `excludedCommands`.

Quando `allowWrite` (o `denyWrite` / `denyRead`) è definito in più ambiti di impostazioni, gli array sono **uniti**, il che significa che i percorsi da ogni ambito sono combinati, non sostituiti. Ad esempio, se le impostazioni gestite consentono scritture in `//opt/company-tools` e un utente aggiunge `~/.kube` nelle sue impostazioni personali, entrambi i percorsi sono inclusi nella configurazione finale della sandbox. Ciò significa che gli utenti e i progetti possono estendere l'elenco senza duplicare o sovrascrivere i percorsi impostati da ambiti con priorità più alta.

I prefissi di percorso controllano come i percorsi vengono risolti:

Prefisso	Significato	Esempio
`//`	Percorso assoluto dalla radice del filesystem	`//tmp/build` diventa `/tmp/build`
`~/`	Relativo alla directory home	`~/.kube` diventa `$HOME/.kube`
`/`	Relativo alla directory del file di impostazioni	`/build` diventa `$SETTINGS_DIR/build`
`./` o nessun prefisso	Percorso relativo (risolto dal runtime della sandbox)	`./output`

È inoltre possibile negare l'accesso in scrittura o lettura utilizzando `sandbox.filesystem.denyWrite` e `sandbox.filesystem.denyRead`. Questi vengono uniti con qualsiasi percorso dalle regole di autorizzazione `Edit(...)` e `Read(...)`.

Tip:

Non tutti i comandi sono compatibili con il sandboxing subito. Alcuni appunti che potrebbero aiutarti a ottenere il massimo dalla sandbox:

- Molti strumenti CLI richiedono l'accesso a determinati host. Man mano che utilizzi questi strumenti, richiederanno l'autorizzazione per accedere a determinati host. Concedere l'autorizzazione consentirà loro di accedere a questi host ora e in futuro, consentendo loro di eseguire in modo sicuro all'interno della sandbox.
- `watchman` è incompatibile con l'esecuzione nella sandbox. Se stai eseguendo `jest`, considera di utilizzare `jest --no-watchman`
- `docker` è incompatibile con l'esecuzione nella sandbox. Considera di specificare `docker` in `excludedCommands` per forzarlo a eseguire al di fuori della sandbox.

Note:

Claude Code include un meccanismo di escape hatch intenzionale che consente ai comandi di eseguire al di fuori della sandbox quando necessario. Quando un comando non riesce a causa di restrizioni della sandbox (come problemi di connettività di rete o strumenti incompatibili), Claude viene richiesto di analizzare l'errore e potrebbe riprovare il comando con il parametro `dangerouslyDisableSandbox`. I comandi che utilizzano questo parametro passano attraverso il flusso di autorizzazioni normale di Claude Code richiedendo l'autorizzazione dell'utente per l'esecuzione. Ciò consente a Claude Code di gestire i casi limite in cui determinati strumenti o operazioni di rete non possono funzionare entro i vincoli della sandbox.

È possibile disabilitare questo escape hatch impostando `"allowUnsandboxedCommands": false` nelle impostazioni della sandbox. Quando disabilitato, il parametro `dangerouslyDisableSandbox` viene completamente ignorato e tutti i comandi devono essere eseguiti sandboxati o essere esplicitamente elencati in `excludedCommands`.

Vantaggi della sicurezza

Protezione contro l'iniezione di prompt

Anche se un attaccante manipola con successo il comportamento di Claude Code attraverso l'iniezione di prompt, la sandbox assicura che il sistema rimanga sicuro:

Protezione del filesystem:

- Non è possibile modificare file di configurazione critici come `~/.bashrc`
- Non è possibile modificare file a livello di sistema in `/bin/`
- Non è possibile leggere file che sono negati nelle impostazioni di autorizzazione di Claude

Protezione della rete:

- Non è possibile esfiltare dati a server controllati dall'attaccante
- Non è possibile scaricare script dannosi da domini non autorizzati
- Non è possibile effettuare chiamate API inaspettate a servizi non approvati
- Non è possibile contattare alcun dominio non esplicitamente consentito

Monitoraggio e controllo:

- Tutti i tentativi di accesso al di fuori della sandbox sono bloccati a livello del sistema operativo
- Ricevi notifiche immediate quando i confini vengono testati
- È possibile scegliere di negare, consentire una volta o aggiornare permanentemente la configurazione

Superficie di attacco ridotta

Il sandboxing limita il danno potenziale da:

- **Dipendenze dannose**: Pacchetti NPM o altre dipendenze con codice dannoso
- **Script compromessi**: Script di compilazione o strumenti con vulnerabilità di sicurezza
- **Ingegneria sociale**: Attacchi che ingannano gli utenti nel far eseguire comandi pericolosi
- **Iniezione di prompt**: Attacchi che ingannano Claude nel far eseguire comandi pericolosi

Funzionamento trasparente

Quando Claude Code tenta di accedere a risorse di rete al di fuori della sandbox:

1. L'operazione viene bloccata a livello del sistema operativo
2. Ricevi una notifica immediata
3. È possibile scegliere di:
    - Negare la richiesta
    - Consentirla una volta
    - Aggiornare la configurazione della sandbox per consentirla permanentemente

Limitazioni della sicurezza

- Limitazioni del sandboxing della rete: Il sistema di filtraggio della rete funziona limitando i domini a cui i processi possono connettersi. Non ispeziona altrimenti il

traffico che passa attraverso il proxy e gli utenti sono responsabili di assicurarsi che consentano solo domini affidabili nella loro politica.

> **Warning:**
>
> Gli utenti dovrebbero essere consapevoli dei potenziali rischi derivanti dal consentire domini ampi come `github.com` che potrebbero consentire l'esfiltrazione di dati. Inoltre, in alcuni casi potrebbe essere possibile aggirare il filtraggio della rete attraverso il domain fronting.

- Escalation dei privilegi tramite Unix Sockets: La configurazione `allowUnixSockets` può inavvertitamente concedere l'accesso a potenti servizi di sistema che potrebbero portare a bypass della sandbox. Ad esempio, se viene utilizzato per consentire l'accesso a `/var/run/docker.sock` questo concederebbe effettivamente l'accesso al sistema host sfruttando il socket docker. Gli utenti sono incoraggiati a considerare attentamente qualsiasi socket unix che consentono attraverso la sandbox.
- Escalation dei permessi del filesystem: I permessi di scrittura del filesystem eccessivamente ampi possono abilitare attacchi di escalation dei privilegi. Consentire scritture a directory contenenti eseguibili in `$PATH`, directory di configurazione di sistema o file di configurazione della shell dell'utente (`.bashrc`, `.zshrc`) può portare all'esecuzione di codice in diversi contesti di sicurezza quando altri utenti o processi di sistema accedono a questi file.
- Forza della sandbox Linux: L'implementazione Linux fornisce un forte isolamento del filesystem e della rete ma include una modalità `enableWeakerNestedSandbox` che le consente di funzionare all'interno di ambienti Docker senza namespace privilegiati. Questa opzione indebolisce considerevolmente la sicurezza e dovrebbe essere utilizzata solo nei casi in cui l'isolamento aggiuntivo è altrimenti applicato.

Come il sandboxing si relaziona alle autorizzazioni

Il sandboxing e le autorizzazioni sono livelli di sicurezza complementari che funzionano insieme:

- **Autorizzazioni** controllano quali strumenti Claude Code può utilizzare e vengono valutate prima che qualsiasi strumento venga eseguito. Si applicano a tutti gli strumenti: Bash, Read, Edit, WebFetch, MCP e altri.
- **Sandboxing** fornisce l'applicazione a livello del sistema operativo che limita ciò che i comandi Bash possono accedere a livello di filesystem e rete. Si applica solo ai comandi Bash e ai loro processi figlio.

Le restrizioni di filesystem e rete sono configurate sia tramite le impostazioni della sandbox che le regole di autorizzazione:

- Utilizza `sandbox.filesystem.allowWrite` per concedere l'accesso in scrittura dei sottoprocessi a percorsi al di fuori della directory di lavoro
- Utilizza `sandbox.filesystem.denyWrite` e `sandbox.filesystem.denyRead` per bloccare l'accesso dei sottoprocessi a percorsi specifici
- Utilizza le regole di negazione `Read` e `Edit` per bloccare l'accesso a file o directory specifici
- Utilizza le regole di consentimento/negazione `WebFetch` per controllare l'accesso al dominio
- Utilizza la sandbox `allowedDomains` per controllare quali domini i comandi Bash possono raggiungere

I percorsi dalle impostazioni `sandbox.filesystem` e dalle regole di autorizzazione vengono uniti insieme nella configurazione finale della sandbox.

Questo repository include configurazioni di impostazioni iniziali per scenari di distribuzione comuni, inclusi esempi specifici della sandbox. Utilizzali come punti di partenza e adattali alle tue esigenze.

Utilizzo avanzato

Configurazione proxy personalizzata

Per le organizzazioni che richiedono una sicurezza di rete avanzata, è possibile implementare un proxy personalizzato per:

- Decrittare e ispezionare il traffico HTTPS
- Applicare regole di filtraggio personalizzate
- Registrare tutte le richieste di rete
- Integrarsi con l'infrastruttura di sicurezza esistente

```
{
  "sandbox": {
    "network": {
      "httpProxyPort": 8080,
      "socksProxyPort": 8081
    }
  }
}
```

Integrazione con gli strumenti di sicurezza esistenti

Lo strumento bash in sandbox funziona insieme a:

- **Regole di autorizzazione**: Combina con impostazioni di autorizzazione per la difesa in profondità
- **Contenitori di sviluppo**: Utilizza con devcontainers per un isolamento aggiuntivo
- **Politiche aziendali**: Applica le configurazioni della sandbox tramite impostazioni gestite

Best practice

1. **Inizia restrittivo**: Inizia con autorizzazioni minime e espandi secondo le necessità
2. **Monitora i log**: Rivedi i tentativi di violazione della sandbox per comprendere le esigenze di Claude Code
3. **Utilizza configurazioni specifiche dell'ambiente**: Diverse regole sandbox per contesti di sviluppo rispetto a produzione
4. **Combina con autorizzazioni**: Utilizza il sandboxing insieme alle politiche IAM per una sicurezza completa
5. **Testa le configurazioni**: Verifica che le impostazioni della sandbox non blocchino i flussi di lavoro legittimi

Open source

Il runtime della sandbox è disponibile come pacchetto npm open source per l'uso nei tuoi progetti di agente. Ciò consente alla comunità più ampia degli agenti AI di costruire sistemi autonomi più sicuri. Questo può anche essere utilizzato per sandboxare altri programmi che potresti desiderare di eseguire. Ad esempio, per sandboxare un server MCP potresti eseguire:

```
npx @anthropic-ai/sandbox-runtime <command-to-sandbox>
```

Per i dettagli di implementazione e il codice sorgente, visita il repository GitHub.

Limitazioni

- **Overhead di prestazioni**: Minimo, ma alcune operazioni del filesystem potrebbero essere leggermente più lente
- **Compatibilità**: Alcuni strumenti che richiedono modelli di accesso al sistema specifici potrebbero necessitare di regolazioni di configurazione, o potrebbero anche dover essere eseguiti al di fuori della sandbox
- **Supporto della piattaforma**: Supporta macOS, Linux e WSL2. WSL1 non è supportato. Il supporto nativo di Windows è pianificato.

Vedi anche

- Security - Funzionalità di sicurezza complete e best practice
- Permissions - Configurazione delle autorizzazioni e controllo dell'accesso
- Settings - Riferimento di configurazione completo
- CLI reference - Opzioni della riga di comando

checkpoint

> *Traccia automaticamente e riavvolgi gli edit di Claude per recuperare rapidamente dai cambiamenti indesiderati.*

Claude Code traccia automaticamente gli edit dei file di Claude mentre lavori, permettendoti di annullare rapidamente i cambiamenti e riavvolgere a stati precedenti se qualcosa va fuori strada.

Come funziona il checkpoint

Mentre lavori con Claude, il checkpointing cattura automaticamente lo stato del tuo codice prima di ogni edit. Questa rete di sicurezza ti permette di perseguire compiti ambiziosi e su larga scala sapendo che puoi sempre tornare a uno stato di codice precedente.

Tracciamento automatico

Claude Code traccia tutti i cambiamenti effettuati dai suoi strumenti di editing dei file:

- Ogni prompt dell'utente crea un nuovo checkpoint
- I checkpoint persistono tra le sessioni, quindi puoi accedervi nelle conversazioni riprese
- Puliti automaticamente insieme alle sessioni dopo 30 giorni (configurabile)

Riavvolgimento dei cambiamenti

Premi `Esc` due volte (`Esc` + `Esc`) o usa il comando `/rewind` per aprire il menu di riavvolgimento. Puoi scegliere di ripristinare:

- **Solo conversazione**: Riavvolgi a un messaggio dell'utente mantenendo i cambiamenti del codice
- **Solo codice**: Ripristina i cambiamenti dei file mantenendo la conversazione
- **Sia codice che conversazione**: Ripristina entrambi a un punto precedente nella sessione

Casi d'uso comuni

I checkpoint sono particolarmente utili quando:

- **Esplorare alternative**: Prova diversi approcci di implementazione senza perdere il tuo punto di partenza
- **Recuperare da errori**: Annulla rapidamente i cambiamenti che hanno introdotto bug o rotto la funzionalità
- **Iterare sulle funzionalità**: Sperimenta variazioni sapendo che puoi tornare a stati funzionanti

Limitazioni

I cambiamenti dei comandi Bash non vengono tracciati

Il checkpointing non traccia i file modificati dai comandi bash. Ad esempio, se Claude Code esegue:

```
rm file.txt
mv old.txt new.txt
cp source.txt dest.txt
```

Queste modifiche ai file non possono essere annullate tramite riavvolgimento. Solo gli edit diretti dei file effettuati attraverso gli strumenti di editing dei file di Claude vengono tracciati.

I cambiamenti esterni non vengono tracciati

Il checkpointing traccia solo i file che sono stati modificati nella sessione corrente. I cambiamenti manuali che effettui ai file al di fuori di Claude Code e gli edit da altre sessioni concorrenti normalmente non vengono acquisiti, a meno che non modifichino gli stessi file della sessione corrente.

Non è un sostituto del controllo di versione

I checkpoint sono progettati per il recupero rapido a livello di sessione. Per la cronologia permanente della versione e la collaborazione:

- Continua a utilizzare il controllo di versione (es. Git) per commit, branch e cronologia a lungo termine
- I checkpoint completano ma non sostituiscono il controllo di versione appropriato
- Pensa ai checkpoint come "annulla locale" e Git come "cronologia permanente"

Vedi anche

- Modalità interattiva - Scorciatoie da tastiera e controlli della sessione
- Comandi integrati - Accesso ai checkpoint usando `/rewind`
- Riferimento CLI - Opzioni della riga di comando

Utilizzo dei dati

> *Scopri le politiche di utilizzo dei dati di Anthropic per Claude*

Politiche sui dati

Politica di addestramento dei dati

Utenti consumer (piani Free, Pro e Max): Vi diamo la possibilità di consentire l'utilizzo dei vostri dati per migliorare i futuri modelli Claude. Addestreremo nuovi modelli utilizzando i dati degli account Free, Pro e Max quando questa impostazione è attiva (incluso quando utilizzate Claude Code da questi account).

Utenti commerciali: (piani Team ed Enterprise, API, piattaforme di terze parti e Claude Gov) mantengono le politiche esistenti: Anthropic non addestra modelli generativi utilizzando codice o prompt inviati a Claude Code secondo i termini commerciali, a meno che il cliente non abbia scelto di fornirci i propri dati per il miglioramento del modello (ad esempio, il Development Partner Program).

Development Partner Program

Se vi iscrivete esplicitamente a metodi per fornirci materiali su cui addestrare, come tramite il Development Partner Program, potremmo utilizzare tali materiali forniti per addestrare i nostri modelli. Un amministratore dell'organizzazione può iscriversi esplicitamente al Development Partner Program per la propria organizzazione. Si noti che questo programma è disponibile solo per l'API di Anthropic di prima parte e non per gli utenti di Bedrock o Vertex.

Feedback utilizzando il comando `/bug`

Se scegliete di inviarci feedback su Claude Code utilizzando il comando `/bug`, potremmo utilizzare il vostro feedback per migliorare i nostri prodotti e servizi. I transcript condivisi tramite `/bug` vengono conservati per 5 anni.

Sondaggi sulla qualità della sessione

Quando vedete il prompt "Come sta andando Claude in questa sessione?" in Claude Code, rispondendo a questo sondaggio (inclusa la selezione di "Ignora"), viene registrato solo il vostro voto numerico (1, 2, 3 o ignora). Non raccogliamo né archiviamo alcun transcript di conversazione, input, output o altri dati di sessione come parte di questo sondaggio. A

differenza del feedback con pollice su/giù o dei report `/bug`, questo sondaggio sulla qualità della sessione è una semplice metrica di soddisfazione del prodotto. Le vostre risposte a questo sondaggio non influiscono sulle vostre preferenze di addestramento dei dati e non possono essere utilizzate per addestrare i nostri modelli di IA.

Per disabilitare questi sondaggi, impostate `CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1`. Il sondaggio viene anche disabilitato automaticamente quando si utilizzano provider di terze parti (Bedrock, Vertex, Foundry) o quando la telemetria è disabilitata.

Conservazione dei dati

Anthropic conserva i dati di Claude Code in base al tipo di account e alle preferenze dell'utente.

Utenti consumer (piani Free, Pro e Max):

- Utenti che consentono l'utilizzo dei dati per il miglioramento del modello: periodo di conservazione di 5 anni per supportare lo sviluppo del modello e i miglioramenti della sicurezza
- Utenti che non consentono l'utilizzo dei dati per il miglioramento del modello: periodo di conservazione di 30 giorni
- Le impostazioni sulla privacy possono essere modificate in qualsiasi momento su claude.ai/settings/data-privacy-controls.

Utenti commerciali (Team, Enterprise e API):

- Standard: periodo di conservazione di 30 giorni
- Zero data retention: disponibile per Claude Code su Claude for Enterprise. ZDR è abilitato su base per organizzazione; ogni nuova organizzazione deve avere ZDR abilitato separatamente dal vostro team di account
- Caching locale: i client di Claude Code possono archiviare le sessioni localmente fino a 30 giorni per abilitare la ripresa della sessione (configurabile)

Potete eliminare le singole sessioni di Claude Code sul web in qualsiasi momento. L'eliminazione di una sessione rimuove permanentemente i dati dell'evento della sessione. Per istruzioni su come eliminare le sessioni, consultate Gestione delle sessioni.

Scopri di più sulle pratiche di conservazione dei dati nel nostro Privacy Center.

Per i dettagli completi, consultate i nostri Termini di servizio commerciali (per gli utenti Team, Enterprise e API) o Termini consumer (per gli utenti Free, Pro e Max) e Informativa sulla privacy.

Accesso ai dati

Per tutti gli utenti di prima parte, potete scoprire di più su quali dati vengono registrati per Claude Code locale e Claude Code remoto. Le sessioni di Remote Control seguono il flusso di dati locale poiché tutta l'esecuzione avviene sulla vostra macchina. Si noti che per Claude Code remoto, Claude accede al repository in cui avviate la vostra sessione di Claude Code. Claude non accede ai repository che avete collegato ma in cui non avete avviato una sessione.

Claude Code locale: flusso di dati e dipendenze

Il diagramma sottostante mostra come Claude Code si connette ai servizi esterni durante l'installazione e il funzionamento normale. Le linee continue indicano connessioni richieste, mentre le linee tratteggiate rappresentano flussi di dati facoltativi o avviati dall'utente.

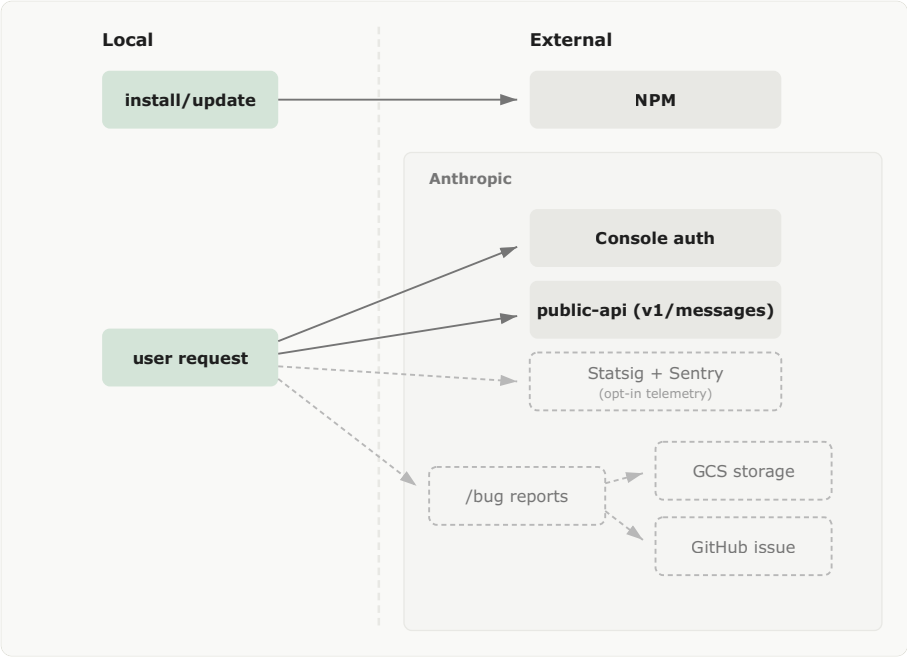


Diagramma che mostra le connessioni esterne di Claude Code: install/update si connette a NPM e le richieste dell'utente si connettono ai servizi Anthropic inclusi Console auth, public-api e facoltativamente Statsig, Sentry e bug reporting

Claude Code viene installato da NPM. Claude Code viene eseguito localmente. Per interagire con l'LLM, Claude Code invia dati sulla rete. Questi dati includono tutti i prompt dell'utente e gli output del modello. I dati vengono crittografati in transito tramite TLS e non vengono crittografati a riposo. Claude Code è compatibile con la maggior parte dei VPN e dei proxy LLM più diffusi.

Claude Code è costruito sulle API di Anthropic. Per i dettagli sui controlli di sicurezza della nostra API, incluse le nostre procedure di registrazione dell'API, consultate gli artefatti di conformità offerti nel Anthropic Trust Center.

Esecuzione nel cloud: flusso di dati e dipendenze

Quando si utilizza Claude Code sul web, le sessioni vengono eseguite in macchine virtuali gestite da Anthropic invece che localmente. Negli ambienti cloud:

- **Archiviazione di codice e dati:** Il vostro repository viene clonato su una VM isolata. Il codice e i dati della sessione sono soggetti alle politiche di conservazione e utilizzo per il vostro tipo di account (consultate la sezione Conservazione dei dati sopra)
- **Credenziali:** L'autenticazione GitHub viene gestita tramite un proxy sicuro; le vostre credenziali GitHub non entrano mai nella sandbox
- **Traffico di rete:** Tutto il traffico in uscita passa attraverso un proxy di sicurezza per la registrazione di audit e la prevenzione degli abusi
- **Dati della sessione:** I prompt, le modifiche al codice e gli output seguono le stesse politiche sui dati dell'utilizzo locale di Claude Code

Per i dettagli sulla sicurezza dell'esecuzione nel cloud, consultate Sicurezza.

Servizi di telemetria

Claude Code si connette dalle macchine degli utenti al servizio Statsig per registrare metriche operative come latenza, affidabilità e modelli di utilizzo. Questa registrazione non include alcun codice o percorso di file. I dati vengono crittografati in transito utilizzando TLS e a riposo utilizzando la crittografia AES a 256 bit. Scopri di più nella documentazione sulla sicurezza di Statsig. Per rinunciare alla telemetria di Statsig, impostate la variabile di ambiente `DISABLE_TELEMETRY`.

Claude Code si connette dalle macchine degli utenti a Sentry per la registrazione degli errori operativi. I dati vengono crittografati in transito utilizzando TLS e a riposo utilizzando la crittografia AES a 256 bit. Scopri di più nella documentazione sulla sicurezza di Sentry. Per rinunciare alla registrazione degli errori, impostate la variabile di ambiente `DISABLE_ERROR_REPORTING`.

Quando gli utenti eseguono il comando `/bug`, una copia della loro cronologia completa della conversazione incluso il codice viene inviata ad Anthropic. I dati vengono crittografati in transito e a riposo. Facoltativamente, viene creato un problema Github nel nostro repository pubblico. Per rinunciare alla segnalazione di bug, impostate la variabile di ambiente `DISABLE_BUG_COMMAND`.

Comportamenti predefiniti per provider API

Per impostazione predefinita, disabilitiamo tutto il traffico non essenziale (inclusa la segnalazione degli errori, la telemetria, la funzionalità di segnalazione dei bug e i sondaggi sulla qualità della sessione) quando si utilizza Bedrock, Vertex o Foundry. Potete anche rinunciare a tutti questi contemporaneamente impostando la variabile di ambiente `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC`. Ecco i comportamenti predefiniti completi:

Servizio	Claude API	Vertex API	Bedrock API	Foundry API
Statsig (Metriche)	Attivo per impostazione predefinita. `DISABLE_TELEMETRY=1` per disabilitare.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_VERTEX` deve essere 1.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_BEDROCK` deve essere 1.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_FOUNDRY` deve essere 1.
Sentry (Errori)	Attivo per impostazione predefinita. `DISABLE_ERROR_REPORTING=1` per disabilitare.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_VERTEX` deve essere 1.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_BEDROCK` deve essere 1.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_FOUNDRY` deve essere 1.
Claude API (report `/bug`)	Attivo per impostazione predefinita. `DISABLE_BUG_COMMAND=1` per disabilitare.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_VERTEX` deve essere 1.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_BEDROCK` deve essere 1.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_FOUNDRY` deve essere 1.

Servizio	Claude API	Vertex API	Bedrock API	Foundry API
Sondaggi sulla qualità della sessione	Attivo per impostazione predefinita. `CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1` per disabilitare.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_VERTEX` deve essere 1.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_BEDROCK` deve essere 1.	Disattivo per impostazione predefinita. `CLAUDE_CODE_USE_FOUNDRY` deve essere 1.

Tutte le variabili di ambiente possono essere controllate in `settings.json` (scopri di più).

Aspetti legali e conformità

> *Accordi legali, certificazioni di conformità e informazioni sulla sicurezza per Claude Code.*

Accordi legali

Licenza

L'uso di Claude Code è soggetto a:

- Termini commerciali - per gli utenti di Team, Enterprise e Claude API
- Termini per i consumatori - per gli utenti di Free, Pro e Max

Accordi commerciali

Che tu stia utilizzando l'API Claude direttamente (1P) o accedendovi tramite AWS Bedrock o Google Vertex (3P), il tuo accordo commerciale esistente si applicherà all'utilizzo di Claude Code, a meno che non abbiamo concordato diversamente.

Conformità

Conformità sanitaria (BAA)

Se un cliente ha un Business Associate Agreement (BAA) con noi e desidera utilizzare Claude Code, il BAA si estenderà automaticamente per coprire Claude Code se il cliente ha eseguito un BAA e ha attivato Zero Data Retention (ZDR). Il BAA sarà applicabile al traffico API di quel cliente che fluisce attraverso Claude Code.

Sicurezza e fiducia

Fiducia e sicurezza

Puoi trovare ulteriori informazioni nel Centro di fiducia di Anthropic e nell'Hub di trasparenza.

Segnalazione di vulnerabilità di sicurezza

Anthropic gestisce il nostro programma di sicurezza tramite HackerOne. Utilizza questo modulo per segnalare vulnerabilità.

Zero data retention

> *Scopri Zero Data Retention (ZDR) per Claude Code su Claude for Enterprise, inclusi ambito, funzionalità disabilitate e come richiedere l'abilitazione.*

Zero Data Retention (ZDR) è disponibile per Claude Code quando utilizzato tramite Claude for Enterprise. Quando ZDR è abilitato, i prompt e le risposte del modello generate durante le sessioni di Claude Code vengono elaborate in tempo reale e non vengono archiviate da Anthropic dopo la restituzione della risposta, tranne dove necessario per conformarsi alla legge o combattere l'uso improprio.

ZDR su Claude for Enterprise offre ai clienti enterprise la possibilità di utilizzare Claude Code con zero data retention e accedere alle funzionalità amministrative:

- Controlli dei costi per utente
- Dashboard Analytics
- Server-managed settings
- Audit log

ZDR per Claude Code su Claude for Enterprise si applica solo alla piattaforma diretta di Anthropic. Per i deployment di Claude su AWS Bedrock, Google Vertex AI o Microsoft Foundry, fare riferimento alle politiche di data retention di quelle piattaforme.

Ambito di ZDR

ZDR copre l'inferenza di Claude Code su Claude for Enterprise.

> **Warning:**
>
> ZDR è abilitato su base per-organizzazione. Ogni nuova organizzazione richiede che ZDR sia abilitato separatamente dal team dell'account Anthropic. ZDR non si applica automaticamente alle nuove organizzazioni create nello stesso account. Contattare il team dell'account per abilitare ZDR per qualsiasi nuova organizzazione.

Cosa copre ZDR

ZDR copre le chiamate di inferenza del modello effettuate tramite Claude Code su Claude for Enterprise. Quando si utilizza Claude Code nel terminale, i prompt inviati e le risposte generate da Claude non vengono conservate da Anthropic. Questo si applica indipendentemente dal modello Claude utilizzato.

Cosa non copre ZDR

ZDR non si estende ai seguenti elementi, anche per le organizzazioni con ZDR abilitato. Queste funzionalità seguono le politiche standard di data retention:

Funzionalità	Dettagli
Chat su claude.ai	Le conversazioni di chat tramite l'interfaccia web Claude for Enterprise non sono coperte da ZDR.
Cowork	Le sessioni Cowork non sono coperte da ZDR.
Claude Code Analytics	Non archivia prompt o risposte del modello, ma raccoglie metadati di produttività come email dell'account e statistiche di utilizzo. Le metriche di contributo non sono disponibili per le organizzazioni ZDR; il dashboard analytics mostra solo metriche di utilizzo.
Gestione utenti e posti	I dati amministrativi come email dell'account e assegnazioni di posti vengono conservati secondo le politiche standard.
Integrazioni di terze parti	I dati elaborati da strumenti di terze parti, MCP servers o altre integrazioni esterne non sono coperti da ZDR. Esaminare indipendentemente le pratiche di gestione dei dati di questi servizi.

Funzionalità disabilitate in ZDR

Quando ZDR è abilitato per un'organizzazione Claude Code su Claude for Enterprise, determinate funzionalità che richiedono l'archiviazione di prompt o completamenti vengono automaticamente disabilitate a livello di backend:

Funzionalità	Motivo
Claude Code on the Web	Richiede l'archiviazione lato server della cronologia delle conversazioni.
Remote sessions dall'app Desktop	Richiede dati di sessione persistenti che includono prompt e completamenti.
Invio di feedback (`/feedback`)	L'invio di feedback invia i dati della conversazione ad Anthropic.

Queste funzionalità sono bloccate nel backend indipendentemente dalla visualizzazione lato client. Se si vede una funzionalità disabilitata nel terminale Claude Code durante l'avvio, il tentativo di utilizzarla restituisce un errore che indica che le politiche dell'organizzazione non consentono tale azione.

Le funzionalità future potrebbero anche essere disabilitate se richiedono l'archiviazione di prompt o completamenti.

Data retention per violazioni delle politiche

Anche con ZDR abilitato, Anthropic può conservare i dati dove richiesto dalla legge o per affrontare violazioni della Usage Policy. Se una sessione viene contrassegnata per una violazione della politica, Anthropic può conservare gli input e gli output associati per un massimo di 2 anni, in linea con la politica ZDR standard di Anthropic.

Richiedere ZDR

Per richiedere ZDR per Claude Code su Claude for Enterprise, contattare il team dell'account Anthropic. Il team dell'account presenterà la richiesta internamente e Anthropic esaminerà e abiliterà ZDR sulla vostra organizzazione dopo aver confermato l'idoneità. Tutte le azioni di abilitazione vengono registrate negli audit log.

Se attualmente si utilizza ZDR per Claude Code tramite chiavi API pay-as-you-go, è possibile passare a Claude for Enterprise per ottenere l'accesso alle funzionalità amministrative mantenendo ZDR per Claude Code. Contattare il team dell'account per coordinare la migrazione.

Part 10: Enterprise & Monitoring

Traccia l'utilizzo del team con l'analittica

> *Visualizza le metriche di utilizzo di Claude Code, traccia l'adozione e misura la velocità di ingegneria nel dashboard di analittica.*

Claude Code fornisce dashboard di analittica per aiutare le organizzazioni a comprendere i modelli di utilizzo degli sviluppatori, tracciare le metriche di contributo e misurare come Claude Code influisce sulla velocità di ingegneria. Accedi al dashboard per il tuo piano:

Piano	URL Dashboard	Include	Leggi di più
Claude for Teams / Enterprise	claude.ai/analytics/claude-code	Metriche di utilizzo, metriche di contributo con integrazione GitHub, leaderboard, esportazione dati	Dettagli
API (Claude Console)	platform.claude.com/claude-code	Metriche di utilizzo, tracciamento della spesa, approfondimenti del team	Dettagli

Access analytics for Teams and Enterprise

Accedi a claude.ai/analytics/claude-code. Gli amministratori e i proprietari possono visualizzare il dashboard.

Il dashboard di Teams e Enterprise include:

- **Metriche di utilizzo**: righe di codice accettate, tasso di accettazione dei suggerimenti, utenti attivi giornalieri e sessioni
- **Metriche di contributo**: PR e righe di codice spedite con assistenza di Claude Code, con integrazione GitHub
- **Leaderboard**: i principali contributori classificati per utilizzo di Claude Code
- **Esportazione dati**: scarica i dati di contributo come CSV per report personalizzati

Enable contribution metrics

> **Note:**
>
> Le metriche di contributo sono in beta pubblica e disponibili sui piani Claude for Teams e Claude for Enterprise. Queste metriche coprono solo gli utenti all'interno della tua organizzazione claude.ai. L'utilizzo tramite l'API Claude Console o integrazioni di terze parti non è incluso.

I dati di utilizzo e adozione sono disponibili per tutti gli account Claude for Teams e Claude for Enterprise. Le metriche di contributo richiedono una configurazione aggiuntiva per connettere la tua organizzazione GitHub.

Hai bisogno del ruolo di proprietario per configurare le impostazioni di analittica. Un amministratore GitHub deve installare l'app GitHub.

> **Warning:**
>
> Le metriche di contributo non sono disponibili per le organizzazioni con Zero Data Retention abilitato. Il dashboard di analittica mostrerà solo le metriche di utilizzo.

Step 1: Installa l'app GitHub

Un amministratore GitHub installa l'app Claude GitHub sull'account GitHub della tua organizzazione su github.com/apps/claude.

Step 2: Abilita l'analittica di Claude Code

Un proprietario Claude accede a claude.ai/admin-settings/claude-code e abilita la funzione di analittica di Claude Code.

Step 3: Abilita l'analittica di GitHub

Sulla stessa pagina, abilita l'interruttore "GitHub analytics".

Step 4: Autentica con GitHub

Completa il flusso di autenticazione GitHub e seleziona quali organizzazioni GitHub includere nell'analisi.

I dati in genere appaiono entro 24 ore dopo l'abilitazione, con aggiornamenti giornalieri. Se non appaiono dati, potresti visualizzare uno di questi messaggi:

- **"GitHub app required"**: installa l'app GitHub per visualizzare le metriche di contributo

- **"Data processing in progress"**: torna tra qualche giorno e conferma che l'app GitHub è installata se i dati non appaiono

Le metriche di contributo supportano GitHub Cloud e GitHub Enterprise Server.

Review summary metrics

> **Note:**
>
> Queste metriche sono deliberatamente conservative e rappresentano una sottostima dell'impatto effettivo di Claude Code. Solo le righe e i PR dove c'è un'alta confidenza nel coinvolgimento di Claude Code vengono conteggiati.

Il dashboard visualizza queste metriche di riepilogo in alto:

- **PRs with CC**: conteggio totale delle pull request unite che contengono almeno una riga di codice scritta con Claude Code
- **Lines of code with CC**: righe totali di codice in tutti i PR uniti che sono stati scritti con assistenza di Claude Code. Solo le "righe effettive" vengono conteggiate: righe con più di 3 caratteri dopo la normalizzazione, escludendo righe vuote e righe con solo parentesi o punteggiatura banale.
- **PRs with Claude Code (%)**: percentuale di tutti i PR uniti che contengono codice assistito da Claude Code
- **Suggestion accept rate**: percentuale di volte in cui gli utenti accettano i suggerimenti di modifica del codice di Claude Code, incluso l'utilizzo degli strumenti Edit, Write e NotebookEdit
- **Lines of code accepted**: righe totali di codice scritte da Claude Code che gli utenti hanno accettato nelle loro sessioni. Questo esclude i suggerimenti rifiutati e non traccia le eliminazioni successive.

Explore the charts

Il dashboard include diversi grafici per visualizzare le tendenze nel tempo.

Track adoption

Il grafico di adozione mostra le tendenze di utilizzo giornaliero:

- **users**: utenti attivi giornalieri
- **sessions**: numero di sessioni attive di Claude Code al giorno

Measure PRs per user

Questo grafico visualizza l'attività dei singoli sviluppatori nel tempo:

- **PRs per user**: numero totale di PR uniti al giorno diviso per utenti attivi giornalieri
- **users**: utenti attivi giornalieri

Usa questo per comprendere come la produttività individuale cambia man mano che aumenta l'adozione di Claude Code.

View pull requests breakdown

Il grafico Pull requests mostra una suddivisione giornaliera dei PR uniti:

- **PRs with CC**: pull request contenenti codice assistito da Claude Code
- **PRs without CC**: pull request senza codice assistito da Claude Code

Attiva la visualizzazione **Lines of code** per vedere la stessa suddivisione per righe di codice anziché per conteggio di PR.

Find top contributors

La leaderboard mostra i 10 utenti principali classificati per volume di contributo. Attiva tra:

- **Pull requests**: mostra PR con Claude Code rispetto a tutti i PR per ogni utente
- **Lines of code**: mostra righe con Claude Code rispetto a tutte le righe per ogni utente

Fai clic su **Export all users** per scaricare i dati di contributo completi per tutti gli utenti come file CSV. L'esportazione include tutti gli utenti, non solo i 10 principali visualizzati.

PR attribution

Quando le metriche di contributo sono abilitate, Claude Code analizza le pull request unite per determinare quale codice è stato scritto con assistenza di Claude Code. Questo viene fatto abbinando l'attività della sessione di Claude Code al codice in ogni PR.

Tagging criteria

I PR sono etichettati come "with Claude Code" se contengono almeno una riga di codice scritta durante una sessione di Claude Code. Il sistema utilizza un abbinamento conservativo: solo il codice dove c'è un'alta confidenza nel coinvolgimento di Claude Code viene conteggiato come assistito.

Attribution process

Quando una pull request viene unita:

1. Le righe aggiunte vengono estratte dal diff del PR
2. Le sessioni di Claude Code che hanno modificato i file corrispondenti entro una finestra temporale vengono identificate
3. Le righe del PR vengono abbinate all'output di Claude Code utilizzando più strategie
4. Le metriche vengono calcolate per le righe assistite da AI e le righe totali

Prima del confronto, le righe vengono normalizzate: gli spazi bianchi vengono eliminati, gli spazi multipli vengono compressi, le virgolette vengono standardizzate e il testo viene convertito in minuscolo.

Le pull request unite contenenti righe assistite da Claude Code sono etichettate come `claude-code-assisted` in GitHub.

Time window

Le sessioni da 21 giorni prima a 2 giorni dopo la data di unione del PR vengono considerate per l'abbinamento dell'attribuzione.

Excluded files

Alcuni file vengono automaticamente esclusi dall'analisi perché sono generati automaticamente:

- File di blocco: package-lock.json, yarn.lock, Cargo.lock e simili
- Codice generato: output Protobuf, artefatti di build, file minificati
- Directory di build: dist/, build/, node_modules/, target/
- Fixture di test: snapshot, cassette, dati mock
- Righe superiori a 1.000 caratteri, che probabilmente sono minificate o generate

Attribution notes

Tieni presenti questi dettagli aggiuntivi quando interpreti i dati di attribuzione:

- Il codice sostanzialmente riscritto dagli sviluppatori, con una differenza superiore al 20%, non viene attribuito a Claude Code
- Le sessioni al di fuori della finestra di 21 giorni non vengono considerate
- L'algoritmo non considera il ramo di origine o di destinazione del PR quando esegue l'attribuzione

Get the most from analytics

Usa le metriche di contributo per dimostrare il ROI, identificare i modelli di adozione e trovare i membri del team che possono aiutare gli altri a iniziare.

Monitor adoption

Traccia il grafico di adozione e i conteggi degli utenti per identificare:

- Utenti attivi che possono condividere le migliori pratiche
- Tendenze di adozione complessiva nella tua organizzazione
- Cali nell'utilizzo che potrebbero indicare attrito o problemi

Measure ROI

Le metriche di contributo aiutano a rispondere a "Vale la pena investire in questo strumento?" con dati dal tuo codebase:

- Traccia i cambiamenti nei PR per utente nel tempo man mano che aumenta l'adozione
- Confronta i PR e le righe di codice spedite con e senza Claude Code
- Usa insieme alle metriche DORA, alla velocità dello sprint o ad altri KPI di ingegneria per comprendere i cambiamenti derivanti dall'adozione di Claude Code

Identify power users

La leaderboard ti aiuta a trovare i membri del team con un'elevata adozione di Claude Code che possono:

- Condividere tecniche di prompt e flussi di lavoro con il team
- Fornire feedback su ciò che funziona bene
- Aiutare a integrare i nuovi utenti

Access data programmatically

Per interrogare questi dati tramite GitHub, cerca i PR etichettati con `claude-code-assisted`.

Access analytics for API customers

I clienti API che utilizzano Claude Console possono accedere all'analittica su platform.claude.com/claude-code. Hai bisogno dell'autorizzazione UsageView per accedere al dashboard, che viene concessa ai ruoli Developer, Billing, Admin, Owner e Primary Owner.

> **Note:**
>
> Le metriche di contributo con integrazione GitHub non sono attualmente disponibili per i clienti API. Il dashboard della console mostra solo le metriche di utilizzo e spesa.

Il dashboard della console visualizza:

- **Lines of code accepted**: righe totali di codice scritte da Claude Code che gli utenti hanno accettato nelle loro sessioni. Questo esclude i suggerimenti rifiutati e non traccia le eliminazioni successive.
- **Suggestion accept rate**: percentuale di volte in cui gli utenti accettano l'utilizzo dello strumento di modifica del codice, inclusi gli strumenti Edit, Write e NotebookEdit.
- **Activity**: utenti attivi giornalieri e sessioni mostrate su un grafico.
- **Spend**: costi API giornalieri in dollari insieme al conteggio degli utenti.

View team insights

La tabella di approfondimenti del team mostra le metriche per utente:

- **Members**: tutti gli utenti che si sono autenticati a Claude Code. Gli utenti con chiave API vengono visualizzati per identificatore di chiave, gli utenti OAuth vengono visualizzati per indirizzo email.
- **Spend this month**: costi API totali per utente per il mese corrente.
- **Lines this month**: totale per utente delle righe di codice accettate per il mese corrente.

> **Note:**
>
> Le cifre di spesa nel dashboard della console sono stime a scopo di analittica. Per i costi effettivi, consulta la tua pagina di fatturazione.

Related resources

- Monitoring with OpenTelemetry: esporta metriche e eventi in tempo reale al tuo stack di osservabilità
- Manage costs effectively: imposta limiti di spesa e ottimizza l'utilizzo dei token
- Permissions: configura ruoli e autorizzazioni

Monitoraggio

> *Scopri come abilitare e configurare OpenTelemetry per Claude Code.*

Claude Code supporta metriche e eventi OpenTelemetry (OTel) per il monitoraggio e l'osservabilità.

Tutte le metriche sono dati di serie temporali esportati tramite il protocollo di metriche standard di OpenTelemetry, e gli eventi vengono esportati tramite il protocollo di log/eventi di OpenTelemetry. È responsabilità dell'utente assicurarsi che i backend di metriche e log siano configurati correttamente e che la granularità dell'aggregazione soddisfi i requisiti di monitoraggio.

Avvio rapido

Configura OpenTelemetry utilizzando variabili di ambiente:

```
## 1. Abilita la telemetria
export CLAUDE_CODE_ENABLE_TELEMETRY=1

## 2. Scegli gli esportatori (entrambi sono facoltativi - configura solo ciò di
cui hai bisogno)
export OTEL_METRICS_EXPORTER=otlp       # Opzioni: otlp, prometheus, console
export OTEL_LOGS_EXPORTER=otlp          # Opzioni: otlp, console

## 3. Configura l'endpoint OTLP (per l'esportatore OTLP)
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## 4. Imposta l'autenticazione (se richiesta)
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer your-token"

## 5. Per il debug: riduci gli intervalli di esportazione
export OTEL_METRIC_EXPORT_INTERVAL=10000  # 10 secondi (predefinito: 60000ms)
export OTEL_LOGS_EXPORT_INTERVAL=5000     # 5 secondi (predefinito: 5000ms)

## 6. Esegui Claude Code
claude
```

> **Note:**
>
> Gli intervalli di esportazione predefiniti sono 60 secondi per le metriche e 5 secondi per i log. Durante la configurazione, potresti voler utilizzare intervalli più brevi per scopi di debug. Ricordati di ripristinare questi valori per l'uso in produzione.

Per le opzioni di configurazione complete, consulta la specifica OpenTelemetry.

Configurazione dell'amministratore

Gli amministratori possono configurare le impostazioni di OpenTelemetry per tutti gli utenti tramite il file di impostazioni gestite. Ciò consente il controllo centralizzato delle impostazioni di telemetria in tutta l'organizzazione. Consulta la precedenza delle impostazioni per ulteriori informazioni su come vengono applicate le impostazioni.

Esempio di configurazione delle impostazioni gestite:

```
{
  "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:

Le impostazioni gestite possono essere distribuite tramite MDM (Mobile Device Management) o altre soluzioni di gestione dei dispositivi. Le variabili di ambiente definite nel file di impostazioni gestite hanno una precedenza elevata e non possono essere sovrascritte dagli utenti.

Dettagli della configurazione

Variabili di configurazione comuni

Variabile di ambiente	Descrizione	Valori di esempio
`CLAUDE_CODE_ENABLE_TELEMETRY`	Abilita la raccolta della telemetria (obbligatorio)	`1`
`OTEL_METRICS_EXPORTER`	Tipo di esportatore di metriche (separati da virgola)	`console`, `otlp`, `prometheus`
`OTEL_LOGS_EXPORTER`	Tipo di esportatore di log/eventi (separati da virgola)	`console`, `otlp`
`OTEL_EXPORTER_OTLP_PROTOCOL`	Protocollo per l'esportatore OTLP (tutti i segnali)	`grpc`, `http/json`, `http/protobuf`
`OTEL_EXPORTER_OTLP_ENDPOINT`	Endpoint del collettore OTLP (tutti i segnali)	`http://localhost:4317`
`OTEL_EXPORTER_OTLP_METRICS_PROTOCOL`	Protocollo per le metriche (sostituisce il generale)	`grpc`, `http/json`, `http/protobuf`

Variabile di ambiente	Descrizione	Valori di esempio
`OTEL_EXPORTER_OTLP_METRICS_ENDPOINT`	Endpoint OTLP per le metriche (sostituisce il generale)	`http://localhost:4318/v1/metrics`
`OTEL_EXPORTER_OTLP_LOGS_PROTOCOL`	Protocollo per i log (sostituisce il generale)	`grpc`, `http/json`, `http/protobuf`
`OTEL_EXPORTER_OTLP_LOGS_ENDPOINT`	Endpoint OTLP per i log (sostituisce il generale)	`http://localhost:4318/v1/logs`
`OTEL_EXPORTER_OTLP_HEADERS`	Intestazioni di autenticazione per OTLP	`Authorization=Bearer token`
`OTEL_EXPORTER_OTLP_METRICS_CLIENT_KEY`	Chiave client per l'autenticazione mTLS	Percorso del file della chiave client
`OTEL_EXPORTER_OTLP_METRICS_CLIENT_CERTIFICATE`	Certificato client per l'autenticazione mTLS	Percorso del file del certificato client
`OTEL_METRIC_EXPORT_INTERVAL`	Intervallo di esportazione in millisecondi (predefinito: 60000)	`5000`, `60000`
`OTEL_LOGS_EXPORT_INTERVAL`	Intervallo di esportazione dei log in millisecondi (predefinito: 5000)	`1000`, `10000`
`OTEL_LOG_USER_PROMPTS`	Abilita la registrazione del contenuto del prompt dell'utente (predefinito: disabilitato)	`1` per abilitare
`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS`	Intervallo per l'aggiornamento delle intestazioni dinamiche (predefinito: 1740000ms / 29 minuti)	`900000`

Controllo della cardinalità delle metriche

Le seguenti variabili di ambiente controllano quali attributi sono inclusi nelle metriche per gestire la cardinalità:

Variabile di ambiente	Descrizione	Valore predefinito	Esempio per disabilitare
`OTEL_METRICS_INCLUDE_SESSION_ID`	Includi l'attributo session.id nelle metriche	`true`	`false`
`OTEL_METRICS_INCLUDE_VERSION`	Includi l'attributo app.version nelle metriche	`false`	`true`
`OTEL_METRICS_INCLUDE_ACCOUNT_UUID`	Includi l'attributo user.account_uuid nelle metriche	`true`	`false`

Queste variabili aiutano a controllare la cardinalità delle metriche, che influisce sui requisiti di archiviazione e sulle prestazioni delle query nel backend delle metriche. Una cardinalità inferiore generalmente significa prestazioni migliori e costi di archiviazione inferiori, ma dati meno granulari per l'analisi.

Intestazioni dinamiche

Per gli ambienti aziendali che richiedono autenticazione dinamica, puoi configurare uno script per generare intestazioni dinamicamente:

Configurazione delle impostazioni

Aggiungi al tuo `.claude/settings.json`:

```
{
  "otelHeadersHelper": "/bin/generate_opentelemetry_headers.sh"
}
```

Requisiti dello script

Lo script deve generare JSON valido con coppie chiave-valore di stringhe che rappresentano intestazioni HTTP:

```
#!/bin/bash
## Esempio: Intestazioni multiple
echo "{\"Authorization\": \"Bearer $(get-token.sh)\", \"X-API-Key\": \"$(get-api-key.sh)\"}"
```

Comportamento di aggiornamento

Lo script dell'helper di intestazioni viene eseguito all'avvio e periodicamente in seguito per supportare l'aggiornamento dei token. Per impostazione predefinita, lo script viene eseguito ogni 29 minuti. Personalizza l'intervallo con la variabile di ambiente `CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS`.

Supporto per organizzazioni multi-team

Le organizzazioni con più team o dipartimenti possono aggiungere attributi personalizzati per distinguere tra diversi gruppi utilizzando la variabile di ambiente `OTEL_RESOURCE_ATTRIBUTES`:

```
## Aggiungi attributi personalizzati per l'identificazione del team
export OTEL_RESOURCE_ATTRIBUTES="department=engineering,team.id=platform,cost_center=eng-123"
```

Questi attributi personalizzati verranno inclusi in tutte le metriche e gli eventi, permettendoti di:

- Filtrare le metriche per team o dipartimento
- Tracciare i costi per centro di costo
- Creare dashboard specifici per team
- Configurare avvisi per team specifici

> **Warning:**
>
> **Requisiti di formattazione importanti per OTEL_RESOURCE_ATTRIBUTES:**
>
> La variabile di ambiente `OTEL_RESOURCE_ATTRIBUTES` segue la specifica W3C Baggage, che ha requisiti di formattazione rigorosi:
>
> - **Nessuno spazio consentito**: I valori non possono contenere spazi. Ad esempio, `user.organizationName=My Company` non è valido
> - **Formato**: Deve essere coppie chiave=valore separate da virgola: `key1=value1,key2=value2`

- **Caratteri consentiti**: Solo caratteri US-ASCII escludendo caratteri di controllo, spazi bianchi, virgolette doppie, virgole, punti e virgola e barre rovesciate
- **Caratteri speciali**: I caratteri al di fuori dell'intervallo consentito devono essere codificati in percentuale

Esempi:

```
## ❌ Non valido - contiene spazi
export OTEL_RESOURCE_ATTRIBUTES="org.name=John's Organization"

## ✅ Valido - usa sottolineature o camelCase invece
export OTEL_RESOURCE_ATTRIBUTES="org.name=Johns_Organization"
export OTEL_RESOURCE_ATTRIBUTES="org.name=JohnsOrganization"

## ✅ Valido - codifica in percentuale i caratteri speciali se necessario
export OTEL_RESOURCE_ATTRIBUTES="org.name=John%27s%20Organization"
```

Nota: racchiudere i valori tra virgolette non sfugge agli spazi. Ad esempio, `org.name="My Company"` risulta nel valore letterale `"My Company"` (con virgolette incluse), non `My Company`.

Configurazioni di esempio

```
## Debug della console (intervalli di 1 secondo)
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

## Esportatori multipli
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console,otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=http/json

## Endpoint/backend diversi per metriche e log
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

## Solo metriche (nessun evento/log)
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

## Solo eventi/log (nessuna metrica)
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
```

Metriche e eventi disponibili

Attributi standard

Tutte le metriche e gli eventi condividono questi attributi standard:

Attributo	Descrizione	Controllato da
`session.id`	Identificatore di sessione univoco	`OTEL_METRICS_INCLUDE_SESSION_ID` (predefinito: true)
`app.version`	Versione corrente di Claude Code	`OTEL_METRICS_INCLUDE_VERSION` (predefinito: false)
`organization.id`	UUID dell'organizzazione (quando autenticato)	Sempre incluso quando disponibile
`user.account_uuid`	UUID dell'account (quando autenticato)	`OTEL_METRICS_INCLUDE_ACCOUNT_UUID` (predefinito: true)
`terminal.type`	Tipo di terminale (ad esempio, `iTerm.app`, `vscode`, `cursor`, `tmux`)	Sempre incluso quando rilevato

Metriche

Claude Code esporta le seguenti metriche:

Nome della metrica	Descrizione	Unità
`claude_code.session.count`	Conteggio delle sessioni CLI avviate	count
`claude_code.lines_of_code.count`	Conteggio delle righe di codice modificate	count
`claude_code.pull_request.count`	Numero di pull request create	count

Nome della metrica	Descrizione	Unità
`claude_code.commit.count`	Numero di commit git creati	count
`claude_code.cost.usage`	Costo della sessione Claude Code	USD
`claude_code.token.usage`	Numero di token utilizzati	tokens
`claude_code.code_edit_tool.decision`	Conteggio delle decisioni di autorizzazione dello strumento di modifica del codice	count
`claude_code.active_time.total`	Tempo attivo totale in secondi	s

Dettagli delle metriche

Contatore di sessione

Incrementato all'inizio di ogni sessione.

Attributi:

- Tutti gli attributi standard

Contatore di righe di codice

Incrementato quando il codice viene aggiunto o rimosso.

Attributi:

- Tutti gli attributi standard
- `type` : (`"added"` , `"removed"`)

Contatore di pull request

Incrementato quando si creano pull request tramite Claude Code.

Attributi:

- Tutti gli attributi standard

Contatore di commit

Incrementato quando si creano commit git tramite Claude Code.

Attributi:

- Tutti gli attributi standard

Contatore di costo

Incrementato dopo ogni richiesta API.

Attributi:

- Tutti gli attributi standard
- `model` : Identificatore del modello (ad esempio, “claude-sonnet-4-5-20250929”)

Contatore di token

Incrementato dopo ogni richiesta API.

Attributi:

- Tutti gli attributi standard
- `type` : (`"input"` , `"output"` , `"cacheRead"` , `"cacheCreation"`)
- `model` : Identificatore del modello (ad esempio, “claude-sonnet-4-5-20250929”)

Contatore di decisione dello strumento di modifica del codice

Incrementato quando l’utente accetta o rifiuta l’utilizzo dello strumento Edit, Write o NotebookEdit.

Attributi:

- Tutti gli attributi standard
- `tool` : Nome dello strumento (`"Edit"` , `"Write"` , `"NotebookEdit"`)
- `decision` : Decisione dell’utente (`"accept"` , `"reject"`)
- `language` : Linguaggio di programmazione del file modificato (ad esempio, `"TypeScript"` , `"Python"` , `"JavaScript"` , `"Markdown"`). Restituisce `"unknown"` per estensioni di file non riconosciute.

Contatore di tempo attivo

Traccia il tempo effettivo trascorso utilizzando attivamente Claude Code (non il tempo di inattività). Questa metrica viene incrementata durante le interazioni dell’utente come la digitazione di prompt o la ricezione di risposte.

Attributi:

- Tutti gli attributi standard

Eventi

Claude Code esporta i seguenti eventi tramite log/eventi OpenTelemetry (quando `OTEL_LOGS_EXPORTER` è configurato):

Evento di prompt dell'utente

Registrato quando un utente invia un prompt.

Nome evento: `claude_code.user_prompt`

Attributi:

- Tutti gli attributi standard
- `event.name`: `"user_prompt"`
- `event.timestamp`: Timestamp ISO 8601
- `prompt_length`: Lunghezza del prompt
- `prompt`: Contenuto del prompt (redatto per impostazione predefinita, abilita con `OTEL_LOG_USER_PROMPTS=1`)

Evento di risultato dello strumento

Registrato quando uno strumento completa l'esecuzione.

Nome evento: `claude_code.tool_result`

Attributi:

- Tutti gli attributi standard
- `event.name`: `"tool_result"`
- `event.timestamp`: Timestamp ISO 8601
- `tool_name`: Nome dello strumento
- `success`: `"true"` o `"false"`
- `duration_ms`: Tempo di esecuzione in millisecondi
- `error`: Messaggio di errore (se non riuscito)
- `decision`: `"accept"` o `"reject"`
- `source`: Fonte della decisione - `"config"`, `"user_permanent"`, `"user_temporary"`, `"user_abort"` o `"user_reject"`
- `tool_parameters`: Stringa JSON contenente parametri specifici dello strumento (quando disponibili)
  - Per lo strumento Bash: include `bash_command`, `full_command`, `timeout`, `description`, `sandbox`

Evento di richiesta API

Registrato per ogni richiesta API a Claude.

Nome evento: `claude_code.api_request`

Attributi:

- Tutti gli attributi standard
- `event.name`: `"api_request"`
- `event.timestamp`: Timestamp ISO 8601
- `model`: Modello utilizzato (ad esempio, “claude-sonnet-4-5-20250929”)
- `cost_usd`: Costo stimato in USD
- `duration_ms`: Durata della richiesta in millisecondi
- `input_tokens`: Numero di token di input
- `output_tokens`: Numero di token di output
- `cache_read_tokens`: Numero di token letti dalla cache
- `cache_creation_tokens`: Numero di token utilizzati per la creazione della cache

Evento di errore API

Registrato quando una richiesta API a Claude non riesce.

Nome evento: `claude_code.api_error`

Attributi:

- Tutti gli attributi standard
- `event.name`: `"api_error"`
- `event.timestamp`: Timestamp ISO 8601
- `model`: Modello utilizzato (ad esempio, “claude-sonnet-4-5-20250929”)
- `error`: Messaggio di errore
- `status_code`: Codice di stato HTTP (se applicabile)
- `duration_ms`: Durata della richiesta in millisecondi
- `attempt`: Numero di tentativo (per le richieste riprovate)

Evento di decisione dello strumento

Registrato quando viene presa una decisione di autorizzazione dello strumento (accetta/rifiuta).

Nome evento: `claude_code.tool_decision`

Attributi:

- Tutti gli attributi standard
- `event.name`: `"tool_decision"`
- `event.timestamp`: Timestamp ISO 8601
- `tool_name`: Nome dello strumento (ad esempio, “Read”, “Edit”, “Write”, “NotebookEdit”)
- `decision`: `"accept"` o `"reject"`
- `source`: Fonte della decisione - `"config"`, `"user_permanent"`, `"user_temporary"`, `"user_abort"` o `"user_reject"`

Interpretazione dei dati di metriche e eventi

Le metriche esportate da Claude Code forniscono informazioni preziose sui modelli di utilizzo e sulla produttività. Ecco alcune visualizzazioni e analisi comuni che puoi creare:

Monitoraggio dell’utilizzo

Metrica	Opportunità di analisi
`claude_code.token.usage`	Suddividi per `type` (input/output), utente, team o modello
`claude_code.session.count`	Traccia l’adozione e l’engagement nel tempo
`claude_code.lines_of_code.count`	Misura la produttività tracciando le aggiunte/rimozioni di codice
`claude_code.commit.count` & `claude_code.pull_request.count`	Comprendi l’impatto sui flussi di lavoro di sviluppo

Monitoraggio dei costi

La metrica `claude_code.cost.usage` aiuta con:

- Tracciare i trend di utilizzo tra team o individui
- Identificare sessioni ad alto utilizzo per l’ottimizzazione

> **Note:**
>
> Le metriche di costo sono approssimazioni. Per i dati di fatturazione ufficiali, consulta il tuo provider API (Claude Console, AWS Bedrock o Google Cloud Vertex).

Avvisi e segmentazione

Avvisi comuni da considerare:

- Picchi di costo
- Consumo di token inusuale
- Alto volume di sessioni da utenti specifici

Tutte le metriche possono essere segmentate per `user.account_uuid`, `organization.id`, `session.id`, `model` e `app.version`.

Analisi degli eventi

I dati degli eventi forniscono informazioni dettagliate sulle interazioni di Claude Code:

Modelli di utilizzo dello strumento: analizza gli eventi di risultato dello strumento per identificare:

- Strumenti più frequentemente utilizzati
- Tassi di successo dello strumento
- Tempi di esecuzione medi dello strumento
- Modelli di errore per tipo di strumento

Monitoraggio delle prestazioni: traccia le durate delle richieste API e i tempi di esecuzione dello strumento per identificare i colli di bottiglia delle prestazioni.

Considerazioni sul backend

La scelta dei backend di metriche e log determina i tipi di analisi che puoi eseguire:

Per le metriche

- **Database di serie temporali (ad esempio, Prometheus)**: Calcoli di velocità, metriche aggregate
- **Archivi colonnari (ad esempio, ClickHouse)**: Query complesse, analisi di utenti univoci
- **Piattaforme di osservabilità complete (ad esempio, Honeycomb, Datadog)**: Query avanzate, visualizzazione, avvisi

Per eventi/log

- **Sistemi di aggregazione dei log (ad esempio, Elasticsearch, Loki)**: Ricerca full-text, analisi dei log
- **Archivi colonnari (ad esempio, ClickHouse)**: Analisi degli eventi strutturati

- **Piattaforme di osservabilità complete (ad esempio, Honeycomb, Datadog)**: Correlazione tra metriche e eventi

Per le organizzazioni che richiedono metriche Daily/Weekly/Monthly Active User (DAU/WAU/MAU), considera backend che supportano query di valori univoci efficienti.

Informazioni sul servizio

Tutte le metriche e gli eventi vengono esportati con i seguenti attributi di risorsa:

- `service.name`: `claude-code`
- `service.version`: Versione corrente di Claude Code
- `os.type`: Tipo di sistema operativo (ad esempio, `linux`, `darwin`, `windows`)
- `os.version`: Stringa della versione del sistema operativo
- `host.arch`: Architettura dell'host (ad esempio, `amd64`, `arm64`)
- `wsl.version`: Numero di versione WSL (presente solo quando si esegue su Windows Subsystem for Linux)
- Nome del contatore: `com.anthropic.claude_code`

Risorse di misurazione del ROI

Per una guida completa sulla misurazione del ritorno sull'investimento per Claude Code, inclusa la configurazione della telemetria, l'analisi dei costi, le metriche di produttività e i report automatizzati, consulta la Guida alla misurazione del ROI di Claude Code. Questo repository fornisce configurazioni Docker Compose pronte all'uso, configurazioni Prometheus e OpenTelemetry, e modelli per generare report di produttività integrati con strumenti come Linear.

Considerazioni sulla sicurezza/privacy

- La telemetria è opt-in e richiede una configurazione esplicita
- Le informazioni sensibili come le chiavi API o i contenuti dei file non vengono mai incluse nelle metriche o negli eventi
- Il contenuto del prompt dell'utente viene redatto per impostazione predefinita - viene registrata solo la lunghezza del prompt. Per abilitare la registrazione del prompt dell'utente, imposta `OTEL_LOG_USER_PROMPTS=1`

Monitoraggio di Claude Code su Amazon Bedrock

Per una guida dettagliata al monitoraggio dell'utilizzo di Claude Code per Amazon Bedrock, consulta Claude Code Monitoring Implementation (Bedrock).

Panoramica della distribuzione aziendale

> *Scopri come Claude Code può integrarsi con vari servizi di terze parti e infrastrutture per soddisfare i requisiti di distribuzione aziendale.*

Questa pagina fornisce una panoramica delle opzioni di distribuzione disponibili e ti aiuta a scegliere la configurazione giusta per la tua organizzazione.

Confronto dei provider

Funzionalità Anthropic Amazon Bedrock Google Vertex AI Microsoft Foundry

Regioni Paesi supportati countries Più regioni AWS Più regioni GCP Più regioni Azure

Caching dei prompt Abilitato per impostazione predefinita Abilitato per impostazione predefinita Abilitato per impostazione predefinita Abilitato per impostazione predefinita

Autenticazione Chiave API Chiave API o credenziali AWS Credenziali GCP Chiave API o Microsoft Entra ID

Tracciamento dei costi Dashboard AWS Cost Explorer GCP Billing Azure Cost Management

Funzionalità aziendali Team, monitoraggio dell'utilizzo Criteri IAM, CloudTrail Ruoli IAM, Cloud Audit Logs Criteri RBAC, Azure Monitor

Provider cloud

- Amazon Bedrock Utilizza i modelli Claude tramite l'infrastruttura AWS con autenticazione basata su chiave API o IAM e monitoraggio nativo di AWS
- Google Vertex AI Accedi ai modelli Claude tramite Google Cloud Platform con sicurezza e conformità di livello aziendale
- Microsoft Foundry Accedi a Claude tramite Azure con autenticazione tramite chiave API o Microsoft Entra ID e fatturazione Azure

Infrastruttura aziendale

- Enterprise Network Configura Claude Code per funzionare con i server proxy e i requisiti SSL/TLS della tua organizzazione

- LLM Gateway Distribuisci l'accesso centralizzato ai modelli con tracciamento dell'utilizzo, budgeting e registrazione di audit

Panoramica della configurazione

Claude Code supporta opzioni di configurazione flessibili che ti permettono di combinare diversi provider e infrastrutture:

> **Note:**
>
> Comprendi la differenza tra:
>
> - **Proxy aziendale**: Un proxy HTTP/HTTPS per l'instradamento del traffico (impostato tramite `HTTPS_PROXY` o `HTTP_PROXY`)
> - **LLM Gateway**: Un servizio che gestisce l'autenticazione e fornisce endpoint compatibili con il provider (impostato tramite `ANTHROPIC_BASE_URL`, `ANTHROPIC_BEDROCK_BASE_URL` o `ANTHROPIC_VERTEX_BASE_URL`)
>
> Entrambe le configurazioni possono essere utilizzate insieme.

Utilizzo di Bedrock con proxy aziendale

Instrada il traffico Bedrock attraverso un proxy HTTP/HTTPS aziendale:

```
## Abilita Bedrock
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1

## Configura proxy aziendale
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Utilizzo di Bedrock con LLM Gateway

Utilizza un servizio gateway che fornisce endpoint compatibili con Bedrock:

```
## Abilita Bedrock
export CLAUDE_CODE_USE_BEDROCK=1

## Configura gateway LLM
export ANTHROPIC_BEDROCK_BASE_URL='https://your-llm-gateway.com/bedrock'
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1  # Se il gateway gestisce l'autenticazione
AWS
```

Utilizzo di Foundry con proxy aziendale

Instrada il traffico Azure attraverso un proxy HTTP/HTTPS aziendale:

```
## Abilita Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1
export ANTHROPIC_FOUNDRY_RESOURCE=your-resource
export ANTHROPIC_FOUNDRY_API_KEY=your-api-key  # O ometti per l'autenticazione
Entra ID

## Configura proxy aziendale
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Utilizzo di Foundry con LLM Gateway

Utilizza un servizio gateway che fornisce endpoint compatibili con Azure:

```
## Abilita Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1

## Configura gateway LLM
export ANTHROPIC_FOUNDRY_BASE_URL='https://your-llm-gateway.com'
export CLAUDE_CODE_SKIP_FOUNDRY_AUTH=1  # Se il gateway gestisce l'autenticazione
Azure
```

Utilizzo di Vertex AI con proxy aziendale

Instrada il traffico Vertex AI attraverso un proxy HTTP/HTTPS aziendale:

```
## Abilita Vertex
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
export ANTHROPIC_VERTEX_PROJECT_ID=your-project-id

## Configura proxy aziendale
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Utilizzo di Vertex AI con LLM Gateway

Combina i modelli Google Vertex AI con un gateway LLM per la gestione centralizzata:

```
## Abilita Vertex
export CLAUDE_CODE_USE_VERTEX=1

## Configura gateway LLM
export ANTHROPIC_VERTEX_BASE_URL='https://your-llm-gateway.com/vertex'
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1  # Se il gateway gestisce l'autenticazione
GCP
```

Configurazione dell'autenticazione

Claude Code utilizza `ANTHROPIC_AUTH_TOKEN` per l'intestazione `Authorization` quando necessario. I flag `SKIP_AUTH` (`CLAUDE_CODE_SKIP_BEDROCK_AUTH`, `CLAUDE_CODE_SKIP_VERTEX_AUTH`) vengono utilizzati negli scenari di gateway LLM in cui il gateway gestisce l'autenticazione del provider.

Scelta della giusta configurazione di distribuzione

Considera questi fattori quando selezioni il tuo approccio di distribuzione:

Accesso diretto al provider

Ideale per le organizzazioni che:

- Desiderano la configurazione più semplice
- Hanno infrastruttura AWS o GCP esistente
- Hanno bisogno di monitoraggio e conformità nativi del provider

Proxy aziendale

Ideale per le organizzazioni che:

- Hanno requisiti di proxy aziendale esistenti
- Hanno bisogno di monitoraggio del traffico e conformità
- Devono instradare tutto il traffico attraverso percorsi di rete specifici

LLM Gateway

Ideale per le organizzazioni che:

- Hanno bisogno di tracciamento dell'utilizzo tra i team
- Desiderano passare dinamicamente tra modelli
- Richiedono limitazione della velocità personalizzata o budget
- Hanno bisogno di gestione dell'autenticazione centralizzata

Debug

Quando esegui il debug della tua distribuzione:

- Utilizza il comando slash `claude /status`. Questo comando fornisce visibilità su qualsiasi autenticazione, proxy e impostazioni URL applicate.
- Imposta la variabile di ambiente `export ANTHROPIC_LOG=debug` per registrare le richieste.

Best practice per le organizzazioni

1. Investire in documentazione e memoria

Ti consigliamo vivamente di investire nella documentazione in modo che Claude Code comprenda il tuo codebase. Le organizzazioni possono distribuire file CLAUDE.md a più livelli:

- **A livello di organizzazione**: Distribuisci in directory di sistema come `/Library/Application Support/ClaudeCode/CLAUDE.md` (macOS) per gli standard aziendali
- **A livello di repository**: Crea file `CLAUDE.md` nelle radici dei repository contenenti l'architettura del progetto, i comandi di build e le linee guida di contribuzione. Archivialo nel controllo del codice sorgente in modo che tutti gli utenti ne traggano beneficio
  Scopri di più.

2. Semplificare la distribuzione

Se hai un ambiente di sviluppo personalizzato, riteniamo che creare un modo “con un clic” per installare Claude Code sia fondamentale per aumentare l’adozione in tutta l’organizzazione.

3. Inizia con un utilizzo guidato

Incoraggia i nuovi utenti a provare Claude Code per domande e risposte sul codebase, o su correzioni di bug più piccole o richieste di funzionalità. Chiedi a Claude Code di fare un piano. Controlla i suggerimenti di Claude e fornisci feedback se non è sulla strada giusta. Nel tempo, man mano che gli utenti comprendono meglio questo nuovo paradigma, saranno più efficaci nel permettere a Claude Code di funzionare in modo più autonomo.

4. Configurare i criteri di sicurezza

I team di sicurezza possono configurare autorizzazioni gestite per ciò che Claude Code è e non è autorizzato a fare, che non può essere sovrascritto dalla configurazione locale. Scopri di più.

5. Sfruttare MCP per le integrazioni

MCP è un ottimo modo per fornire a Claude Code più informazioni, come la connessione a sistemi di gestione dei ticket o registri di errori. Ti consigliamo che un team centrale configuri i server MCP e archivi una configurazione `.mcp.json` nel codebase in modo che tutti gli utenti ne traggano beneficio. Scopri di più.

In Anthropic, confidiamo in Claude Code per alimentare lo sviluppo in ogni codebase di Anthropic. Speriamo che tu apprezzi l’utilizzo di Claude Code quanto lo apprezziamo noi.

Passaggi successivi

- Configura Amazon Bedrock per la distribuzione nativa di AWS
- Configura Google Vertex AI per la distribuzione GCP
- Configura Microsoft Foundry per la distribuzione Azure
- Configura Enterprise Network per i requisiti di rete
- Distribuisci LLM Gateway per la gestione aziendale
- Impostazioni per le opzioni di configurazione e le variabili di ambiente

Configurare le impostazioni gestite dal server (beta pubblico)

> *Configurare centralmente Claude Code per la vostra organizzazione tramite impostazioni consegnate dal server, senza richiedere infrastruttura di gestione dei dispositivi.*

Le impostazioni gestite dal server consentono agli amministratori di configurare centralmente Claude Code tramite un'interfaccia basata sul web su Claude.ai. I client di Claude Code ricevono automaticamente queste impostazioni quando gli utenti si autenticano con le credenziali della loro organizzazione.

Questo approccio è progettato per le organizzazioni che non dispongono di infrastruttura di gestione dei dispositivi, o che hanno bisogno di gestire le impostazioni per gli utenti su dispositivi non gestiti.

Note:

Le impostazioni gestite dal server sono in beta pubblico e disponibili per i clienti di Claude for Teams e Claude for Enterprise. Le funzionalità potrebbero evolversi prima della disponibilità generale.

Requisiti

Per utilizzare le impostazioni gestite dal server, è necessario:

- Piano Claude for Teams o Claude for Enterprise
- Claude Code versione 2.1.38 o successiva per Claude for Teams, o versione 2.1.30 o successiva per Claude for Enterprise
- Accesso di rete a `api.anthropic.com`

Scegliere tra impostazioni gestite dal server e gestite dall'endpoint

Claude Code supporta due approcci per la configurazione centralizzata. Le impostazioni gestite dal server forniscono la configurazione dai server di Anthropic. Le impostazioni gestite dall'endpoint vengono distribuite direttamente ai dispositivi tramite criteri nativi del sistema operativo (preferenze gestite macOS, registro Windows) o file di impostazioni gestiti.

Approccio	Ideale per	Modello di sicurezza
Impostazioni gestite dal server	Organizzazioni senza MDM, o utenti su dispositivi non gestiti	Impostazioni consegnate dai server di Anthropic al momento dell'autenticazione
Impostazioni gestite dall'endpoint	Organizzazioni con MDM o gestione degli endpoint	Impostazioni distribuite ai dispositivi tramite profili di configurazione MDM, criteri del registro, o file di impostazioni gestiti

Se i vostri dispositivi sono registrati in una soluzione MDM o di gestione degli endpoint, le impostazioni gestite dall'endpoint forniscono garanzie di sicurezza più forti perché il file di impostazioni può essere protetto dalla modifica dell'utente a livello del sistema operativo.

Configurare le impostazioni gestite dal server

Step 1: Aprire la console di amministrazione

In Claude.ai, navigate a **Admin Settings > Claude Code > Managed settings**.

Step 2: Definire le impostazioni

Aggiungere la configurazione come JSON. Tutte le impostazioni disponibili in `settings.json` sono supportate, incluse le impostazioni solo gestite come `disableBypassPermissionsMode`.

Questo esempio applica un elenco di negazione delle autorizzazioni e impedisce agli utenti di aggirare le autorizzazioni:

```
{
  "permissions": {
    "deny": [
      "Bash(curl *)",
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(./secrets/**)"
    ]
  },
  "disableBypassPermissionsMode": "disable"
}
```

Step 3: Salvare e distribuire

Salvare le modifiche. I client di Claude Code ricevono le impostazioni aggiornate al prossimo avvio o ciclo di polling orario.

Verificare la consegna delle impostazioni

Per confermare che le impostazioni vengono applicate, chiedere a un utente di riavviare Claude Code. Se la configurazione include impostazioni che attivano la finestra di dialogo di approvazione della sicurezza, l'utente vede un prompt che descrive le impostazioni gestite all'avvio. È inoltre possibile verificare che le regole di autorizzazione gestite siano attive facendo eseguire a un utente `/permissions` per visualizzare le regole di autorizzazione effettive.

Controllo di accesso

I seguenti ruoli possono gestire le impostazioni gestite dal server:

- **Primary Owner**
- **Owner**

Limitare l'accesso al personale di fiducia, poiché le modifiche alle impostazioni si applicano a tutti gli utenti dell'organizzazione.

Limitazioni attuali

Le impostazioni gestite dal server hanno le seguenti limitazioni durante il periodo beta:

- Le impostazioni si applicano uniformemente a tutti gli utenti dell'organizzazione. Le configurazioni per gruppo non sono ancora supportate.

- Le configurazioni MCP server non possono essere distribuite tramite impostazioni gestite dal server.

Consegna delle impostazioni

Precedenza delle impostazioni

Le impostazioni gestite dal server e le impostazioni gestite dall'endpoint occupano entrambe il livello più alto nella gerarchia delle impostazioni di Claude Code. Nessun altro livello di impostazioni può sostituirle, inclusi gli argomenti della riga di comando. Quando entrambe sono presenti, le impostazioni gestite dal server hanno la precedenza e le impostazioni gestite dall'endpoint non vengono utilizzate.

Comportamento di recupero e caching

Claude Code recupera le impostazioni dai server di Anthropic all'avvio e esegue il polling per gli aggiornamenti ogni ora durante le sessioni attive.

Primo avvio senza impostazioni memorizzate nella cache:

- Claude Code recupera le impostazioni in modo asincrono
- Se il recupero non riesce, Claude Code continua senza impostazioni gestite
- C'è una breve finestra prima che le impostazioni si carichino in cui le restrizioni non sono ancora applicate

Avvii successivi con impostazioni memorizzate nella cache:

- Le impostazioni memorizzate nella cache si applicano immediatamente all'avvio
- Claude Code recupera le impostazioni aggiornate in background
- Le impostazioni memorizzate nella cache persistono attraverso i guasti di rete

Claude Code applica gli aggiornamenti delle impostazioni automaticamente senza un riavvio, ad eccezione delle impostazioni avanzate come la configurazione di OpenTelemetry, che richiedono un riavvio completo per avere effetto.

Finestre di dialogo di approvazione della sicurezza

Determinate impostazioni che potrebbero comportare rischi di sicurezza richiedono l'approvazione esplicita dell'utente prima di essere applicate:

- **Impostazioni dei comandi shell**: impostazioni che eseguono comandi shell
- **Variabili di ambiente personalizzate**: variabili non nell'elenco di sicurezza noto
- **Configurazioni hook**: qualsiasi definizione di hook

Quando queste impostazioni sono presenti, gli utenti vedono una finestra di dialogo di sicurezza che spiega cosa viene configurato. Gli utenti devono approvare per procedere. Se un utente rifiuta le impostazioni, Claude Code esce.

> **Note:**
>
> In modalità non interattiva con il flag `-p`, Claude Code salta le finestre di dialogo di sicurezza e applica le impostazioni senza approvazione dell'utente.

Disponibilità della piattaforma

Le impostazioni gestite dal server richiedono una connessione diretta a `api.anthropic.com` e non sono disponibili quando si utilizzano provider di modelli di terze parti:

- Amazon Bedrock
- Google Vertex AI
- Microsoft Foundry
- Endpoint API personalizzati tramite `ANTHROPIC_BASE_URL` o gateway LLM

Registrazione di audit

Gli eventi del registro di audit per le modifiche alle impostazioni sono disponibili tramite l'API di conformità o l'esportazione del registro di audit. Contattare il vostro team di account Anthropic per l'accesso.

Gli eventi di audit includono il tipo di azione eseguita, l'account e il dispositivo che ha eseguito l'azione, e riferimenti ai valori precedenti e nuovi.

Considerazioni sulla sicurezza

Le impostazioni gestite dal server forniscono l'applicazione centralizzata delle politiche, ma operano come un controllo lato client. Su dispositivi non gestiti, gli utenti con accesso amministratore o sudo possono modificare il binario di Claude Code, il filesystem, o la configurazione di rete.

Scenario	Comportamento
L'utente modifica il file di impostazioni memorizzato nella cache	Il file manomesso si applica all'avvio, ma le impostazioni corrette si ripristinano al prossimo recupero dal server

Scenario	Comportamento
L'utente elimina il file di impostazioni memorizzato nella cache	Si verifica il comportamento del primo avvio: le impostazioni vengono recuperate in modo asincrono con una breve finestra non applicata
L'API non è disponibile	Le impostazioni memorizzate nella cache si applicano se disponibili, altrimenti le impostazioni gestite non vengono applicate fino al prossimo recupero riuscito
L'utente si autentica con un'organizzazione diversa	Le impostazioni non vengono consegnate per gli account al di fuori dell'organizzazione gestita
L'utente imposta un `ANTHROPIC_BASE_URL` non predefinito	Le impostazioni gestite dal server vengono ignorate quando si utilizzano provider API di terze parti

Per rilevare le modifiche della configurazione in fase di esecuzione, utilizzare gli hook `ConfigChange` per registrare le modifiche o bloccare le modifiche non autorizzate prima che abbiano effetto.

Per garanzie di applicazione più forti, utilizzare le impostazioni gestite dall'endpoint su dispositivi registrati in una soluzione MDM.

Vedere anche

Pagine correlate per la gestione della configurazione di Claude Code:

- Settings: riferimento di configurazione completo incluse tutte le impostazioni disponibili
- Endpoint-managed settings: impostazioni gestite distribuite ai dispositivi da IT
- Authentication: configurare l'accesso degli utenti a Claude Code
- Security: salvaguardie di sicurezza e best practice

Part 11: Cloud Providers

Claude Code su Amazon Bedrock

> *Scopri come configurare Claude Code tramite Amazon Bedrock, inclusa la configurazione, la configurazione IAM e la risoluzione dei problemi.*

Prerequisiti

Prima di configurare Claude Code con Bedrock, assicurati di avere:

- Un account AWS con accesso a Bedrock abilitato
- Accesso ai modelli Claude desiderati (ad esempio, Claude Sonnet 4.6) in Bedrock
- AWS CLI installato e configurato (facoltativo - necessario solo se non hai un altro meccanismo per ottenere le credenziali)
- Autorizzazioni IAM appropriate

> **Note:**
>
> Se stai distribuendo Claude Code a più utenti, fissa le versioni del tuo modello per evitare interruzioni quando Anthropic rilascia nuovi modelli.

Configurazione

1. Invia i dettagli del caso d'uso

I nuovi utenti dei modelli Anthropic devono inviare i dettagli del caso d'uso prima di invocare un modello. Questa operazione viene eseguita una sola volta per account.

1. Assicurati di avere le giuste autorizzazioni IAM (vedi ulteriori informazioni di seguito)
2. Accedi alla console di Amazon Bedrock
3. Seleziona **Chat/Text playground**
4. Scegli un modello Anthropic qualsiasi e ti verrà chiesto di compilare il modulo del caso d'uso

2. Configura le credenziali AWS

Claude Code utilizza la catena di credenziali predefinita di AWS SDK. Configura le tue credenziali utilizzando uno di questi metodi:

Opzione A: Configurazione AWS CLI

```
aws configure
```

Opzione B: Variabili di ambiente (chiave di accesso)

```
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
```

Opzione C: Variabili di ambiente (profilo SSO)

```
aws sso login --profile=<your-profile-name>

export AWS_PROFILE=your-profile-name
```

Opzione D: Credenziali della console di gestione AWS

```
aws login
```

Scopri di più su `aws login`.

Opzione E: Chiavi API Bedrock

```
export AWS_BEARER_TOKEN_BEDROCK=your-bedrock-api-key
```

Le chiavi API Bedrock forniscono un metodo di autenticazione più semplice senza la necessità di credenziali AWS complete. Scopri di più sulle chiavi API Bedrock.

Configurazione avanzata delle credenziali

Claude Code supporta l'aggiornamento automatico delle credenziali per AWS SSO e provider di identità aziendali. Aggiungi queste impostazioni al file di impostazioni di Claude Code (vedi Impostazioni per i percorsi dei file).

Quando Claude Code rileva che le tue credenziali AWS sono scadute (sia localmente in base al loro timestamp che quando Bedrock restituisce un errore di credenziale), eseguirà automaticamente i tuoi comandi `awsAuthRefresh` e/o `awsCredentialExport` configurati per ottenere nuove credenziali prima di riprovare la richiesta.

Configurazione di esempio

```
{
  "awsAuthRefresh": "aws sso login --profile myprofile",
  "env": {
    "AWS_PROFILE": "myprofile"
  }
}
```

Impostazioni di configurazione spiegate

`awsAuthRefresh` : Usa questo per i comandi che modificano la directory `.aws` , come l'aggiornamento delle credenziali, della cache SSO o dei file di configurazione. L'output del comando viene visualizzato all'utente, ma l'input interattivo non è supportato. Funziona bene per i flussi SSO basati su browser in cui la CLI visualizza un URL o un codice e completi l'autenticazione nel browser.

`awsCredentialExport` : Usa questo solo se non puoi modificare `.aws` e devi restituire direttamente le credenziali. L'output viene acquisito silenziosamente e non mostrato all'utente. Il comando deve restituire JSON in questo formato:

```
{
  "Credentials": {
    "AccessKeyId": "value",
    "SecretAccessKey": "value",
    "SessionToken": "value"
  }
}
```

3. Configura Claude Code

Imposta le seguenti variabili di ambiente per abilitare Bedrock:

```
## Abilita integrazione Bedrock
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1  # o la tua regione preferita

## Facoltativo: Sovrascrivi la regione per il modello piccolo/veloce (Haiku)
export ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION=us-west-2
```

Quando abiliti Bedrock per Claude Code, tieni presente quanto segue:

- `AWS_REGION` è una variabile di ambiente obbligatoria. Claude Code non legge dal file di configurazione `.aws` per questa impostazione.
- Quando si utilizza Bedrock, i comandi `/login` e `/logout` sono disabilitati poiché l'autenticazione viene gestita tramite credenziali AWS.
- Puoi utilizzare file di impostazioni per variabili di ambiente come `AWS_PROFILE` che non desideri perdere in altri processi. Vedi Impostazioni per ulteriori informazioni.

4. Fissa le versioni del modello

> **Warning:**
>
> Fissa versioni specifiche del modello per ogni distribuzione. Se utilizzi alias di modello (`sonnet` , `opus` , `haiku`) senza fissare, Claude Code potrebbe tentare di utilizzare una versione di modello più recente che non è disponibile nel tuo account Bedrock, interrompendo gli utenti esistenti quando Anthropic rilascia aggiornamenti.

Imposta queste variabili di ambiente su ID di modello Bedrock specifici:

```
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'
```

Queste variabili utilizzano ID di profili di inferenza tra regioni (con il prefisso `us.`). Se utilizzi un prefisso di regione diverso o profili di inferenza dell'applicazione, regola di conseguenza. Per gli ID di modello attuali e legacy, vedi Panoramica dei modelli. Vedi Configurazione del modello per l'elenco completo delle variabili di ambiente.

Claude Code utilizza questi modelli predefiniti quando non sono impostate variabili di fissaggio:

Tipo di modello	Valore predefinito
Modello primario	`global.anthropic.claude-sonnet-4-6`
Modello piccolo/veloce	`us.anthropic.claude-haiku-4-5-20251001-v1:0`

Per personalizzare ulteriormente i modelli, utilizza uno di questi metodi:

```
## Utilizzo dell'ID del profilo di inferenza
export ANTHROPIC_MODEL='global.anthropic.claude-sonnet-4-6'
export ANTHROPIC_SMALL_FAST_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'

## Utilizzo dell'ARN del profilo di inferenza dell'applicazione
export ANTHROPIC_MODEL='arn:aws:bedrock:us-east-2:your-account-id:application-
inference-profile/your-model-id'

## Facoltativo: Disabilita il caching dei prompt se necessario
export DISABLE_PROMPT_CACHING=1
```

> **Note:**
>
> Prompt caching potrebbe non essere disponibile in tutte le regioni.

Mappa ogni versione del modello a un profilo di inferenza

Le variabili di ambiente `ANTHROPIC_DEFAULT_*_MODEL` configurano un profilo di inferenza per famiglia di modelli. Se la tua organizzazione ha bisogno di esporre diverse versioni della stessa famiglia nel selettore `/model`, ciascuna instradato al suo ARN del profilo di inferenza dell'applicazione, utilizza invece l'impostazione `modelOverrides` nel tuo file di impostazioni.

Questo esempio mappa tre versioni di Opus a ARN distinti in modo che gli utenti possano passare da uno all'altro senza aggirare i profili di inferenza della tua organizzazione:

```
{
  "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"
  }
}
```

Quando un utente seleziona una di queste versioni in `/model`, Claude Code chiama Bedrock con l'ARN mappato. Le versioni senza un override tornano all'ID del modello Bedrock integrato o a qualsiasi profilo di inferenza corrispondente scoperto all'avvio. Vedi Sovrascrivi ID di modello per versione per i dettagli su come gli override interagiscono con `availableModels` e altre impostazioni del modello.

Configurazione IAM

Crea una policy IAM con le autorizzazioni richieste per Claude Code:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowModelAndInferenceProfileAccess",
      "Effect": "Allow",
      "Action": [
        "bedrock:InvokeModel",
        "bedrock:InvokeModelWithResponseStream",
        "bedrock:ListInferenceProfiles"
      ],
      "Resource": [
        "arn:aws:bedrock:*:*:inference-profile/*",
        "arn:aws:bedrock:*:*:application-inference-profile/*",
        "arn:aws:bedrock:*:*:foundation-model/*"
      ]
    },
    {
      "Sid": "AllowMarketplaceSubscription",
      "Effect": "Allow",
      "Action": [
        "aws-marketplace:ViewSubscriptions",
        "aws-marketplace:Subscribe"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "bedrock.amazonaws.com"
        }
      }
    }
  ]
}
```

Per autorizzazioni più restrittive, puoi limitare la Resource a ARN di profili di inferenza specifici.

Per i dettagli, vedi Documentazione IAM di Bedrock.

> **Note:**
>
> Crea un account AWS dedicato per Claude Code per semplificare il tracciamento dei costi e il controllo degli accessi.

AWS Guardrails

Amazon Bedrock Guardrails ti consente di implementare il filtro dei contenuti per Claude Code. Crea un Guardrail nella console di Amazon Bedrock, pubblica una versione, quindi aggiungi le intestazioni Guardrail al tuo file di impostazioni. Abilita l'inferenza tra regioni sul tuo Guardrail se stai utilizzando profili di inferenza tra regioni.

Configurazione di esempio:

```
{
  "env": {
    "ANTHROPIC_CUSTOM_HEADERS": "X-Amzn-Bedrock-GuardrailIdentifier: your-
guardrail-id\nX-Amzn-Bedrock-GuardrailVersion: 1"
  }
}
```

Risoluzione dei problemi

Se riscontri problemi di regione:

- Controlla la disponibilità del modello: `aws bedrock list-inference-profiles --region your-region`
- Passa a una regione supportata: `export AWS_REGION=us-east-1`
- Considera l'utilizzo di profili di inferenza per l'accesso tra regioni

Se ricevi un errore "on-demand throughput isn't supported":

- Specifica il modello come ID di profilo di inferenza

Claude Code utilizza l'API Bedrock Invoke e non supporta l'API Converse.

Risorse aggiuntive

- Documentazione di Bedrock
- Prezzi di Bedrock
- Profili di inferenza di Bedrock

- Claude Code su Amazon Bedrock: Guida di configurazione rapida
- Implementazione del monitoraggio di Claude Code (Bedrock)

Claude Code su Google Vertex AI

> *Scopri come configurare Claude Code tramite Google Vertex AI, inclusa la configurazione, la configurazione IAM e la risoluzione dei problemi.*

Prerequisiti

Prima di configurare Claude Code con Vertex AI, assicurati di avere:

- Un account Google Cloud Platform (GCP) con fatturazione abilitata
- Un progetto GCP con Vertex AI API abilitata
- Accesso ai modelli Claude desiderati (ad esempio, Claude Sonnet 4.5)
- Google Cloud SDK (`gcloud`) installato e configurato
- Quota allocata nella regione GCP desiderata

Configurazione della regione

Claude Code può essere utilizzato sia con endpoint Vertex AI globali che regionali.

> **Note:**
>
> Vertex AI potrebbe non supportare i modelli predefiniti di Claude Code in tutte le regioni. Potrebbe essere necessario passare a una regione o modello supportato.

> **Note:**
>
> Vertex AI potrebbe non supportare i modelli predefiniti di Claude Code su endpoint globali. Potrebbe essere necessario passare a un endpoint regionale o a un modello supportato.

Configurazione

1. Abilita Vertex AI API

Abilita Vertex AI API nel tuo progetto GCP:

```
## Imposta il tuo ID progetto
gcloud config set project YOUR-PROJECT-ID

## Abilita Vertex AI API
gcloud services enable aiplatform.googleapis.com
```

2. Richiedi accesso al modello

Richiedi accesso ai modelli Claude in Vertex AI:

1. Accedi a Vertex AI Model Garden
2. Cerca i modelli “Claude”
3. Richiedi accesso ai modelli Claude desiderati (ad esempio, Claude Sonnet 4.5)
4. Attendi l’approvazione (potrebbe richiedere 24-48 ore)

3. Configura le credenziali GCP

Claude Code utilizza l’autenticazione standard di Google Cloud.

Per ulteriori informazioni, consulta la documentazione di autenticazione di Google Cloud.

> **Note:**
>
> Durante l’autenticazione, Claude Code utilizzerà automaticamente l’ID progetto dalla variabile di ambiente `ANTHROPIC_VERTEX_PROJECT_ID`. Per eseguire l’override, imposta una di queste variabili di ambiente: `GCLOUD_PROJECT`, `GOOGLE_CLOUD_PROJECT` o `GOOGLE_APPLICATION_CREDENTIALS`.

4. Configura Claude Code

Imposta le seguenti variabili di ambiente:

```
## Abilita integrazione Vertex AI
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=global
export ANTHROPIC_VERTEX_PROJECT_ID=YOUR-PROJECT-ID

## Facoltativo: Disabilita prompt caching se necessario
export DISABLE_PROMPT_CACHING=1

## Quando CLOUD_ML_REGION=global, esegui l'override della regione per i modelli non supportati
export VERTEX_REGION_CLAUDE_3_5_HAIKU=us-east5

## Facoltativo: Esegui l'override delle regioni per altri modelli specifici
export VERTEX_REGION_CLAUDE_3_5_SONNET=us-east5
export VERTEX_REGION_CLAUDE_3_7_SONNET=us-east5
export VERTEX_REGION_CLAUDE_4_0_OPUS=europe-west1
export VERTEX_REGION_CLAUDE_4_0_SONNET=us-east5
export VERTEX_REGION_CLAUDE_4_1_OPUS=europe-west1
```

Note:

Prompt caching è automaticamente supportato quando specifichi il flag effimero `cache_control`. Per disabilitarlo, imposta `DISABLE_PROMPT_CACHING=1`. Per limiti di velocità aumentati, contatta il supporto di Google Cloud.

Note:

Quando si utilizza Vertex AI, i comandi `/login` e `/logout` sono disabilitati poiché l'autenticazione viene gestita tramite le credenziali di Google Cloud.

5. Configurazione del modello

Claude Code utilizza questi modelli predefiniti per Vertex AI:

Tipo di modello	Valore predefinito
Modello primario	`claude-sonnet-4-5@20250929`
Modello piccolo/veloce	`claude-haiku-4-5@20251001`

> **Note:**
>
> Per gli utenti di Vertex AI, Claude Code non eseguirà automaticamente l'aggiornamento da Haiku 3.5 a Haiku 4.5. Per passare manualmente a un modello Haiku più recente, imposta la variabile di ambiente `ANTHROPIC_DEFAULT_HAIKU_MODEL` sul nome completo del modello (ad esempio, `claude-haiku-4-5@20251001`).

Per personalizzare i modelli:

```
export ANTHROPIC_MODEL='claude-opus-4-1@20250805'
export ANTHROPIC_SMALL_FAST_MODEL='claude-haiku-4-5@20251001'
```

Configurazione IAM

Assegna i permessi IAM richiesti:

Il ruolo `roles/aiplatform.user` include i permessi richiesti:

- `aiplatform.endpoints.predict` - Richiesto per l'invocazione del modello e il conteggio dei token

Per permessi più restrittivi, crea un ruolo personalizzato con solo i permessi di cui sopra.

Per i dettagli, consulta la documentazione IAM di Vertex.

> **Note:**
>
> Consigliamo di creare un progetto GCP dedicato per Claude Code per semplificare il tracciamento dei costi e il controllo degli accessi.

Finestra di contesto da 1M token

Claude Sonnet 4 e Sonnet 4.5 supportano la finestra di contesto da 1M token su Vertex AI.

> **Note:**
>
> La finestra di contesto da 1M token è attualmente in beta. Per utilizzare la finestra di contesto estesa, includi l'intestazione beta `context-1m-2025-08-07` nelle tue richieste Vertex AI.

Risoluzione dei problemi

Se riscontri problemi di quota:

- Controlla le quote attuali o richiedi un aumento della quota tramite Cloud Console

Se riscontri errori "model not found" 404:

- Conferma che il modello sia abilitato in Model Garden
- Verifica di avere accesso alla regione specificata
- Se utilizzi `CLOUD_ML_REGION=global`, verifica che i tuoi modelli supportino endpoint globali in Model Garden sotto "Supported features". Per i modelli che non supportano endpoint globali, puoi:
  - Specificare un modello supportato tramite `ANTHROPIC_MODEL` o `ANTHROPIC_SMALL_FAST_MODEL`, oppure
  - Impostare un endpoint regionale utilizzando le variabili di ambiente `VERTEX_REGION_<MODEL_NAME>`

Se riscontri errori 429:

- Per endpoint regionali, assicurati che il modello primario e il modello piccolo/veloce siano supportati nella tua regione selezionata
- Considera di passare a `CLOUD_ML_REGION=global` per una migliore disponibilità

Risorse aggiuntive

- Documentazione di Vertex AI
- Prezzi di Vertex AI
- Quote e limiti di Vertex AI

Claude Code su Microsoft Foundry

> *Scopri come configurare Claude Code tramite Microsoft Foundry, inclusi setup, configurazione e risoluzione dei problemi.*

Prerequisiti

Prima di configurare Claude Code con Microsoft Foundry, assicurati di avere:

- Un abbonamento Azure con accesso a Microsoft Foundry
- Autorizzazioni RBAC per creare risorse e distribuzioni di Microsoft Foundry
- Azure CLI installato e configurato (facoltativo - necessario solo se non hai un altro meccanismo per ottenere le credenziali)

Setup

1. Provisioning della risorsa Microsoft Foundry

Per prima cosa, crea una risorsa Claude in Azure:

1. Accedi al portale Microsoft Foundry
2. Crea una nuova risorsa, annotando il nome della risorsa
3. Crea distribuzioni per i modelli Claude:
    - Claude Opus
    - Claude Sonnet
    - Claude Haiku

2. Configurare le credenziali Azure

Claude Code supporta due metodi di autenticazione per Microsoft Foundry. Scegli il metodo che meglio si adatta ai tuoi requisiti di sicurezza.

Opzione A: Autenticazione tramite chiave API

1. Accedi alla tua risorsa nel portale Microsoft Foundry
2. Vai alla sezione **Endpoint e chiavi**
3. Copia **Chiave API**
4. Imposta la variabile di ambiente:

```
export ANTHROPIC_FOUNDRY_API_KEY=your-azure-api-key
```

Opzione B: Autenticazione Microsoft Entra ID

Quando `ANTHROPIC_FOUNDRY_API_KEY` non è impostato, Claude Code utilizza automaticamente la catena di credenziali predefinita di Azure SDK. Questo supporta una varietà di metodi per autenticare carichi di lavoro locali e remoti.

Negli ambienti locali, puoi comunemente utilizzare Azure CLI:

```
az login
```

> **Note:**
>
> Quando si utilizza Microsoft Foundry, i comandi `/login` e `/logout` sono disabilitati poiché l'autenticazione viene gestita tramite le credenziali Azure.

3. Configurare Claude Code

Imposta le seguenti variabili di ambiente per abilitare Microsoft Foundry. Nota che i nomi delle tue distribuzioni sono impostati come identificatori di modello in Claude Code (potrebbe essere facoltativo se si utilizzano i nomi di distribuzione suggeriti).

```
## Abilita l'integrazione Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1

## Nome della risorsa Azure (sostituisci {resource} con il nome della tua risorsa)
export ANTHROPIC_FOUNDRY_RESOURCE={resource}
## Oppure fornisci l'URL di base completo:
## export ANTHROPIC_FOUNDRY_BASE_URL=https://{resource}.services.ai.azure.com

## Imposta i modelli sui nomi di distribuzione della tua risorsa
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'
```

Per ulteriori dettagli sulle opzioni di configurazione del modello, vedi Configurazione del modello.

Configurazione RBAC di Azure

I ruoli predefiniti `Azure AI User` e `Cognitive Services User` includono tutte le autorizzazioni necessarie per invocare i modelli Claude.

Per autorizzazioni più restrittive, crea un ruolo personalizzato con quanto segue:

```
{
  "permissions": [
    {
      "dataActions": [
        "Microsoft.CognitiveServices/accounts/providers/*"
      ]
    }
  ]
}
```

Per i dettagli, vedi Documentazione RBAC di Microsoft Foundry.

Risoluzione dei problemi

Se ricevi un errore “Failed to get token from azureADTokenProvider: ChainedTokenCredential authentication failed”:

- Configura Entra ID nell’ambiente, oppure imposta `ANTHROPIC_FOUNDRY_API_KEY`.

Risorse aggiuntive

- Documentazione di Microsoft Foundry
- Modelli di Microsoft Foundry
- Prezzi di Microsoft Foundry

Configurazione del gateway LLM

> *Scopri come configurare Claude Code per funzionare con soluzioni di gateway LLM. Copre i requisiti del gateway, la configurazione dell'autenticazione, la selezione del modello e la configurazione degli endpoint specifici del provider.*

I gateway LLM forniscono un livello proxy centralizzato tra Claude Code e i provider di modelli, spesso fornendo:

- **Autenticazione centralizzata** - Punto singolo per la gestione delle chiavi API
- **Tracciamento dell'utilizzo** - Monitora l'utilizzo tra team e progetti
- **Controlli dei costi** - Implementa budget e limiti di velocità
- **Registrazione di audit** - Traccia tutte le interazioni del modello per la conformità
- **Instradamento dei modelli** - Passa da un provider all'altro senza modifiche al codice

Requisiti del gateway

Affinché un gateway LLM funzioni con Claude Code, deve soddisfare i seguenti requisiti:

Formato API

Il gateway deve esporre ai client almeno uno dei seguenti formati API:

1. **Anthropic Messages**: `/v1/messages`, `/v1/messages/count_tokens`
   - Deve inoltrare le intestazioni della richiesta: `anthropic-beta`, `anthropic-version`
2. **Bedrock InvokeModel**: `/invoke`, `/invoke-with-response-stream`
   - Deve preservare i campi del corpo della richiesta: `anthropic_beta`, `anthropic_version`
3. **Vertex rawPredict**: `:rawPredict`, `:streamRawPredict`, `/count-tokens:rawPredict`
   - Deve inoltrare le intestazioni della richiesta: `anthropic-beta`, `anthropic-version`

Il mancato inoltro delle intestazioni o la mancata preservazione dei campi del corpo potrebbe causare una riduzione della funzionalità o l'impossibilità di utilizzare le funzionalità di Claude Code.

> **Note:**

> Claude Code determina quali funzionalità abilitare in base al formato API. Quando si utilizza il formato Anthropic Messages con Bedrock o Vertex, potrebbe essere necessario impostare la variabile di ambiente `CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS=1`.

Configurazione

Selezione del modello

Per impostazione predefinita, Claude Code utilizzerà nomi di modelli standard per il formato API selezionato.

Se hai configurato nomi di modelli personalizzati nel tuo gateway, utilizza le variabili di ambiente documentate in Configurazione del modello per abbinare i tuoi nomi personalizzati.

Configurazione di LiteLLM

> **Note:**
>
> LiteLLM è un servizio proxy di terze parti. Anthropic non approva, mantiene o controlla la sicurezza o la funzionalità di LiteLLM. Questa guida è fornita a scopo informativo e potrebbe diventare obsoleta. Utilizzala a tua discrezione.

Prerequisiti

- Claude Code aggiornato all'ultima versione
- LiteLLM Proxy Server distribuito e accessibile
- Accesso ai modelli Claude attraverso il provider scelto

Configurazione di base di LiteLLM

Configura Claude Code:

Metodi di autenticazione

Chiave API statica

Metodo più semplice utilizzando una chiave API fissa:

```
## Imposta nell'ambiente
export ANTHROPIC_AUTH_TOKEN=sk-litellm-static-key

## O nelle impostazioni di Claude Code
{
  "env": {
    "ANTHROPIC_AUTH_TOKEN": "sk-litellm-static-key"
  }
}
```

Questo valore verrà inviato come intestazione `Authorization`.

Chiave API dinamica con helper

Per chiavi rotanti o autenticazione per utente:

1. Crea uno script helper per la chiave API:

```
#!/bin/bash
## ~/bin/get-litellm-key.sh

## Esempio: Recupera la chiave dal vault
vault kv get -field=api_key secret/litellm/claude-code

## Esempio: Genera token JWT
jwt encode \
  --secret="${JWT_SECRET}" \
  --exp="+1h" \
  '{"user":"'${USER}'","team":"engineering"}'
```

1. Configura le impostazioni di Claude Code per utilizzare l'helper:

```
{
  "apiKeyHelper": "~/bin/get-litellm-key.sh"
}
```

1. Imposta l'intervallo di aggiornamento del token:

```
## Aggiorna ogni ora (3600000 ms)
export CLAUDE_CODE_API_KEY_HELPER_TTL_MS=3600000
```

Questo valore verrà inviato come intestazioni `Authorization` e `X-Api-Key`.
L'`apiKeyHelper` ha una precedenza inferiore rispetto a `ANTHROPIC_AUTH_TOKEN` o `ANTHROPIC_API_KEY`.

Endpoint unificato (consigliato)

Utilizzando l'endpoint in formato Anthropic di LiteLLM:

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000
```

Vantaggi dell'endpoint unificato rispetto agli endpoint pass-through:

- Bilanciamento del carico
- Fallback
- Supporto coerente per il tracciamento dei costi e il tracciamento dell'utente finale

Endpoint pass-through specifici del provider (alternativa)

Claude API attraverso LiteLLM

Utilizzando l'endpoint pass-through:

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000/anthropic
```

Amazon Bedrock attraverso LiteLLM

Utilizzando l'endpoint pass-through:

```
export ANTHROPIC_BEDROCK_BASE_URL=https://litellm-server:4000/bedrock
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1
export CLAUDE_CODE_USE_BEDROCK=1
```

Google Vertex AI attraverso LiteLLM

Utilizzando l'endpoint pass-through:

```
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
```

Per informazioni più dettagliate, consulta la documentazione di LiteLLM.

Risorse aggiuntive

- Documentazione di LiteLLM
- Impostazioni di Claude Code
- Configurazione della rete aziendale
- Panoramica delle integrazioni di terze parti

Configurazione di rete aziendale

> *Configurare Claude Code per ambienti aziendali con server proxy, Autorità di Certificazione (CA) personalizzate e autenticazione Transport Layer Security (mTLS) reciproca.*

Claude Code supporta varie configurazioni di rete e sicurezza aziendali attraverso variabili di ambiente. Ciò include l'instradamento del traffico attraverso server proxy aziendali, la fiducia in Autorità di Certificazione (CA) personalizzate e l'autenticazione con certificati Transport Layer Security (mTLS) reciproco per una sicurezza migliorata.

> **Note:**
>
> Tutte le variabili di ambiente mostrate in questa pagina possono essere configurate anche in `settings.json`.

Configurazione del proxy

Variabili di ambiente

Claude Code rispetta le variabili di ambiente proxy standard:

```
## HTTPS proxy (consigliato)
export HTTPS_PROXY=https://proxy.example.com:8080

## HTTP proxy (se HTTPS non disponibile)
export HTTP_PROXY=http://proxy.example.com:8080

## Bypass proxy per richieste specifiche - formato separato da spazi
export NO_PROXY="localhost 192.168.1.1 example.com .example.com"
## Bypass proxy per richieste specifiche - formato separato da virgole
export NO_PROXY="localhost,192.168.1.1,example.com,.example.com"
## Bypass proxy per tutte le richieste
export NO_PROXY="*"
```

> **Note:**

Claude Code non supporta proxy SOCKS.

Autenticazione di base

Se il proxy richiede l'autenticazione di base, includere le credenziali nell'URL del proxy:

```
export HTTPS_PROXY=http://username:password@proxy.example.com:8080
```

Warning:

Evitare di codificare le password negli script. Utilizzare variabili di ambiente o archiviazione sicura delle credenziali.

Tip:

Per proxy che richiedono autenticazione avanzata (NTLM, Kerberos, ecc.), considerare l'utilizzo di un servizio LLM Gateway che supporti il metodo di autenticazione.

Certificati CA personalizzati

Se l'ambiente aziendale utilizza CA personalizzate per le connessioni HTTPS (sia attraverso un proxy che per l'accesso diretto all'API), configurare Claude Code per fidarsi di esse:

```
export NODE_EXTRA_CA_CERTS=/path/to/ca-cert.pem
```

Autenticazione mTLS

Per ambienti aziendali che richiedono l'autenticazione del certificato client:

```
## Certificato client per l'autenticazione
export CLAUDE_CODE_CLIENT_CERT=/path/to/client-cert.pem

## Chiave privata del client
export CLAUDE_CODE_CLIENT_KEY=/path/to/client-key.pem

## Facoltativo: Passphrase per la chiave privata crittografata
export CLAUDE_CODE_CLIENT_KEY_PASSPHRASE="your-passphrase"
```

Requisiti di accesso alla rete

Claude Code richiede l'accesso ai seguenti URL:

- `api.anthropic.com` : endpoint dell'API Claude
- `claude.ai` : autenticazione per gli account claude.ai
- `platform.claude.com` : autenticazione per gli account Anthropic Console

Assicurarsi che questi URL siano inseriti nella whitelist nella configurazione del proxy e nelle regole del firewall. Ciò è particolarmente importante quando si utilizza Claude Code in ambienti di rete containerizzati o limitati.

Risorse aggiuntive

- Impostazioni di Claude Code
- Riferimento delle variabili di ambiente
- Guida alla risoluzione dei problemi

Part 12: Environment Setup

Contenitori di sviluppo

> *Scopri il contenitore di sviluppo Claude Code per i team che necessitano di ambienti coerenti e sicuri.*

La configurazione devcontainer di riferimento e il relativo Dockerfile offrono un contenitore di sviluppo preconfigurato che puoi utilizzare così com'è o personalizzare secondo le tue esigenze. Questo devcontainer funziona con l'estensione Dev Containers di Visual Studio Code e strumenti simili.

Le misure di sicurezza avanzate del contenitore (isolamento e regole firewall) ti permettono di eseguire `claude --dangerously-skip-permissions` per ignorare i prompt di autorizzazione per l'operazione automatica.

> **Warning:**
>
> Sebbene il devcontainer fornisca protezioni sostanziali, nessun sistema è completamente immune da tutti gli attacchi. Quando eseguito con `--dangerously-skip-permissions`, i devcontainer non impediscono a un progetto dannoso di estrarre qualsiasi cosa accessibile nel devcontainer, incluse le credenziali di Claude Code. Ti consigliamo di utilizzare i devcontainer solo quando sviluppi con repository affidabili. Mantieni sempre buone pratiche di sicurezza e monitora le attività di Claude.

Caratteristiche principali

- **Node.js pronto per la produzione**: Basato su Node.js 20 con dipendenze di sviluppo essenziali
- **Sicurezza per design**: Firewall personalizzato che limita l'accesso di rete solo ai servizi necessari
- **Strumenti user-friendly**: Include git, ZSH con miglioramenti di produttività, fzf e altro ancora
- **Integrazione perfetta con VS Code**: Estensioni preconfigurate e impostazioni ottimizzate
- **Persistenza della sessione**: Preserva la cronologia dei comandi e le configurazioni tra i riavvii del contenitore

- **Funziona ovunque**: Compatibile con ambienti di sviluppo macOS, Windows e Linux

Iniziare in 4 passaggi

1. Installa VS Code e l'estensione Remote - Containers
2. Clona il repository dell'implementazione di riferimento di Claude Code
3. Apri il repository in VS Code
4. Quando richiesto, fai clic su "Reopen in Container" (o usa Command Palette: Cmd+Shift+P → "Remote-Containers: Reopen in Container")

Analisi della configurazione

La configurazione devcontainer è composta da tre componenti principali:

- **devcontainer.json**: Controlla le impostazioni del contenitore, le estensioni e i mount dei volumi
- **Dockerfile**: Definisce l'immagine del contenitore e gli strumenti installati
- **init-firewall.sh**: Stabilisce le regole di sicurezza della rete

Caratteristiche di sicurezza

Il contenitore implementa un approccio di sicurezza multi-livello con la sua configurazione firewall:

- **Controllo di accesso preciso**: Limita le connessioni in uscita solo ai domini nella whitelist (registro npm, GitHub, API Claude, ecc.)
- **Connessioni in uscita consentite**: Il firewall consente connessioni DNS e SSH in uscita
- **Politica di default-deny**: Blocca tutti gli altri accessi alla rete esterna
- **Verifica all'avvio**: Convalida le regole firewall quando il contenitore si inizializza
- **Isolamento**: Crea un ambiente di sviluppo sicuro separato dal tuo sistema principale

Opzioni di personalizzazione

La configurazione devcontainer è progettata per essere adattabile alle tue esigenze:

- Aggiungi o rimuovi estensioni di VS Code in base al tuo flusso di lavoro
- Modifica le allocazioni di risorse per diversi ambienti hardware
- Regola le autorizzazioni di accesso alla rete
- Personalizza le configurazioni della shell e gli strumenti per sviluppatori

Esempi di casi d’uso

Lavoro sicuro con i clienti

Utilizza i devcontainer per isolare diversi progetti client, assicurando che il codice e le credenziali non si mescolino mai tra gli ambienti.

Onboarding del team

I nuovi membri del team possono ottenere un ambiente di sviluppo completamente configurato in pochi minuti, con tutti gli strumenti e le impostazioni necessarie preinstallate.

Ambienti CI/CD coerenti

Rispecchia la configurazione del tuo devcontainer nelle pipeline CI/CD per assicurare che gli ambienti di sviluppo e produzione corrispondano.

Risorse correlate

- Documentazione devcontainer di VS Code
- Migliori pratiche di sicurezza di Claude Code
- Configurazione della rete aziendale

Ottimizza la configurazione del tuo terminale

> *Claude Code funziona al meglio quando il tuo terminale è correttamente configurato. Segui queste linee guida per ottimizzare la tua esperienza.*

Temi e aspetto

Claude non può controllare il tema del tuo terminale. Questo è gestito dall'applicazione del tuo terminale. Puoi far corrispondere il tema di Claude Code al tuo terminale in qualsiasi momento tramite il comando `/config`.

Per ulteriori personalizzazioni dell'interfaccia di Claude Code stesso, puoi configurare una linea di stato personalizzata per visualizzare informazioni contestuali come il modello corrente, la directory di lavoro o il ramo git nella parte inferiore del tuo terminale.

Interruzioni di riga

Hai diverse opzioni per inserire interruzioni di riga in Claude Code:

- **Escape rapido**: Digita `\` seguito da Invio per creare una nuova riga
- **Shift+Invio**: Funziona immediatamente in iTerm2, WezTerm, Ghostty e Kitty
- **Scorciatoia da tastiera**: Configura un keybinding per inserire una nuova riga in altri terminali

Configura Shift+Invio per altri terminali

Esegui `/terminal-setup` all'interno di Claude Code per configurare automaticamente Shift+Invio per VS Code, Alacritty, Zed e Warp.

> **Note:**
>
> Il comando `/terminal-setup` è visibile solo nei terminali che richiedono una configurazione manuale. Se stai utilizzando iTerm2, WezTerm, Ghostty o Kitty, non vedrai questo comando perché Shift+Invio funziona già nativamente.

Configura Option+Invio (VS Code, iTerm2 o macOS Terminal.app)

Per Mac Terminal.app:

1. Apri Impostazioni → Profili → Tastiera
2. Seleziona "Usa Option come Meta Key"

Per il terminale iTerm2 e VS Code:

1. Apri Impostazioni → Profili → Tasti
2. In Generale, imposta il tasto Option sinistro/destro su “Esc+”

Configurazione delle notifiche

Non perdere mai quando Claude completa un’attività con una corretta configurazione delle notifiche:

Notifiche di sistema iTerm 2

Per gli avvisi di iTerm 2 quando le attività si completano:

1. Apri Preferenze di iTerm 2
2. Vai a Profili → Terminale
3. Abilita “Silence bell” e Filtra avvisi → “Invia avvisi generati da sequenza di escape”
4. Imposta il ritardo di notifica preferito

Nota che queste notifiche sono specifiche di iTerm 2 e non disponibili nel Terminal predefinito di macOS.

Hook di notifica personalizzati

Per la gestione avanzata delle notifiche, puoi creare hook di notifica per eseguire la tua logica personalizzata.

Gestione di input di grandi dimensioni

Quando lavori con codice esteso o istruzioni lunghe:

- **Evita l’incollamento diretto**: Claude Code potrebbe avere difficoltà con contenuti incollati molto lunghi
- **Usa flussi di lavoro basati su file**: Scrivi il contenuto in un file e chiedi a Claude di leggerlo
- **Sii consapevole delle limitazioni di VS Code**: Il terminale di VS Code è particolarmente soggetto al troncamento di incollamenti lunghi

Modalità Vim

Claude Code supporta un sottoinsieme di keybinding Vim che può essere abilitato con `/vim` o configurato tramite `/config`.

Il sottoinsieme supportato include:

- Cambio di modalità: `Esc` (a NORMAL), `i` / `I`, `a` / `A`, `o` / `O` (a INSERT)

- Navigazione: `h` / `j` / `k` / `l`, `w` / `e` / `b`, `0` / `$` / `^`, `gg` / `G`, `f` / `F` / `t` / `T` con ripetizione `;` / `,`
- Modifica: `x`, `dw` / `de` / `db` / `dd` / `D`, `cw` / `ce` / `cb` / `cc` / `C`, `.` (ripeti)
- Copia/incolla: `yy` / `Y`, `yw` / `ye` / `yb`, `p` / `P`
- Oggetti di testo: `iw` / `aw`, `iW` / `aW`, `i"` / `a"`, `i'` / `a'`, `i(` / `a(`, `i[` / `a[`, `i{` / `a{`
- Indentazione: `>>` / `<<`
- Operazioni di riga: `J` (unisci righe)

Vedi Modalità interattiva per il riferimento completo.

Personalizza la tua barra di stato

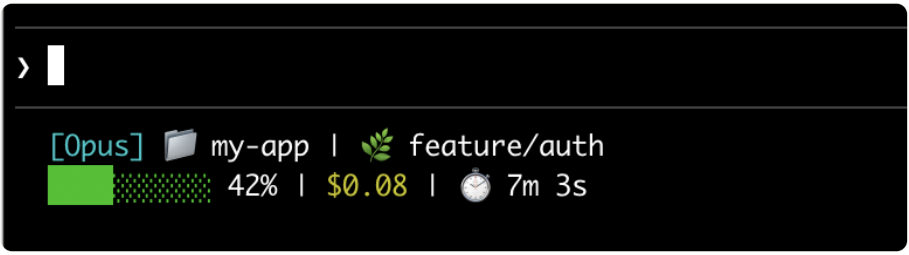
> *Configura una barra di stato personalizzata per monitorare l'utilizzo della finestra di contesto, i costi e lo stato git in Claude Code*

La barra di stato è una barra personalizzabile nella parte inferiore di Claude Code che esegue qualsiasi script di shell che configuri. Riceve dati di sessione JSON su stdin e visualizza tutto ciò che il tuo script stampa, fornendoti una visualizzazione persistente e immediata dell'utilizzo del contesto, dei costi, dello stato git o di qualsiasi altra cosa tu voglia tracciare.

Le barre di stato sono utili quando:

- Vuoi monitorare l'utilizzo della finestra di contesto mentre lavori
- Hai bisogno di tracciare i costi della sessione
- Lavori su più sessioni e hai bisogno di distinguerle
- Vuoi che il ramo git e lo stato siano sempre visibili

Ecco un esempio di una barra di stato multi-riga che visualizza le informazioni git sulla prima riga e una barra di contesto codificata a colori sulla seconda.



Una barra di stato multi-riga che mostra il nome del modello, la directory, il ramo git sulla prima riga, e una barra di progresso dell'utilizzo del contesto con costo e durata sulla seconda riga

Questa pagina illustra come configurare una barra di stato di base, spiega come fluiscono i dati da Claude Code al tuo script, elenca tutti i campi che puoi visualizzare, e fornisce esempi pronti all'uso per modelli comuni come lo stato git, il tracciamento dei costi e le barre di progresso.

Configura una barra di stato

Usa il comando `/statusline` per far generare uno script a Claude Code, oppure crea manualmente uno script e aggiungilo alle tue impostazioni.

Usa il comando /statusline

Il comando `/statusline` accetta istruzioni in linguaggio naturale che descrivono cosa vuoi visualizzare. Claude Code genera un file di script in `~/.claude/` e aggiorna automaticamente le tue impostazioni:

```
/statusline show model name and context percentage with a progress bar
```

Configura manualmente una barra di stato

Aggiungi un campo `statusLine` alle tue impostazioni utente (`~/.claude/settings.json` , dove `~` è la tua directory home) o alle impostazioni del progetto. Imposta `type` su `"command"` e punta `command` a un percorso di script o a un comando di shell inline. Per una procedura dettagliata sulla creazione di uno script, vedi Costruisci una barra di stato passo dopo passo.

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh",
    "padding": 2
  }
}
```

Il campo `command` viene eseguito in una shell, quindi puoi anche usare comandi inline invece di un file di script. Questo esempio usa `jq` per analizzare l'input JSON e visualizzare il nome del modello e la percentuale di contesto:

```
{
  "statusLine": {
    "type": "command",
    "command": "jq -r '\"[\\(.model.display_name)] \\(.context_window.used_percent
age // 0)% context\"'"
  }
}
```

Il campo opzionale `padding` aggiunge spazi orizzontali extra (in caratteri) al contenuto della barra di stato. Il valore predefinito è `0`. Questo padding è in aggiunta alla spaziatura integrata dell'interfaccia, quindi controlla l'indentazione relativa piuttosto che la distanza assoluta dal bordo del terminale.

Disabilita la barra di stato

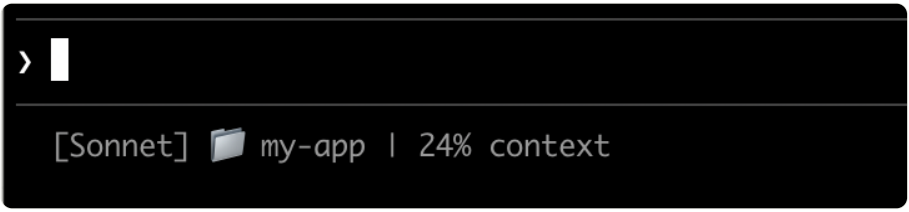
Esegui `/statusline` e chiedigli di rimuovere o cancellare la tua barra di stato (ad esempio, `/statusline delete`, `/statusline clear`, `/statusline remove it`). Puoi anche eliminare manualmente il campo `statusLine` dal tuo settings.json.

Costruisci una barra di stato passo dopo passo

Questa procedura mostra cosa sta accadendo dietro le quinte creando manualmente una barra di stato che visualizza il modello corrente, la directory di lavoro e la percentuale di utilizzo della finestra di contesto.

> **Note:**
>
> Eseguire `/statusline` con una descrizione di quello che vuoi configura tutto questo automaticamente per te.

Questi esempi usano script Bash, che funzionano su macOS e Linux. Su Windows, vedi Configurazione Windows per esempi PowerShell e Git Bash.



Una barra di stato che mostra il nome del modello, la directory e la percentuale di contesto

Step 1: Crea uno script che legge JSON e stampa l'output

Claude Code invia dati JSON al tuo script tramite stdin. Questo script usa `jq`, un parser JSON da riga di comando che potrebbe essere necessario installare, per estrarre il nome del modello, la directory e la percentuale di contesto, quindi stampa una riga formattata.

Salva questo in `~/.claude/statusline.sh` (dove `~` è la tua directory home, come `/Users/username` su macOS o `/home/username` su Linux):

```
#!/bin/bash
## Read JSON data that Claude Code sends to stdin
input=$(cat)

## Extract fields using jq
MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
## The "// 0" provides a fallback if the field is null
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## Output the status line - ${DIR##*/} extracts just the folder name
echo "[$MODEL] 📁 ${DIR##*/} | ${PCT}% context"
```

Step 2: Rendilo eseguibile

Contrassegna lo script come eseguibile in modo che la tua shell possa eseguirlo:

```
chmod +x ~/.claude/statusline.sh
```

Step 3: Aggiungi alle impostazioni

Dì a Claude Code di eseguire il tuo script come barra di stato. Aggiungi questa configurazione a `~/.claude/settings.json`, che imposta `type` su `"command"` (che significa "esegui questo comando di shell") e punta `command` al tuo script:

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh"
  }
}
```

La tua barra di stato appare nella parte inferiore dell'interfaccia. Le impostazioni si ricaricano automaticamente, ma le modifiche non appariranno fino alla tua prossima interazione con Claude Code.

Come funzionano le barre di stato

Claude Code esegue il tuo script e invia i dati di sessione JSON ad esso tramite stdin. Il tuo script legge il JSON, estrae quello di cui ha bisogno e stampa il testo su stdout. Claude Code visualizza tutto ciò che il tuo script stampa.

Quando si aggiorna

Il tuo script viene eseguito dopo ogni nuovo messaggio dell'assistente, quando cambia la modalità di autorizzazione o quando la modalità vim si attiva/disattiva. Gli aggiornamenti sono debounced a 300ms, il che significa che i cambiamenti rapidi si raggruppano insieme e il tuo script viene eseguito una volta che le cose si stabilizzano. Se un nuovo aggiornamento si attiva mentre il tuo script è ancora in esecuzione, l'esecuzione in corso viene annullata. Se modifichi il tuo script, le modifiche non appariranno fino al prossimo aggiornamento attivato da un'interazione con Claude Code.

Cosa può produrre il tuo script

- **Più righe**: ogni istruzione `echo` o `print` viene visualizzata come una riga separata. Vedi l'esempio multi-riga.
- **Colori**: usa codici di escape ANSI come `\033[32m` per il verde (il terminale deve supportarli). Vedi l'esempio di stato git.
- **Link**: usa sequenze di escape OSC 8 per rendere il testo cliccabile (Cmd+clic su macOS, Ctrl+clic su Windows/Linux). Richiede un terminale che supporti i hyperlink come iTerm2, Kitty o WezTerm. Vedi l'esempio di link cliccabili.

> **Note:**

> La barra di stato viene eseguita localmente e non consuma token API. Si nasconde temporaneamente durante determinate interazioni dell'interfaccia utente, inclusi i suggerimenti di completamento automatico, il menu della guida e i prompt di autorizzazione.

Dati disponibili

Claude Code invia i seguenti campi JSON al tuo script tramite stdin:

Campo	Descrizione
`model.id`, `model.display_name`	Identificatore del modello corrente e nome visualizzato
`cwd`, `workspace.current_dir`	Directory di lavoro corrente. Entrambi i campi contengono lo stesso valore; `workspace.current_dir` è preferito per coerenza con `workspace.project_dir`.
`workspace.project_dir`	Directory in cui Claude Code è stato avviato, che potrebbe differire da `cwd` se la directory di lavoro cambia durante una sessione
`cost.total_cost_usd`	Costo totale della sessione in USD
`cost.total_duration_ms`	Tempo totale trascorso dal momento dell'avvio della sessione, in millisecondi
`cost.total_api_duration_ms`	Tempo totale trascorso in attesa delle risposte API in millisecondi
`cost.total_lines_added`, `cost.total_lines_removed`	Righe di codice modificate
`context_window.total_input_tokens`, `context_window.total_output_tokens`	Conteggi cumulativi dei token nella sessione
`context_window.context_window_size`	Dimensione massima della finestra di contesto in token. 200000 per impostazione predefinita, o 1000000 per i modelli con contesto esteso.
`context_window.used_percentage`	Percentuale pre-calcolata della finestra di contesto utilizzata
`context_window.remaining_percentage`	Percentuale pre-calcolata della finestra di contesto rimanente

Campo	Descrizione
`context_window.current_usage`	Conteggi dei token dall'ultima chiamata API, descritti in campi della finestra di contesto
`exceeds_200k_tokens`	Se il conteggio totale dei token (token di input, cache e output combinati) dalla risposta API più recente supera 200k. Questo è un limite fisso indipendentemente dalla dimensione effettiva della finestra di contesto.
`session_id`	Identificatore univoco della sessione
`transcript_path`	Percorso del file di trascrizione della conversazione
`version`	Versione di Claude Code
`output_style.name`	Nome dello stile di output corrente
`vim.mode`	Modalità vim corrente (`NORMAL` o `INSERT`) quando la modalità vim è abilitata
`agent.name`	Nome dell'agente quando si esegue con il flag `--agent` o le impostazioni dell'agente configurate
`worktree.name`	Nome del worktree attivo. Presente solo durante le sessioni `--worktree`
`worktree.path`	Percorso assoluto della directory del worktree
`worktree.branch`	Nome del ramo git per il worktree (ad esempio, `"worktree-my-feature"`). Assente per i worktree basati su hook
`worktree.original_cwd`	La directory in cui Claude si trovava prima di entrare nel worktree
`worktree.original_branch`	Ramo git estratto prima di entrare nel worktree. Assente per i worktree basati su hook

Il tuo comando della barra di stato riceve questa struttura JSON tramite stdin:

```
{
  "cwd": "/current/working/directory",
  "session_id": "abc123...",
  "transcript_path": "/path/to/transcript.jsonl",
  "model": {
    "id": "claude-opus-4-6",
    "display_name": "Opus"
  },
  "workspace": {
    "current_dir": "/current/working/directory",
    "project_dir": "/original/project/directory"
  },
  "version": "1.0.80",
  "output_style": {
    "name": "default"
  },
  "cost": {
    "total_cost_usd": 0.01234,
    "total_duration_ms": 45000,
    "total_api_duration_ms": 2300,
    "total_lines_added": 156,
    "total_lines_removed": 23
  },
  "context_window": {
    "total_input_tokens": 15234,
    "total_output_tokens": 4521,
    "context_window_size": 200000,
    "used_percentage": 8,
    "remaining_percentage": 92,
    "current_usage": {
      "input_tokens": 8500,
      "output_tokens": 1200,
      "cache_creation_input_tokens": 5000,
      "cache_read_input_tokens": 2000
    }
  },
  "exceeds_200k_tokens": false,
  "vim": {
```

```
    "mode": "NORMAL"
  },
  "agent": {
    "name": "security-reviewer"
  },
  "worktree": {
    "name": "my-feature",
    "path": "/path/to/.claude/worktrees/my-feature",
    "branch": "worktree-my-feature",
    "original_cwd": "/path/to/project",
    "original_branch": "main"
  }
}
```

Campi che potrebbero essere assenti (non presenti in JSON):

- `vim` : appare solo quando la modalità vim è abilitata
- `agent` : appare solo quando si esegue con il flag `--agent` o le impostazioni dell'agente configurate
- `worktree` : appare solo durante le sessioni `--worktree` . Quando presente, `branch` e `original_branch` potrebbero anche essere assenti per i worktree basati su hook

Campi che potrebbero essere `null` :

- `context_window.current_usage` : `null` prima della prima chiamata API in una sessione
- `context_window.used_percentage` , `context_window.remaining_percentage` : potrebbero essere `null` all'inizio della sessione

Gestisci i campi mancanti con accesso condizionale e i valori null con fallback predefiniti nei tuoi script.

Campi della finestra di contesto

L'oggetto `context_window` fornisce due modi per tracciare l'utilizzo del contesto:

- **Totali cumulativi** (`total_input_tokens` , `total_output_tokens`): somma di tutti i token nell'intera sessione, utile per tracciare il consumo totale
- **Utilizzo corrente** (`current_usage`): conteggi dei token dall'ultima chiamata API, usa questo per una percentuale di contesto accurata poiché riflette lo stato effettivo del contesto

L'oggetto `current_usage` contiene:

- `input_tokens` : token di input nel contesto corrente
- `output_tokens` : token di output generati
- `cache_creation_input_tokens` : token scritti nella cache
- `cache_read_input_tokens` : token letti dalla cache

Il campo `used_percentage` viene calcolato solo dai token di input: `input_tokens + cache_creation_input_tokens + cache_read_input_tokens` . Non include `output_tokens` .

Se calcoli manualmente la percentuale di contesto da `current_usage` , usa la stessa formula solo per l'input per corrispondere a `used_percentage` .

L'oggetto `current_usage` è `null` prima della prima chiamata API in una sessione.

Esempi

Questi esempi mostrano modelli comuni di barre di stato. Per usare qualsiasi esempio:

1. Salva lo script in un file come `~/.claude/statusline.sh` (o `.py` / `.js`)
2. Rendilo eseguibile: `chmod +x ~/.claude/statusline.sh`
3. Aggiungi il percorso alle tue impostazioni

Gli esempi Bash usano `jq` per analizzare JSON. Python e Node.js hanno l'analisi JSON integrata.

Utilizzo della finestra di contesto

Visualizza il modello corrente e l'utilizzo della finestra di contesto con una barra di progresso visiva. Ogni script legge JSON da stdin, estrae il campo `used_percentage` e costruisce una barra di 10 caratteri dove i blocchi pieni (▓) rappresentano l'utilizzo:

Una barra di stato che mostra il nome del modello e una barra di progresso con percentuale

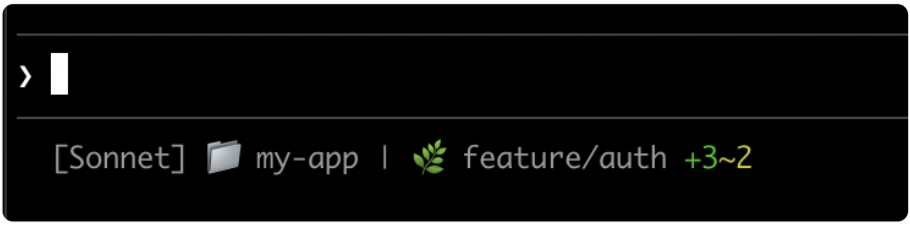
```bash
#!/bin/bash
## Read all of stdin into a variable
input=$(cat)

## 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}%`);
});
```

Stato git con colori

Mostra il ramo git con indicatori codificati a colori per i file in staging e modificati. Questo script usa codici di escape ANSI per i colori del terminale: `\033[32m` è verde, `\033[33m` è giallo e `\033[0m` ripristina il valore predefinito.



Una barra di stato che mostra il modello, la directory, il ramo git e indicatori codificati a colori per i file in staging e modificati

Ogni script verifica se la directory corrente è un repository git, conta i file in staging e modificati e visualizza indicatori codificati a colori:

```
#!/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
```

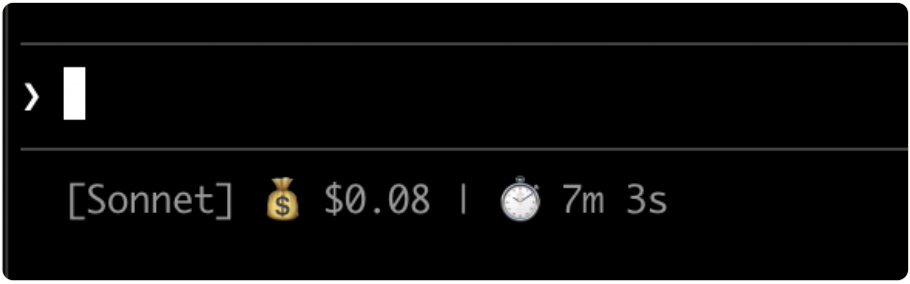
```
#!/usr/bin/env python3
import json, sys, subprocess, os

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}`);
    }
});
```

Tracciamento di costi e durata

Traccia i costi API della tua sessione e il tempo trascorso. Il campo `cost.total_cost_usd` accumula il costo di tutte le chiamate API nella sessione corrente. Il campo `cost.total_duration_ms` misura il tempo totale trascorso dall'inizio della sessione, mentre `cost.total_api_duration_ms` traccia solo il tempo trascorso in attesa delle risposte API.

Ogni script formatta il costo come valuta e converte i millisecondi in minuti e secondi:



Una barra di stato che mostra il nome del modello, il costo della sessione e la durata

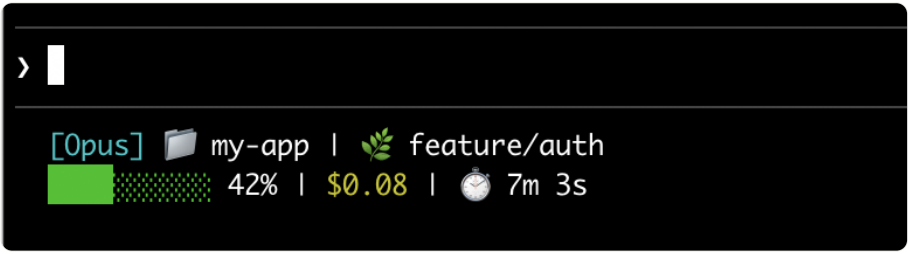
```
#!/bin/bash
input=$(cat)

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")
```

```javascript
#!/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`);
});
```

Visualizza più righe

Il tuo script può produrre più righe per creare una visualizzazione più ricca. Ogni istruzione `echo` produce una riga separata nell'area di stato.



Una barra di stato multi-riga che mostra il nome del modello, la directory, il ramo git sulla prima riga, e una barra di progresso dell'utilizzo del contesto con costo e durata sulla seconda riga

Questo esempio combina diverse tecniche: colori basati su soglie (verde sotto il 70%, giallo 70-89%, rosso 90%+), una barra di progresso e informazioni sul ramo git. Ogni istruzione `print` o `echo` crea una riga separata:

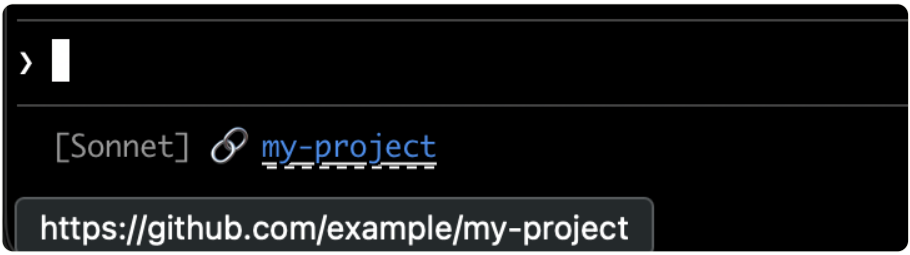
```
#!/bin/bash
input=$(cat)

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`);
});
```

Link cliccabili

Questo esempio crea un link cliccabile al tuo repository GitHub. Legge l'URL del remote git, converte il formato SSH in HTTPS con `sed` e avvolge il nome del repository nei codici di escape OSC 8. Tieni premuto Cmd (macOS) o Ctrl (Windows/Linux) e fai clic per aprire il link nel tuo browser.



Una barra di stato che mostra un link cliccabile a un repository GitHub

Ogni script ottiene l'URL del remote git, converte il formato SSH in HTTPS e avvolge il nome del repository nei codici di escape OSC 8. La versione Bash usa `printf '%b'` che interpreta gli escape di backslash in modo più affidabile rispetto a `echo -e` su diverse shell:

```
#!/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
```

```python
#!/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}]`);
    }
});
```

Memorizza nella cache le operazioni costose

Il tuo script della barra di stato viene eseguito frequentemente durante le sessioni attive. Comandi come `git status` o `git diff` possono essere lenti, specialmente in repository di grandi dimensioni. Questo esempio memorizza nella cache le informazioni git in un file temporaneo e le aggiorna solo ogni 5 secondi.

Usa un nome di file stabile e fisso per il file di cache come `/tmp/statusline-git-cache`. Ogni invocazione della barra di stato viene eseguita come un nuovo processo, quindi gli identificatori basati su processi come `$$`, `os.getpid()` o `process.pid` producono un valore diverso ogni volta e la cache non viene mai riutilizzata.

Ogni script verifica se il file di cache è mancante o più vecchio di 5 secondi prima di eseguire i comandi git:

```
#!/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}`);
    }
});
```

Configurazione Windows

Su Windows, Claude Code esegue i comandi della barra di stato tramite Git Bash. Puoi invocare PowerShell da quella shell:

```
{
  "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]"
}
```

Oppure esegui uno script Bash direttamente:

```
{
  "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]"
```

Suggerimenti

- **Testa con input simulato**: `echo '{"model":{"display_name":"Opus"},"context_window":{"used_percentage":25}}' | ./statusline.sh`
- **Mantieni l'output breve**: la barra di stato ha una larghezza limitata, quindi l'output lungo potrebbe essere troncato o andare a capo in modo sgradevole
- **Memorizza nella cache le operazioni lente**: il tuo script viene eseguito frequentemente durante le sessioni attive, quindi comandi come `git status` possono causare lag. Vedi l'esempio di caching per come gestire questo.

Progetti della comunità come ccstatusline e starship-claude forniscono configurazioni precostruite con temi e funzionalità aggiuntive.

Risoluzione dei problemi

La barra di stato non appare

- Verifica che il tuo script sia eseguibile: `chmod +x ~/.claude/statusline.sh`
- Controlla che il tuo script stampi su stdout, non su stderr
- Esegui il tuo script manualmente per verificare che produca output
- Se `disableAllHooks` è impostato su `true` nelle tue impostazioni, anche la barra di stato è disabilitata. Rimuovi questa impostazione o impostala su `false` per riabilitarla.

- Esegui `claude --debug` per registrare il codice di uscita e stderr dalla prima invocazione della barra di stato in una sessione
- Chiedi a Claude di leggere il tuo file di impostazioni ed eseguire il comando `statusLine` direttamente per far emergere gli errori

La barra di stato mostra `--` o valori vuoti

- I campi potrebbero essere `null` prima che la prima risposta API si completi
- Gestisci i valori null nel tuo script con fallback come `// 0` in jq
- Riavvia Claude Code se i valori rimangono vuoti dopo più messaggi

La percentuale di contesto mostra valori inaspettati

- Usa `used_percentage` per uno stato di contesto accurato piuttosto che i totali cumulativi
- `total_input_tokens` e `total_output_tokens` sono cumulativi nella sessione e potrebbero superare la dimensione della finestra di contesto
- La percentuale di contesto potrebbe differire dall'output `/context` a causa di quando ciascuno viene calcolato

I link OSC 8 non sono cliccabili

- Verifica che il tuo terminale supporti i hyperlink OSC 8 (iTerm2, Kitty, WezTerm)
- Terminal.app non supporta i link cliccabili
- Le sessioni SSH e tmux potrebbero eliminare le sequenze OSC a seconda della configurazione
- Se le sequenze di escape appaiono come testo letterale come `\e]8;;`, usa `printf '%b'` invece di `echo -e` per una gestione più affidabile degli escape

Glitch di visualizzazione con sequenze di escape

- Le sequenze di escape complesse (colori ANSI, link OSC 8) possono occasionalmente causare output corrotto se si sovrappongono ad altri aggiornamenti dell'interfaccia utente
- Se vedi testo corrotto, prova a semplificare il tuo script in output di testo semplice
- Le barre di stato multi-riga con codici di escape sono più soggette a problemi di rendering rispetto al testo semplice su una sola riga

Errori di script o blocchi

- Gli script che escono con codici diversi da zero o non producono output causano il vuoto della barra di stato

- Gli script lenti bloccano l'aggiornamento della barra di stato fino al completamento. Mantieni gli script veloci per evitare output obsoleto.
- Se un nuovo aggiornamento si attiva mentre uno script lento è in esecuzione, lo script in corso viene annullato
- Testa il tuo script indipendentemente con input simulato prima di configurarlo

Le notifiche condividono la riga della barra di stato

- Le notifiche di sistema come errori del server MCP, aggiornamenti automatici e avvisi di token vengono visualizzate sul lato destro della stessa riga della tua barra di stato
- L'abilitazione della modalità verbose aggiunge un contatore di token a questa area
- Su terminali stretti, queste notifiche potrebbero troncare l'output della tua barra di stato

Part 13: Troubleshooting & Changelog

Troubleshooting

> *Scopri soluzioni ai problemi comuni con l'installazione e l'utilizzo di Claude Code.*

Risolvi i problemi di installazione

> **Tip:**
>
> Se preferisci evitare completamente il terminale, l'app Claude Code Desktop ti consente di installare e utilizzare Claude Code tramite un'interfaccia grafica. Scaricala per macOS o Windows e inizia a programmare senza alcuna configurazione da riga di comando.

Trova il messaggio di errore o il sintomo che stai riscontrando:

Cosa vedi	Soluzione
`command not found: claude` o `'claude' is not recognized`	Correggi il tuo PATH
`syntax error near unexpected token '<'`	Lo script di installazione restituisce HTML
`curl: (56) Failure writing output to destination`	Scarica lo script prima, poi eseguilo
`Killed` durante l'installazione su Linux	Aggiungi spazio di swap per server con poca memoria
`TLS connect error` o `SSL/TLS secure channel`	Aggiorna i certificati CA
`Failed to fetch version` o impossibile raggiungere il server di download	Controlla la connettività di rete e le impostazioni proxy

Cosa vedi	Soluzione
`irm is not recognized` o `&& is not valid`	Usa il comando corretto per la tua shell
`Claude Code on Windows requires git-bash`	Installa o configura Git Bash
`Error loading shared library`	Variante binaria sbagliata per il tuo sistema
`Illegal instruction` su Linux	Mancata corrispondenza dell'architettura
`dyld: cannot load` o `Abort trap` su macOS	Incompatibilità binaria
`Invoke-Expression: Missing argument in parameter list`	Lo script di installazione restituisce HTML
`App unavailable in region`	Claude Code non è disponibile nel tuo paese. Vedi paesi supportati.
`unable to get local issuer certificate`	Configura i certificati CA aziendali
`OAuth error` o `403 Forbidden`	Correggi l'autenticazione

Se il tuo problema non è elencato, segui questi passaggi diagnostici.

Esegui il debug dei problemi di installazione

Controlla la connettività di rete

Il programma di installazione scarica da `storage.googleapis.com`. Verifica di poterlo raggiungere:

```
curl -sI https://storage.googleapis.com
```

Se questo fallisce, la tua rete potrebbe bloccare la connessione. Le cause comuni sono:

- Firewall aziendali o proxy che bloccano Google Cloud Storage
- Restrizioni di rete regionali: prova una VPN o una rete alternativa
- Problemi TLS/SSL: aggiorna i certificati CA del tuo sistema, oppure controlla se `HTTPS_PROXY` è configurato

Se sei dietro un proxy aziendale, imposta `HTTPS_PROXY` e `HTTP_PROXY` all'indirizzo del tuo proxy prima di installare. Chiedi al tuo team IT l'URL del proxy se non lo conosci, oppure controlla le impostazioni del proxy del tuo browser.

Questo esempio imposta entrambe le variabili proxy, quindi esegue il programma di installazione attraverso il tuo proxy:

```
export HTTP_PROXY=http://proxy.example.com:8080
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

Verifica il tuo PATH

Se l'installazione è riuscita ma ricevi un errore `command not found` o `not recognized` quando esegui `claude`, la directory di installazione non è nel tuo PATH. La tua shell cerca i programmi nelle directory elencate in PATH, e il programma di installazione posiziona `claude` in `~/.local/bin/claude` su macOS/Linux o `%USERPROFILE%\.local\bin\claude.exe` su Windows.

Controlla se la directory di installazione è nel tuo PATH elencando le voci di PATH e filtrando per `local/bin`:

macOS/Linux

```
echo $PATH | tr ':' '\n' | grep local/bin
```

Se non c'è output, la directory manca. Aggiungila alla configurazione della tua shell:

```
## Zsh (default su macOS)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc

## Bash (default su Linux)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

In alternativa, chiudi e riapri il tuo terminale.

Verifica che la correzione abbia funzionato:

```
claude --version
```

Windows PowerShell

```
$env:PATH -split ';' | Select-String 'local\\bin'
```

Se non c'è output, aggiungi la directory di installazione al tuo User PATH:

```
$currentPath = [Environment]::GetEnvironmentVariable('PATH', 'User')
[Environment]::SetEnvironmentVariable('PATH', "$currentPath;$env:USERPROFILE\.local\bin", 'User')
```

Riavvia il tuo terminale affinché la modifica abbia effetto.

Verifica che la correzione abbia funzionato:

```
claude --version
```

Windows CMD

```
echo %PATH% | findstr /i "local\bin"
```

Se non c'è output, apri Impostazioni di sistema, vai a Variabili di ambiente, e aggiungi `%USERPROFILE%\.local\bin` alla tua variabile User PATH. Riavvia il tuo terminale.

Verifica che la correzione abbia funzionato:

```
claude --version
```

Controlla le installazioni in conflitto

Più installazioni di Claude Code possono causare mancate corrispondenze di versione o comportamenti inaspettati. Controlla cosa è installato:

macOS/Linux

Elenca tutti i binari `claude` trovati nel tuo PATH:

```
which -a claude
```

Controlla se sono presenti le versioni del programma di installazione nativo e npm:

```
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"
```

Se trovi più installazioni, mantieni solo una. L'installazione nativa in `~/.local/bin/claude` è consigliata. Rimuovi eventuali installazioni extra:

Disinstalla un'installazione globale npm:

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

Rimuovi un'installazione Homebrew su macOS:

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

Controlla i permessi della directory

Il programma di installazione ha bisogno di accesso in scrittura a `~/.local/bin/` e `~/.claude/`. Se l'installazione fallisce con errori di permesso, controlla se queste directory sono scrivibili:

```
test -w ~/.local/bin && echo "writable" || echo "not writable"
test -w ~/.claude && echo "writable" || echo "not writable"
```

Se una delle directory non è scrivibile, crea la directory di installazione e imposta il tuo utente come proprietario:

```
sudo mkdir -p ~/.local/bin
sudo chown -R $(whoami) ~/.local
```

Verifica che il binario funzioni

Se `claude` è installato ma si arresta in modo anomalo o si blocca all'avvio, esegui questi controlli per restringere la causa.

Conferma che il binario esiste ed è eseguibile:

```
ls -la $(which claude)
```

Su Linux, controlla le librerie condivise mancanti. Se `ldd` mostra librerie mancanti, potrebbe essere necessario installare pacchetti di sistema. Su Alpine Linux e altre distribuzioni basate su musl, vedi Configurazione di Alpine Linux.

```
ldd $(which claude) | grep "not found"
```

Esegui un rapido controllo di sanità mentale che il binario possa essere eseguito:

```
claude --version
```

Problemi di installazione comuni

Questi sono i problemi di installazione più frequentemente riscontrati e le loro soluzioni.

Lo script di installazione restituisce HTML invece di uno script shell

Quando esegui il comando di installazione, potresti vedere uno di questi errori:

```
bash: line 1: syntax error near unexpected token `<'
bash: line 1: `<!DOCTYPE html>'
```

Su PowerShell, lo stesso problema appare come:

```
Invoke-Expression: Missing argument in parameter list.
```

Questo significa che l'URL di installazione ha restituito una pagina HTML invece dello script di installazione. Se la pagina HTML dice "App unavailable in region," Claude Code non è disponibile nel tuo paese. Vedi paesi supportati.

Altrimenti, questo può accadere a causa di problemi di rete, routing regionale, o un'interruzione temporanea del servizio.

Soluzioni:

1. **Usa un metodo di installazione alternativo**:
   Su macOS o Linux, installa tramite Homebrew:

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

Su Windows, installa tramite WinGet:

```
winget install Anthropic.ClaudeCode
```

1. **Riprova dopo alcuni minuti**: il problema è spesso temporaneo. Aspetta e prova di nuovo il comando originale.

`command not found: claude` dopo l'installazione

L'installazione è terminata ma `claude` non funziona. L'errore esatto varia in base alla piattaforma:

Piatta-forma	Messaggio di errore
macOS	`zsh: command not found: claude`
Linux	`bash: claude: command not found`

Piattaforma	Messaggio di errore
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`

Questo significa che la directory di installazione non è nel percorso di ricerca della tua shell. Vedi Verifica il tuo PATH per la correzione su ogni piattaforma.

`curl: (56) Failure writing output to destination`

Il comando `curl ... | bash` scarica lo script e lo passa direttamente a Bash per l'esecuzione usando una pipe (`|`). Questo errore significa che la connessione si è interrotta prima che lo script finisse di scaricarsi. Le cause comuni includono interruzioni di rete, il download bloccato a metà flusso, o limiti di risorse di sistema.

Soluzioni:

1. **Controlla la stabilità della rete**: i binari di Claude Code sono ospitati su Google Cloud Storage. Testa di poter raggiungerlo:

```
curl -fsSL https://storage.googleapis.com -o /dev/null
```

Se il comando si completa silenziosamente, la tua connessione è buona e il problema è probabilmente intermittente. Riprova il comando di installazione. Se vedi un errore, la tua rete potrebbe bloccare il download.

1. **Prova un metodo di installazione alternativo**:
   Su macOS o Linux:

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

Su Windows:

```
winget install Anthropic.ClaudeCode
```

Errori di connessione TLS o SSL

Errori come `curl: (35) TLS connect error`, `schannel: next InitializeSecurityContext failed`, o il `Could not establish trust relationship for the SSL/TLS secure channel` di PowerShell indicano fallimenti dell'handshake TLS.

Soluzioni:

1. **Aggiorna i certificati CA del tuo sistema**:
   Su Ubuntu/Debian:

```
sudo apt-get update && sudo apt-get install ca-certificates
```

Su macOS tramite Homebrew:

```
brew install ca-certificates
```

1. **Su Windows, abilita TLS 1.2** in PowerShell prima di eseguire il programma di installazione:

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
irm https://claude.ai/install.ps1 | iex
```

1. **Controlla l'interferenza del proxy o del firewall**: i proxy aziendali che eseguono l'ispezione TLS possono causare questi errori, incluso `unable to get local issuer certificate`. Imposta `NODE_EXTRA_CA_CERTS` sul tuo bundle di certificati CA aziendale:

```
export NODE_EXTRA_CA_CERTS=/path/to/corporate-ca.pem
```

Chiedi al tuo team IT il file del certificato se non lo hai. Puoi anche provare su una connessione diretta per confermare che il proxy è la causa.

`Failed to fetch version from storage.googleapis.com`

Il programma di installazione non ha potuto raggiungere il server di download. Questo in genere significa che `storage.googleapis.com` è bloccato sulla tua rete.

Soluzioni:

1. **Testa la connettività direttamente**:

```
curl -sI https://storage.googleapis.com
```

1. **Se dietro un proxy**, imposta `HTTPS_PROXY` in modo che il programma di installazione possa instradarlo attraverso. Vedi configurazione del proxy per i dettagli.

```
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

1. **Se su una rete ristretta**, prova una rete diversa o una VPN, oppure usa un metodo di installazione alternativo:

   Su macOS o Linux:

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

Su Windows:

```
winget install Anthropic.ClaudeCode
```

Windows: `irm` o `&&` non riconosciuto

Se vedi `'irm' is not recognized` o `The token '&&' is not valid`, stai eseguendo il comando sbagliato per la tua shell.

- `irm` **non riconosciuto**: sei in CMD, non in PowerShell. Hai due opzioni:

  Apri PowerShell cercando “PowerShell” nel menu Start, quindi esegui il comando di installazione originale:

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

Oppure rimani in CMD e usa il programma di installazione CMD:

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

- `&&` **non valido**: sei in PowerShell ma hai eseguito il comando del programma di installazione CMD. Usa il programma di installazione PowerShell:

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

Installazione interrotta su server Linux con poca memoria

Se vedi `Killed` durante l'installazione su un VPS o un'istanza cloud:

```
Setting up Claude Code...
Installing Claude Code native build latest...
bash: line 142: 34803 Killed    "$binary_path" install ${TARGET:+"$TARGET"}
```

Il killer OOM di Linux ha terminato il processo perché il sistema ha esaurito la memoria. Claude Code richiede almeno 4 GB di RAM disponibile.

Soluzioni:

1. **Aggiungi spazio di swap** se il tuo server ha RAM limitata. Lo swap utilizza lo spazio su disco come memoria di overflow, consentendo al programma di installazione di completarsi anche con poca RAM fisica.
   Crea un file di swap da 2 GB e abilitalo:

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```

Quindi riprova l'installazione:

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

1. **Chiudi altri processi** per liberare memoria prima di installare.
2. **Usa un'istanza più grande** se possibile. Claude Code richiede almeno 4 GB di RAM.

L'installazione si blocca in Docker

Quando installi Claude Code in un contenitore Docker, installare come root in `/` può causare blocchi.

Soluzioni:

1. **Imposta una directory di lavoro** prima di eseguire il programma di installazione. Quando eseguito da `/`, il programma di installazione scansiona l'intero filesystem, il che causa un utilizzo eccessivo della memoria. Impostare `WORKDIR` limita la scansione a una piccola directory:

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

1. **Aumenta i limiti di memoria di Docker** se usi Docker Desktop:

```
docker build --memory=4g .
```

Windows: Claude Desktop sostituisce il comando CLI `claude`

Se hai installato una versione precedente di Claude Desktop, potrebbe registrare un `Claude.exe` nella directory `WindowsApps` che ha priorità nel PATH rispetto a Claude Code CLI. Eseguire `claude` apre l'app Desktop invece della CLI.

Aggiorna Claude Desktop all'ultima versione per risolvere questo problema.

Windows: "Claude Code on Windows requires git-bash"

Claude Code su Windows nativo ha bisogno di Git for Windows, che include Git Bash.

Se Git non è installato, scaricalo e installalo da git-scm.com/downloads/win. Durante la configurazione, seleziona "Add to PATH." Riavvia il tuo terminale dopo l'installazione.

Se Git è già installato ma Claude Code ancora non riesce a trovarlo, imposta il percorso nel tuo file settings.json:

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

Se il tuo Git è installato da qualche altra parte, trova il percorso eseguendo `where.exe git` in PowerShell e usa il percorso `bin\bash.exe` da quella directory.

Linux: variante binaria sbagliata installata (mancata corrispondenza musl/glibc)

Se vedi errori su librerie condivise mancanti come `libstdc++.so.6` o `libgcc_s.so.1` dopo l'installazione, il programma di installazione potrebbe aver scaricato la variante binaria sbagliata per il tuo sistema.

```
Error loading shared library libstdc++.so.6: No such file or directory
```

Questo può accadere su sistemi basati su glibc che hanno pacchetti di cross-compilazione musl installati, causando al programma di installazione di rilevare erroneamente il sistema come musl.

Soluzioni:

1. **Controlla quale libc usa il tuo sistema**:

```
ldd /bin/ls | head -1
```

Se mostra `linux-vdso.so` o riferimenti a `/lib/x86_64-linux-gnu/`, sei su glibc. Se mostra `musl`, sei su musl.

1. **Se sei su glibc ma hai il binario musl**, rimuovi l'installazione e reinstalla. Puoi anche scaricare manualmente il binario corretto dal bucket GCS in `https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json`. Apri un problema su GitHub con l'output di `ldd /bin/ls` e `ls /lib/libc.musl*`.
2. **Se sei effettivamente su musl** (Alpine Linux), installa i pacchetti richiesti:

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

`Illegal instruction` su Linux

Se il programma di installazione stampa `Illegal instruction` invece del messaggio `Killed` dell'OOM, il binario scaricato non corrisponde all'architettura della tua CPU. Questo accade comunemente su server ARM che ricevono un binario x86, o su CPU più vecchie che mancano di set di istruzioni richiesti.

```
bash: line 142: 2238232 Illegal instruction    "$binary_path" install ${TARGET:
+"$TARGET"}
```

Soluzioni:

1. **Verifica la tua architettura**:

```
uname -m
```

`x86_64` significa 64-bit Intel/AMD, `aarch64` significa ARM64. Se il binario non corrisponde, apri un problema su GitHub con l'output.

1. **Prova un metodo di installazione alternativo** mentre il problema dell'architettura viene risolto:

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

`dyld: cannot load` su macOS

Se vedi `dyld: cannot load` o `Abort trap: 6` durante l'installazione, il binario è incompatibile con la tua versione di macOS o hardware.

```
dyld: cannot load 'claude-2.1.42-darwin-x64' (load command 0x80000034 is unknown)
Abort trap: 6
```

Soluzioni:

1. **Controlla la tua versione di macOS**: Claude Code richiede macOS 13.0 o successivo. Apri il menu Apple e seleziona About This Mac per controllare la tua versione.
2. **Aggiorna macOS** se sei su una versione precedente. Il binario utilizza comandi di caricamento che le versioni precedenti di macOS non supportano.
3. **Prova Homebrew** come metodo di installazione alternativo:

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

Problemi di installazione su Windows: errori in WSL

Potresti incontrare i seguenti problemi in WSL:

Problemi di rilevamento del sistema operativo/piattaforma: se ricevi un errore durante l'installazione, WSL potrebbe utilizzare npm di Windows. Prova:

- Esegui `npm config set os linux` prima dell'installazione
- Installa con `npm install -g @anthropic-ai/claude-code --force --no-os-check`. Non usare `sudo`.

Errori Node non trovato: se vedi `exec: node: not found` quando esegui `claude`, il tuo ambiente WSL potrebbe utilizzare un'installazione di Node.js di Windows. Puoi confermarlo con `which npm` e `which node`, che dovrebbero puntare a percorsi Linux che iniziano con `/usr/` piuttosto che `/mnt/c/`. Per risolvere questo, prova a installare Node tramite il gestore di pacchetti della tua distribuzione Linux o tramite `nvm`.

Conflitti di versione nvm: se hai nvm installato sia in WSL che in Windows, potresti riscontrare conflitti di versione quando cambi versioni di Node in WSL. Questo accade perché WSL importa il PATH di Windows per impostazione predefinita, causando a npm/nvm di Windows di avere priorità rispetto all'installazione di WSL.

Puoi identificare questo problema da:

- Eseguire `which npm` e `which node` - se puntano a percorsi di Windows (che iniziano con `/mnt/c/`), vengono utilizzate le versioni di Windows
- Sperimentare funzionalità interrotte dopo aver cambiato versioni di Node con nvm in WSL

Per risolvere questo problema, correggi il tuo PATH Linux per assicurarti che le versioni Linux di node/npm abbiano priorità:

Soluzione primaria: assicurati che nvm sia correttamente caricato nella tua shell

La causa più comune è che nvm non è caricato in shell non interattive. Aggiungi quanto segue al tuo file di configurazione della shell (`~/.bashrc`, `~/.zshrc`, ecc.):

```
## Carica nvm se esiste
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

Oppure esegui direttamente nella tua sessione corrente:

```
source ~/.nvm/nvm.sh
```

Alternativa: regola l'ordine di PATH

Se nvm è correttamente caricato ma i percorsi di Windows hanno ancora priorità, puoi esplicitamente anteporre i tuoi percorsi Linux a PATH nella configurazione della tua shell:

```
export PATH="$HOME/.nvm/versions/node/$(node -v)/bin:$PATH"
```

> **Warning:**
>
> Evita di disabilitare l'importazione del PATH di Windows tramite `appendWindowsPath = false` poiché questo interrompe la capacità di chiamare eseguibili di Windows da WSL. Allo stesso modo, evita di disinstallare Node.js da Windows se lo usi per lo sviluppo di Windows.

Configurazione sandbox WSL2

Il Sandboxing è supportato su WSL2 ma richiede l'installazione di pacchetti aggiuntivi. Se vedi un errore come "Sandbox requires socat and bubblewrap" quando esegui `/sandbox`, installa le dipendenze:

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

WSL1 non supporta il sandboxing. Se vedi "Sandboxing requires WSL2", devi eseguire l'upgrade a WSL2 o eseguire Claude Code senza sandboxing.

Errori di permesso durante l'installazione

Se il programma di installazione nativo fallisce con errori di permesso, la directory di destinazione potrebbe non essere scrivibile. Vedi Controlla i permessi della directory.

Se hai precedentemente installato con npm e stai riscontrando errori di permesso specifici di npm, passa al programma di installazione nativo:

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

Permessi e autenticazione

Queste sezioni affrontano i fallimenti di accesso, i problemi di token e il comportamento dei prompt di permesso.

Prompt di permesso ripetuti

Se ti trovi a dover approvare ripetutamente gli stessi comandi, puoi consentire a strumenti specifici di essere eseguiti senza approvazione utilizzando il comando `/permissions`. Vedi Documentazione Permissions.

Problemi di autenticazione

Se stai riscontrando problemi di autenticazione:

1. Esegui `/logout` per disconnetterti completamente
2. Chiudi Claude Code
3. Riavvia con `claude` e completa di nuovo il processo di autenticazione

Se il browser non si apre automaticamente durante l'accesso, premi `c` per copiare l'URL OAuth negli appunti, quindi incollalo nel tuo browser manualmente.

Errore OAuth: codice non valido

Se vedi `OAuth error: Invalid code. Please make sure the full code was copied`, il codice di accesso è scaduto o è stato troncato durante la copia-incolla.

Soluzioni:

- Premi Invio per riprovare e completa l'accesso rapidamente dopo che il browser si apre
- Digita `c` per copiare l'URL completo se il browser non si apre automaticamente
- Se usi una sessione remota/SSH, il browser potrebbe aprirsi sulla macchina sbagliata. Copia l'URL visualizzato nel terminale e aprilo nel tuo browser locale.

403 Forbidden dopo l'accesso

Se vedi `API Error: 403 {"error":{"type":"forbidden","message":"Request not allowed"}}` dopo l'accesso:

- **Utenti Claude Pro/Max**: verifica che il tuo abbonamento sia attivo in claude.ai/settings

- **Utenti Console**: conferma che il tuo account abbia il ruolo “Claude Code” o “Developer” assegnato dal tuo amministratore
- **Dietro un proxy**: i proxy aziendali possono interferire con le richieste API. Vedi configurazione di rete per la configurazione del proxy.

L’accesso OAuth fallisce in WSL2

L’accesso basato su browser in WSL2 potrebbe fallire se WSL non riesce ad aprire il tuo browser di Windows. Imposta la variabile di ambiente `BROWSER`:

```
export BROWSER="/mnt/c/Program Files/Google/Chrome/Application/chrome.exe"
claude
```

Oppure copia l’URL manualmente: quando appare il prompt di accesso, premi `c` per copiare l’URL OAuth, quindi incollalo nel tuo browser di Windows.

“Not logged in” o token scaduto

Se Claude Code ti chiede di accedere di nuovo dopo una sessione, il tuo token OAuth potrebbe essere scaduto.

Esegui `/login` per autenticarti di nuovo. Se questo accade frequentemente, controlla che l’orologio di sistema sia accurato, poiché la convalida del token dipende da timestamp corretti.

Posizioni dei file di configurazione

Claude Code memorizza la configurazione in diverse posizioni:

File	Scopo
`~/.claude/settings.json`	Impostazioni utente (permessi, hooks, override del modello)
`.claude/settings.json`	Impostazioni del progetto (controllate nel controllo del codice sorgente)
`.claude/settings.local.json`	Impostazioni del progetto locale (non committate)
`~/.claude.json`	Stato globale (tema, OAuth, server MCP)
`.mcp.json`	Server MCP del progetto (controllati nel controllo del codice sorgente)

File	Scopo
`managed-mcp.json`	Server MCP gestiti
Impostazioni gestite	Impostazioni gestite (gestite dal server, politiche MDM/a livello di sistema operativo, o basate su file)

Su Windows, `~` si riferisce alla tua directory home dell'utente, come `C:\Users\YourName`.

Per i dettagli sulla configurazione di questi file, vedi Settings e MCP.

Ripristino della configurazione

Per ripristinare Claude Code alle impostazioni predefinite, puoi rimuovere i file di configurazione:

```
## Ripristina tutte le impostazioni utente e lo stato
rm ~/.claude.json
rm -rf ~/.claude/

## Ripristina le impostazioni specifiche del progetto
rm -rf .claude/
rm .mcp.json
```

> **Warning:**
>
> Questo rimuoverà tutte le tue impostazioni, configurazioni del server MCP, e cronologia della sessione.

Prestazioni e stabilità

Queste sezioni affrontano i problemi relativi all'utilizzo delle risorse, alla reattività e al comportamento della ricerca.

Utilizzo elevato di CPU o memoria

Claude Code è progettato per funzionare con la maggior parte degli ambienti di sviluppo, ma potrebbe consumare risorse significative durante l'elaborazione di grandi basi di codice. Se stai riscontrando problemi di prestazioni:

1. Usa `/compact` regolarmente per ridurre la dimensione del contesto
2. Chiudi e riavvia Claude Code tra i compiti principali

3. Considera di aggiungere grandi directory di build al tuo file `.gitignore`

Il comando si blocca o si congela

Se Claude Code sembra non reattivo:

1. Premi Ctrl+C per tentare di annullare l'operazione corrente
2. Se non reattivo, potrebbe essere necessario chiudere il terminale e riavviare

Problemi di ricerca e scoperta

Se lo strumento Search, le menzioni `@file`, gli agenti personalizzati e le skill personalizzate non funzionano, installa il sistema `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
```

Quindi imposta `USE_BUILTIN_RIPGREP=0` nel tuo ambiente.

Risultati di ricerca lenti o incompleti su WSL

Le penalità di prestazioni di lettura del disco quando lavori tra file system su WSL possono risultare in meno corrispondenze del previsto quando usi Claude Code su WSL. La ricerca funziona ancora, ma restituisce meno risultati rispetto a un file system nativo.

> **Note:**
>
> `/doctor` mostrerà Search come OK in questo caso.

Soluzioni:

1. **Invia ricerche più specifiche**: riduci il numero di file cercati specificando directory o tipi di file: “Search for JWT validation logic in the auth-service package” o “Find use of md5 hash in JS files”.
2. **Sposta il progetto al file system Linux**: se possibile, assicurati che il tuo progetto si trovi sul file system Linux (`/home/`) piuttosto che sul file system di Windows (`/mnt/c/`).
3. **Usa Windows nativo**: considera di eseguire Claude Code nativamente su Windows invece che tramite WSL, per migliori prestazioni del file system.

Problemi di integrazione IDE

Se Claude Code non si connette al tuo IDE o si comporta inaspettatamente all’interno di un terminale IDE, prova le soluzioni di seguito.

IDE JetBrains non rilevato su WSL2

Se stai usando Claude Code su WSL2 con IDE JetBrains e ricevi errori “No available IDEs detected”, questo è probabilmente dovuto alla configurazione di rete di WSL2 o al Windows Firewall che blocca la connessione.

Modalità di rete WSL2

WSL2 utilizza il networking NAT per impostazione predefinita, il che può impedire il rilevamento dell’IDE. Hai due opzioni:

Opzione 1: Configura Windows Firewall (consigliato)

1. Trova il tuo indirizzo IP WSL2:

```
wsl hostname -I
## Esempio di output: 172.21.123.45
```

1. Apri PowerShell come amministratore e crea una regola del firewall:

```
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
```

Regola l'intervallo IP in base alla tua subnet WSL2 dal passaggio 1.

1. Riavvia sia il tuo IDE che Claude Code

Opzione 2: Passa al networking con mirroring

Aggiungi a `.wslconfig` nella tua directory utente di Windows:

```
[wsl2]
networkingMode=mirrored
```

Quindi riavvia WSL con `wsl --shutdown` da PowerShell.

> **Note:**
>
> Questi problemi di rete interessano solo WSL2. WSL1 utilizza direttamente la rete dell'host e non richiede queste configurazioni.

Per ulteriori suggerimenti di configurazione di JetBrains, vedi la guida IDE JetBrains.

Segnala problemi di integrazione IDE su Windows

Se stai riscontrando problemi di integrazione IDE su Windows, crea un problema con le seguenti informazioni:

- Tipo di ambiente: Windows nativo (Git Bash) o WSL1/WSL2
- Modalità di rete WSL, se applicabile: NAT o mirroring
- Nome e versione dell'IDE
- Versione dell'estensione/plugin di Claude Code
- Tipo di shell: Bash, Zsh, PowerShell, ecc.

Il tasto Esc non funziona nei terminali IDE JetBrains

Se stai usando Claude Code nei terminali JetBrains e il tasto `Esc` non interrompe l'agente come previsto, questo è probabilmente dovuto a uno scontro di scorciatoie da tastiera con i tasti di scelta rapida predefiniti di JetBrains.

Per risolvere questo problema:

1. Vai a Settings → Tools → Terminal
2. Uno di:
    - Deseleziona "Move focus to the editor with Escape", oppure

    - Fai clic su “Configure terminal keybindings” e elimina la scorciatoia “Switch focus to Editor”

3. Applica le modifiche

Questo consente al tasto `Esc` di interrompere correttamente le operazioni di Claude Code.

Problemi di formattazione Markdown

Claude Code a volte genera file markdown con tag di linguaggio mancanti sui recinti di codice, il che può influire sull’evidenziazione della sintassi e sulla leggibilità in GitHub, editor e strumenti di documentazione.

Tag di linguaggio mancanti nei blocchi di codice

Se noti blocchi di codice come questo nel markdown generato:

````
```
function example() {
  return "hello";
}
```text
````

Invece di blocchi correttamente taggati come:

````
```javascript
function example() {
  return "hello";
}
```text
````

Soluzioni:

1. **Chiedi a Claude di aggiungere tag di linguaggio**: richiedi “Add appropriate language tags to all code blocks in this markdown file.”
2. **Usa hook di post-elaborazione**: configura hook di formattazione automatica per rilevare e aggiungere tag di linguaggio mancanti. Vedi Auto-format code after edits per un esempio di hook PostToolUse di formattazione.
3. **Verifica manuale**: dopo aver generato file markdown, esamina la formattazione corretta dei blocchi di codice e richiedi correzioni se necessario.

Spaziatura e formattazione incoerenti

Se il markdown generato ha righe vuote eccessive o spaziatura incoerente:

Soluzioni:

1. **Richiedi correzioni di formattazione**: chiedi a Claude di "Fix spacing and formatting issues in this markdown file."
2. **Usa strumenti di formattazione**: configura hook per eseguire formattatori markdown come `prettier` o script di formattazione personalizzati su file markdown generati.
3. **Specifica preferenze di formattazione**: includi requisiti di formattazione nei tuoi prompt o nei file memory del progetto.

Riduci i problemi di formattazione markdown

Per minimizzare i problemi di formattazione:

- **Sii esplicito nelle richieste**: chiedi "properly formatted markdown with language-tagged code blocks"
- **Usa convenzioni di progetto**: documenta il tuo stile markdown preferito in `CLAUDE.md`
- **Configura hook di convalida**: usa hook di post-elaborazione per verificare e correggere automaticamente i problemi di formattazione comuni

Ottieni più aiuto

Se stai riscontrando problemi non affrontati qui:

1. Usa il comando `/bug` all'interno di Claude Code per segnalare i problemi direttamente ad Anthropic
2. Controlla il repository GitHub per i problemi noti
3. Esegui `/doctor` per diagnosticare i problemi. Controlla:
    - Tipo di installazione, versione e funzionalità di ricerca
    - Stato dell'aggiornamento automatico e versioni disponibili
    - File di impostazioni non validi (JSON malformato, tipi non corretti)
    - Errori di configurazione del server MCP
    - Problemi di configurazione delle scorciatoie da tastiera
    - Avvisi di utilizzo del contesto (file CLAUDE.md di grandi dimensioni, utilizzo elevato di token MCP, regole di permesso non raggiungibili)
    - Errori di caricamento di plugin e agenti

4. Chiedi a Claude direttamente sulle sue capacità e funzionalità - Claude ha accesso integrato alla sua documentazione